

Oracle Endeca Commerce

Assembler Application Developer's Guide

Version 11.0 • January 2014



Contents

Preface.....	9
About this guide.....	9
Who should use this guide.....	9
Conventions used in this guide.....	9
Contacting Oracle Support.....	10
Part I: About the Assembler.....	11
Chapter 1: Introduction to the Assembler.....	13
What is the Assembler?.....	13
Configuring Assembler applications in Experience Manager.....	16
Using the Assembler with the MDEX Engine.....	17
Chapter 2: Assembler Architectural Overview.....	21
The Assembler processing model.....	21
About serialization and de-serialization.....	24
The Assembler eventing framework.....	25
About Assembler error handling.....	26
Chapter 3: About cartridges and content items.....	27
About cartridges.....	27
Anatomy of a cartridge.....	27
About the CartridgeHandler interface.....	28
Part II: Designing an Assembler Application.....	31
Chapter 4: Planning an Assembler Application.....	33
About planning your application sitemap.....	33
About page types.....	34
About content folders.....	38
Chapter 5: Creating Experience Manager Templates.....	45
About creating templates.....	45
Anatomy of a template.....	46
About the template XML schema.....	46
Template identifiers.....	47
About the type of a template.....	47
Specifying the description and thumbnail image for a template.....	48
Specifying the default name for a cartridge.....	49
About defining the content properties and editing interface.....	49
Structural properties.....	52
About multiple locales.....	55
Managing Experience Manager Templates.....	56
Part III: Developing an Assembler Application.....	61
Chapter 6: Deploying the Assembler.....	63
Assembler environment requirements.....	63
Assembler dependencies.....	64
About deploying the Assembler.....	64
Assembler configuration.....	65

Chapter 7: Invoking the Assembler.....	71
Invoking the Assembler in Java.....	71
Querying the Assembler Service.....	73
About building an Assembler query string.....	75
About retrieving Assembler results using the packaged services.....	75
About handling the Assembler response.....	83
Chapter 8: Implementing Multichannel Applications.....	87
Overview of multichannel applications with the Endeca Assembler.....	87
About creating templates for mobile channels.....	87
Chapter 9: Tuning an Assembler application.....	89
Setting up the Preview Application for Workbench.....	89
Configuring Logging for an Assembler Application.....	97
Configuring cartridge performance logging.....	101
Understanding and Debugging Query Results.....	101
Part IV: Optimizing Application URLs.....	105
Chapter 10: About the URL optimization classes.....	107
Package contents.....	107
Introduction to URL optimization.....	107
Overview of URL optimization capabilities.....	107
About URL canonicalization.....	109
Chapter 11: Working with Application URLs.....	111
About application URLs.....	111
About Actions.....	111
About working with URL parameters.....	114
About URL configuration in the reference application.....	114
About working with canonical links.....	118
Chapter 12: Preparing your application.....	121
Preparing your dimensions.....	121
Preparing your properties.....	121
Handling images and external JavaScript files.....	122
URL transitioning.....	122
Chapter 13: Building optimized URLs.....	123
Core URL optimization classes.....	123
Overview of building URLs using the URL optimization classes.....	123
Parsing an incoming query and sending it to an MDEX Engine.....	124
Informing the UrlState of the navigation state.....	124
Creating link URLs from a UrlState.....	125
Chapter 14: Configuring URLs.....	127
Anatomy of an optimized Endeca URL	127
About the URL configuration file.....	128
Creating a URL configuration file.....	129
About optimizing the misc-path.....	132
Configuring the path-param-separator.....	149
About optimizing the path-params and query string.....	150
Using the URL configuration file with your application.....	154
Chapter 15: Integrating with the Sitemap Generator.....	155
The Sitemap Generator urlconfig.xml file.....	155
Using the URL configuration file with the Sitemap Generator.....	155

Part V: Extending the Assembler.....	157
Chapter 16: Extending and Developing Cartridges.....	159
Cartridge Basics.....	159
First steps with a new cartridge.....	159
Adding a basic renderer.....	163
Deeper dive on the example cartridge.....	164
Overview of cartridge extension points.....	167
Customizing the Experience Manager interface.....	168
About Cartridge Handlers and the Assembler.....	172
About using event listeners to extend the navigation cartridges.....	176
Sample Cartridges.....	178
Chapter 17: Developing Editors for Workbench.....	205
Setting up the Experience Manager Editor SDK.....	205
Flex prerequisites and resources.....	205
About setting up a Flex development environment.....	206
Developing Editors With the Experience Manager Editor SDK.....	207
Specifying dependencies between editors.....	215
Appendix A: Template Property and Editor Reference.....	221
Editor property mapping reference.....	221
Editor label configuration reference.....	224
Basic content properties.....	224
Adding a string property.....	225
About numeric properties.....	231
Adding a Boolean property.....	234
Adding a list property.....	235
Adding an item property.....	236
Adding a group label.....	237
Complex property editors.....	238
About the microbrowser.....	238
About the Select Records dialog.....	240
About the Dynamic Slot editor.....	241
Adding a Link Builder.....	244
About the Media editor.....	246
Adding a Boost-Bury Record editor.....	258
Adding a Guided Navigation editor.....	259
Adding a Dimension Selector.....	260
Adding a Dimension List editor.....	261
Adding a Dimension Value Boost-Bury editor.....	262
Adding a Dimension Value List editor.....	264
Adding an Image Preview.....	265
Adding a Record List editor.....	266
Adding a Record Stratification editor.....	269
Adding a Rich Text editor.....	270
Adding a Sort editor.....	271
Adding a Spotlight Selection editor.....	273
Application feature property reference.....	275
Appendix B: Navigation Cartridge Configuration Reference.....	279
Navigation cartridge URL parameter reference.....	279
About this section.....	279
Core URL query parameters.....	280
Cartridge-specific URL query parameters.....	287
About the navigation cartridge configuration models.....	294
Overview of the navigation cartridge configuration models.....	294
Search cartridges.....	297
Guided Navigation cartridges.....	310
Results cartridges.....	316
Record details cartridges.....	320

Content and spotlighting cartridges.....	321
Dynamic triggering cartridges.....	323
Request Event Attributes.....	324
Base request event attributes.....	325
Navigation cartridge request event attributes.....	325

Copyright and disclaimer

Copyright © 2003, 2014, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information on content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services.

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc>.

Oracle customers have access to electronic support through My Oracle Support. For information, visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info> or visit <http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs> if you are hearing impaired.

Preface

Oracle Endeca Commerce 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 Commerce enables businesses to help guide and influence customers in each step of their search experience. At the core of Oracle Endeca Commerce 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 into results that can be rendered for display.

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 determine the conditions for displaying content in response to any search, category selection, or facet refinement.

About this guide

This guide provides an overview of Assembler application development. It covers the architecture of a typical Assembler application, as well as the tasks required to enable configuration through the Experience Manager or Rule Manager tools in Workbench.



Note: Rule Manager is deprecated in the Tools and Frameworks 11.0.0 release.

The Tools and Frameworks package includes a Java implementation of the Assembler, so examples in this document are Java-based.

Who should use this guide

This guide is intended for developers who are building applications using the Assembler, and are supporting business users who configure these applications using Endeca Workbench. You should familiarize yourself with the concepts in the *Oracle Endeca Commerce Concepts Guide* before reading this 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 Support

Oracle 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 Support through Oracle's Support portal, My Oracle Support at <https://support.oracle.com>.

Part 1

About the Assembler

- *Introduction to the Assembler*
- *Assembler Architectural Overview*
- *About cartridges and content items*

Chapter 1

Introduction to the Assembler

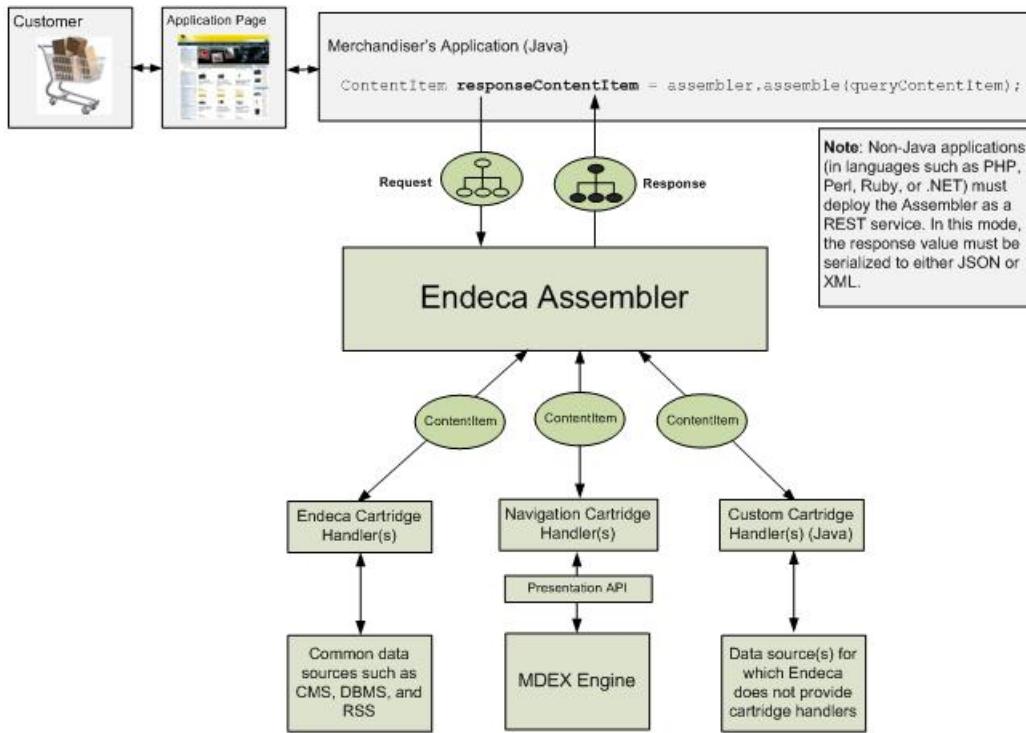
This section provides a conceptual overview of the Endeca Assembler.

What is the Assembler?

The Assembler is a microcontroller framework for building view-friendly response models from external data sources (also known as a server-side mashup framework). You can develop extensions to the framework to interact with your resources, centralizing runtime data retrieval and manipulation in your application. For these reasons, the Assembler also integrates well with organizations that use Service-Oriented Architecture.

The Assembler acts as the single source for data within an application, marshaling data from external sources, including Digital Asset Management systems, social media feeds, and databases such as the MDEX Engine.

The following diagram illustrates the role of the Assembler in an Oracle Commerce implementation:



Note: A cartridge is a configurable region of your application page, typically used to display one type of information. A content item is set of key:value pairs which can include other content items, and contains the data passed to your application. Cartridges are represented by their component content items in the data model.

As shown in the preceding diagram, the following things happen when customers request information through your application page:

1. Your Endeca application invokes the `assemble()` method as follows:

```
ContentItem responseContentItem = assembler.assemble(contentItem)
```

`responseContentItem` is a tree of other content items, each of which is associated with a content type.

2. The `assemble()` method sends `responseContentItem` to the Assembler.
3. The Assembler passes the individual content items in `responseContentItem` to cartridge handlers that are mapped to handle a given content type. Each content item specifies a request for information.
4. The cartridge handlers pass the requests on to the appropriate sources of information, such as an MDEX Engine, a relational database system, a content management system, and so on.
5. The cartridge handlers receive and process information from their respective sources. The handlers contain all the logic needed to process the information, though they may also process requests without requiring input from an external data source.
6. Each cartridge handler returns a content item to the Assembler that contains the requested information.
7. The Assembler combines the content items that it receives from all of the cartridge handlers into a `responseContentItem`, which is structured as a tree that contains all of the information required by the front end application.
8. The Assembler returns `responseContentItem` to the front end application.
9. Rendering code in the application converts the information in `responseContentItem` into a form that can be displayed in the appropriate cartridges on your application page. Typically, a cartridge renderer (a

separate module of rendering code) processes and displays the information for each content item in the `responseContentItem` tree.



Note: The Assembler can return XML or JSON representations of objects for consumption by a variety of rendering engines, such as .NET, PHP, or Flash-based applications. It can also return model objects as POJOs (plain old Java objects) when embedded in a native Java application.

The Tools and Frameworks package includes a Java Assembler implementation that uses Spring to resolve cartridge handlers and services. It also features a set of Navigation Cartridges with handlers and renderers for communicating with the MDEX Engine. You can write additional cartridges in Java to communicate with other data sources.

Core Assembler concepts

The Assembler stores and manipulates data as content items, sets of `key:value` pairs. These content items can represent cartridges, which map to front-end features in an application.

About Content Items

Content items are maps: objects that contain data as `key:value` pairs. Typically, a content item is either a structural component (such as an application page) that contains additional content items, or it maps to some front-end component in an application (such as an image banner).

For example, in the Discover Electronics reference implementation, the entire default "browse" page is represented by a content item that contains the page template. Each section of the page is also a content item, nested within the containing "three column page" content item. Within those sections are additional content items that represent front-end features:

- `ThreeColumnPage`
 - `headerContent`
 - Search Box
 - `leftContent`
 - Breadcrumbs
 - Guided Navigation
 - `mainContent`
 - SearchAdjustments
 - ContentSlotMain
 - ResultsList
 - `rightContent`
 - RecordSpotlight

Each of the entries above is a content item in the reference application. Because they are organized as a tree, the resulting data object is easy to traverse for rendering.

About Cartridges and Cartridge Templates

You may notice that the cartridges mentioned correspond to the content items listed above. A cartridge defines a content item with a specific role in your application.

To a business user, a cartridge maps to a GUI component in the front-end application. The navigation cartridges included with the Assembler are designed around this idea, and feature a Breadcrumbs cartridge, a Search Box cartridge, a Results List, and so on. Your business users will expect similar structure from the cartridges for your own application.

A cartridge is defined by its template. These templates serve two functions in Experience Manager:

- A cartridge template defines the structure and initial configuration for a content item.
- A cartridge template defines a set of configurable properties and the associated editors with which the business user can configure them.

Experience Manager instantiates each content item from its cartridge template. This includes any configuration made by the business user, and results in a content item with instance configuration that is passed to the Assembler.

About Cartridge Handlers

A cartridge handler takes a content item as input, processes it, and returns a content item as output.

The input content item typically includes instance configuration, which consists of any properties specified by a business user using the Experience Manager or Rule Manager tool in Endeca Workbench. The content item is typically initialized by layering configuration from other sources: your application may include default values, or URL parameters that represent end user selections in the front-end application.

A cartridge handler can optionally perform further processing, such as querying a search engine for data. Once processing is finished, the handler returns a completed content item to the application.



Note: Not all cartridges require cartridge handlers. In the case of a content item with no associated cartridge handler, the Assembler returns the unmodified content item.

For detailed information regarding the `CartridgeHandler` interface, see [About the CartridgeHandler interface](#) on page 172, or refer to the [Assembler API Reference \(Javadoc\)](#).

Example: The Results List Cartridge

Consider the Results List cartridge included with Tools and Frameworks. The `ResultsList` object that backs the cartridge is a content item. The `MainContent-ResultsList.xml` cartridge template exposes a subset of object properties for configuration in Experience Manager. The remaining properties are configurable through the UI in the front-end application.

The Results List cartridge handler combines the default, instance, and URL configuration values to create a query to send to the MDEX Engine. The values in the query response are used to populate the `ResultsList` content item and return it to the application for rendering.

Configuring Assembler applications in Experience Manager

The Assembler interacts with the Rule Manager or Experience Manager tools in Endeca Workbench to expose content configuration to business users.

Experience Manager instantiates each content item from its content XML. In an unconfigured cartridge, this XML is identical to the cartridge template (including any default values specified in the template). When a business user opens and modifies a cartridge in Experience Manager or Rule Manager, their settings are saved in the content XML. In an authoring environment, this XML is stored in the Endeca Configuration Repository. In a production environment, it is published to a content server.

At runtime, the Assembler deserializes the content XML to create the appropriate content item object, and passes it to its corresponding cartridge handler for processing.

In addition to creating instances of front-end application components in this manner, the business user can also use structural content item templates (such as the Three-Column Navigation Page template) to create the organizational elements of a site. For example, your business user can create an "About Us" page, a "Frequently Asked Questions" page, and other static elements of a site by selecting and populating suitable cartridge templates.

License restrictions for Rule Manager

You can use Rule Manager to create business rules to control content spotlighting. Using the record spotlight cartridge to spotlight records and spotlighting content such as promotional images is supported.

You cannot use Rule Manager to provide business user control over contextual, rules-driven dimensions or dimension values (for example, functionality similar to the guided navigation cartridges). You cannot provide contextual business user control over results lists (for example, the search results cartridge). In short, if you use Rule Manager, guided navigation and results lists must be configured globally or through APIs. Business user tooling for rules-driven control of either of these aspects requires an Experience Manager license.

Using the Assembler with the MDEX Engine

The Assembler implementation in the Tools and Frameworks package includes cartridge handlers designed to interact with the MDEX Engine through the Presentation API. The reference application includes templates that use these navigation cartridges to enable configuration in Experience Manager and render the resulting data in a front end application.

Implementing cartridge handlers to communicate with an external system requires the following considerations:

- What use cases does this work address in the end user application?
- How do the cartridges accomplish this?
- What set of cartridges best fits the defined use cases?

In the case of the navigation cartridges, the work exposes MDEX Engine features to an Assembler application. It also enables a business user to configure powerful Guided Navigation features using UI components that can be customized by an application developer to fit business needs.

The navigation cartridges accomplish this by communicating with the MDEX Engine through the Presentation API (PAPI). They expose most of the PAPI features through editors that simplify the configuration process (such as the Endeca Browser, which allows business users to navigate through the data set and select specific records or a dynamic search and navigation state in an intuitive manner).

The navigation cartridges include the following:

- Search Box
- Auto-Suggest Search Results
- Dimension Search Results
- Search Adjustments
- Refinement Menu
- Breadcrumbs
- Dimension Navigation
- Results List
- Record Details
- Media Banner

- Record Spotlight and Horizontal Record Spotlight

Example: The Results List cartridge

The Results List cartridge exists to display MDEX Engine search results for an end user query. It is backed by a `com.endeca.infront.cartridge.ResultsList` content item object, which extends the `com.endeca.infrontAssembler.BasicContentItem` interface.

The input to the Assembler consists of a configuration model -- a content item with MDEX Engine query information such as the end user's search terms, selected search refinements, sorting options, and records per page. These are passed in as a `com.endeca.infront.cartridge.ResultsListConfig` object.

