Endeca® URL Optimization API for Java

Developer's Guide

Version 2.1.0 • December 2011

Contents

Copyright and disclaimer

Preface

Endeca® InFront enables businesses to deliver targeted experiences for any customer, every time, in any channel. Utilizing all underlying product data and content, businesses are able to influence customer behavior regardless of where or how customers choose to engage — online, in-store, or on-the-go. And with integrated analytics and agile business-user tools, InFront solutions help businesses adapt to changing market needs, influence customer behavior across channels, and dynamically manage a relevant and targeted experience for every customer, every time.

InFront Workbench with Experience Manager provides a single, flexible platform to create, deliver, and manage content-rich, multichannel customer experiences. Experience Manager allows non-technical users to control how, where, when, and what type of content is presented in response to any search, category selection, or facet refinement.

At the core of InFront is the Endeca MDEX Engine,™ a hybrid search-analytical database specifically designed for high-performance exploration and discovery. InFront Integrator provides a set of extensible mechanisms to bring both structured data and unstructured content into the MDEX Engine from a variety of source systems. InFront Assembler dynamically assembles content from any resource and seamlessly combines it with results from the MDEX Engine.

These components — along with additional modules for SEO, Social, and Mobile channel support — make up the core of Endeca InFront, 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 Java.

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 Java, 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 the Endeca URL Optimization API.

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, aggregated records, etc.
- Configuring Endeca dimensions using Developer Studio
- Endeca Presentation API, specifically:
  - `UrlGen` class
  - `ENEQueryToolkit` class

- Guided Navigation classes, e.g. `DimVal`, `Dimension`, `DimLocation`, etc.

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

The Endeca Support Center provides registered users with important information regarding Endeca software, implementation questions, product and solution help, training and professional services consultation as well as overall news and updates from Endeca.

You can contact Endeca Standard Customer Support through the Support section of the Endeca Developer Network (EDeN) at *http://eden.endeca.com*.

Chapter 1

Installation

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

System requirements

Below is a list of minimum system requirements for the Endeca URL Optimization API for Java.

Software requirements

The Endeca URL Optimization API for Java requires the following software:

- Platform Services
- MDEX Engine
- The Endeca HTTP service or a standalone Java application server

Note: While the Endeca URL Optimization API reference Web application can be installed into any standalone Java application server, this guide assumes that you are installing into an existing instance of the Endeca HTTP service.

To determine the compatibility of the URL Optimization API with other Endeca installation packages, see the *Endeca InFront Compatibility Matrix* available on EDeN.

Supported operating systems

The URL Optimization API is supported on all hardware and operating system platforms that are supported by Endeca Platform Services.

The URL Optimization API for Java can be run on any of the supported platforms, using these versions of Java:

- Sun JDK 1.4.2, 5.0 (1.5), and 6.0 (1.6)
- IBM JDK 1.4.2, 5.0 (1.5), and 6.0 (1.6)

Installing the URL Optimization API

The URL Optimization API for Java is distributed as a zip file, `UrlOptimizationAPIJava-`*`version`*`.zip`.

The package unpacks into a self-contained directory structure tree:

```
Endeca\SEM\URL Optimization APIs\Java\version\
```

1. Extract the zip package using WinZip or an alternate decompression utility.

   Endeca recommends that you install the URL Optimization API to the same directory as your Endeca installation. For example:

Endeca location (6.x)	extract zip archive to:
C:\Endeca\PlatformServices*version*\	C:\
/usr/local/endeca/PlatformServices/*version*/	/usr/local/

2. Stop the Endeca HTTP service.
3. Navigate to the `\lib` subdirectory of your URL Optimization API installation directory.
   For example: `C:\Endeca\SEM\URL Optimization APIs\Java\version\lib`
4. Copy the `urlFormatterSeo.jar` and the `urlFormatterCore.jar` into the `WEB-INF/lib` subdirectory of your Web application directory.
   For example: `C:\Endeca\MyApps\WEB-INF\lib`
5. Add `urlFormatterSeo.jar` and `urlFormatterCore.jar` to your application's classpath.
6. Start the Endeca HTTP service.

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.

The `C:\Endeca\SEM\URLOptimizationAPIs\Java\version` directory is the root for URL Optimization API for Java. The directory contains the following components:

Component	Description
Root directory	Contains the release notes `README_UOJ.txt`.
`\doc`	Contains the *Licensing Guide* and generated API documentation for the core and SEO Java library files.
`\lib`	Contains the `urlFormatterSeo.jar` and the `urlFormatterCore.jar` files. These are the Java library files that contain the full implementation of the URL Optimization API. The `urlFormatterSeo.jar` contains the SEO-specific classes and is dependant upon the `urlFormatterCore.jar` which contains the basic formatting classes.
`\webapps`	Contains the `urlformatter_jspref.xml` context file as well as the

Component	Description
	`urlformatter_jspref` reference application subdirectory.
`\webapps\urlformatter_jspref`	Contains the reference Web application that demonstrates the full functionality of the URL Optimization API within the context of a traditional Endeca application. The sample URL configuration file (`urlconfig.xml`) is included in the `\WEB-INF` subdirectory.

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
```

About 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:8888/urlformatter_jspref/controller/Wine-Red/Region-Na¬pa/_/N-1z141vcZ66t`
- `http://localhost:8888/urlformatter_jspref/controller/Region-Napa/Wine-Red/_/N-1z141vcZ66t`

To the search engine, each version of the URL appears to be its own unique page with identical or near-identical content, 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.

By design, the URL Optimization API prevents the creation of syntactically different URLs by canonicalizing keywords, ensuring that equivalent pages have URLs with the same syntax even if they can be navigated to through different paths. You can choose from a number of configuration options to control the arrangement of keywords. For example, you can configure your `UrlFormatter` to arrange dimensions alphabetically in an ascending order:

- `http://localhost:8888/urlformatter_jspref/controller/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`.

Related Links

Canonicalization configuration options on page 41
You can customize the canonicalization of URLs for navigation pages by choosing a sort method, for example by dimension name or dimension ID, and then a sort direction.

Chapter 3

Setting up the Reference Application

The reference Web application included with the URL Optimization API is a means to demonstrate the functionality of the URL Optimization API. It contains a sample configuration that is specifically designed to work with the Endeca wine data set.

About the reference application

The URL Optimization API reference Web application demonstrates the basic capabilities of the URL Optimization API.

The URL Optimization API reference application is designed to show a typical approach to URL optimization. The reference application relies on the `urlconfig.xml` file, which uses the Spring Framework to configure optimized URLs. While the URL Optimization API does not require the Spring Framework, it supplies a convenient and flexible mechanism for configuring optimized URLs.

The reference application may be used as a starting point for your own application code. You can customize it to suit your data and business requirements and extend its functionality as needed.

Reference application prerequisites

Before installing the URL Optimization API reference application, you must have a properly configured MDEX Engine running with the Endeca wine data set.

For information about setting up the sample wine data project, please refer to the *Endeca Commerce Suite 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 21

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 21

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

Installing the reference application

While the Endeca URL Optimization API reference Web application can be installed into any standalone Java application server, this section assumes that you are installing into an existing instance of the Endeca Tools Service.

Before proceeding with the installation, you must have a properly configured MDEX Engine running with the wine data set.

The URL Optimization API Web application is distributed and unpacked with the zip file `UrlOptimizationAPIJava-`*`version`*`.zip`.

To install the reference Web application:

1. Stop the Endeca Tools Service.
2. Navigate to the `\lib\java` subdirectory of your Endeca installation, for example:
   `C:\Endeca\PlatformServices\`*`version`*`\lib\java`
3. Copy and transfer the `commons-logging-1.0.4.jar`, `endeca_navigation.jar`, and `endeca_logging.jar` files into the `\webapps\urlformatter_jspref\WEB-INF\lib` subdirectory of your URL Optimization API installation directory.
   For example: `C:\Endeca\SEO\URL Optimization APIs\Java\`*`version`*`\webapps\urlformatter_jspref\WEB-INF\lib`
4. Navigate to the `\webapps` subdirectory of your URL Optimization API installation.
   For example: `C:\Endeca\SEO\URL Optimization APIs\Java\`*`version`*`\webapps`
5. Copy and transfer the `urlformatter_jspref.xml` file into `\%ENDECA_TOOLS_CONF%\conf\Standalone\localhost`.
6. In the `urlformatter_jspref.xml` file, edit the value of the `docBase` attribute so that it points to the location of the `\webapps\urlformatter_jspref` subdirectory of the URL Optimization API installation.
   For example:

```
<Context
    path="/urlformatter_jspref"
   docBase="C:\Endeca\Solutions\urlOptimizationApiJava-1.2.0\webapps\url¬
formatter_jspref"
    debug="0"
    privileged="false"
/>
```

7. Start the Endeca Tools Service.

Testing your reference Web application

Once you have installed the URL Optimization API and configured the reference Web application, you can verify that the reference application URLs are optimized.

To test the reference application:

1. In your Web browser, navigate to the reference application.
   The default URL is *http://localhost:8006/urlformatter_jspref/controller*
2. If the MDEX Engine for your reference application is not running on the default host and port (`localhost:15000`), specify the correct host and port values.
3. Navigate to **Wine Type** > **Red**.
   You should see an optimized URL similar to the following: `http://localhost:8888/urlformatter_jspref/controller/Wine-Red/_/N-66t/Ne-4s8`

The traditional Endeca URL for the **Wine Type** > **Red** page displays as: `http://localhost:8888/endeca_jspref/controller.jsp?sid=122C75137F30&enePort=15000&Ne=6200&eneHost=localhost&N=8021`. Notice that the optimized URL includes the dimension names `Wine` and `Red`, and does not contain the host and port values or the session ID.

Chapter 4

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 *Endeca Developers Studio Help* documentation.

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 *Endeca Developers Studio Help* documentation.

Handling images and external JavaScript files

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.

The URL Optimization API reference Web application uses a constant base path for all URLs in order to ensure that external references do not break. To review this logic, open the `constants.jsp` file located in the `\webapps\urlformatter_jspref` subdirectory of your URL Optimization API installation. For example:

`C:\Endeca\SEO\URL Optimization APIs\Java\`*`version`*`\webapps\urlformatter_jspref\`

Locate the `FILE_PATH` code:

```
// Specifies path to image files, js files.
private static String FILE_PATH = "/urlformatter_jspref/";
```

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.

Endeca 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 5

Building URLs with the URL Optimization API

This section describes the basic tasks for using the URL Optimization API to build search engine-optimized URLs.

Core components in the URL Optimization API

The primary classes and interfaces of the URL Optimization API are `UrlState`, `UrlFormatter`, and `QueryBuilder`.

UrlState

A `UrlState` instance represents the URL, including any parameters, for a particular navigation state in your Endeca application. You typically create a `UrlState` by using a `UrlFormatter` to parse a URL string. You then inform the `UrlState` of the navigation state that it represents by passing it a set of Endeca query results. Once the `UrlState` is informed, you can modify it in order to generate URLs representing links to other states in your application, such as selecting refinements.

UrlFormatter

A `UrlFormatter` is responsible for parsing URL strings into `UrlState` objects and transforming `UrlState` objects back into URLs. The `SeoUrlFormatter` is a highly configurable implementation of `UrlFormatter` that parses and generates search engine-optimized URLs.

QueryBuilder

A `QueryBuilder` marshals `UrlState` objects into MDEX Engine queries. The `BasicQueryBuilder` is an implementation of `QueryBuilder` that creates `ENEQuery` objects from a given `UrlState`.

For further information about these and other classes in the URL Optimization API, please refer to the *URL Optimization API Reference*.

Overview of building URLs using the URL Optimization API

The URL Optimization API requires a different approach to building Endeca URLs than you would use to build URLs with the Endeca Presentation API.

The high-level process is as follows:

1. Set up your basic application configuration with a `BasicQueryBuilder` and `SeoUrlFormatter`.

   How you create and configure the `QueryBuilder` and `UrlFormatter` may vary depending on your application, but they should be should be scoped at a global or application level.

2. Handle requests by parsing the incoming query and sending it to an MDEX Engine.
3. Inform a `UrlState` object of the navigation state.
4. Modify the `UrlState` object by adding or removing URL parameters.
5. Generate a URL from the `UrlState`.

Parsing an incoming query and sending it to an MDEX Engine

Because it is possible for optimized URLs not to contain query string parameters (these parameters can be stored in the path), you cannot rely on the `UrlENEQuery` class to create an `ENEQuery` object from a URL.

Instead, use a `UrlFormatter` to parse the incoming request URL in order to populate the `UrlState` with the current URL query parameters, then use a `QueryBuilder` to create the `ENEQuery` from the `UrlState`.

To parse an incoming request and query an MDEX Engine:

1. Parse the request into a `UrlState` instance.

   For example:

   ```
   UrlState requestUrlState = urlFormatter.parseRequest(request);
   ```

2. Build an `ENEQuery` based on the `UrlState`.

   For example:

   ```
   ENEQuery eneQuery = queryBuilder.buildQuery(requestUrlState);
   ```

3. Execute the request and retrieve the results.

   For example:

   ```
   HttpENEConnection conn = new HttpENEConnection(mdexHost, mdexPort);
   ENEQueryResults eneQueryResults = conn.query(eneQuery);
   ```

Informing the UrlState of the navigation state

Informing is the process of providing the `UrlState` object with information about the current query results.

From this information, the `UrlState` object creates either a `NavStateUrlParam` if the query results are from a navigation query, an `ERecUrlParam` if the query results are from a record detail query, or an `AggrERecUrlParam` if the query results are from an aggregated record detail query.

The `SeoUrlFormatter` can use the extra information in these objects to generate customized URLs based on the current navigation state or properties and dimensions associated with these results.

To inform a `UrlState` of the current navigation state:

Add code similar to the following:

```
urlState.inform(eneQueryResults);
```

You can generate properly formatted URLs representing either the current navigation state, a record detail link, or an aggregated record detail link. Note that of these three possiblities, only the record detail link is guaranteed to be complete when calling `inform` on an empty `UrlState`. A navigation URL would be correct but, without further modification, only reflects the selected dimension values (the N parameter values). An aggregated record detail URL would not work without adding the required An and Au parameters.

The intent of the `inform()` method is to give the `UrlFormatter` and `UrlState` access to property and dimension information, not to copy your query. In some cases a complete query URL can only be created through a combination of using `UrlFormatter.parseRequest()` on the initial request and calling `UrlState.setParam()` as needed in addition to using `inform()`.

Creating link URLs from a UrlState

In order to create link URLs on a particular page to different navigation states within your application, you modify the `UrlState` and then transform the modified `UrlState` to a URL string.

This procedure assumes that you already have an informed `UrlState` that represents the current navigation state of your page.

To create a link URL:

1. Modify the `UrlState` to reflect a different navigation state in your application.

   For example, you can use the following to create a refinement link for a Guided Navigation component in your application:

   ```
   UrlState refinedUrlState =
          informedUrlState.selectRefinement(refDim, refDimVal, true);
   ```

   The final parameter indicates whether the modification should be performed on a cloned version of the current `UrlState`, and should typically be `true`. For instance, in the case of a Guided Navigation component, you would loop through the possible refinements and create a modified `UrlState` based on the current `UrlState` for each refinement link. If you wanted to select several refinements in the same URL, you would pass `false` as the value of this parameter.

   For further details about additional methods that can be used to modify a `UrlState`, please refer to the *URL Optimization API Reference*.

2. Generate the URL string from the modified `UrlState`.

   ```
   String refinedUrl = refinedUrlState.toString();
   ```

   The `UrlState.toString()` method calls the `formatString()` method of the `UrlFormatter` that constructed the `UrlState` instance.

Chapter 6

Configuring URLs

The following sections provide information about creating and using a URL configuration file similiar to the `urlconfig.xml` file included with the URL Optimization API to optimize your URLs. 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.

About the URL configuration file

The URL Optimization API reference application uses an XML file named `urlconfig.xml` to configure the format of the URLs that it generates.

The reference application uses the Spring Framework for this configuration file. Although the URL Optimization API does not require the Spring Framework, it supplies a convenient and flexible configuration mechanism. In addition, if you plan to use the Sitemap Generator with your application, Endeca strongly recommends using the `urlconfig.xml` file to configure your optimized URLs, because the Sitemap Generator relies on the same format for configuration. If you need further information about the Spring Framework syntax, please consult the documentation provided with the Spring Framework.

The URL configuration file contains basic configurations for the following objects:

- A `BasicQueryBuilder` to transform `UrlState` objects into `ENEQuery` objects
- An `SeoUrlFormatter` to transform `UrlState` objects into optimized URL strings

By specifying settings for additional components in the configuration file, you can configure the following aspects of your URLs:

- the dimension values and properties to display in the misc-path
- canonicalization options for dimensions in the misc-path
- the path-param-separator
- Endeca parameters to be included in the path-params instead of the query string
- base-36 encoding for numeric Endeca parameters

Creating a URL configuration file

A simple URL configuration file defines a `BasicQueryBuilder` and a top-level `SeoUrlFormatter`.

To create a URL configuration file:

1. Create a basic query builder that invokes the `com.endeca.soleng.urlformatter.basic.BasicQueryBuilder` class:
   For example:

```
  <bean id="queryBuilder" class="com.endeca.soleng.urlformatter.basic.BasicQueryBuilder">
  </bean>
```

2. Add the following properties:

Option	Description
`queryEncoding`	Specifies the query encoding. For example: `<value>UTF-8</value>`
`baseUrlENEQuery`	Sets the `baseUrLENEQuery`. This query is used to create the `UrlENEQuery` if the `UrlState` is not associated with a record or navigation state. If this value is `<null/>`, a new query is created.
`baseNavigationUrlENEQuery`	Sets the `baseNavigationUrlENEQuery`. This query is used to create the `UrlENEQuery` if the `UrlState` is associated with a navigation state (but not a record or aggregate record). If this value is `<null/>`, a new query is created.
`baseERecUrlENEQuery`	Sets the `baseERecUrlENEQuery`. This query is used to create the `UrlENEQuery` if the `UrlState` is associated with a record (but not an aggregate record). If this value is `<null/>`, a new query is created.
`baseAggrERecUrlENEQuery`	Sets the `baseAggrERecUrlENEQuery`. This query is used to create the `UrlENEQuery` if the `UrlState` is associated with an aggregate record. If this value is `<null/>`, a new query is created.
`defaultUrlENEQuery`	Sets the `detaultUrlENEQuery`. This query is used to create the `UrlENEQuery` if the `UrlState` contains no parameters.

   For example:

```
  <bean id="queryBuilder" class="com.endeca.soleng.urlformatter.basic.BasicQueryBuilder">

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

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

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

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

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

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

  </bean>
```

3. Create a top-level `seoUrlFormatter` bean to invoke the `com.endeca.soleng.urlformat¬ter.seo.SeoUrlFormatter` class:
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">
  </bean>
```

4. Add the following properties:

Option	Description
`defaultEncoding`	Specifies the default query encoding. For example: `<val¬ue>UTF-8</value>`
`pathSeparatorToken`	Specifies the character used to separate the misc-path from the path-params section in URLs.
`pathKeyValueSeparator`	Specifies the character used to separate key-value pairs in the path parameter section of the URL.