The `ResultsListHandler` generates a query from the properties on `ResultsListConfig`, then sends the query to the MDEX Engine. It instantiates a `ResultsList` content item using the query response, and copies over some of the properties from the configuration model (such as records per page and sorting) directly. This view-friendly `ResultsList` object is then returned to the application for rendering.

In the Discover Electronics reference application, the cartridge configuration comes from the following sources:

- Default configuration — In the Discover Electronics reference application, this is specified in the Spring context file.
- Instance configuration — Specified by the business user in the Results List cartridge in Experience Manager.
- Request-based configuration — Specified by the application end user; this includes any search terms or selected dimension refinements, among other things.

Default Cartridge Configuration

The default cartridge configuration is specified in the Spring context file, located at `ToolsAndFrameworks\<version>\reference\discover-electronics-authoring\WEB-INF\assembler-context.xml` for the authoring instance of the Discover Electronics reference application. This includes values for the following properties on the `ResultsListConfig` content item:

- `sortOption` — The sorting options available to the end user when viewing the list of query results.
- `relRankStrategy` — The Relevance Ranking strategy applied to search results. For more information on Relevance Ranking, see the *MDEX Engine Developer's Guide*.
- `recordsPerPage` — The number of records to display per page of results.



Note: The above list is a subset of configured properties and provided as an example.

Instance Configuration

The cartridge instance configuration comes from the values in the `ToolsAndFrameworks\<version>\reference\discover-data\cartridge_templates\MainContent-ResultsList.xml` cartridge template. The template exposes the following properties to the business user in Experience Manager:

- `boostStrata` — A list of records to elevate to the top of the Results List.
- `buryStrata` — A list of records to move to the bottom of the Results List.
- `sortOption` — The business user can override the default sorting options.
- `relRankStrategy` — The business user can override the default Relevance Ranking strategy.
- `recordsPerPage` — The business user can override the default number of records to display on each page.

Request-Based Configuration

The application end user's configuration in Discover Electronics is passed to the `ResultsListConfig` object as URL parameters, though you may choose to implement such functionality differently in your own application.

- `offset` — Controls the record offset of the displayed results in order to control record display while paging through results.
- `relRankTerms` — The end user's search terms.
- `sortOption` — The end user can override the default values and the instance configuration.
- `recordsPerPage` — The end user can override the default values and the instance configuration.

The Results List cartridge handler combines the default, instance, and request-based values to create a query to send to the MDEX Engine. The values are used to populated the `ResultsList` content item and return it to the application for rendering.

Chapter 2

Assembler Architectural Overview

This section provides an architectural overview of the Endeca Assembler.

The Assembler processing model

The core of the Assembler is the `assemble()` method, which takes a content item representing a cartridge instance configuration and invokes cartridge handlers to process it into a response content item.

The Assembler uses the visitor pattern to traverse the input content item and any child content items, and invokes the appropriate cartridge handler (if any) for each of them.

The Assembler makes two passes over the input content item:

1. In the first pass, the Assembler calls `CartridgeHandler.initialize()` followed by `CartridgeHandler.preprocess()` on each content item in the tree. This is a pre-order traversal of the tree (working from the top of the tree down through its children), so cartridge handlers may add or modify child content items at this stage.
2. In the second pass, the Assembler calls `CartridgeHandler.process()` on each content item, which returns the response content item for that cartridge. This is a post-order traversal of the tree (working from the bottom up), so child content items are processed before the parent. The response object for the root content item of the tree (the content item originally passed in as input to the first `assemble()` call) contains the response objects for all its child cartridges.

The default implementation of the Assembler uses Spring to map each cartridge to the appropriate handler based on its content type. This content type corresponds to the template identifier that was used to create the content item object. If no cartridge handler is defined for a particular content type, the instance configuration is passed through as the response model.

Example

For example, consider the following content item:

```
NestingDollContentItemSubclass nestingDoll
```

This content item represents a Russian Nesting Doll. It includes properties for its name, color, and its child content item:

```
nestingDoll.name = "Nesting Doll"  
nestingDoll.color = red  
nestingDoll.child = secondNestingDoll
```

The `secondNestingDoll` contained within is green. It contains a `thirdNestingDoll`, which is blue. Assuming there is no cartridge handler for `NestingDollContentItemSubclass`, an `assemble(nestingDoll)` call executes the following:

1. The pre-order traversal starts. There is no cartridge handler for `NestingDollContentItemSubclass`, so no `initialize()` or `preprocess()` calls are made for `nestingDoll`.
2. Similarly, no calls are made for `secondNestingDoll` or `thirdNestingDoll`. At this point, the pre-order traversal is complete.
3. The post-order traversal starts. The `thirdNestingDoll` object is returned as-is, since there is no handler to invoke a `process()` method.
4. Similarly, the `secondNestingDoll` and `nestingDoll` objects are returned, unmodified. Serialized to JSON, the response looks like the following:

```
@type": "NestingDollTemplateType",
"name": "Nesting Doll",
"color": "red",
"child": [
  {
    @type": "NestingDollTemplateType",
    "name": "Second Nesting Doll",
    "color": "green"
    "child": [
      {
        @type": "NestingDollTemplateType",
        "name": "Third Nesting Doll",
        "color": "blue"
        "child": []
      }
    ],
  },
]
}
```

What if you create a cartridge handler for `NestingDollContentItemSubclass` that doesn't override the `initialize()` or `preprocess()` methods, but implements logic to add a property `colorType` of value `warm` or `cool`, based on the `color` property? Steps 1-2 above don't change, but Step 3 invokes the new logic, and the property shows up in the response:

```
@type": "NestingDollTemplateType",
"name": "Nesting Doll",
"color": "red",
"colorType": "warm",
"child": [
  {
    @type": "NestingDollTemplateType",
    "name": "Second Nesting Doll",
    "color": "green",
    "colorType": "cool",
    "child": [
      {
        @type": "NestingDollTemplateType",
        "name": "Third Nesting Doll",
        "color": "blue",
        "colorType": "cool",
      }
    ],
  },
]
}
```

About content items

A content item is a set of `key:value` pairs where the key is a property name and the value may be any primitive type, or another content item. The `com.endeca.infront.assembler.ContentItem` interface extends `java.util.Map`.

Content items in the Assembler represent either structural components of an application page, or GUI components on the page itself. A call to the `Assembler.assemble(ContentItem)` method takes a `ContentItem` containing configuration as input, and returns a content item as output. The response content item can encompass an entire page in an application, with each sub-section of the page (such as the search box or the search results list) represented as its own nested content item.



Note: In the default implementation of the Assembler, the `ContentItem` interface is implemented by the `com.endeca.infront.assembler.BasicContentItem` class. The navigation cartridges in the `com.endeca.infront.cartridge` package extend this implementation for their individual use cases.

About ContentInclude and ContentSlotConfig objects

The default Assembler implementation typically takes a `ContentInclude` or `ContentSlotConfig` object as input to the Assembler. The first specifies a content item by URI, while the second retrieves a content item from a specified folder according to template type and ID restrictions, trigger criteria, and content item priority.

Both methods retrieve the associated configuration for the content item in Workbench.

Defining a ContentInclude object

A `ContentInclude` object defines a single content item to pass into the Assembler (though keep in mind that a content item may contain additional content items as children). It resolves a URI to a content item within a configured content source (typically the Endeca Configuration Repository).

Defining a ContentSlotConfig object

Unlike a `ContentInclude` object, which explicitly specifies a content item to pass as input to the Assembler, the `ContentSlotConfig` object defines a set of criteria for dynamically retrieving one or more content items at runtime. In most cases the content administrator creates and populates `ContentSlotConfig` objects through editors in Experience Manager, although you can still programmatically instantiate them if necessary.

The dynamic content slot is populated based on the following restrictions:

- **Content paths** — The path or paths to content folders in Experience Manager. Any content items within the specified folders or within sub-folders are considered valid for retrieval.
- **Template types (Optional)** — The types of content item to retrieve, based on the `type` attribute of the cartridge template used to create it. For example, a Record Spotlight slot in the Discover Electronics reference application is restricted to content items created from a template with `type="SecondaryContent"`.
- **Template IDs (Optional)** — The template IDs to match against. This is a narrower restriction than restricting by template `type`, and instead restricts based on a unique template `id`. For example, a Record Spotlight slot in the Discover Electronics reference application is restricted to content items created from a template with `id="RecordSpotlight"`.
- **Rule Limit** — The number of matching content items to retrieve. This is applied after the above restrictions, and after checking for triggered content items.

Once the list of possible content items has been narrowed down, the `ContentSlotHandler` issues a content trigger request. This checks valid content items against any triggers defined in Experience Manager. Trigger criteria can include:

- The user's search terms or refinement selections, also referred to as their "navigation state."
- Characteristics of the user, such as past buying habits or geographical location. This information constitutes the user's "user segment."
- The current date and time, referred to as "schedule triggers."

The list of results is limited to triggered content items and ordered by the priority assigned to each content item in Experience Manager. The number of results is truncated to the value specified for the content slot (also specified on `ContentSlotConfig`). The Assembler then processes the content items and returns them in its response.

About nesting content items

Content items may contain other content items, which can include both `ContentInclude` references and `ContentSlotConfig` definitions

For example, in Discover Electronics the `/browse` path corresponds to a page within the sitemap. The `browse` page consists of a content slot that references the Web Browse Pages folder. Most of the pages within the Web Browse Pages folder contain a mixture of static and dynamic content items. As the Assembler processes the query for `http://www.example.com/discover/browse` (assuming no search terms or refinement selections), the following steps occur:

1. The Assembler is invoked with a `ContentInclude` item with the URI `/pages/browse`.
2. The Assembler invokes the `ContentIncludeHandler` to retrieve the configuration for the `browse` page, which is a `ContentSlotConfig` that specifies a single content item from the Three-Column Page collection.
3. The Assembler invokes the `ContentSlotHandler` to retrieve the highest priority content item within the Three-Column Page collection. In this case, it is the `Default Browse Page`, which is a `ThreeColumnPage`.
4. There is no cartridge handler configured to process the `ThreeColumnPage`, but it contains child content items, so the Assembler goes on to process the child content items:
 - a. It passes the configuration for the search box cartridge through to the response object.
 - b. It invokes the `BreadcrumbsHandler` to process the breadcrumbs cartridge.
 - c. It invokes the `ContentSlotHandler` to process the navigation slot, which in turn retrieves the `Default Guided Navigation` configuration from the `Guided Navigation` collection and invokes `DimensionNavigationHandler` to process it.
 - d. It invokes the `SearchAdjustmentsHandler` to process the search adjustments cartridge.
 - e. It invokes the `ContentSlotHandler` to process the results list slot, which in turn retrieves the `Default Results List` configuration from the `Results List` collection and invokes `ResultsListHandler` to process it.
 - f. It invokes the `RecordSpotlightHandler` to process the spotlight records.

About serialization and de-serialization

The Assembler serializes content items, including any Workbench configuration, as XML in the Endeca Configuration Repository (or on a content server in a production environment). This XML is deserialized during an `assemble()` call when retrieving a content item to pass it to its cartridge handler.

You can also use the included classes to serialize the Assembler response to a format that is more convenient for use in your front end application. For example:

```
// Invoke the Assembler on myContentItem
ContentItem responseContentItem = assembler.assemble(myContentItem);
// Serialize the Assembler response to JSON
response.setCharacterEncoding("UTF-8");
```

```
JsonSerializer serializer = new JsonSerializer(response.getWriter());
serializer.write(responseContentItem);
```

If you deploy the Assembler as a service, then responses are serialized by default. Whether you retrieve the XML or JSON version depends on the URL that you access.



Note: The Assembler includes default implementations of a `JSONResponseWriter` and an `XMLResponseWriter`. You can provide your own implementation if you need to output the Assembler response to a different format (such as a different XML representation).

Refer to the documentation for the `com.endeca.infront.serialization` package in the *Assembler API Reference (Javadoc)* for detailed information.

The Assembler eventing framework

The Assembler includes an eventing framework that fires events at different points in an `assemble()` call. Creating listeners for these events enables cartridge handlers to retrieve or modify data at more granular points in the Assembly process.

Note that logic included in an event listener is evaluated for every cartridge handler, and that event listeners do not have access to the current Assembler request or to the navigation state.

Related Links

[Creating an event listener](#) on page 176

The Assembler provides an empty implementation of the `AssemblerEventListener`, `AssemblerEventAdapter`. You can extend this implementation to create a listener that triggers on an Assembler event.

Assembler event framework reference

The Assembler includes an `AssemblerEventListener` interface that you can use to create and register event listeners.

The Assembler fires the following events:

Event	Condition
<code>assemblyStarting</code>	Fires when an <code>assemble()</code> call starts.
<code>assemblyComplete</code>	Fires when an <code>assemble()</code> call completes.
<code>assemblyError</code>	Fires when an <code>assemble()</code> call is aborted due to an unrecoverable error.
<code>cartridgeInitializeStarting</code>	Fires when a cartridge handler calls the <code>initialize()</code> method.
<code>cartridgeInitializeComplete</code>	Fires when a call to the <code>initialize()</code> method completes.
<code>cartridgePreprocessStarting</code>	Fires when a cartridge handler calls the <code>preprocess()</code> method.
<code>cartridgePreprocessComplete</code>	Fires when a call to the <code>preprocess()</code> method completes.

Event	Condition
cartridgeProcessStarting	Fires when a cartridge handler calls the <code>process()</code> method.
cartridgeProcessComplete	Fires when a call to the <code>process()</code> method completes.
cartridgeError	Fires when a cartridge fails due to a local error. This stops execution of the cartridge handler workflow, and prevents any additional events from firing.

Event payload

Each Assembler event has an `AssemblerEvent` payload consisting of three objects:

- `Assembler` — the `Assembler` object responsible for servicing the request.
- `ContentItem` — the content item currently undergoing processing within the `assemble()` call.
- `CartridgeHandler` — the cartridge handler associated with the event.

About Assembler error handling

In case of an error during processing, the `Assembler` API defines two kinds of exceptions: `AssemblerException` and `CartridgeHandlerException`.

The exceptions are distinguished as follows:

Exception	Description
<code>AssemblerException</code>	Indicates that an exception occurred while creating or processing an <code>Assembler</code> request. Exceptions of this type indicate that the entire assembly process was terminated.
<code>CartridgeHandlerException</code>	Indicates that an exception occurred while invoking a single cartridge handler. Exceptions of this type do not terminate the entire assembly process.

Both types of exceptions are returned as part of the `Assembler` response.

Error handling in the Assembler service

The `Assembler` service returns an HTTP status code of 200 (OK) regardless of whether any exceptions occurred during `Assembler` processing. Error conditions are serialized as exceptions in the `Assembler` response, as with the following example:

```
{
  @error: "com.endeca.infront.assembler.CartridgeHandlerException"
  description: "Detailed cartridge handler error description"
}
```

Unchecked exceptions result in the `Assembler` service returning HTTP status code 500 (Internal Server Error).

Chapter 3

About cartridges and content items

This section describes how cartridges expose content in an application.

About cartridges

The component model consists of configurable *content items*. Cartridges expose these content items in a rendered format for the front-end application.

A content item is a map of properties or key-value pairs, where the key is a string representing the property name and the value may be any primitive type (including String, Boolean, List, and Map) or another content item. This allows for content items to be nested within other content items, forming a content tree that represents the structure of a Web page and all its components.

There are generally two kinds of content items within an application:

- *Container content items* are primarily structural components. They define the logical (and sometimes physical) structure of the content to be rendered by an application. The top-level container typically represents a Web page with sections that can contain other content items (leaf content items or, occasionally, other containers). In a Web application, these sections may correspond to areas on the page with certain assumptions about layout and rendering. In other applications, they may represent logical groupings of related components.
- *Leaf content items* are typically functional components. They contain information about content to be displayed in the application, and typically encapsulate the configuration for a particular feature, such as a Guided Navigation component, spotlight, or results list. Leaf content items are also referred to as *cartridges*.

A page may contain cartridges directly (in which case the configuration for the cartridges is triggered along with the page) or the page can contain a dynamic slot, which serves as a placeholder for cartridges that can be triggered independently of the page in which they display.

Anatomy of a cartridge

A cartridge is a functional component that a content administrator can choose to display on a page.

The core aspects of a cartridge are the following:

- The **cartridge instance configuration**, which consists of a content item created by a business user in Experience Manager
- The **cartridge handler**, which is the Assembler component that contains the processing logic for the associated feature

- The **response model**, which is the content item returned from the Assembler to the application for rendering

The configuration model for a cartridge is defined by a *cartridge template*, which describes the properties that can be configured as well as the interface through which the content administrator can specify their values in Experience Manager. Cartridges typically have configuration options specific to the cartridge's function, such as the number of refinements to display (and the order in which to display them) for a Dimension Navigation cartridge; the records to promote for a Spotlight cartridge; or the sort options and records per page for a Results List cartridge.

1. At query time, the configured values of the cartridge properties become an input to the Assembler.

The Discover Electronics reference application contains several sample templates or cartridges that demonstrate core Endeca functionality. You can customize them for your own application or write your own templates in order to add or remove configuration options or to pass additional information to the Assembler or the front-end application.

2. At query time, the Assembler invokes the appropriate cartridge handler to process the cartridge configuration.

The core cartridge handlers also have access to information about the initial request context that triggered the cartridge. The cartridge handler is responsible for generating a response model based on this configuration. In most cases this involves fetching content from an external resource.

In the case where the configuration model is the same as the response model, no cartridge handler is needed; the default behavior of the Assembler is to pass the configuration properties through to the response model.

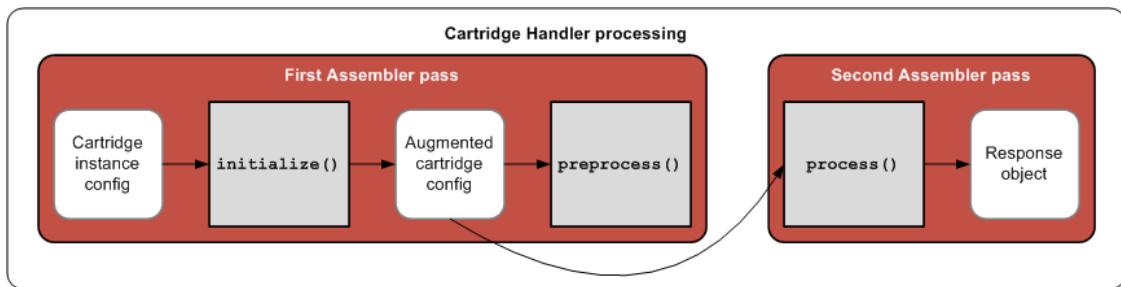
3. The Assembler passes the response model to the corresponding renderer in the application.