   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">

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

    <property name="pathSeparatorToken">
      <value>_</value>
    </property>

    <property name="pathKeyValueSeparator">
      <value>-</value>
    </property>

<!-- additional elements deleted from this example --!>
```

```
  </bean>
```

5. Set any required properties to specify configuration beans.

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

For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">

    <property name="pathParamKeys">
      <list>
        <value>R</value>
        <value>A</value>
        <value>An</value>
        <value>Au</value>
        <value>N</value>
        <value>No</value>
        <value>Np</value>
        <value>Nu</value>
        <value>D</value>
        <value>Ntt</value>
        <value>Ne</value>
      </list>
    </property>

    <property name="navStateFormatter">
      <ref bean="navStateFormatter"/>
    </property>

    <property name="ERecFormatter">
      <ref bean="erecFormatter"/>
    </property>

    <property name="aggrERecFormatter">
      <ref bean="aggrERecFormatter"/>
    </property>

    <property name="navStateCanonicalizer">
      <ref bean="navStateCanonicalizer"/>
    </property>

    <property name="urlParamEncoders">
      <list>
        <ref bean="N-paramEncoder"/>
        <ref bean="Ne-paramEncoder"/>
        <ref bean="An-paramEncoder"/>
      </list>
    </property>

  </bean>
```

After you have created the basic URL configuration file, you create additional beans to specify further configuration for the misc-path and path-params. Follow the procedures in the sections below to complete your URL configuration.

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 dimension and dimension values display. The `urlconfig.xml` file provides a simple and convenient method for configuring these options.

navStateFormatter

The `navStateFormatter` bean invokes the `com.endeca.soleng.urlformatter.seo.SeoNavStateFormatter` class to define `dimLocationFormatters` for each dimension that you want to configure.

Using the `dimLocationFormatters` defined in the `navStateFormatter` bean, you can configure URLs for navigation pages to include dimension names, roots, ancestors, and dimension value names in the misc-path of URLs for navigation pages.

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

```
http://localhost:8888/endeca_jspref/controller.jsp?&Ne=8&N=4294967160
```

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

```
http://localhost:8888/urlformatter_jspref/controller/Napa/_/N-1z141vc/Ne-8
```

navStateCanonicalizer

The `navStateCanonicalizer` bean invokes the `com.endeca.soleng.urlformatter.seo.SeoNavStateCanonicalizer` to order 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 `navStateCanonicalizer` to standardize the order of dimension and dimension values in the misc-path.

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 keywords, but the URLs are always canonicalized.

erecFormatter

URL optimization for record detail pages is configured separately from navigation pages and aggregate record details pages. The `erecFormatter` bean invokes the `com.endeca.soleng.urlformatter.seo.SeoERecFormatter` class to define `dimLocationFormatters` for each dimension that you want to configure.

The same options for including dimension names, roots, ancestors, and dimension value names are available for record detail pages as are available for navigation pages. While the `urlconfig.xml` configuration file uses the same `dimLocationFormatters` for the `erecFormatter` and the `aggErecFormatter` as are used for the `navStateFormatter`, this is not a requirement. You can create separate `dimLocationFormatters` for navigation pages, record detail pages, and aggregate record detail pages.

aggrERecFormatter

URL optimization for aggregate record detail pages is configured separately from navigation pages and record details pages as are available for navigation pages. The `aggrERecFormatter` bean invokes the `com.endeca.soleng.urlformatter.seo.SeoAggrERecFormatter` class to define `dimLocationFormatters` for each dimension that you want to configure. The same options for including dimension names, roots, ancestors, a nd dimension value names are available for aggregate record detail pages. While the `urlconfig.xml` configuration file uses the same `dimLocationFor¬matters` for the `aggrERecFormatter` and the `erecFormatter` as are used for the `navStateFor¬matter`, this is not a requirement. You can create separate `dimLocationFormatters` for navigation pages, record detail pages, and aggregate record detail pages.

Formatting misc-path strings in optimized URLs

The `SeoNavStateFormatter`, `SeoERecFormatter`, and `SeoAggrERecFormatter` use `StringFormatter` objects to format dimension and record property strings that display in URLs.

You can format the strings in the misc-path section of a URL by using string formatters that are predefined in the URL Optimization API. Formatting may include changing capitalization or applying a regular expression to replace portions of the string.

There are several `StringFormatter` objects in the URL Optimization API:

- `LowerCaseStringFormatter` — formats path-keyword data into lower case.
- `UpperCaseStringFormatter`— formats path-keyword data into upper case.
- `UrlEncodedStringFormatter` — URL-encodes strings.
- `RegexStringFormatter` — You can create a new `RegexStringFormatter` object and customize the `pattern`, `replacement`, and `replaceAll` properties to perform custom string formatting. For more information on the properties, please refer to the generated API documentation for the `api-seo` library.

To define `StringFormatter` objects in the `urlconfig.xml` file:

1. Create a bean to invoke a `StringFormatter` class.
   This example shows the configuration for a `RegexStringFormatter` that replaces all non-word character sequences with a single `"-"` character:

```
        <bean class="com.endeca.soleng.urlformatter.seo.RegexStringFor¬
matter">
          <property name="pattern">
            <value><![CDATA[[\W_&&[^\u00C0-\u00FF]]+]]></value>
          </property>

          <property name="replacement">
            <value>-</value>
          </property>

          <property name="replaceAll">
            <value>true</value>
          </property>
        </bean>
```

2. Optionally, you can build a `StringFormatterChain` to apply more than one `StringFormatter` to a string in series.

The following example shows the `defaultStringFormatterChain` that is used throughout the sample `urlconfig.xml` file.

```
  <bean name="defaultStringFormatterChain"
      class="com.endeca.soleng.urlformatter.seo.StringFormatterChain">

    <property name="stringFormatters">
      <list>
        <!--
          ###################################################################

          # replace all non-word character sequences with a single '-'
          #
        -->
        <bean class="com.endeca.soleng.urlformatter.seo.RegexStringFor¬
matter">
          <property name="pattern">
            <value><![CDATA[[\W_&&[^\u00C0-\u00FF]]+]]></value>
          </property>

          <property name="replacement">
            <value>-</value>
          </property>

          <property name="replaceAll">
            <value>true</value>
          </property>
        </bean>

        <!--
          ###################################################################

          # trim leading and trailing '-' characters (if any)
          #
        -->
        <bean class="com.endeca.soleng.urlformatter.seo.RegexStringFor¬
matter">
          <property name="pattern">
            <value><![CDATA[^-?([\w\u00C0-\u00FF][\w-\u00C0-
\u00FF]*[\w\u00C0-\u00FF])-?$]]></value>
          </property>

          <property name="replacement">
            <value>$1</value>
          </property>

          <property name="replaceAll">
            <value>false</value>
          </property>
        </bean>

      </list>
    </property>
  </bean>
```

Note that because `StringFormatterChain` implements `StringFormatter`, you can nest chains. For example:

```
    <bean class="com.endeca.soleng.urlformatter.seo.StringFormatterChain">
```

```
        <property name="stringFormatters">
          <list>

            <!-- replace 'Wine Type' with 'Wine' -->

           <bean class="com.endeca.soleng.urlformatter.seo.RegexStringFor¬
matter">
              <property name="pattern">
                <value>Wine Type</value>
              </property>

              <property name="replacement">
                <value>Wine</value>
              </property>

              <property name="replaceAll">
                <value>false</value>
              </property>
            </bean>

            <!-- execute the default string formatter chain -->

            <ref bean="defaultStringFormatterChain"/>

          </list>
        </property>
      </bean>
```

Optimizing URLs for navigation pages

Using the URL Optimization API, you can include dimension and dimension value names in the misc-path of URLs. You can also choose to canonicalize these dimension and dimension value names in order to avoid duplicate content and to increase your natural search rankings.

Note: For dimensions to display properly in the URL, they must be enabled for display with the record list.

You must create a URL configuration file before completing this procedure.

To optimize URLs for navigation pages:

1. Create a `navStateFormatter` bean to invoke the `com.endeca.soleng.urlformat¬ ter.seo.SeoNavStateFormatter`:

   For example:

   ```
     <bean id="navStateFormatter" class="com.endeca.soleng.urlformat¬
   ter.seo.SeoNavStateFormatter">
     </bean>
   ```

2. Add a `navStateFormatter` property to your top-level `seoUrlFormatter` bean.
   For example:

   ```
     <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
   ter.seo.SeoUrlFormatter">

   <!-- additional elements deleted from this example --!>
   ```

```
    <property name="navStateFormatter">
      <ref bean="navStateFormatter"/>
    </property>

  </bean>
```

3. Add a `useDimensionNameAsKey` property on the `navStateFormatter`.
   For example:

```
  <bean id="navStateFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>
  </bean>
```

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

4. Add a `dimLocationFormatters` property and list each `dimLocationFormatter` bean you plan to define.
   For example:

```
  <bean id="navStateFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="dimLocationFormatters">
      <list>
        <ref bean="wineTypeFormatter"/>
        <ref bean="regionFormatter"/>
        <ref bean="wineryFormatter"/>
        <ref bean="flavorsFormatter"/>
      </list>
    </property>

  </bean>
```

5. Create a `dimLocationFormatter` for each of the dimensions in the `dimLocationFormatters` list.
   For example:

```
  <bean id="regionFormatter"
     class="com.endeca.soleng.urlformatter.seo.SeoDimLocationFormatter">

  </bean>
```

Note: The sample `urlconfig.xml` file uses the same `dimLocationFormatter` for navigation pages, record detail pages, and aggregate record detail pages. You can choose to create unique `dimLocationFormatters` for each page type.

6. Add the following properties to each `dimLocationFormatter`:

Property	Description
key	In the `navStateFormatter` bean, the `useDimensionNameAsKey` property sets the key type. If you set the `useDimensionNameAsKey` to true, then use the dimension name as the value for this property (for

Property	Description
	example <value>Region</value>). If you set the useDimension¬NameAsKey to false, use the dimension ID number.
appendRoot	Specifies whether or not to append root dimension values to the URL. Set to true to append root dimension values.
appendAncestors	Specifices whether or not to append ancestor dimension values to the URL. Set to true to append ancestor dimension values.
appendDescriptor	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to true to append selected or descriptor dimension values.
separator	Specifies the character used to separate dimension roots, ancestors, and descriptor values.
rootStringFormat¬ter	Specifies the bean to format the dimension name. The reference application uses a defaultStringFormatterChain bean to invoke the com.en¬deca.soleng.urlformatter.seo.StringFormatterChain.
dimValStringFor¬matter	Specifies the bean to format the dimension value names. The reference application uses a defaultStringFormatterChain bean to invoke the com.endeca.soleng.urlformatter.seo.StringFormatter¬Chain. The examples below also use a defaultStringFormatter¬Chain bean.

For example:

```
<bean id="regionFormatter"
    class="com.endeca.soleng.urlformatter.seo.SeoDimLocationFormatter">


  <property name="key">
    <value>Region</value>
  </property>

  <property name="appendRoot">
    <value>false</value>
  </property>

  <property name="appendAncestors">
    <value>false</value>
  </property>

  <property name="appendDescriptor">
    <value>true</value>
  </property>

  <property name="separator">
    <value>-</value>
  </property>

  <property name="rootStringFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>

  <property name="dimValStringFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>
```

```
  </bean>
```

7. Create a `navStateCanonicalizer` bean to invoke the `com.endeca.soleng.urlformat¬ter.seo.SeoNavStateCanonicalizer` class.
   For example:

```
  <bean name="navStateCanonicalizer" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateCanonicalizer">
  </bean>
```

Note: Canonicalizing the dimension and dimension value names in the misc-path also 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.

8. Add a `navStateCanonicalizer` property to your top-level `seoUrlFormatter` bean.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">

<!-- additional elements deleted from this example --!>

    <property name="navStateCanonicalizer">
      <ref bean="navStateCanonicalizer"/>
    </property>

  </bean>
```

9. Configure the `navStateCanonicalizer`.
   For example, the following configuration creates URLs sorted by dimension ID in descending order:

```
  <bean name="navStateCanonicalizer" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateCanonicalizer">