As a best practice, the application should contain several modular *renderers*, each intended to handle the output model for a particular cartridge or cartridge type. The Discover Electronics application includes reference JSP pages to render each cartridge. These renderers are intended to be updated for styling or otherwise customized for your application.

About the CartridgeHandler interface

A cartridge handler takes a content item representing the cartridge instance configuration as input and is responsible for returning the response as a content item.

The CartridgeHandler interface defines three methods: `initialize()`, `preprocess()`, and `process()`.



The `initialize()` method provides an opportunity for the cartridge handler to augment the cartridge instance configuration specified in Experience Manager with configuration from other sources. This can be used to define default behavior for a cartridge in the case where there is no Experience Manager configuration, or to override the Experience Manager configuration for the current query. The `initialize()` method should return a content item containing the complete configuration for the cartridge from all possible configuration

sources. This augmented configuration item can either be the mutated input content item or a new instance of `ContentItem`, and is used as input to both the `preprocess()` and `process()` methods.

Because the `preprocess()` method is called on all cartridges before `process()` is called on any cartridges, it provides an opportunity to coordinate processing between cartridges. Many of the core Endeca cartridges make use of this mechanism in order to consolidate queries to an MDEX Engine among several cartridges during the course of a single assembly cycle.

The `process()` method is responsible for returning a `ContentItem` that represents the cartridge response.

A cartridge handler need not define any behavior for `initialize()` or `preprocess()`. The `AbstractCartridgeHandler` class exists to simplify the task of implementing the `CartridgeHandler` interface. It provides empty implementations for `initialize()` and `preprocess()`. Subclasses of `AbstractCartridgeHandler` need only implement the `process()` method to return the response object. They can optionally override the `initialize()` and `preprocess()` methods.

Part 2

Designing an Assembler Application

- *Planning an Assembler Application*
- *Creating Experience Manager Templates*

Chapter 4

Planning an Assembler Application

This section covers considerations for designing your Assembler application.

About planning your application sitemap

An Assembler application consists of a combination of static pages and dynamic pages that contain content related to an end user's navigation state. Your planned sitemap helps determine what pages and content folders you should create for your application.

Consider a site with the following structure:

- about
 - contact
 - faq
- promotions
 - christmas
 - mothersDay
- browse
- details

In this case, each of the pages maps directly to a set of content. To separate most of the content out from the site structure and support dynamic triggering, the organization of an Assembler application is divided into the pages within an application, and the content that populates them:

- pages
 - about
 - contact
 - faq
- browse
- details

- content
 - guided navigation
 - record details
 - browse pages

- default
- christmas
- mothers day
- spotlights
 - top rated
 - best sellers

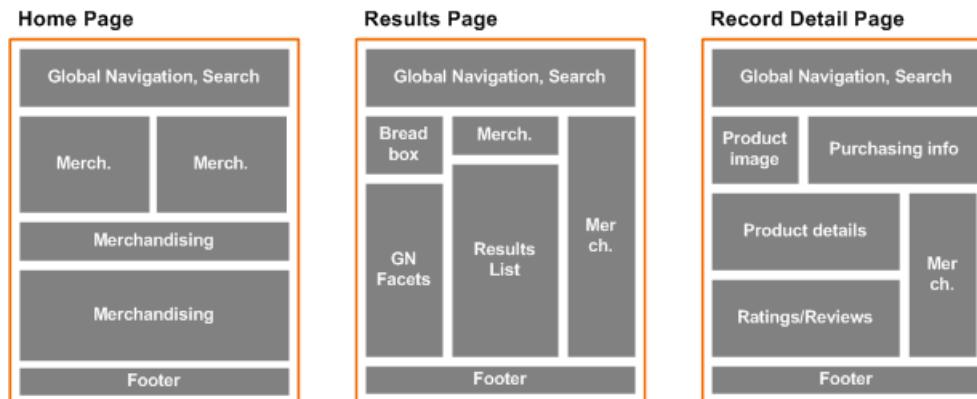
In the example above, the promotional Christmas and Mother's Day pages no longer exist as explicit pages. Instead, the content associated with those promotions is stored as available "browse" page configurations that each trigger during a specified date range.

You can refer to the Discover Electronics reference application for a further example of how content and pages can be separated. When planning your own application, you should consider which locations in your site are best represented as pages, and which locations consist of triggered content on an existing page.

About page types

A typical application has several types of pages that may display under different circumstances or contain different content.

For example, a site may have the following three basic page types:



These pages may differ in the following ways:

- **They are intended to display in different contexts.** The home page displays before the user has made any selections. The results page displays only when the user has performed some search or navigation query. The record detail page displays only when a user has selected a specific product. These conditions are configured in Experience Manager as triggering criteria.
- **They display different types of content.** A home page or category page typically displays high-level promotions and merchandising. A results page displays a list of record results as well as additional controls for the user to select additional facets or otherwise refine the search. A record detail page displays detailed product information as well as controls for transactions (such as add to cart, wishlist, and so on). These differences in content imply differences in layout, which is configured at the template level.
- **They are accessed via different URLs.** The home page is accessed at the base URL for the site. Search results pages may be accessed at a URL that includes the path `/browse/`. Record details pages may be accessed at a URL that includes the path `/detail/`. These URL mappings are typically achieved by setting up individual services for each page type.

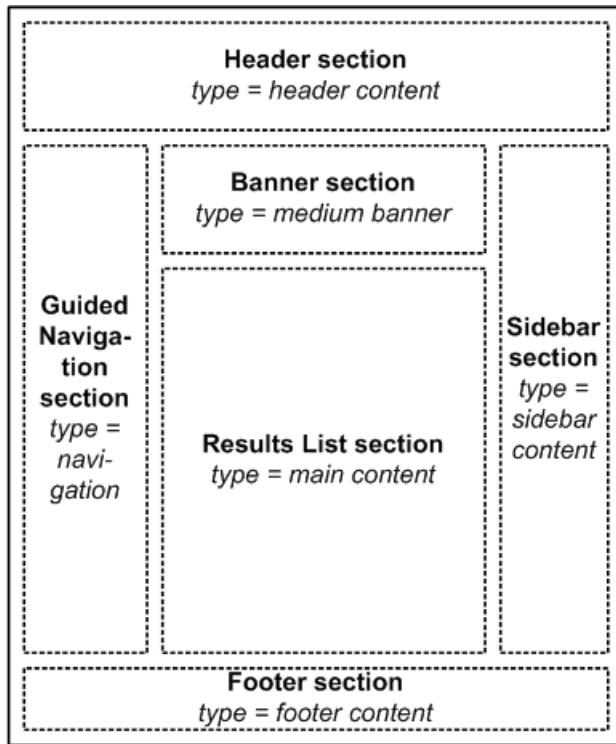
The Discover Electronics reference application includes servlets for results pages and record details pages.

About page structure and content types

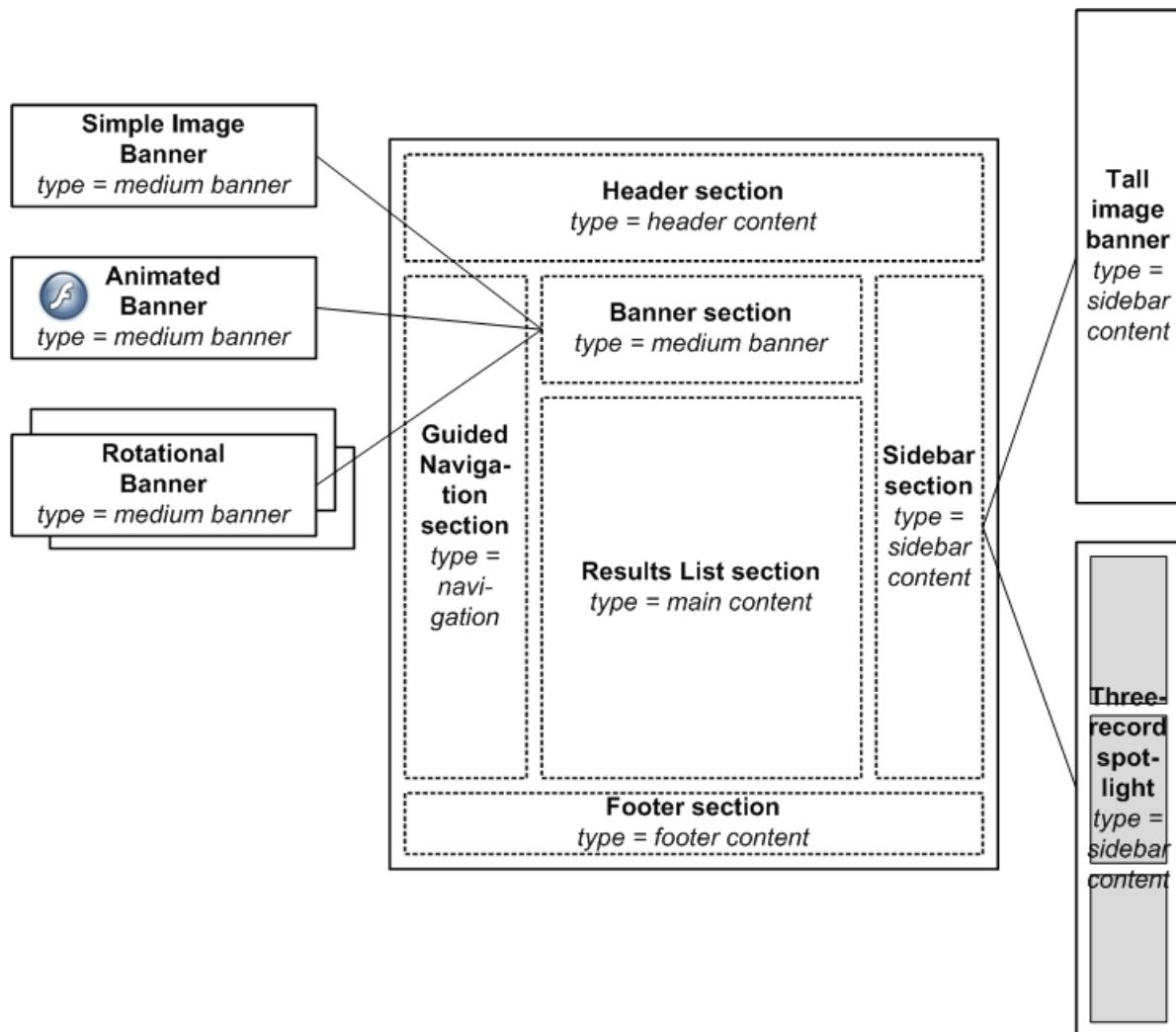
The logical structure of a page, including the types of content it can contain, is defined in an Experience Manager template.

Every template defines a content item that can be processed by the Assembler. A content item describes the logic of how to promote content for display to application users. Content items have several parts: the records in a data set, the conditions that must be met for those records to display, and the templates that determine how those records are rendered in the application.

A page template defines a container content item with sections that can be populated with other content items, such as the following:



Typically, a section represents a physical area on the page, but it can also represent a functional grouping, including content that may not be visible to an end user. Each section has an associated *content type* that determines what kind of content items can be inserted in that section. An application may have multiple cartridges of each type, providing greater flexibility for the content administrator to configure the content for a specific page.



You can create templates for different page types within your application and define which content types are valid for each type of page. You can create templates for high-level page structures and different layouts for a single page type. Each of the content items that can be inserted into a template is itself defined by a template, and may be either another container content item or (more commonly) a leaf content item associated with a front end feature.

About mapping pages to services

You can map the URL paths of pages in your application to specific services.

Services can be used to set attributes on the incoming request before it is processed by the Assembler depending on the type of page being requested, which can control what content is triggered in response to the request, and the format in which the response is returned.

The following is an example from the `WEB-INF\web.xml` file for the Discover Electronics reference application, which maps end user requests to `/services` via a URL redirect and sends them to the application controller, `WEB-INF\services\assemble.jsp`.

```
<!-- Services Definition. For reference, this is implemented as simple jsp pages,
-->
```

```

<!-- but this could also be done with a web framework, such as Spring MVC --
->
<servlet>
    <servlet-name>assemble</servlet-name>
    <jsp-file>/WEB-INF/services/assemble.jsp</jsp-file>
</servlet>

<servlet>
    <servlet-name>assemble-stats</servlet-name>
    <jsp-file>/WEB-INF/services/assemble-stats.jsp</jsp-file>
</servlet>

<servlet>
    <servlet-name>autosuggest.json</servlet-name>
    <jsp-file>/WEB-INF/services/autosuggest-json.jsp</jsp-file>
</servlet>
<servlet>
    <servlet-name>link</servlet-name>
    <!-- link service content omitted for brevity -->
</servlet>

<servlet-mapping>
    <servlet-name>autosuggest.json</servlet-name>
    <url-pattern>/servlet/autosuggest.json/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>link</servlet-name>
    <url-pattern>/servlet/link.json/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>assemble-stats</servlet-name>
    <url-pattern>/servlet/stats/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
    <servlet-name>assemble</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
</servlet-mapping>

```

When a content administrator defines a new application page in the reference application, requests on that page are mapped to the `/services` servlet. Your application should include similar logic for mapping arbitrary pages to a controller, though you may also choose to explicitly define additional services for certain pages within your site. Additionally, your UI tier must be able to resolve whatever links you expect your content administrators to create. For more information about handling application URLs, see "Working with Application URLs."

Creating a page

The Content Tree in the left pane of Experience Manager is divided into two sections: **Pages** and **Content**. You create pages within the **Pages** section.

You must deploy and provision your application with the EAC in order to modify it in Workbench.

To create a page:

1. Login to Workbench and navigate to Experience Manager.
2. Mouse over the **Pages** heading in the Content Tree.

The drop-down menu arrow appears on the right.

3. Click the drop-down menu arrow and select **Add Page**.

The **Add Page** panel appears.

4. Enter a **Name/Path** for the new page.

This is the part of the URL path that uniquely identifies the page within your application.

5. Click **Create**.

The new page is added to your application.

A page exists as a content item in Experience Manager. A content administrator can configure it directly by selecting a template with included editors, or they can specify a template with a dynamic slot to populate the page from one or more selected content folders.

About content folders

Before a content administrator can configure dynamic content items within an application, you must create content folders to contain those items. Content items within the same folder are evaluated against each other at runtime to determine which item (or items) should be returned to populate a defined section of the current page.

In Experience Manager, content folders define the top-level organizational structure of an application, in which the content administrator can browse for content. If a query satisfies the trigger criteria for multiple content items within a folder, items with higher priority take precedence over those with lower priority. A single application request may trigger content items from multiple folders.

Content folders have the following properties:

- **Template type** — Specifies the type of content items that can be created in this collection, as defined by the `type` attribute of the content template.
- **Template ID** — Specifies the type of content items that can be created in this collection, as defined by the `ID` attribute of the content template. This is more restrictive than specifying by template type, as an ID is unique to a single template.

Oracle recommends that you create at least one content folder for pages and one for each slot on the page that can contain either shared or variable content. This provides a logical organization of content within Experience Manager. It enables content to be triggered independently of the pages that contain them and also enables content in one slot to be triggered independently of content in another slot.

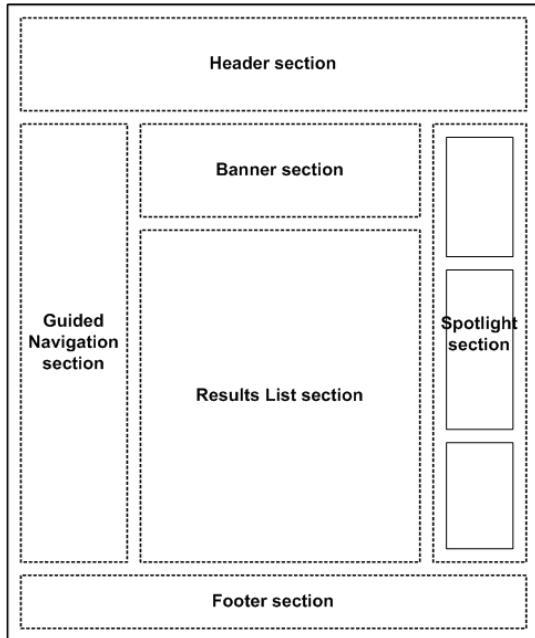
For example, the Discover Electronics reference application includes the following content folders :

- **Mobile \ Mobile Browse Pages** — Top-level page configuration for pages viewed from a mobile device. Mobile pages must be more streamlined than Web pages, so they require a different page template.
- **Shared \ Auto-Suggest Panels** — Configuration for the auto-suggest panel that displays when a user starts to enter a search query. The Shared collections return the same response model for both the Mobile and Web versions of the application, but the renderers vary based on the client.
- **Shared \ Detail Pages** — Configuration for record details pages within the application.
- **Shared \ Guided Navigation** — Configuration for the Guided Navigation menu.
- **Shared \ Results List** — Configuration for a list of search results.
- **Web \ Category Spotlights** — Category-specific product spotlights that display above the search results when a user navigates to those products.
- **Web \ Web Browse Pages** — Top-level page configuration for Web pages. These templates are structural and primarily consist of dynamic slots that pull in content items from other collections to populate the page.

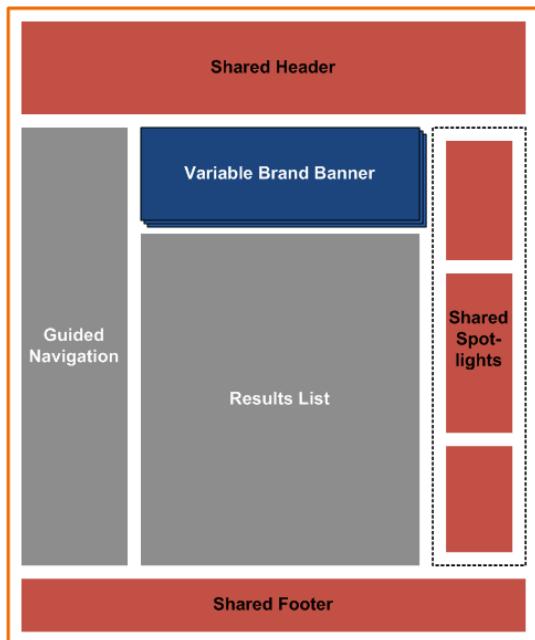
Content folders example

Content folders determine which content items are evaluated and returned when populating a dynamic section of an application page.