    <property name="sortByName">
      <value>false</value>
    </property>

    <property name="sortByDimension">
      <value>true</value>
    </property>

    <property name="ascending">
      <value>false</value>
    </property>

  </bean>
```

Note: There a number of possible configuration options for canonicalization.

Related Links

Preparing your dimensions on page 21
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 21

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 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.

Formatting misc-path strings in optimized URLs on page 35
The `SeoNavStateFormatter`, `SeoERecFormatter`, and `SeoAggrERecFormatter` use `StringFormatter` objects to format dimension and record property strings that display in URLs.

Canonicalization configuration options

You can customize the canonicalization of URLs for navigation pages by choosing a sort method, for example by dimension name or dimension ID, and then a sort direction.

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)

Sort direction

Sort Direction	Configuration	Example base URL (sorted by dimension ID)

Ascending	`<property name="ascending">`
	`  <value>true</value>`
	`</property>`
		`http://localhost/urlformat¬`
		`ter_jspref/controller/region-`
		`Napa/Wine-red/`

Descending	`<property name="ascending">`
	`  <value>false</value>`
	`</property>`
		`http://localhost/urlformat¬`
		`ter_jspref/controller/Wine-`
		`red/region-Napa/`

Sort method

Sort by	Configuration	Example base URL (sort direction ascending)

Dimension name, case sensitive	`<property name="sortByName">`
	`  <value>true</value>`
	`</property>`
	
	`<property name="sortByDimension">`
	
	`  <value>true</value>`
	`</property>`
	
	`<property name="ignoreCase">`
	`  <value>false</value>`
	`</property>`
		`http://localhost/urlformat¬`
		`ter_jspref/controller/Wine-`
		`red/region-Napa/`

Dimension name, case insensitive	`<property name="sortByName">`
	`  <value>true</value>`
	`</property>`
	
	`<property name="sortByDimension">`
	
	`  <value>true</value>`
	`</property>`
	
	`<property name="ignoreCase">`
	`  <value>true</value>`
	`</property>`
		`http://localhost/urlformat¬`
		`ter_jspref/controller/region-`
		`Napa/Wine-red/`

Dimension ID	`<property name="sortByName">`
	`  <value>false</value>`
	`</property>`
	
	`<property name="sortByDimension">`
	
	`  <value>true</value>`
	`</property>`
		`http://localhost/urlformat¬`
		`ter_jspref/controller/region-`
		`Napa/Wine-red/`

Dimension value name,	`<property name="sortByName">` `  <value>true</value>`	`http://localhost/urlformat¬` `ter_jspref/controller/region-`

Sort by	Configuration	Example base URL (sort direction ascending)

case sensitive	`</property>`
	
	`<property name="sortByDimension">`
	
	`  <value>false</value>`
	`</property>`
	
	`<property name="ignoreCase">`
	`  <value>false</value>`
	`</property>`
		`Napa/Wine-red/`

Dimension value name, case insensitive	`<property name="sortByName">`
	`  <value>true</value>`
	`</property>`
	
	`<property name="sortByDimension">`
	
	`  <value>false</value>`
	`</property>`
	
	`<property name="ignoreCase">`
	`  <value>true</value>`
	`</property>`
		`http://localhost/urlformat¬ter_jspref/controller/region-Napa/Wine-red/`

Dimension value ID	`<property name="sortByName">`
	`  <value>false</value>`
	`</property>`
	
	`<property name="sortByDimension">`
	
	`  <value>false</value>`
	`</property>`
		`http://localhost/urlformat¬ter_jspref/controller/Wine-red/region-Napa/`

Example 1: the following code sample creates a canonicalized URL that sorts by dimension name, case sensitive, in an ascending order:

```
<bean name="navStateCanonicalizer" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateCanonicalizer">

    <property name="sortByName">
      <value>true</value>
    </property>

    <property name="sortByDimension">
      <value>true</value>
    </property>

    <property name="ascending">
      <value>true</value>
    </property>

    <property name="ignoreCase">
      <value>false</value>
    </property>
```

```
  </bean>
```

The resulting base URL: `http://localhost/urlformatter_jspref/controller/Wine-red/region-Napa/`

Example 2: the following code sample creates a canonicalized URL that sorts by dimension value ID in a descending order:

```
<bean name="navStateCanonicalizer" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateCanonicalizer">

    <property name="sortByName">
      <value>false</value>
    </property>

    <property name="sortByDimension">
      <value>true</value>
    </property>

    <property name="ascending">
      <value>false</value>
    </property>

  </bean>
```

The resulting base URL: `http://localhost/urlformatter_jspref/controller/region-Napa/Wine-red/`

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 enabled for display with the record list.

You must create a URL configuration file before completing this procedure.

To optimize URLs for record detail pages:

1. Create an `erecFormatter` bean to invoke the `com.endeca.soleng.urlformatter.seo.SeoERecFormatter`:

   For example:

   ```
     <bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.SeoERecFormatter">
     </bean>
   ```

2. Add an `ERecFormatter` property to your top-level `seoUrlFormatter` bean.

For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">

<!-- additional elements deleted from this example --!>

    <property name="ERecFormatter">
      <ref bean="erecFormatter"/>
    </property>

  </bean>
```

3. Add a `useDimensionNameAsKey` property on the `erecFormatter`.
   For example:

```
  <bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.Seo¬
ERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

  </bean>
```

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

4. Add a `propertyKeys` property to include record properties in the URLs of record details pages.
   For example:

```
  <bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.Seo¬
ERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="propertyKeys">
      <list>
        <value>P_Name</value>
      </list>
    </property>

  </bean>
```

5. Add a `propertyFormatter` property to format record properties included in the URLs of record details pages.
   For example:

```
  <bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.Seo¬
ERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="propertyKeys">
      <list>
        <value>P_Name</value>
      </list>
    </property>
```

```
    <property name="propertyFormatter">
      <ref bean="defaultStringFormatterChain"/>
    </property>

  </bean>
```

6. Add a `dimLocationFormatters` property and list each `dimLocationFormatter` bean you plan to define.
   For example:

```
  <bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.Seo¬
ERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="dimLocationFormatters">
      <list>
        <ref bean="regionFormatter"/>
        <ref bean="wineryFormatter"/>
        <ref bean="wineTypeFormatter"/>
        <ref bean="vintageFormatter"/>
      </list>
    </property>

    <property name="propertyKeys">
      <list>
        <value>P_Name</value>
      </list>
    </property>

    <property name="propertyFormatter">
      <ref bean="defaultStringFormatterChain"/>
    </property>

  </bean>
```

7. Create a `dimLocationFormatter` for each of the dimensions in the `dimLocationFormatters` list.
   For example:

```
  <bean id="regionFormatter"
     class="com.endeca.soleng.urlformatter.seo.SeoDimLocationFormatter">

  </bean>
```

Note: The sample `urlconfig.xml` file uses the same `dimLocationFormatter` for navigation pages, record detail pages, and aggregate record detail pages. You can choose to create unique `dimLocationFormatters` for each page type.

8. Add the following properties to each `dimLocationFormatter`:

Property	Description
key	In the `navStateFormatter` bean, the `useDimensionNameAsKey` property sets the key type. If you set the `useDimensionNameAsKey` to true, then use the dimension name as the value for this property (for example `<value>Region</value>`). If you set the `useDimension¬ NameAsKey` to false, use the dimension ID number.

Property	Description
appendRoot	Specifies whether or not to append root dimension values to the URL. Set to `true` to append root dimension values.
appendAncestors	Specifices whether or not to append ancestor dimension values to the URL. Set to `true` to append ancestor dimension values.
appendDescriptor	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to `true` to append selected or descriptor dimension values.
separator	Specifies the character used to separate dimension roots, ancestors, and descriptor values.
rootStringFormatter	Specifies the bean to format the dimension name. The reference application uses a `defaultStringFormatterChain` bean to invoke the `com.endeca.soleng.urlformatter.seo.StringFormatterChain`.
dimValStringFormatter	Specifies the bean to format the dimension value names. The reference application uses a `defaultStringFormatterChain` bean to invoke the `com.endeca.soleng.urlformatter.seo.StringFormatterChain`. The examples below also use a `defaultStringFormatterChain` bean.

For example:

```
<bean id="regionFormatter"
   class="com.endeca.soleng.urlformatter.seo.SeoDimLocationFormatter">


  <property name="key">
    <value>Region</value>
  </property>

  <property name="appendRoot">
    <value>false</value>
  </property>

  <property name="appendAncestors">
    <value>false</value>
  </property>

  <property name="appendDescriptor">
    <value>true</value>
  </property>

  <property name="separator">
    <value>-</value>
  </property>

  <property name="rootStringFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>

  <property name="dimValStringFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>

</bean>
```

Related Links

Preparing your dimensions on page 21
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 21
If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

Formatting misc-path strings in optimized URLs on page 35
The `SeoNavStateFormatter`, `SeoERecFormatter`, and `SeoAggrERecFormatter` use `StringFormatter` objects to format dimension and record property strings that display in URLs.

Optimizing URLs for aggregate 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 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 enabled for display with the record list.

You must create a URL configuration file before completing this procedure.

To optimize URLs for aggregate record detail pages:

1. Create an `aggrERecFormatter` bean to invoke the `com.endeca.soleng.urlformat¬ter.seo.SeoAggrERecFormatter` class:
   For example:
   ```
     <bean id="aggrERecFormatter" class="com.endeca.soleng.urlformat¬
   ter.seo.SeoAggrERecFormatter">
     </bean>
   ```
2. Add an `aggrERecFormatter` property to your top-level `seoUrlFormatter` bean.
   For example:
   ```
     <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
   ter.seo.SeoUrlFormatter">

   <!-- additional elements deleted from this example --!>

       <property name="aggrERecFormatter">
         <ref bean="aggrERecFormatter"/>
       </property>

     </bean>
   ```
3. Add a `useDimensionNameAsKey` property on the `aggrERecFormatter`.
   For example:
   ```
     <bean id="aggrERecFormatter" class="com.endeca.soleng.urlformat¬
   ter.seo.SeoAggrERecFormatter">

       <property name="useDimensionNameAsKey">
         <value>true</value>
   ```

```
    </property>
  </bean>
```

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

4. Add a `propertyKeys` property to include record properties in the URLs of record details pages.
   For example:

```
  <bean id="aggrERecFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoAggrERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="propertyKeys">
      <list>
        <value>P_Name</value>
      </list>
    </property>

  </bean>
```

5. Add a `propertyFormatter` property to format record properties included in the URLs of record details pages.
   For example:

```
  <bean id="aggrERecFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoAggrERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="propertyKeys">
      <list>
        <value>P_Name</value>
      </list>
    </property>
    <!-- use default string formatter chain -->

    <property name="propertyFormatter">
      <ref bean="defaultStringFormatterChain"/>
    </property>

  </bean>
```

6. Add a `dimLocationFormatters` property and list each `dimLocationFormatter` bean you plan to define.
   For example:

```
  <bean id="aggrERecFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoAggrERecFormatter">

    <property name="useDimensionNameAsKey">
      <value>true</value>
    </property>

    <property name="dimLocationFormatters">
      <list>
        <ref bean="regionFormatter"/>
        <ref bean="wineryFormatter"/>
```

```
    </list>
  </property>

  <property name="propertyKeys">
    <list>
      <value>P_Name</value>
    </list>
  </property>

  <property name="propertyFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>

</bean>
```

Note: The sample `urlconfig.xml` file uses the same `dimLocationFormatter` for navigation pages, record detail pages, and aggregate record detail pages. You can choose to create unique `dimLocationFormatters` for each page type.

7. Create a `dimLocationFormatter` for each of the dimensions in the `dimLocationFormatters` list.
   For example:

```
<bean id="regionFormatter"
   class="com.endeca.soleng.urlformatter.seo.SeoDimLocationFormatter">

</bean>
```

8. Add the following properties to each `dimLocationFormatter`:

Property	Description
key	In the `navStateFormatter` bean, the `useDimensionNameAsKey` property sets the key type. If you set the `useDimensionNameAsKey` to true, then use the dimension name as the value for this property (for example `<value>Region</value>`). If you set the `useDimension¬NameAsKey` to false, use the dimension ID number.
appendRoot	Specifies whether or not to append root dimension values to the URL. Set to `true` to append root dimension values.
appendAncestors	Specifices whether or not to append ancestor dimension values to the URL. Set to `true` to append ancestor dimension values.
appendDescriptor	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to `true` to append selected or descriptor dimension values.
separator	Specifies the character used to separate dimension roots, ancestors, and descriptor values.
rootStringFormat¬ter	Specifies the bean to format the dimension name. The reference application uses a `defaultStringFormatterChain` bean to invoke the `com.en¬deca.soleng.urlformatter.seo.StringFormatterChain`.
dimValStringFor¬matter	Specifies the bean to format the dimension value names. The reference application uses a `defaultStringFormatterChain` bean to invoke the `com.endeca.soleng.urlformatter.seo.StringFormatter¬Chain`. The examples below also use a `defaultStringFormatter¬Chain` bean.

For example:

```
<bean id="regionFormatter"
   class="com.endeca.soleng.urlformatter.seo.SeoDimLocationFormatter">


  <property name="key">
    <value>Region</value>
  </property>

  <property name="appendRoot">
    <value>false</value>
  </property>

  <property name="appendAncestors">
    <value>false</value>
  </property>

  <property name="appendDescriptor">
    <value>true</value>
  </property>

  <property name="separator">
    <value>-</value>
  </property>

  <property name="rootStringFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>

  <property name="dimValStringFormatter">
    <ref bean="defaultStringFormatterChain"/>
  </property>

</bean>
```

Related Links

Preparing your dimensions on page 21
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 21
If you intend to display record properties in your URLs, you must configure each property to **Show with record** and **Show with record list**.

Formatting misc-path strings in optimized URLs on page 35
The `SeoNavStateFormatter`, `SeoERecFormatter`, and `SeoAggrERecFormatter` use `StringFormatter` objects to format dimension and record property strings that display in URLs.

Configuring the path-param-separator

You can customize the string that displays between the misc-path and the path-params components of URLs.

The sample `urlconfig.xml` file provided with the URL Optimization API uses an underscore to separate the misc-path from the path-params in URLs. For example: `http://localhost/urlformatter_jspref/controller/Wine-Red-Pinot-Noir/_/N-66w`

You must create a URL configuration file before completing this procedure.