Suppose you have a site where a typical structure for a search and navigation page looks like the following:



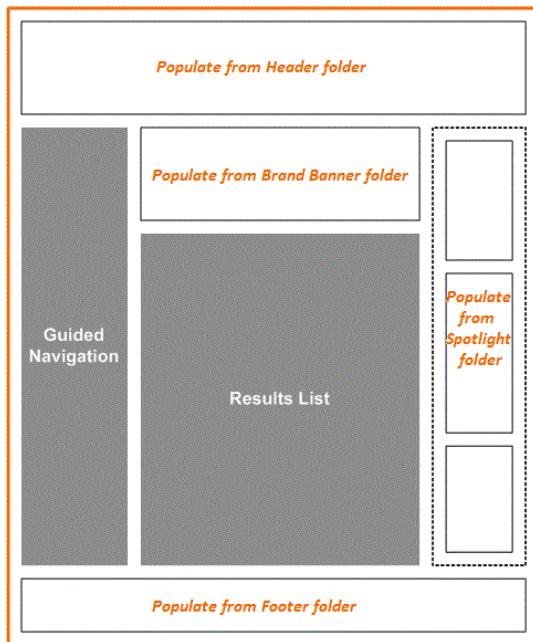
Based on this template, the content administrator wants to configure a page for a specific trigger (for example, Category > Cameras > Digital Cameras) using contextual, shared, and variable content as in this picture:



- The header and footer are populated as dynamic slots with default triggering criteria, in order to avoid defining them multiple times for a large number of pages.

- The Guided Navigation and Results List cartridges are configured specifically for this page and do not need to vary based on criteria other than the page triggers.
- The Banner area is configured to display a different image depending on the brand that the site visitor has selected.
- The Spotlight area displays a mix of promotions based on triggers that are independent of the triggering criteria for the page itself. For example, a "Holiday Specials" spotlight may display for the date range between November 1 and January 2.

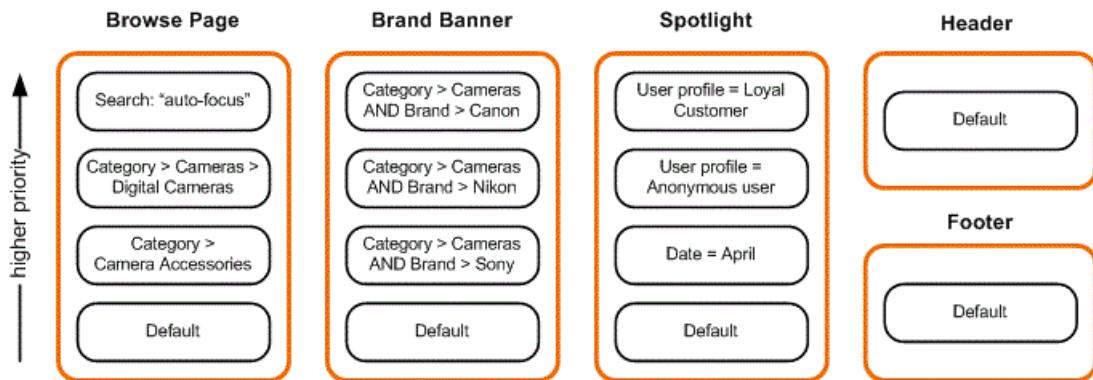
The configuration for the page (as specified in Experience Manager) looks like this:



The configuration for Guided Navigation (including which dimensions to display and which dimension values to boost or bury within those dimensions) and for the Results List (including default sort options and record boost and bury) are specified as part of the page configuration. The other slots on the page contain only placeholders. The actual Header, Footer, Banner, and Spotlight content items that display when someone visits the site are defined in their respective content folders.

The mechanism for populating these slots is the same regardless of whether the content that should display in each slot is shared or variable content. The only difference between the two kinds of content is in the trigger criteria on the content items within those collections: variable content, such as the Spotlight, has triggers that are more specific than the page trigger. Reusable content, such as the generic header and footer, has triggers that are more general than or orthogonal to the page trigger.

When the content administrator has created all the content needed to populate this page (and a few other pages), the application may include the following content items in the following folders:

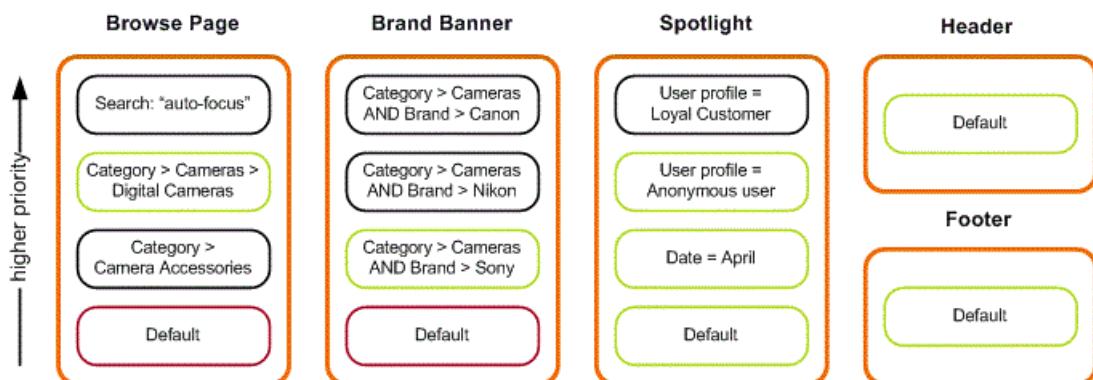


The content folders are configured as follows:

- The Browse Page folder contains all the content items representing search and navigation pages in the site.
- The Brand Banner folder contains cartridges of type `MediumBanner` that are appropriate to display in the Banner slot. This dynamic slot on the Browse page has an evaluation limit of 1, since the page is designed to display only one banner at a time.
- The Spotlight folder contains cartridges of type `SidebarItem` because items created in this collection are intended to display in the Spotlight slot in the right column. Because this space is intended to display several independently triggered spotlight items, the evaluation limit for the dynamic slot on the Browse page is 3.
- The Header and Footer folders each contain cartridges of type `FullWidthContent`.

Each page or content item within these folders has an associated trigger and priority (relative to the other items in the same folder) specified by the content administrator in Experience Manager.

When a site visitor refines on `Category > Cameras > Digital Cameras` and `Brand > Sony`, the following content triggers:



- The Digital Cameras page is returned as a Page, which includes the content administrator's configuration for Guided Navigation and for Results List. Note that the Default page (with a trigger of "Applies at all locations") is also eligible to fire, but the Digital Cameras page has a higher priority, therefore it takes precedence and the Default page does not fire.
- The Banner slot is populated by the highest priority content item in the Brand Banner folder that matches the user's navigation state. In this case, it is the Sony cameras banner. Again, there is a Default banner but it does not fire because it has a lower priority.
- The Spotlight slot is populated by the highest priority content items in the Spotlight folder that match the user's navigation state. In this case, the Default spotlight does fire because there is room for three spotlights in this slot and that item has a high enough priority (among those that satisfy the user's context) to be included. These three content items display in the Spotlight area in order of priority.

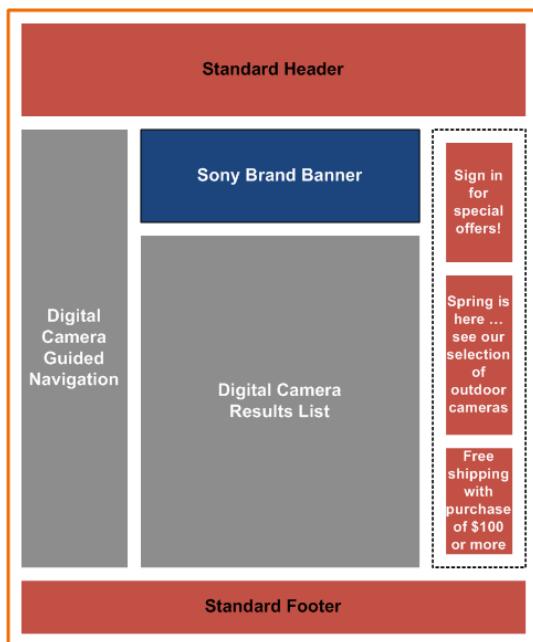
- The Header and Footer folders have only one content item each, which is set to display at all locations, therefore the same content is returned for this page as for all pages.

In this example, content is returned from five content folders. Priority between items is specified within each folder. It does not make sense to prioritize the Sony cameras banner against the April spotlight cartridge, for example, because they are not competing against each other to be displayed on the page. In general, content items with more specific trigger criteria should have a higher priority than those with more general criteria, especially if they are used in a dynamic slot with an evaluation limit of 1.

Oracle recommends that you create separate content folders for each area on the page, even if they have the same content type. For example, if you want to have two banners on the page, each populated via dynamic slots, they should reference two different folders, or else the same banner (the one with the highest priority for the current navigation state) is returned for both sections of the page.

Oracle also recommends that you do not mix reusable and variable content within the same folder. For example, if a slot (such as the Spotlight slot) can be populated with either reusable or variable content, create two different folders, Reusable Spotlights and Variable Spotlights. The content administrator can configure a particular page to populate the Spotlight slot from either folder as applicable. In order to populate the same slot with a mixture of reusable and variable content, the content administrator can insert two (or more) placeholders in the Spotlight slot, each referencing the corresponding folder for each type of content.

The final result for the site visitor who is looking at Sony cameras looks something like the following:



Creating a content folder

The Content Tree in the left pane of Experience Manager is divided into two sections: **Pages** and **Content**. You create content folders within the **Content** section.

You must deploy and provision your application with the EAC in order to modify it in Workbench.

To create a content folder:

1. Login to Workbench and navigate to Experience Manager.
2. Mouse over the **Content** heading in the Content Tree.

The drop-down menu arrow appears on the right.

3. Click the drop-down menu arrow and select **Add Folder**.

The **Add Content Folder** panel appears.

4. Enter the **Name** of the folder you wish to add.

5. Optionally, select a content type restriction.

The drop-down list is populated based on the available `type` values for the set of templates uploaded to the application.

This selection restricts the content items within the folder to the specified type.

6. Click **Add**.

The new content folder is added to the Content Tree in Experience Manager.

About moving content folders

You can move and re-organize content folders in the Content Tree within Experience Manager.

If you move a content folder that includes dynamic content referenced elsewhere in the application, a warning dialog appears with a list of content items that rely on the content you are moving. You must manually update these content items if you proceed with the move.

Chapter 5

Creating Experience Manager Templates

This section describes the process of creating templates that enable the configuration of content items in Experience Manager.

About creating templates

Templates define the content structure of a content item as well as the editing interface that content administrators can use to configure instances of content items in Experience Manager.

In general, you create one or more templates that define the high-level structure of the pages in your application. These templates define sections that can be populated with other content items, or cartridges. Cartridge templates specify the properties required to display the content for that component. This may include values that the client application uses directly to render the information, or inputs into the Assembler for processing (such as query parameters to the MDEX Engine).

While cartridges and template properties typically determine aspects of the visual appearance of the page, keep in mind that they can also represent page elements that are not visible in the application. For example, a property can contain meta keywords used for search engine optimization, or a cartridge can include embedded code that does not render in the page but enables functionality such as Web analytics beaconing.

The Discover Electronics reference application provides sample page templates for some standard page types, as well as templates that enable configuration of the core set of cartridges in Experience Manager. These cartridges cover basic Endeca functionality, and are provided as a starting point for your application. You can customize them to suit your needs.



Note: In some cases, the reference application includes more than one template for the same functional cartridge. This is in order to enforce the proper constraints on which cartridges are available to insert in specific template slots. The only difference between the different versions of these templates is the template type.

This section concentrates on the basic template elements that enable you to create top-level page templates appropriate to your application. Details about the template configuration for core cartridges are covered in the "Feature Configuration" section. Reference information about the full range of properties and editors that can be used in templates is provided in the appendix to this guide.

Anatomy of a template

Top-level templates, which define an entire page, and cartridge templates, which drive the content of individual components, are both XML documents that share the same structure.

Templates can be broken down into three parts:

- **General information** such as the template type, description, and thumbnail image. This information is used in Experience Manager to help the content administrator select the appropriate template for a page or section.
- **Content item definition.** In this part of the template, you explicitly declare all the properties of the content item that is described by this template. Property types can include Strings, Lists, and Booleans. You can also specify the default values of properties here.
- **Editor panel definition.** These allow you to define the editing interface in Experience Manager for this content item. Properties are generally associated with an editor that enables content administrators to configure the content items that they create within the tool.

```

<?xml version="1.0" encoding="UTF-8"?>

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors" type="MainColumnContent" >
  <Description>Displays dimension search results.</Description>
  <ThumbnailUrl>/thumbnails/PageTemplate/DimensionSearchResults.png</ThumbnailUrl>

  <ContentItem>
    <Name>New Dimension Search Results</Name>
    <Property name="displayCompoundDimensions">
      <Boolean>false</Boolean>
    </Property>
    <Property name="maxEntriesPerDimension">
      <String>3</String>
    </Property>
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <editors:BooleanEditor propertyName="displayCompoundDimensions" label="Compound
      dimensions" />
      <editors:NumericStepperEditor propertyName="maxEntriesPerDimension" label="Max
      entries / dimension" minValue="1"/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

By defining the properties in the template along with how they can be configured in the tool, you ensure that the content configured in Experience Manager matches the expected configuration model for the corresponding cartridge handler in the Assembler.

About the template XML schema

All templates share the same primary schema. In addition, there are several other namespaces that are commonly used in templates.

The template schema

The template schema describes the overall structure of page and cartridge templates. It is also used for primitive property types such as String and Boolean.

All templates must include the following schema declaration:

```
<ContentTemplate xmlns="http://endeaca.com/schema/content-template/2008"
    type="PageTemplate" >
```

The Xavia schema

The Xavia namespace is used for properties that are lists or items (collections of key-value pairs). Include the following namespace declaration in templates that use these properties:

```
<ContentTemplate xmlns="http://endeaca.com/schema/content-template/2008"
    xmlns:xavia="http://endeaca.com/schema/xavia/2010"
    type="PageTemplate" >
```

The editors schema

There is no formal schema for editor configuration, however, by convention, they are associated with an `editors` namespace to distinguish these elements from the template schema. Include the following namespace declaration in all templates:

```
<ContentTemplate xmlns="http://endeaca.com/schema/content-template/2008"
    xmlns:xavia="http://endeaca.com/schema/xavia/2010"
    xmlns:editors="editors"
    type="PageTemplate" >
```

Template identifiers

Templates are saved as XML files named `template.xml` that are then uploaded to Experience Manager. Each template is required to have a unique identifier.

The template identifier is the folder name where the `template.xml` file resides. For example, in `ThreeColumnNavigationPage\template.xml`, the folder name, `ThreeColumnNavigationPage`, is the template identifier. The identifier displays as the name of the cartridge in the cartridge selector in Experience Manager. The value should be as descriptive as possible to help the user select the appropriate template, for instance, "ThreeColumnWithLargeBanner" or "HolidaySalePromotion."

Template folder names cannot have spaces in them. Folder names must be unique within your application. Templates with non-unique identifiers are not available in Experience Manager. Oracle recommends that you treat templates as part of your application's configuration and store them in a version control system. It can also be useful to include a template version number in a property for debugging purposes.

About the type of a template

Each template has a `type` that indicates where the template fits in an application page.

The type restriction serves two purposes. For top-level container templates, such as those that define a page, a type restriction can be specified for each section of the page. This limits the cartridges that can be inserted into that section. For example, if a template that includes a "HorizontalBanner" section, only cartridges of type "HorizontalBanner" are available to insert into that section in Experience Manager.

Additionally, you can specify a template type in a dynamic slot to restrict the content that appears in that slot. This restriction applies at runtime when content items are evaluated against each other and ranked by priority for display in the application; any content items that do not match the specified template type for a dynamic slot are removed from consideration.

Setting a template type

The template type is specified as a required attribute on the `<ContentTemplate>` element of the `template.xml`. For example:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  xmlns:editors="editors"
  type="PageTemplate" >
```



Note: The `type` attribute is defined as type `xs:Name` in the template schema. This means that valid values for these attributes must:

- be a single string token (no spaces or commas)
- begin with a letter, a colon (:), or an underscore (_)

Numbers are allowed as long as they do not appear at the beginning of the string.

Specifying the description and thumbnail image for a template

The description and thumbnail image for a template display in the template selector and cartridge selector dialog boxes in Experience Manager. Adding a description and thumbnail image to a template is optional.

To specify the description and thumbnail image for a template:

Insert the following elements within `<ContentTemplate>`:

Element	Description
<code><Description></code>	One or two brief sentences to help the content administrator identify the template in Experience Manager. This can include information about the visual layout of the template ("Three-column layout with large top banner") or its intended purpose ("Back to school promotion").
<code><ThumbnailUrl></code>	The absolute URL to a thumbnail image that shows a sample page or section that is based on the template. The images must be hosted on a Web server accessible from the Experience Manager server. Any URL without a protocol or leading slash will be treated as relative to the root of the template structure. If your thumbnail is in the same folder as your <code>template.xml</code> file, you can omit the path altogether. For example, <code><ThumbnailUrl>thumbnail.png</ThumbnailUrl></code> .

Example

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  xmlns:editors="editors"
  type="PageTemplate" >
  <RuleInfo zone="NavigationPageZone" style="PageStyle"/>
  <Description>A page layout with left and right sidebars intended for general category pages.</Description>
  <ThumbnailUrl>http://images.mycompany.com-thumbnails/PageTemplate/ThreeColumn-
NavigationPage.png</ThumbnailUrl>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

About using thumbnail images in Experience Manager

Thumbnail images can help the content administrator identify the appropriate template to use for the pages they create.

The suggested size for thumbnail images is 81 x 81 pixels; smaller images are stretched to fill this size and larger images are cropped to show only the top left corner.

The images must be hosted on a Web server accessible from the Experience Manager server. If the thumbnail image for a template is either not specified or not accessible, a default image displays in the dialog box.

Specifying the default name for a cartridge

The value of `<Name>` within the `<ContentItem>` displays as a label for the cartridge in the Content Tree in Experience Manager.