To change the path-param-separator string:

1. Locate the top-level URL formatter bean in your URL configuration file.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformatter.seo.SeoUrlFormatter">
  </bean>
```

2. Customize the value of the `pathSeparatorToken` property:
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformatter.seo.SeoUrlFormatter">
    <property name="pathSeparatorToken">
      <value>separator</value>
    </property>
  </bean>
```

The new URL displays as: `http://localhost/urlformatter_jspref/controller/Wine-Red-Pinot-Noir/separator/N-66w`

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/Bordeaux?N=4294966952&fromsearch=false&Ntk=All&Ntt=red&Ntx=mode%2bmatchallpartial`

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`

Note: To ensure the best possible natural search-engine ranking, it is recommended that you limit the number of parameters you include in the path-params section.

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 Region > Napa contains the dimension value ID for Napa (4294966952):

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

By base-36 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

You can add non-Endeca parameters to URLs by passing them through the API.

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.

You must create a URL configuration file before completing this procedure.

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

1. In your URL configuration file, locate the top-level URL formatter.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">

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

    <property name="pathSeparatorToken">
      <value>_</value>
    </property>

    <!-- additional elements deleted from this example --!>

  </bean>
```

2. Add a `pathParamKeys` property.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">
```

```
    <property name="pathParamKeys">
    </property>

  </bean>
```

3. Add a `list` attribute containing all of the Endeca parameters you want moved from the query string.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">

    <property name="pathParamKeys">
      <list>
        <value>R</value>
        <value>A</value>
        <value>An</value>
      </list>
    </property>

  </bean>
```

Encoding Endeca parameters

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

You must create a URL configuration file before completing this procedure.

Only the numeric Endeca parameters can be encoded:

- N
- Ne
- An
- Dn

The following procedure provides instructions for applying base-36 encoding to the An parameter. You can apply base-36 encoding to any numeric Endeca parameter, but each parameter requires a separately configured `paramEncoder` bean.

To encode numeric Endeca parameters:

1. Create a `paramEncoder` bean to invoke the `com.endeca.soleng.urlformatter.seo.SeoN¬avStateEncoder`:
   For example:

```
  <bean name="An-paramEncoder" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateEncoder">
  </bean>
```

Remember: You need to create a separate `paramEncoder` bean for each numeric Endeca parameter you want to encode.

2. Add a `paramKey` property to specify which numeric Endeca parameter to encode.
   For example:

```
  <bean name="An-paramEncoder" class="com.endeca.soleng.urlformat¬
ter.seo.SeoNavStateEncoder">
```

```
    <property name="paramKey">
      <value>An</value>
    </property>
  </bean>
```

3. Repeat steps one and two for each Endeca parameter you want to encode.
4. Locate the top-level URL formatter bean in your URL configuration file.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">
  </bean>
```

5. Add a `urlParamEncoders` property:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">
    <property name="urlParamEncoders">
    </property>
  </bean>
```

6. Add a `list` attribute and specify each of the parameter encoder beans.
   For example:

```
  <bean id="seoUrlFormatter" class="com.endeca.soleng.urlformat¬
ter.seo.SeoUrlFormatter">
    <property name="urlParamEncoders">
      <list>
        <ref bean="N-paramEncoder"/>
        <ref bean="Ne-paramEncoder"/>
        <ref bean="An-paramEncoder"/>
      </list>
    </property>
  </bean>
```

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:8888/endeca_jspref/controller.jsp?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:8888/endeca_jspref/controller.jsp
```

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. To set the attribute, use the following code:

```
 session.setAttribute("eneHost", eneHost);
```

2. To retrieve the attribute value, use the following code:

```
eneHost = (String)session.getAttribute("eneHost");
```

About passing non-Endeca parameters to the API

You can add non-Endeca parameters to URLs by passing them through the API.

For example, you could add information about how many records per page should display in each results set:

In the reference application's `controller.jsp` file, find the following section:

```
UrlState baseUrlState = urlFormatter.parseRequest(request);

ENEQuery usq = queryBuilder.buildQuery(baseUrlState);
```

and add code similar to the following:

```
baseUrlState.setParam("records_per_page", "25");
```

Note: Endeca recommends limiting the number of parameters that display in URLs. It is recommended that session-scope parameters be removed from the URL and stored as session objects.

Using the URL configuration file with your application

Before you can create optimized URLs with your own application, you need to include the URL configuration file in your application's classpath.

To use the URL configuration file with your application:

1. Stop the Endeca HTTP service.
2. Locate your URL configuration file.
3. Copy the URL configuration file into the `WEB-INF` subdirectory of your Web application directory. For example: `C:\Endeca\MyApps\WEB-INF`
4. Start the Endeca HTTP service.

To verify that the URL configurations are working properly, open a Web browser and navigate to your Web application. Check that the URLs display as you configured them with the URL configuration file.

Related Links

Creating a URL configuration file on page 31
A simple URL configuration file defines a `BasicQueryBuilder` and a top-level `SeoUrlFor¬matter`.

Creating a URL configuration file on page 31
A simple URL configuration file defines a `BasicQueryBuilder` and a top-level `SeoUrlFor¬matter`.

Chapter 7

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 a URL 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 configured appropriately in the Sitemap Generator's `urlconfig.xml` file.

Because the `urlconfig.xml` file included with the Sitemap Generator uses the same format as the sample `urlconfig.xml` file for the URL Optimization API, you can use your URL Optimization API `urlconfig.xml` file for sitemap generation.

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*.

Using the URL Optimization API urlconfig.xml file for sitemap generation

You can use the same `urlconfig.xml` file you created for URL optimization as the URL configuration file for sitemap generation.

To use the URL Optimization API URL configuration file with the Sitemap Generator:

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 `URL_FORMAT_FILE`:
   For example:

```
<URL_FORMAT_FILE>urlconfig.xml</URL_FORMAT_FILE>
```

3. Edit the `<URL_FORMAT_FILE>` value so that it points to the `urlconfig.xml` file you created with the URL Optimization API.
   For example:

```
<URL_FORMAT_FILE>C:\Endeca\SEM\URL Optimization
APIs\Java\version\webapps\urlformatter_jspref\WEB-INF\urlcon¬
fig.xml</URL_FORMAT_FILE>
```

4. Save and close the `conf.xml` file.

You can also copy your URL Optimization API `urlconfig.xml` file to the `\conf\` subdirectory of your Sitemap Generator installation directory. If you choose to do this, you need to make sure that the two `urlconfig.xml` files maintain identical configurations.

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

Related Links

Creating a URL configuration file on page 31
A simple URL configuration file defines a `BasicQueryBuilder` and a top-level `SeoUrlFor¬matter`.

About the URL configuration file on page 30
The URL Optimization API reference application uses an XML file named `urlconfig.xml` to configure the format of the URLs that it generates.

Index

O

P

Q

R

S

T

U

W