To specify a default name for a cartridge:

Insert the `<Name>` element inside `<ContentItem>` as in the following example:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  xmlns:editors="editors"
  type="PageTemplate">
  <RuleInfo zone="NavigationPageZone" style="PageStyle"/>
  <Description>A page layout with left and right sidebars intended for general
  category pages.</Description>
  <ThumbnailUrl>http://images.mycompany.com-thumbnails/PageTemplate/ThreeColumn-
  NavigationPage.png</ThumbnailUrl>
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <!-- additional elements deleted from this example -->
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

`<Name>` is a required element. The value you specify in the template becomes the default name when a content administrator creates the page or adds a cartridge. If you insert an empty `<Name/>` element, an empty text field displays in Experience Manager and the content administrator can supply a value.

About defining the content properties and editing interface

A template defines the properties of a content item and also the interface that enables a content administrator to configure the properties.

You define properties within the `<ContentItem>` element in the template. For each property, you specify a name and a property type. You can optionally specify a default value for a property.

You associate editors with properties to enable the content administrator to configure their values within Experience Manager. Properties are generally primitive types such as Strings, Booleans, or Lists. Another type of property is a section, which allows content administrators to insert and configure another content item.

You can choose not to expose a particular property in Experience Manager and simply specify a default value to pass to the Assembler and ultimately to the client application. This is useful for values that do not need to

be configured by the content administrator, but are needed by the Assembler for content processing or by the client application to determine how to render the content.

About template properties

You can define the properties of a content item by nesting any number of `<Property>` elements within the `<ContentItem>` element.

Cartridge properties are typically used for one of the following purposes:

- The property values may be intended to be used directly by the client application. For example, the content administrator may be able to enter text to use a heading or link text, or she may supply a URL to an image. Property values can also contain information such as meta keywords that are part of the page but do not affect its display.
- The values may be intended for the relevant cartridge handler in the assembler to use for processing, for example, parameters for a query to the MDEX Engine (or another external resource) to return the actual content that the application should display.
- Occasionally, a cartridge has no properties (and therefore no configuration options in Experience Manager), but exists only as a placeholder to indicate that a certain functional component should be included on a page. The Assembler inserts the necessary information for this cartridge at query time.

Each property must have a name that is unique within the template. If the property is to be passed through directly to the renderer, this can be any name that makes sense for your application. However, some properties are part of the configuration model for the cartridge. In this case the associated cartridge handler depends on the presence of specific properties in the template.

The property name is specified in the `name` attribute of the `<Property>` element.



Note: The `name` attribute is defined as type `xs:Name` in the template schema. This means that valid values for these attributes must:

- be a single string token (no spaces or commas)
- begin with a letter, a colon (:), or a hyphen (-)

Numbers are allowed as long as they do not appear at the beginning of the string.

You specify the property type by adding a child element of `<Property>`. Properties can be one of two kinds:

- content properties (described by the template schema for primitive properties and Xavia for lists and items)
- structural properties (described by the template schema)

About defining the editing interface for properties

After you have defined the content properties in your template, you can define how those properties can be configured by the content administrator in Experience Manager.

You add content editors inside the `<EditorPanel>` element in the template. The `<BasicContentItemEditor>` element enables you to specify individual property editors that display in Experience Manager and associate them with a particular property.

For example, this excerpt from a sample template defines a configurable string property named `title`:

```
<ContentTemplate xmlns="http://endeeca.com/schema/content-template/2008"
  xmlns:editors="editors"
    type="ResultsPage">
  <!-- additional elements deleted from this example -->
  <ContentItem>
```

```

<Name>Three-Column Navigation Page</Name>
<!-- First define the content property -->
<Property name="title">
  <String>Discover Electronics</String>
</Property>
<!-- additional properties deleted from this example -->
</ContentItem>
<EditorPanel>
  <BasicContentItemEditor>
    <!-- Define an editor for each property that should
        be configurable -->
    <StringEditor propertyName="title" label="Title"/>
    <!-- additional editors deleted from this example -->
  </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

Editors are defined in templates with the `editors` namespace. By convention, the `propertyName` is a required attribute and specifies the property that this editor is associated with. The property must be defined in the `<ContentItem>` part of the template, and must be of the appropriate type for that editor. For example, an `<editors:StringEditor>` cannot be associated with a `<xavia>List>` property. If you define a content editor for a property that does not exist, or that is of the wrong type, a warning displays in Experience Manager when a content administrator attempts to configure the content.

Property editors do not have to be defined in the same order as the properties in the template. The `<BasicContentItemEditor>` renders the editors in a vertical layout in Experience Manager, in the order in which you define them in the template. If you do not want a property to be exposed in the Experience Manager interface, do not define an editor associated with it.

It is possible to create more than one editor associated with the same property. However, be aware that all editors that you define in the template are displayed in Experience Manager, which may be confusing to the content administrator. When the value of a property is changed, any other editors associated with that property are instantly updated with the new value.

Related Links

[Editor property mapping reference](#) on page 221

This section provides an overview of which property types are associated with the different Oracle Endeca Commerce Suite editors.

About configuring editor default values

You can configure default values for Experience Manager editors across the entire application by modifying the editor configuration file, or on a per-template basis by modifying cartridge templates directly.

You can configure Experience Manager editors through the following methods:

- You can configure editors in the editor configuration file, `editors.xml`. This configuration applies to all instances of a specific editor within an application.
- You can configure editors within a cartridge template. This configuration applies to all instances of a specific editor created based on that template. In the case of shared properties, configuration in the cartridge template overrides configuration in `editors.xml`.

For details about configuring the core editors packaged with Oracle Endeca Tools and Frameworks, see the "Template Property and Editor Reference" Appendix.

Related Links

[Template Property and Editor Reference](#) on page 221

This section describes how to define basic content properties and associated editing interfaces in Experience Manager templates.

[About defining the editing interface for properties](#) on page 50

After you have defined the content properties in your template, you can define how those properties can be configured by the content administrator in Experience Manager.

Specifying editor-specific configuration

You can modify the editor configuration file to set configuration that is common to all instances of a specific editor within an application. This can include basic values for the editor, or information used to communicate with an external resource.



Note: Oracle recommends configuring a data service for cases where different editors all need to access a common set of configuration for an external resource.

To add configuration information to the editor configuration file:

1. Navigate to the editor configuration file at `<app dir>\config\editors_config\editors.xml`.
2. Insert an `<EditorConfig>` element directly inside the `<Editor>` tag of the editor you wish to modify.
3. Add your arbitrary configuration information.

The example below includes the configuration inside a nested element, but you can also specify the information as attributes of the `EditorConfig` element:

```
<Editor name="editors:MyEditor">
  <EditorConfig>
    <Arbitrary foo="bar" size="10" resizeable="false"/>
  </EditorConfig>
</Editor>
```

4. Save and close the file.
5. Navigate to the `<app dir>\control` directory.
6. Run the `set_editors_config` script to publish your changes to the Endeca Configuration Repository.

Structural properties

You can define a section within a template by inserting a `<ContentItem>` or `<ContentItemList>` element within a `<Property>`.

Adding a content item property

A content item property defines a template section by creating a placeholder for a nested content item defined by a cartridge template.

Content administrators can configure a section in Experience Manager by choosing a cartridge to insert in the section then configuring the properties of the cartridge.

To add a content item property to a template:

1. Insert a `<ContentItem>` element inside a `<Property>` element.
2. Specify the section type.

Only cartridge templates with a type that matches the section type are presented as options for the content administrator to choose from in Experience Manager. For example, when a content administrator inserts a cartridge in a `RecommendedContent` section, only templates of type `RecommendedContent` display in the **Select Cartridge** dialog box. (Recall that the cartridge template is the part of a cartridge that is exposed in Experience Manager). Because the type of the section property and cartridge templates must match exactly, the type attribute is also defined as type `xs:Name` in the schema and all restrictions that apply to template types also apply to section types.

The following example defines two sections within a template. Note that more than one section in a template can have the same type, as long as your client application expects this kind of content.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:xavia="http://endeca.com/schema/xavia/2010"
                  xmlns:editors="editors"
                  type="PageTemplate">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <!-- additional properties deleted from this example -->
    <Property name="leftColumn">
      <ContentItem type="SidebarItem" />
    </Property>
    <Property name="rightColumn">
      <ContentItem type="SidebarItem" />
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

Adding a content item list property

A content item list allows content administrators to add an arbitrary number of items to a section and to reorder those items within the list using the **Content Tree** in Experience Manager.

Using content item properties to define the subsections of a cartridge restricts the number of subsections available to the content administrator in Experience Manager. For example, the right column of this page template can contain exactly four cartridges:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:xavia="http://endeca.com/schema/xavia/2010"
                  xmlns:editors="editors"
                  type="PageTemplate">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
  <!-- additional elements deleted from this example -->
  <Property name="rightColumn1">
    <ContentItem type="SidebarItem" />
  </Property>
  <Property name="rightColumn2">
    <ContentItem type="SidebarItem" />
  </Property>
  <Property name="rightColumn3">
    <ContentItem type="SidebarItem" />
  </Property>
  <Property name="rightColumn4">
    <ContentItem type="SidebarItem" />
  </Property>
</ContentTemplate>
```

```

    </Property>
  </ContentItem>
<!-- additional elements deleted from this example -->
</ContentTemplate>

```

Although some of the sections can be left empty, no more than four cartridges can be added to the right column.

Using a content item list removes the restriction and allows the content administrator to add an arbitrary number of content items to the right column of the page:

```

<ContentTemplate xmlns="http://endecea.com/schema/content-template/2008"
                  xmlns:xavia="http://endecea.com/schema/xavia/2010"
                  xmlns:editors="editors"
                  type="PageTemplate">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
  <!-- additional elements deleted from this example -->
  <Property name="rightColumn">
    <ContentItemList type="SidebarItem" />
  </Property>
  </ContentItem>
<!-- additional elements deleted from this example -->
</ContentTemplate>

```

To add a content item list to a template:

1. Insert a `<ContentItemList>` element inside a `<Property>` element.
2. Specify the template type.

Only cartridge templates with a type that matches the content item list type are presented as options for the content administrator to choose from in Experience Manager. In the above example, when a content administrator inserts a cartridge in a `RightColumn` section, only templates of type `SidebarItem` display in the **Select Cartridge** dialog box.

3. Optionally, specify a maximum number of content items using the `maxContentItems` attribute. For example:

```

<Property name="RightColumn">
  <ContentItemList type="SidebarItem" maxContentItems="4" />
</Property>

```

By default, the value of `maxContentItems` is 0, which means that there is no limit to the number of cartridges that can be added to a content item list.

About cartridge selectors

Unlike other types of content properties, section properties are always editable; you do not need to explicitly specify an editor in the template.

In Experience Manager, content administrators can select cartridges to insert in sections either by clicking the cartridge **Add** button in the content detail panel or by right-clicking the section in the Content Tree. Both options bring up the cartridge selector dialog box and are enabled automatically when you define a section in the template.

About multiple locales

If your implementation supports multiple locales, you can localize your custom templates.

You can create resource property files for each locale for storing localized strings. Each resource property file name must follow this format: Resources_<locale>.properties where <locale> is the ISO language code. For example Resources_fr.properties indicates that French values are stored in it. Place these files in a locales folder for your custom template: <app dir>\config\cartridge_templates\<template_identifier>\locales. You can specify values that do not change for locale (thumbnail URLs for example) in the single Resources.properties file or directly in the template.xml file.

In the template itself, you can use \${property.name} notation in element content and attributes to reference a localized string in the Resources_<locale>.properties. Only content in the Description, ThumbnailURL, and EditorPanel sections can reference localized strings in the resources properties files.

The following example shows a template that uses notation to reference strings in resource properties files and two resource property files containing the strings that are being referenced.

```

<ContentTemplate xmlns="http://endeeca.com/schema/content-template/2008"
    xmlns:editors="editors"
    type="Type">
    <Description>${my.template.description}</Description>
    <ThumbnailUrl>${my.template.thumbnailurl}</ThumbnailUrl>
    <ContentItem>
        <Name>Dimension Search Auto-Suggest</Name>
        <Property name="title">
            <String>Search Suggestions:</String>
        </Property>
        <Property name="displayImage">
            <Boolean>true</Boolean>
        </Property>
    </ContentItem>
    <EditorPanel>
        <BasicContentItemEditor>
            <GroupLabel label="${my.template.displaySettingsLabel}" />
            <editors:StringEditor propertyName="title" label="${my.template.titleLabel}" enabled="true" />
            <editors:BooleanEditor propertyName="displayImage" label="${my.template.displayImageLabel}" enabled="true" />
            <editors:NumericStepperEditor propertyName="maxResults" label="${my.template.maxSearchSuggestionsLabel}" maxValue="100" enabled="true" />
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

The English resources property file, Resources_en.properties, for this template contains the following:

```

# Dimension Search Auto-Suggest
my.template.description = Display dimension matches as part of the auto-suggest
panel below the search box.
my.template.displaySettingsLabel = Display Settings
my.template.titleLabel = Title
my.template.displayImageLabel = Display Image

```

```
my.template.maxSearchSuggestionsLabel = Max Search Suggestions
```

In the template example, the thumbnail URL is the same for all locales, so the \${my.template.thumbnailurl} notation is only referenced in the Resources.properties file.

```
# Dimension Search Auto-Suggest
my.template.thumbnailurl = /ifcr/tools/xmgr/img/template_thumbnails/type_ahead_2.jpg
```

Managing Experience Manager Templates

You must upload templates to Endeca Workbench before they are available to users in Experience Manager.

Updating Experience Manager templates

All deployment template applications include a set_templates script in the control directory to update Experience Manager templates. You run the script after you locally modify XML template files and you want the templates available in Experience Manager.

This script requires that the templates you modify are stored locally in <app dir>\config\cartridge_templates.

To send updated templates to Experience Manager:

1. In a command prompt, navigate to the control directory of your deployed application.

This is located under your application directory. For example: C:\Endeca\apps\<app name>\control.

2. Run the set_templates script.

For example:

```
C:\Endeca\apps\Discover\control>set_templates.bat
Setting new cartridge templates for Discover
[06.05.13 10:46:52] INFO: Checking definition from AppConfig.xml against existing
EAC provisioning.
[06.05.13 10:46:54] INFO: Updating provisioning for component 'DailyReportGen-
erator'.
[06.05.13 10:46:54] INFO: Updating definition for component 'DailyReportGener-
ator'.
[06.05.13 10:46:55] INFO: Definition updated.
[06.05.13 10:46:55] INFO: Packaging contents for upload...
[06.05.13 10:46:55] INFO: Finished packaging contents.
[06.05.13 10:46:55] INFO: Uploading contents to: http://local-
host:8006/ifcr/sites/Discover/templates
[06.05.13 10:47:05] INFO: Finished uploading contents.
Finished setting templates
```

Troubleshooting problems with uploading templates

Template errors are detailed in the ifcr.log file.

The ifcr.log file is located in:

- %ENDECA_TOOLS_CONF%\logs on Windows
- \$ENDECA_TOOLS_CONF/logs on UNIX

If any templates fail validation, the upload is canceled, and the previous templates remain in Workbench.

Schema validation

Schema validation failure issues an error similar to the following:

```
Setting new cartridge templates for Discover
[06.05.13 11:02:25] INFO: Checking definition from AppConfig.xml against existing
EAC provisioning.
[06.05.13 11:02:26] INFO: Definition has not changed.
[06.05.13 11:02:26] INFO: Packaging contents for upload...
[06.05.13 11:02:26] INFO: Finished packaging contents.
[06.05.13 11:02:26] INFO: Uploading contents to: http://local-
host:8006/ifcr/sites/Discover/templates
[06.05.13 11:02:28] SEVERE: Caught an exception while invoking method 'loadContent'
on object 'IFCR'. Releasing locks.

Caused by java.lang.reflect.InvocationTargetException
sun.reflect.NativeMethodAccessorImpl invoke0 - null
Caused by com.endeca.soleng.eac.toolkit.utility.IFCRUtility$HttpStatusException
com.endeca.soleng.eac.toolkit.utility.IFCRUtility execute - Internal Server Error
(500): com.endeca.ifcr.content.entity.ValidationFailureException:

The following 2 validation failures occurred:
  [/sites/Discover/templates/BadImage] Error reading template XML:
javax.xml.bind.UnmarshalException
- with linked exception:
[org.xml.sax.SAXParseException: cvc-complex-type.4: Attribute 'type' must appear
on element 'ContentTemplate'.]
  [/sites/Discover/templates/BadPageSlot] Error reading template XML:
javax.xml.bind.UnmarshalException
- with linked exception:
[org.xml.sax.SAXParseException: cvc-complex-type.2.4.a: Invalid content was found
starting with element 'EditorPanel'. One of '{"http://endeca.com/schema/content-
template/2008":ContentItem}' is expected.]
```

The errors are written to the `ifcr.log` file. The upload in the example has two errors:

- The `BadImage` template is missing its `type` attribute.
- The `BadPageSlot` template has no `ContentItem` element.

Troubleshooting invalid templates

Some templates may be successfully uploaded to Workbench, but still contain errors that lead to unexpected behavior in Experience Manager.

The most common scenario is when a property is associated with an editor that has constraints, such as a choice editor that can only accept certain string properties. If the default value of the property does not meet the editor's constraints, the editor may discard the value and display the following message in the Content Details Panel when a user adds the cartridge to a page:

Some fields or cartridges within this cartridge may have been updated or removed. Your content has been converted to the new cartridge. To accept these changes click OK and Save All Changes from the List View. To reject these changes, click Cancel. For more information, see "Troubleshooting pages" in the Oracle Endeca Workbench Help.

To avoid this message, ensure that all property defaults are valid options in the associated property editor.

About modifying templates that are used by existing pages

During the development and testing phase of your application deployment, you may need to make adjustments to your templates and update them in Experience Manager.

When Experience Manager populates the **Content Detail Panel** for a content item, it checks the content configuration of the loaded page against the template. If the template has been changed such that it is no longer compatible with the content, Experience Manager displays a warning and attempts to upgrade existing content to fit the new template definition.



Note: Existing configurations are not upgraded to the new template until a content administrator edits and saves the affected content item in Experience Manager.

Experience Manager does the following to ensure that the content and template are in sync:

- If a property has not changed its name or type, the existing values are migrated to the new template.
- If new properties are added to a template, any corresponding property editors become available in Experience Manager when a content administrator edits a content item based on the updated template. If you specify default values for the new properties, they are applied when a content administrator edits and saves the content item using the updated template.
- If properties are removed from a template, the corresponding property editors no longer display in Experience Manager when a content administrator edits a content item based on the updated template. The properties and their values are deleted from the page configuration.
- If the type of a property has changed (for example from string to list) within a template, the corresponding property editor (if one is specified) becomes available in the Experience Manager when a content administrator edits a content item based on the updated template. The existing value for the property does not display in Experience Manager until the content administrator saves the new value, replacing the previous value.
- If a content item or content item list property has changed to specify a different content type, then any existing cartridge in that section is ejected and its configured properties deleted.
- If the default value of an existing property has changed, it is only applied to new content items that are created based on the updated template. In existing pages, the previously saved value of the property (even if it is an empty string) is preserved regardless of whether it was originally a default or user-specified value.
- Some editors may implement specific update-handling logic in cases where an existing value does not meet the editor's constraints.



Note: Changing the `name` of a property is equivalent to removing the property with the old name and adding a property with the new name. Avoid changing the names of properties that are being used by existing pages. To change the display name of a property on Experience Manager, use the `label` attribute instead.

Managing template changes

Because existing content is not automatically updated to the new templates, and default values are never updated in existing pages, any changes that you make to your rendering code to reflect changes to a template should be backward-compatible. You can trigger the content upgrade process manually by accessing all affected content, but this approach is not recommended.

For this reason, you should avoid making changes to existing templates that are being used in production. You should limit updates to templates to the early stages of application development when you have little or no legacy content to support.

Retrieving the current templates from Experience Manager

If you need to view or edit an existing template on a local machine, you can run the `get_templates` script to download templates from Experience Manager to the local `<app dir>\config\cartridge_templates` directory.

To get templates from Experience Manager:

1. In a command prompt, navigate to the `control` directory of your deployed application.

This is located under your application directory. For example: `C:\Endeca\apps\<app dir>\control`.

2. Run the `get_templates` script.

Part 3

Developing an Assembler Application

- *Deploying the Assembler*
- *Invoking the Assembler*
- *Implementing Multichannel Applications*
- *Tuning an Assembler application*

Chapter 6

Deploying the Assembler

The Assembler can run in process as part of a Java application, or it can be deployed as a standalone servlet. This section covers both deployment options, as well as environment requirements and Assembler dependencies.

Assembler environment requirements

Review the requirements in this section before you deploy an Assembler.

Port usage

Before you begin your deployment, you might need to request an open port. You must assign a port for the Assembler client port. If this port is set to `-1`, the system uses an ephemeral port. An ephemeral port is allocated automatically for a short time and is only used for the duration of a communication session. When the session ends, it is available for another request.

For a complete list of ports used by Oracle Endeca Commerce, see the *Oracle Endeca Commerce Administrator's Guide*.

Threads

The Assembler spawns threads to monitor and query various components for updates. This has an impact on how you manage and prioritize threads.

About authoring and production environments

When designing your application and deploying the Assembler, consider the deployment requirements that come with maintaining an authoring environment and a live environment.

You should monitor the performance of your application and make adjustments as necessary to handle the expected load in a production situation.



Note: The Assembler has no dependencies on Workbench in a live environment; rule information is published to the MDEX Engine, and content items are exported from Workbench and maintained in an external location accessible from the live server(s). All live Assembler instances for a given application access the same exported content.

For additional information, including the necessary steps for exporting content from Workbench, see the *Oracle Endeca Commerce Administrator's Guide*.

Assembler dependencies

Assembler dependencies are packaged in the %ENDECA_TOOLS_ROOT%\assembler\lib directory. You must include them in any custom Assembler application that you build.

The Assembler relies on the following libraries:

- AOP Alliance 1.0
- Apache Commons Logging 1.1.1
- Endeca Navigation API 6.5.0
- Endeca Logging API 6.5.0
- Spring AOP 3.0.1
- Spring ASM 3.0.1
- Spring Beans 3.0.1
- Spring Context 3.0.1
- Spring Core 3.0.1
- Spring Expression 3.0.1
- Spring Web 3.0.1

About deploying the Assembler

The Assembler can run in process as part of a Java application that powers a Web site, or it can be deployed as a standalone servlet. Non-Java applications must use the Assembler servlet.

The Tools and Frameworks package includes an example of each deployment mode in `/reference/discover-electronics` (for the Assembler running in process) and `/reference/discover-service` (for the standalone Assembler servlet). The standalone servlet, or Assembler Service, provides a RESTful interface for Assembler queries that returns results in either JSON or XML.

Both deployment modes depend on a Spring context file for application-specific configuration. The deployment descriptor files for the reference implementations specify a context file located in `/WEB-INF/assembler-context.xml`, as follows:

```
<listener>
  <listener-class>
    org.springframework.web.context.ContextLoaderListener
  </listener-class>
</listener>
<listener>
  <listener-class>
    org.springframework.web.context.request.RequestContextListener
  </listener-class>
</listener>
<context-param>
  <param-name>contextConfigLocation</param-name>
  <param-value>/WEB-INF/assembler-context.xml</param-value>
</context-param>
```

Assembler configuration

The Assembler implementation included with Tools and Frameworks is configured through Spring. The configuration in the Spring context file applies to both the in-process Assembler, and the Assembler Service.

This guide assumes an application based around the included Assembler implementations. You can provide your own implementation if you wish to use an alternate means of configuring the Assembler.

In the reference implementations, application-specific Assembler configuration is specified in the Spring context file located in `WEB-INF\assembler-context.xml`.

Assembler factory

The `AssemblerFactory` is an interface for creating a new Assembler. In the reference implementation, it is implemented in the `SpringAssemblerFactory` class and defined as follows:

```
<bean id="assemblerFactory" class="com.endeca.infront.assembler.spring.SpringAssemblerFactory"
    scope="singleton">
    <constructor-arg>
        <bean class="com.endeca.infront.assemblerAssemblerSettings">
            <property name="previewEnabled" value="${preview.enabled}" />
            <property name="previewModuleUrl" value="http://${workbench.host}:${workbench.port}/preview" />
        </bean>
    </constructor-arg>
</bean>
```

For details on the `AssemblerFactory` interface and `SpringAssemblerFactory` implementation, see the [Assembler API Reference \(Javadoc\)](#).

Endeca Configuration Repository

The `SiteHandler` provides the credentials and URL for the Endeca Configuration Repository. It is defined as follows:

```
<spr:bean id="SiteHandler" class="com.endeca.dt.ifcr.SiteHandler">
    <spr:property name="appName" value="@@PROJECT_NAME@@" />
    <spr:property name="repoUrl" value="http://@@HOST@@:@@WORKBENCH_PORT@@/ifcr"/>

    <spr:property name="repoUserName" value="myRepoAdminUser" />
    <spr:property name="repoPassword" value="myRepoPassword" />
</spr:bean>
```

The bean is referenced in the `InitialSetup` script:

```
<script id="InitialSetup">
    <bean-shell-script>
        <![CDATA[
        if (ConfigManager.isWebStudioEnabled()) {
            log.info("Updating Workbench configuration... ");
            ConfigManager.updateWsConfig();
            log.info("Finished updating Workbench.");
        }
        SiteHandler.provisionSite();
    ]]>
</bean-shell-script>
</script>
```

About configuring cartridge handlers

A cartridge handler is an Assembler component that takes the configuration model for a specific cartridge and interacts with an external system to produce a response model. Cartridge handler configuration is a subset of Assembler configuration.

HTTP servlet request access

The `httpServletRequest` bean provides access to the `HttpServletRequest` object for the current request.

```
<bean id="httpServletRequest" scope="request"
      factory-bean="springUtility"
      factory-method="getHttpServletRequest" />
```

Cartridge handlers that need access to the servlet request can specify a reference to this bean as follows:

```
<property name="httpServletRequest" ref="httpServletRequest" />
```

Search and navigation request configuration

The Assembler provides several utilities for parsing incoming requests and forming MDEX Engine queries.

MDEX resource configuration

The MDEX resource provides access to the MDEX Engine and manages information about the MDEX Engine and its schema configuration. Cartridge handlers can request data from their MDEX resource during the course of processing a cartridge.

The MDEX resource has the following properties:

MDEX resource property	Description
host	The hostname or IP address of your MDEX Engine server.
port	The port on which the MDEX Engine server listens.
recordSpecName	The name of the property that serves as the record spec in your data set.

Navigation state builder configuration

The navigation state builder is responsible for parsing the request URL into a `NavigationState` object and for generating Endeca URLs based on a `NavigationState`.

Navigation state builder property	Description
urlFormatter	Specifies the <code>UrlFormatter</code> object to use for parsing the request URL into a <code>NavigationState</code> object and for generating Endeca URLs based on a <code>NavigationState</code> .  Note: In the Discover Electronics application, this bean is configured in <code>endeca-url-config.xml</code> .
mdexRequestBuilder	The <code>MdexRequestBuilder</code> implementation to use for forming MDEX Engine requests. For more information, see "About

Navigation state builder property	Description
	configuring cartridge handlers that make search and navigation queries."
contentPathProvider	Specifies the ContentPathProvider implementation that provides the URL path info for a navigation query or a record query. A reference implementation, BasicContentPathProvider, is included as part of Discover Electronics. As configured in the example below, it returns /browse for navigation queries and /detail for record detail queries.
defaultSearchKey	The name of a property, dimension, or search interface against which searches (using the Search Box cartridge) are performed.
defaultMatchMode	The match mode to use for text searches. Valid values for this property follow the syntax of URL parameters for search mode, without the mode+match prefix.
defaultFilterState	A default filter state to apply to all queries. See below for more details about default filter state configuration; all properties of the default filter state are optional.
removeAlways removeOnUpdateFilterState removeOnClearFilterState	These properties configure which URL parameters from the request URL are preserved when generating action strings and which ones are removed, depending on the type of transition the action URL represents.
recordDetailsDimensionNames	A list of dimensions whose dimension values should be applied to the navigation state for a record query (based on the values that are tagged on that record). This navigation state can be used for triggering configuration for the associated record detail page or for a spotlight cartridge that has the "restrict to refinement state" option enabled.

Filter state property	Description
rollupKey	A rollup key (used for aggregated records) to apply to all queries made with the default filter state.
autoPhraseEnabled	Specifies whether to apply automatic phrasing to text search queries. By default, automatic phrasing is enabled. For more information about automatic phrasing configuration, see "About implementing automatic phrasing" in this guide.
securityFilter	A default record filter to apply to MDEX Engine queries. For information about the record filter syntax, refer to the <i>MDEX Engine Advanced Development Guide</i> .
languageID	The language ID (as a valid RFC-3066 or ISO-639 code) to specify for MDEX Engine queries. For information about working with internationalized data, refer to the <i>MDEX Engine Advanced Development Guide</i> .

About configuring cartridge handlers that make search and navigation queries

Cartridge handlers that need to make MDEX Engine queries can reference the navigation state, record state, and MDEX request builder beans configured in the cartridge support section of the Spring context file.

The navigation state and record state represent the query parameters for each type of MDEX Engine query. The MDEX request builder consolidates requests from all the cartridge handlers in a single Assembler processing cycle into as few MDEX queries as possible. These beans are defined in terms of previously configured beans; their configuration should not need to vary between applications..

The `NavigationCartridgeHandler` references the `navigationState` and `mdexRequestBuilder` beans for making navigation queries. The `RecordDetailsHandler` references the `recordState` for record detail queries. Cartridge handlers (including many of the core cartridges) that need access to the navigation state, record state, or the MDEX request builder typically extend one of these handlers. Note that `RecordDetailsHandler` itself extends `NavigationCartridgeHandler` as shown below, thereby inheriting the references to the navigation state and MDEX request builder specified in the `NavigationCartridgeHandler` bean.

```
<bean id="NavigationCartridgeHandler" abstract="true">
    <property name="navigationState" ref="navigationState" />
    <property name="mdexRequestBuilder" ref="mdexRequestBuilder" />
</bean>

<bean id="CartridgeHandler_RecordDetails"
    class="com.endeca.infront.cartridge.RecordDetailsHandler"
    parent="NavigationCartridgeHandler" scope="prototype" >
    <property name="recordState" ref="recordState" />
</bean>
```

About configuring cartridges to retrieve dynamic content

Cartridge handlers that retrieve dynamic content based on trigger criteria can reference the content manager bean configured in the cartridge support section of the Spring context file.

The content manager depends on the content trigger state builder and its associated content trigger state, which perform similar functions to the navigation state builder and navigation state, only for the trigger query that retrieves dynamic content configuration, rather than the main navigation query.

Application-specific configuration for these beans relates to preview and auditing functionality. For more information about configuring preview, see "Setting up the Preview Application for Workbench."

The `ContentSlotHandler` references the content manager to make dynamic content queries. Other handlers that need to retrieve content items from a folder in Experience Manager should extend from this handler.

```
<bean id="CartridgeHandler_ContentSlot"
    class="com.endeca.infront.cartridge.ContentSlotHandler"
    scope="prototype" >
    <property name="contentManager" ref="contentManager" />
</bean>
```

About configuring the Assembler servlet

The Spring Assembler servlet extends the `AbstractAssemblerServlet` class, which requires a method for retrieving an `AssemblerFactory`, and another for retrieving a `ResponseWriter` that processes Assembler output.

The Assembler servlet references the same Spring configuration as the rest of the Assembler, with an additional dependency on response writer configuration.

Response writers

The Assembler servlet uses JSON or XML response writers to serialize the results of a query. The Assembler includes default implementations of a `JSONResponseWriter` and an `XMLResponseWriter`. You can provide your own implementation if you need to output the Assembler response to a different format (such as a different XML representation).

```
<bean id="jsonResponseWriter"
  class="com.endeca.infront.assembler.servlet.JsonResponseWriter"
  scope="singleton"/>

<bean id="xmlResponseWriter"
  class="com.endeca.infront.assembler.servlet.XmlResponseWriter"
  scope="singleton"/>
```

Reference implementations

The reference content includes two Web applications that run the Spring Assembler servlet with the appropriate configuration for Discover Electronics in either an authoring or a live environment:

- The implementation for an authoring environment is located at `reference\discover-service-authoring`.
- The implementation for a live environment is located at `reference\discover-service`.

Chapter 7

Invoking the Assembler

This section describes how to invoke the Assembler in process or as a service.

Invoking the Assembler in Java

You invoke the Assembler by passing in a content item object for assembly.

If a cartridge handler exists for the input content item, the Assembler invokes that handler to process it. If not, the content item is passed through as output. Upon invoking the cartridge handler, the Assembler may in turn invoke additional cartridge handlers to process child content items. The end result of the processing cycle is an output content item representing the Assembler response.



Note: If you have purchased Oracle Endeca Guided Search, you typically query the Assembler using one of the packaged services, either with a ContentInclude item or via the Assembler service.

The examples in this topic are specific to a Spring implementation of the Assembler.

To invoke the Assembler in Java:

1. Create an `AssemblerFactory` object.

Note that the example implementation below first fetches configuration via the `WebApplicationContext` in the Spring framework:

```
// Get the Spring Web Application Context
ServletContext servletCtx = this.getServletContext();
WebApplicationContext webappCtx =
    WebApplicationContextUtils.getRequiredWebApplicationContext(servletCtx);

// Get an assembler factory and create an assembler
AssemblerFactory assemblerFactory =
    (AssemblerFactory)webappCtx.getBean("assemblerFactory", AssemblerFactory.class);
```

2. Use the `AssemblerFactory` to create an `Assembler`:

```
Assembler assembler = assemblerFactory.createAssembler();
```

3. Optionally, add event listeners to the newly-created `Assembler`:

```
assembler.addAssemblerEventListener(new MyLogger());
```

4. Pass in the content item object to assemble:

```
ContentItem responseContentItem = assembler.assemble(myContentItem);
```



Note: You can instantiate any content item programmatically and pass it to the Assembler, but typically an assembly cycle begins with a `ContentInclude` or `ContentSlotConfig` item. Both of these methods retrieve content items created in Workbench, the former by URI, and the latter by triggering content from a folder populated either in Experience Manager or Rule Manager.

After invoking the Assembler, you may wish to serialize the response:

```
// Serialize the results to JSON
response.setCharacterEncoding("UTF-8");
JsonSerializer serializer = new JsonSerializer(response.getWriter());
serializer.write(responseContentItem);
```

The Assembler implementation included with Tools and Frameworks comes with two classes for this purpose, `JsonSerializer` and `XmlSerializer`. See the [Assembler API Reference \(Javadoc\)](#) for details.

Related Links

[About retrieving Assembler results using the packaged services](#) on page 75

If you have purchased Oracle Endeca Guided Search (without Oracle Endeca Experience Manager), you must retrieve Assembler results via the packaged services. These services are also available for Experience Manager implementations.

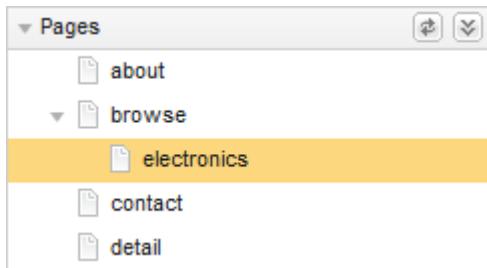
Invoking the Assembler with a ContentInclude item

A `ContentInclude` object specifies the URI from which to retrieve a content item.

In an authoring instance the content configuration is stored in the Endeca Configuration Repository. In a live instance, the Assembler retrieves content configuration from the live content source, specified in the configuration for the `ContentIncludeHandler`.

- In Oracle Endeca Experience Manager implementations, the URI typically begins with `/pages` (a hard-coded path to distinguish the sitemap from content folders in Experience Manager), with the path info from the request URL appended.
- In Oracle Endeca Guided Search implementations, the URI must begin with `/services` and specify one of the packaged Assembler services.

The `ContentIncludeHandler` retrieves the content that matches the deepest path in the URI. For example, if the request URL is `http://www.example.com/browse/electronics/Cameras`, the URI passed to the Assembler is `/pages/browse/electronics/Cameras`. Suppose that the sitemap for this site looks like the following:



The cartridge handler first tries to retrieve the content at the exact URI. There is no content at that location, so it attempts to find the deepest matching path, which in this case is the content configuration at

/browse/electronics. The Assembler then processes the content item at that location and returns the response for rendering.

Example

The following example of a content include query retrieves page content for the Discover Electronics application with Experience Manager:

```
// Construct a content include to query the content source
// for content, given the path info of the request
ContentItem contentItem =
    new ContentInclude("/pages" + request.getPathInfo());
```

Invoking the Assembler with a ContentSlotConfig item

A ContentSlotConfig object specifies one or more paths to a content folder in Experience Manager. The Assembler dynamically retrieves content items from the folder based on the trigger criteria and priorities set by the content administrator. It returns a number of content items equal to the evaluation limit configured for the specified content folder.

The Endeca Configuration Repository stores all Workbench content configuration for a given application within a content node. For example, the path to a **Web > Spotlights** content folder in the Discover Electronics reference application would be content/Web/Spotlights.

Example

The following example creates a ContentSlotConfig object that is intended to populate the sidebar of an application page with three content items pulled from a **Web > Spotlights** content folder in Experience Manager:

```
ContentItem dynamicContentItem = new ContentSlotConfig();
dynamicContentItem.setContentPaths("/content/Web/Spotlights");
dynamicContentItem.setTemplateTypes("SecondaryContent");
dynamicContentItem.setRuleLimit(3);
```

It specifies a template type restriction to retrieve only "SecondaryContent" for the sidebar, but does not restrict results by template ID. This allows the query to pull in content items created from multiple cartridge templates, as long as those templates have the correct type; for example, it might return a Breadcrumbs cartridge, a Record Spotlight cartridge, and a Rich Text cartridge.

The call to the Assembler is the same as for any other content item:

```
ContentItem responseContentItem = assembler.assemble(dynamicContentItem);
```

Querying the Assembler Service

The Assembler Service provides a RESTful interface for making Assembler queries and retrieving results in either JSON or XML.

You query the Assembler Service by making a GET request to a URL that specifies the location of a content item that you wish to assemble. The URL should be of the following form:

```
http://[hostname:port]/[servlet-path]/[content-URI]?[query-params]
```

In the reference deployment of the Assembler Service, the servlet path determines the format of the Assembler response. The deployment descriptor file (`web.xml`) in the reference deployment defines two servlets:

Servlet path	Servlet description
/json	Returns the Assembler response as JSON.
/xml	Returns the Assembler response as XML.

The difference between the servlets is in the `ResponseWriter` implementation that they use. It is also possible to write an Assembler response writer that forwards the results to another servlet rather than serializing them.

The `content-URI` is the path to the content item to be assembled.

- In Experience Manager implementations, the URI typically begins with `/pages` (a hard-coded path to distinguish the sitemap from content folders in Experience Manager), with the path info from the request URL appended.
- In Oracle Endeca Guided Search-only implementations, the URI must begin with `/services` and specify one of the Assembler packaged services.

The Assembler servlet request URL `http://www.example.com/json/pages/browse` is equivalent to passing a `ContentInclude` item to the Assembler `assemble()` method with the URI `/pages/browse` and retrieving the results in JSON format.

Query parameters in an Assembler servlet request URL are processed the same way as in the embedded Java Assembler. For example, the URL `http://www.example.com/json/browse?N=101022` with the reference Assembler servlet deployment returns the same results as `http://www.example.com/discover/browse?N=101022` in the reference Java application.

Making dynamic content queries to the Assembler servlet

Unlike the Assembler in embedded mode, which allows assembly of any configuration content item, all queries to the Assembler servlet are treated as content include queries. To request content dynamically from a content folder based on a set of trigger criteria, you can create a content slot at a location in the sitemap that you can then specify in your Assembler request URL. In the reference implementation, the `browse` page is one example of a content item that is addressable by URI that then references content items within a specified folder path.

Related Links

[Invoking the Assembler with a ContentInclude item](#) on page 72

A `ContentInclude` object specifies the URI from which to retrieve a content item.

The Assembler servlet response format

The Assembler provides response writer implementations that serialize the Assembler response to JSON or XML.

The Assembler response takes the form of a content item (that is, a map of properties). The properties are key-value pairs where the key is a string and the value may be one of the following types:

- String
- Boolean
- Integer
- List (of any property type)
- Item (a nested map of properties)

This structure makes it straightforward to serialize the response to JSON.

The XML output of the Assembler (using the default `XmlResponseWriter`) is not SOAP-compliant. The default XML format has the following characteristics:

- The root element of the response is `<Item>`.
- `<Item>` may have either a `type` attribute whose value is equivalent to the template `id` that defined the content item, or a `class` attribute in the case of a strongly typed response model for a content item.
- The child elements of `<Item>` are `<Property>` elements.
- Each `<Property>` element has a `name` attribute whose value is the property key, and contains either a `<String>`, `<Boolean>`, `<Integer>`, `<List>`, or `<Item>` element whose contents represent the property value.

For convenience, the Discover Electronics reference application provides links to the JSON and XML representations of the Assembler response, which are identical to the output of the Assembler servlet. This link can be useful for debugging purposes, but it is not recommended as a primary means of querying the Assembler for JSON or XML-formatted results.

Copyright © 2003, 2012, Oracle and/or its affiliates. All rights reserved.
 Product Data © 2002-2012 [ICECAT BIZ](#) | [Disclaimer](#) | [json](#) | [xml](#)

About building an Assembler query string

Whether you invoke the Assembler programmatically in Java or as a service, the content URI that you pass into the Assembler includes any MDEX Engine query parameters.

For more information on MDEX Engine query parameter syntax, refer to the *Assembler API Reference (Javadoc)* content for the `UrlNavigationStateBuilder` class.

About retrieving Assembler results using the packaged services

If you have purchased Oracle Endeca Guided Search (without Oracle Endeca Experience Manager), you must retrieve Assembler results via the packaged services. These services are also available for Experience Manager implementations.

The packaged services include the following:

Service URI	Description
<code>/services/dimensionsearch</code>	Returns dimension search results based on a text search.
<code>/services/recorddetails</code>	Returns record detail information for a given record.
<code>/services/guidedsearch</code>	Returns search and navigation results including core Endeca features such as Guided Navigation, along with dynamic content returned from content folders.

You query the services by passing a `ContentInclude` item to the Assembler with the relevant service URI or making an Assembler servlet request specifying the service URI. The services are configured to return results for a specific cartridge or set of cartridges.

The cartridges that are returned by the services cannot be configured on a per-instance basis in Rule Manager or Experience Manager, but application-wide default configuration for the cartridges can be specified based on your configuration framework (such as Spring). The exception is the dynamic content that can be configured in content folders and that is returned by the Guided Search Service, which can be configured in Rule Manager or Experience Manager.

The services are populated in the Endeca Configuration Repository (for use by the authoring instance) when you run `initialize_services` after deploying an application. They are promoted to the live content source when you promote the site configuration for the live instance.

The Dimension Search Service

The Dimension Search Service returns dimension search results for a keyword search.

The service returns a single `DimensionSearchResults` object in a `dimensionSearchResults` property, representing the list of dimensions that match the search term.

The default behavior of this cartridge is configured as part of the `CartridgeHandler_DimensionSearchResults` bean in the Spring context file for the Assembler. For information about the configuration options for the Dimension Search Results cartridge, refer to the *Assembler API Reference (Javadoc)* for the `DimensionSearchResultsConfig` class.

This service exists for cases where you want to retrieve dimension search results only (such as in the case of an auto-suggest dimension search feature). Dimension search results are also returned as part of the response from the Guided Search Service.

The following is an example of the results of a Dimension Search Service query for the URI `http://localhost:8006/assembler-authoring/json/services/dimensionsearch?Ntt=fla*&Dy=1`, serialized to JSON:

```
{
  "@type": "DimensionSearchService",
  "name": "Dimension Search Service",
  "dimensionSearchResults": {
    "@type": "DimensionSearchResults",
    "totalNumResults": 13,
    "dimensionSearchGroups": [
      {
        "@class": "com.endeca.infront.cartridge.model.DimensionSearchGroup",
        "dimensionSearchValues": [ ... ],
        "dimensionName": "camera.flash"
      },
      {
        "@class": "com.endeca.infront.cartridge.model.DimensionSearchGroup",
        "dimensionSearchValues": [ ... ],
        "dimensionName": "product.features"
      },
      {
        "@class": "com.endeca.infront.cartridge.model.DimensionSearchGroup",
        "dimensionSearchValues": [ ... ],
        "dimensionName": "product.category"
      }
    ],
    "endeca:contentPath": "/services/dimensionsearch",
    "previewModuleUrl": "http://localhost:8006/preview"
  }
}
```

The Record Details Service

The Record Details Service returns record detail information for a given record.

The service returns a single `RecordDetails` object in a `recordDetails` property, representing the details for a single record or an aggregate record.

The default behavior of this cartridge is configured as part of the `CartridgeHandler_RecordDetails` bean in the Spring context file for the Assembler. For information about the configuration options for the Record Details cartridge, refer to the *Assembler API Reference (Javadoc)* for the `RecordDetailsConfig` class.

The following is an example of the results of a record details service query for the URI `http://localhost:8006/assembler-authoring/json/services/recorddetails/Canon/Prima-Super-130U-Date/_/A-266556`, serialized to JSON:

```
{
  "@type": "RecordDetailsService",
  "name": "Record Details Service",
  "recordDetails": {
    "@type": "ProductDetail",
    "record": {
      "@class": "com.endeca.infront.cartridge.model.Record",
      "numRecords": 1,
      "attributes": { ... },
      "records": [
        {
          "@class": "com.endeca.infront.cartridge.model.Record",
          "numRecords": 0,
          "attributes": { ... }
        }
      ]
    },
    "endeca:siteRootPath": "/services",
    "endeca:contentPath": "/recorddetails",
    "previewModuleUrl": "http://localhost:8006/preview",
    "endeca:assemblerRequestInformation": { ... }
  }
}
```

The Guided Search Service

The Guided Search Service returns search and navigation results including core Endeca features such as Guided Navigation, along with dynamic content returned for content slots.

The properties returned as part of the response model, as well as their associated configuration, are listed below:

Property name	Response model	Configuration bean	Configuration model
navigation	GuidedNavigation	CartridgeHandler_GuidedNavigation	GuidedNavigationConfig
breadcrumbs	Breadcrumbs	CartridgeHandler_Breadcrumbs	BreadcrumbsConfig
resultsList	ResultsList	CartridgeHandler_ResultsList	ResultsListConfig

Property name	Response model	Configuration bean	Configuration model
searchAdjustments	SearchAdjustments	CartridgeHandler_SearchAdjustments	SearchAdjustmentsConfig
dimensionSearchResults	DimensionSearchResults	CartridgeHandler_DimensionSearchResults	DimensionSearchResultsConfig
zones	Depends on contents of referenced content folders.	CartridgeHandler_ContentSlotList	ContentSlotConfig

The following is an example of the results of a guided search service query for the URI `http://localhost:8006/assembler-authoring/json/services/guidedsearch?Ntt=pink+camera`, serialized to JSON:

```
{
  "@type": "GuidedSearchService",
  "name": "Guided Search Service",
  "navigation": {
    "@type": "GuidedNavigation"
  },
  "breadcrumbs": {
    "@type": "Breadcrumbs",
    "removeAllAction": "/services/guidedsearch",
    "refinementCrumbs": [ ],
    "searchCrumbs": [ ... ],
    "rangeFilterCrumbs": [ ]
  },
  "resultsList": {
    "@type": "ResultsList",
    "totalNumRecs": 213,
    "sortOptions": [ ... ],
    "firstRecNum": 1,
    "lastRecNum": 10,
    "pagingActionTemplate": "/services/guidedsearch?No=%7Boffset%7D&Nrpp=%7BrecordsPerPage%7D&Ntt=pink+camera",
    "recsPerPage": 10,
    "records": [ ... ]
  },
  "searchAdjustments": {
    "@type": "SearchAdjustments",
    "originalTerms": [
      "pink camera"
    ]
  },
  "zones": {
    "@type": "ContentSlotList"
  },
  "endeca:contentPath": "/services/guidedsearch",
  "previewModuleUrl": "http://localhost:8006/preview"
}
```



Note:

For details about the contents of the `zones` property, see "About dynamic content and the Guided Search Service."

Configuring dynamic content for the Guided Search Service

For each dynamic slot that you wish to populate as part of the response from the Guided Search Service, you must configure a `ContentSlotConfig` object. Each of these objects is set as a property of the default input content item for the `ContentSlotHandler`.

Specify the following properties for each instance of `ContentSlotConfig`:

Property name	Value
<code>contentPaths</code>	A List of String typed paths to the content folders from which you want to return results.
<code>templateTypes</code>	(Optional) A List of String typed template type restrictions for the dynamic slot.
<code>templateIds</code>	(Optional) A List of String typed template ID restrictions for the dynamic slot.
<code>ruleLimit</code>	The maximum number of content items to return from this collection. The Assembler returns up to this number of items that match the trigger criteria, based on priority.



Note: The content within a folder depends on the template type or ID restrictions configured for that folder in Experience Manager. While it is possible to configure your default `ContentSlotConfig` objects with any restrictions you wish, you should ensure that the type and ID restrictions you enter match those in Experience Manager. For example, it is possible to create a `ContentSlotConfig` object that is restricted by template type "MainContent," while the `contentPaths` property points to folders in Experience Manager that are restricted to "SecondaryContent" (and thus will never contain any "MainContent" content items).

Example

In the example below, the input content item to the `ContentSlotHandler` is a `ContentSlotListConfig` object. It is instantiated as "contentSlotList," and contains a `ContentSlotConfig` object for each dynamic slot in the application. The `contentSlotList` is passed in to the `ConfigInitializer` that instantiates it as the input content item for the cartridge handler.

The `contentSlotList` for the Discover Electronics reference application is configured in the `Cartridge-Handler_ContentSlotList` bean in the Spring context file, `assembler-context.xml`. For each content folder that is enabled for the Guided Search Service, a `ContentSlotConfig` bean appears in the `contentSlotList` as in the example below:

```

<bean id="CartridgeHandler_ContentSlotList" class="com.endeca.infront.cartridge.ContentSlotListHandler"
  scope="prototype">
  <property name="contentItemInitializer">
    <bean class="com.endeca.infront.cartridge.ConfigInitializer" scope="request">
      <property name="defaults">
        <bean class="com.endeca.infront.cartridge.ContentSlotListConfig"
          scope="singleton">
          <property name="contentSlotList">
            <list>
              <bean class="com.endeca.infront.cartridge.ContentSlotConfig"
                scope="singleton">
                <property name="contentPaths">
                  <list>
                    <value>/content/Right Column Spotlights</value>

```

For detailed information on the `ContentSlotConfig` configuration model and its included properties, see the [Assembler API Reference \(Javadoc\)](#).

Handling the Guided Search Service response

The Assembler returns the matching content items for each configured `ContentSlotConfig`, so the response consists of a list of lists of content items:

- ContentSlotList response content item
 - 1st content item, returned from a ContentSlotConfig with a ruleLimit of 3
 - Highest priority matching content item
 - Second highest priority matching content item
 - Third highest priority matching content item
 - 2nd content item, returned from a ContentSlotConfig with a ruleLimit of 2
 - Highest priority matching content item
 - Second highest priority matching content item

Note that the Guided Search Service response is not view-friendly. You must parse it in your application logic to determine if any of the content items returned in the tree correspond to page sections you wish to populate for the end user's current location in the application.

Below is a sample JSON response from the Guided Search Service in the Discover Electronics reference application when the user selects the "Cameras" category:

```
"zones": {
    "@type": "ContentSlotList",
    "contentSlotList": [
        {
            "@type": "ContentSlot",
            "templateTypes": [
                "RecordSpotlight"
            ]
        }
    ]
}
```

```

"contents": [
  {
    "@type": "RecordSpotlight",
    "title": "Most Popular Cameras",
    "name": "Spotlight Records",
    "records": [
      { ... },
      { ... },
      { ... },
      { ... }
    ]
  },
  {
    "@type": "RecordSpotlight",
    "title": "Top Rated Products",
    "name": "Spotlight Records",
    "records": [
      { ... },
      { ... },
      { ... }
    ]
  }
],
"contentPaths": [
  "/content/Right Column Spotlights"
],
"ruleLimit": 3,
"templateIds": [ ]
}
]
},
]
}

```

It populates two sidebar Record Spotlight cartridges, the first with four records, and the second with three.

About retrieving content item properties from packaged services

This topic outlines the logic required for retrieving properties from the Assembler response model for the included Guided Search Service.

The example below includes processing logic within a renderer JSP file. Oracle recommends including most of your logic for handling Assembler responses in your cartridge handlers, as this minimizes the amount of duplicate code required across multiple renderers.



Note: API documentation for the Assembler packages is available in the `assembler\apidoc\assembler` directory of your Tools and Frameworks installation.

Retrieving information from the Assembler response

Recall the serialized Assembler response for the URI `http://localhost:8006/assembler-authoring/json/services/guidedsearch?Ntt=pink+camera`:

```
{
  "@type": "GuidedSearchService",
  "name": "Guided Search Service",
  "navigation": {
    "@type": "GuidedNavigation"
  },
  "breadcrumbs": {
    "@type": "Breadcrumbs",
    "path": [
      {
        "label": "Home"
      },
      {
        "label": "Pink"
      },
      {
        "label": "Pink Cameras"
      }
    ]
  }
}
```

```

        "removeAllAction": "/services/guidedsearch",
        "refinementCrumbs": [ ],
        "searchCrumbs": [ ... ],
        "rangeFilterCrumbs": [ ]
    },
    "resultsList": {
        "@type": "ResultsList",
        "totalNumRecs": 213,
        "sortOptions": [ ... ],
        "firstRecNum": 1,
        "lastRecNum": 10,
        "pagingActionTemplate": "/services/guidedsearch?No=%7Boff-
set%7D&Nrpp=%7BrecordsPerPage%7D&Ntt=pink+camera",
        "recsPerPage": 10,
        "records": [ ... ]
    },
    "searchAdjustments": {
        "@type": "SearchAdjustments",
        "originalTerms": [
            "pink camera"
        ]
    },
    "zones": {
        "@type": "ContentSlotList"
    },
    "endeca:contentPath": "/services/guidedsearch",
    "previewModuleUrl": "http://localhost:8006/preview"
}

```

To create a sample JSP file that invokes the Assembler:

1. Import the required packages and create the necessary objects for supporting the Assembler:

```

<%@page language="java" contentType="text/html; charset=UTF-8" %>
<%@page import="com.endeca.infront.assemblerAssembler" %>
<%@page import="com.endeca.infront.assemblerAssemblerFactory" %>
<%-- additional imports removed from this example --%>
<%@page import="org.springframework.web.context.WebApplicationContext" %>
<%@taglib prefix="discover" tagdir="/WEB-INF/tags/discover" %>
<%
    // Create the Web Application Context object
    WebApplicationContext webappCtx = WebApplicationContextUtils.getRequiredWe-
    bApplicationContext(application);

    // Get the AssemblerFactory from the Spring context file
    AssemblerFactory assemblerFactory = (AssemblerFactory)webappCtx.getBean("as-
    semblerFactory");

```

2. Recall that the packaged services invoke the Assembler with a `ContentInclude` item. Create the `Assembler` object and the `ContentInclude` item, and pass it into the Assembler as the configuration model:

```

    // Create an Assembler object
    Assembler assembler = assemblerFactory.createAssembler();

    // Construct a content include item for the Guided Search service
    ContentItem contentItem = new ContentInclude("/services/guidedsearch");

    // Assemble the content
    ContentItem responseContentItem = assembler.assemble(contentItem);

```

The Assembler returns a `com.endeca.infront.assembler.ContentItem` interface as the response model; this extends the Java Map interface. Assign this response to a `responseContentItem` object.

3. get the resultsList object from the responseContentItem:

```
ContentItem resultsListItem = (ContentItem) responseContentItem.get("resultsList");
```

This retrieves the top-level resultsList object, which is itself an extension of BasicContentItem, from the Assembler response.

4. You can now retrieve and use the individual values stored on the resultsList object, for example, the total number of records:

```
String totalNumRecs = resultsListItem.get("totalNumRecs");
```

This assigns a value of "213" to the totalNumRecs variable (based on the sample response presented at the start of this topic). Similarly, you could retrieve any other value from the key/value pairs that comprise resultsList, including other objects that extend the Map interface and are themselves made up of key/value pairs.

Refer to the Assembler API documentation for additional information on available Assembler interfaces, implementations, and methods. Following the pattern described in Steps 3-4, you can continue to retrieve values from the Assembler response by calling the get method on the response model object to traverse the nested values.

About handling the Assembler response

As a best practice, your application should have modular renderers to handle the response model for each content item.

A typical page consists of a content item that contains several child content items representing the individual feature cartridges. The Discover Electronics application maps each response model to the proper renderer by convention, based on the @type. The model @type corresponds to the template identifier (the directory name) of the template that was used to configure it. (Recall that the template type determines where a cartridge can be placed in another content item, while the template ID uniquely identifies the cartridge and its associated content definition.) For each cartridge, the associated renderer is located in WEB-INF/views/<channel>/<TemplateID>/<TemplateID>.jsp. For example, the renderer for the Breadcrumbs cartridge is located in WEB-INF/views/desktop/Breadcrumbs/Breadcrumbs.jsp.

In the Discover Electronics application, this logic is implemented in include.tag. Your application should implement a similar mapping of response models to their corresponding rendering code.

Source code for the renderers in the Discover Electronics application is provided as an example of how to work with the model objects returned by the Assembler in Java. The sample rendering code is intentionally lightweight, enabling it to be more easily customized for your own site. For information about the response models for the core cartridges, refer to the *Assembler API Reference (Javadoc)*.

Some features in the Discover Electronics application are designed with certain assumptions about the data set, such as property and dimension names. Mirroring the Discover Electronics data schema for your own data can facilitate reuse of the reference cartridges, reducing the need to update rendering logic and Assembler configuration for your data set.

About rendering the Assembler response

Once you have retrieved the necessary information for your page, Oracle recommends subdividing your view logic to correspond to the hierarchy of content items returned by the Assembler.

The renderer for the Three Column Navigation Page content item in Discover Electronics provides an example of the page rendering process as implemented in the reference application. It is located in your Tools and Frameworks installation directory under

reference\discover-electronics-authoring\WEB-INF\views\desktop\ThreeColumnPage\ThreeColumnPage.jsp. You can use this JSP file as a point of reference for developing your own application pages. While the details are specific to the Discover Electronics implementation of the Assembler API, your general approach should be similar.

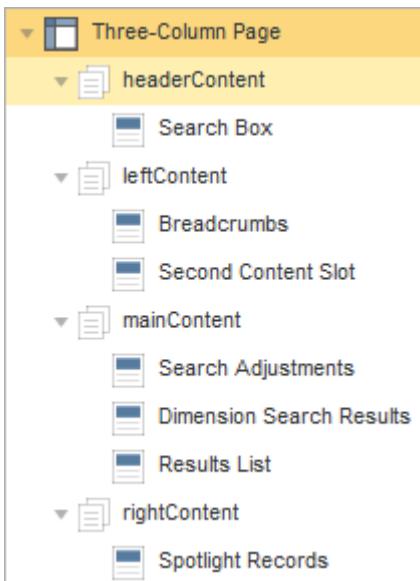
Recall that each of the `<div>` elements that make up the page uses a custom `<discover:include>` tag, defined in `WEB-INF\tags\discover\include.jsp`, to include the rendering code for the associated page component:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
    <!-- Additional elements removed from this sample -->
</head>
<body>
    <endeca:pageBody rootContentItem="${rootComponent}">
        <div class="PageContent">
            <%--include user panel --%>
            <%@ include file="/WEB-INF/views/userPanel.jsp" %>
            <%--include user page logo --%>
            <%@ include file="/WEB-INF/views/pageLogo.jsp" %>
            <div class="PageHeader">
                <c:forEach var="element" items="${component.headerContent}">
                    <discover:include component="${element}" />
                </c:forEach>
            </div>
            <div class="PageLeftColumn">
                <c:forEach var="element" items="${component.leftContent}">
                    <discover:include component="${element}" />
                </c:forEach>
            </div>
            <div class="PageCenterColumn">
                <c:forEach var="element" items="${component.mainContent}">
                    <discover:include component="${element}" />
                </c:forEach>
            </div>
            <div class="PageRightColumn">
                <c:forEach var="element" items="${component.rightContent}">
                    <discover:include component="${element}" />
                </c:forEach>
            </div>
            <div class="PageFooter">
                <%--include copyright --%>
                <%@include file="/WEB-INF/views/copyright.jsp" %>
            </div>
        </endeca:pageBody>
    </body>
</html>
```

For the example above, the JSP is composed as follows:

1. The static `<div class="UserPanel">` and `<div class="PageLogo">` elements are included from the specified JSP files.
2. The `<div class="PageHeader">` element retrieves the list of `headerContent` content items from the component.

- In an Oracle Endeca Experience Manager installation, this is the list of content items defined by the content administrator in Experience Manager:



- In an Oracle Endeca Guided Search installation, this is the list of content items specified application-wide under WEB-INF\services\browse.jsp:

```

<div class="PageContent">
    <%--include user panel --%>
    <%@ include file="/WEB-INF/views/userPanel.jsp" %>
    <%@ include file="/WEB-INF/views/pageLogo.jsp" %>

    <div class="PageHeader">
        <discover:include component="${searchBox}" />
    </div>
    <div class="PageLeftColumn">
        <discover:include component="${component.breadcrumbs}" />
        <discover:include component="${component.navigation}" />
    </div>
    ...
  
```

3. For each of the included content items, the JSP includes the output of the associated renderer.
4. The `<div class="PageLeftColumn">`, `<div class="PageCenterColumn">`, and `<div class="PageRightColumn">` elements are included in the same fashion.
5. The static `<div class="Copyright">` element is included from the specified JSP file.

For additional information on cartridge handlers and renderers, as well as a walkthrough of creating a sample cartridge, see [Extending and Developing Cartridges](#) on page 159.

Chapter 8

Implementing Multichannel Applications

This section covers how to design and implement multichannel applications built on the Assembler and managed using Workbench with Experience Manager.

Overview of multichannel applications with the Endeca Assembler

The Assembler provides an API for delivering content across an entire site, allowing content configuration to be shared between channels when appropriate, and also enabling a more targeted channel-specific experience where desired.

Enabling the full flexibility of the cross-channel experience involves the following:

- **Creating channel-specific templates.** Content administrators may wish to configure different features or cartridges for different channels. For example, pages designed for mobile devices typically have a simpler structure and present fewer options than pages designed for desktop Web.
- **Writing channel-specific rendering code.** Due to the limitations of mobile browsers and device bandwidth, renderers for mobile Web applications are typically more lightweight than those for desktop Web, while native applications for mobile devices require platform-specific client code.
- **Enabling device detection.** The UI tier of your Assembler application should include logic for handling device detection. Typically, this also includes redirecting a client to the appropriate service for their user agent.



Note: Endeca for Mobile is licensed separately from Oracle Endeca Guided Search and Oracle Endeca Experience Manager. It requires an additional software license.

About creating templates for mobile channels

Templates for mobile-specific content in a multichannel application can give content administrators the flexibility to manage channel-specific content in Experience Manager. However, when planning the set of templates for your application, it is a good idea to use more general templates where possible in order to share configuration across multiple channels.

The following general practices help enable this combination of flexibility and consistency:

- Create different top-level page templates for channels that have a different high-level structure. For example, the same range of cartridges may not be appropriate to a page designed to display on a mobile device as

opposed to a page designed to display on a desktop computer. Native applications for mobile devices may display content in simplified "pages" that differ from those intended for Web browsers.

- Use dynamic slots for configuration that should be shared across channels, since they enable reuse of content between pages. For example, if the same refinement configuration (such as overall dimension order, refinement ordering, and boost and bury options) should apply at a specific navigation state regardless of channel, it may make sense to configure it within a separate content folder and reference it from the appropriate pages for each channel.

To enable the greatest flexibility in Experience Manager while ensuring that content administrators create configurations that are appropriate to each channel, you can restrict the cartridges that can be placed on a page or in a content folder by content type. These content types may vary depending on the intended purpose of a page or dynamic slot. For example, you may have the following in your application:

- Page templates for desktop Web, which may define a section of type `SecondaryContent`. This section may be populated with Guided Navigation cartridges, Spotlight cartridges, or dynamic slots serving as a placeholder for either type.
- A content folder designed for Guided Navigation cartridges. This is similar to the Navigation section of the mobile page, but it should not allow a content administrator to create a dynamic slot within a dynamic slot, so it should have a third content type (such as `Navigation`) to enforce this restriction.

In most cases, the set of Dimension Navigation cartridges in an application should be identical. Variance between different output channels tends to manifest at the page design level, rather than at the level of the individual components of a page.

Chapter 9

Tuning an Assembler application

The Assembler and the MDEX Engine both include logging functionality that you can use to debug and fine tune your application. In addition, Workbench includes Preview functionality that your Content Administrator can use to evaluate the results of their changes.

Setting up the Preview Application for Workbench

If you are using Experience Manager, you can use a preview application to simulate sets of trigger conditions, such as time-based triggers, in order to determine which content items display when specific conditions are met. This section describes how to set up a custom Endeca application to function as the preview application in Workbench.

About the preview application

The preview application is the end-user application that displays in a new browser tab or window when selected. It allows content administrators to determine why each content item does or does not fire for specified navigation query and trigger combinations. This chapter describes how to set up your own custom application as the Workbench preview application.

You can launch the preview application for a specific page, or for an individual cartridge. A selected cartridge will display in the context of a page that includes it.

It is not necessary for the preview application to be an exact representation of your final front-end application, as long as it is using the correct data. The business logic that is built into Workbench is not tied to the physical representation of the front-end application. It is good practice, however, to make sure that your preview application represents your final application closely enough so that business users know if their changes are correct.

By default, Workbench is configured to use the Discover Electronics reference application as the preview application. This application is located under %ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring (%ENDECA_TOOLS_ROOT/reference/discover-electronics-authoring on UNIX).

Workbench communicates with the preview application via settings you specify in the **Preview Settings** tool. The **Preview URL** field lets you specify the preview application URL.



Note: The preview application must not use frames, because they are likely to collide with the frames of the Workbench preview toolbar.

About auditing content using a preview application

The Workbench preview feature includes auditing functionality that enables business users to view the underlying triggers for the set of displayed content items. Instrumenting a custom application for preview enables auditing.

If a content administrator wants to know which content items trigger for a specified navigation state, such as **Category > Cameras**, they can audit content by navigating to the desired state in the preview application. Clicking the **Audit** button for a content item displays the Audit Panel, which includes a **Status** column as shown below:

Status
<i>Not considered</i>
<i>Not considered</i>
<i>Not considered</i>
 Fired
Collection full

Content Items can exist in any of the following states:

Status	Meaning
Fired	The content item triggers and displays for the current navigation and trigger states.
Collection full	The content item triggers for the current navigation and trigger states, but does not display. This is due to one or more content items in the same collection having higher priority, causing the collection to reach its limit for the maximum number of allowable content items.
Not considered	The content item does not trigger for the current navigation and trigger states.
Navigation trigger not satisfied	The content item does not trigger. The navigation state conditions are met, but additional trigger conditions (such as a schedule trigger) are not.

About previewing specific devices

Workbench preview functionality includes support for specifying a device to preview against for a specified user agent.

The **Preview Settings** tool includes a Device Manager that allows you to map a user agent to a device with an associated preview skin. The Discover Electronics reference application includes the following configured devices:

- Mobile (landscape and portrait orientations)
- Tablet (landscape and portrait orientations)

Adding a device for preview

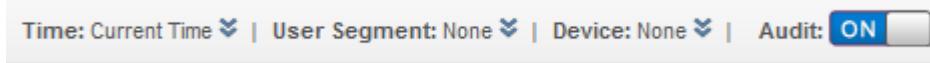
You can add additional device profiles and map them to the device skins included with Workbench.

To add a device for preview:

1. In Workbench, navigate to the **Application Settings > Preview Settings** tool.
2. Click **Add Device**.

The **Add Device** panel appears.

3. In the **Device Name** field, enter the device name to associate with your user agent.
This is the unique name that appears in the **Device** dropdown when previewing your application.
4. Select the device skin from the **Skin Name** dropdown.
5. In the **User Agent** field, enter the user agent string that you wish to add.
6. Click **Save**.
7. To confirm that your device is available for preview:
 - a) Navigate to the **Experience Manager** tool in Workbench.
 - b) Navigate to a content item of your choosing.
 - c) Mouseover the content item.
The Action dropdown button appears.
 - d) Select **Preview** from the Action dropdown.
The preview application launches in a new browser tab.
 - e) Click the arrow beside the Device: None entry in the Preview Toolbar:



The Preview Toolbar expands to show configuration options.

- f) Click the **Device** dropdown and confirm that your device is included in the dropdown list.
- g) Select the device name and click **Submit**.
- h) Confirm that the appropriate device skin displays.



Note: The preview application itself must have rendering logic for the selected user agent in order to display correctly.

Modifying a preview device

You can modify any of the fields for an existing preview device by clicking the device name in the Preview Settings tool.

To modify a preview device:

1. In Workbench, navigate to the **Application Settings > Preview Settings** tool.
2. In the **Device Name** column, select the device you wish to edit.
The **Edit Device** panel appears.
3. Modify the desired fields.
4. Click **Save**.

Removing a preview device

If a device is not required for previewing your application, you can remove it from the preview toolbar.

To remove a preview device:

1. In Workbench, navigate to the **Application Settings > Preview Settings** tool.
2. In the **Delete** column, click the **X** for the device you wish to remove.
A confirmation dialog appears.
3. Click **Delete**.

About instrumenting your application for preview

In order to enable auditing in your custom preview application, your rendering code must include logic for adding preview frames and an "Audit" button to content items that your content administrator mouses over.

Your custom preview application should include tags that specify paths to the required JavaScript and CSS resources, as well as tags for enabling audit functionality. These are provided in the `endeca` tag library.



Note: These requirements assume an application that uses JSP files for cartridge renderers (as in the case of the Discover Electronics reference application). If you are using a different technology stack to implement your Assembler application, you must write your own auditing functionality.

Adding Preview resources

All JSP files must include the `endeca` tag library, as shown below:

```
<%@ taglib prefix="endeca" uri="/endeca-infront-assembler/utilityTags"%>
```

Each `<head>` tag must contain a reference to the `pageHead` tag. This includes paths to the Preview JavaScript and CCS files:

```
<head>
  <endeca:pageHead rootContentItem="${rootComponent}" />
  <title><c:out value="${component.title}" /></title>
  <meta name="keywords" content="${component.metaKeywords}" />
  <meta name="description" content="${component.metaDescription}" />
</head>
```

Enabling auditing

Each `<body>` tag must contain a reference to the `pageBody` tag. This wraps content slot components in a `<div class="endeca-slot">` element and enables audit functionality:

```
</head>
<body>
  <endeca:pageBody rootContentItem="${rootComponent}" />
  <div class="PageContent" ...>
    <script type="text/javascript" ...>
  </endeca:pageBody>
</body>
</html>
```

Additionally, all JSP files that define a cartridge with content slots must include an `includeSlot` tag. This serves the same purpose as the reference to the `pageBody` tag for JSP files that do not contain a `<head>` tag:

```
<%@include file="/WEB-INF/views/include.jsp"%>

<endeca:includeSlot contentItem="${component}">
  <c:forEach var="element" items="${component.contents}">
    <discover:include component=${element}" />
  </c:forEach>
</endeca:includeSlot>
```

Device-specific auditing

In order to handle preview for different devices, you must implement conditional rendering logic for different user agent strings. The rendering code should include the tags described above.

You can retrieve the user agent String by getting a reference to the `UserState` object and calling `getUserAgent()` on it. The `UserState` class is documented in the Javadoc for the `com.endeca.infront.navigation` package.

For example, the Discover Electronics reference application includes the following logic in the `WEB-INF\services\assemble.jsp` page:

```
UserState userState = webappCtx.getBean(properties.getProperty("user.state.ref"),
    UserState.class);

String userAgent = userState.getUserAgent();

//If the userAgent is null, then no user-agent was specified and we need to get
//the user agent from the request header.
if(userAgent == null){
    userAgent = request.getHeader("user-agent");
}
```

Enabling your preview application

Once you have finished instrumenting your preview application, you can enable it for use in Workbench.

Ensure that your application has been correctly instrumented before enabling it for preview in Workbench.

All examples shown below are taken from the configuration files for the Discover Electronics authoring application, located in `%ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring` (on Windows) or `$ENDECA_TOOLS_ROOT/reference/discover-electronics-authoring` (on UNIX). The exact mechanisms used for configuring your Assembler and content sources will vary depending on your implementation details.

For a full description of the properties described below, see the Assembler API Javadoc for the `AssemblerFactory` and `ContentSource` interfaces and their corresponding implementations.

To enable your custom preview application:

1. In the constructor arguments for your `AssemblerSettings`, set the following:

Property	Value
<code>previewEnabled</code>	<code>true</code>
<code>previewModuleUrl</code>	<code>http://localhost:8006/preview</code>

In the Discover Electronics reference implementation, these are configured as properties in `WEB-INF\assembler.properties`:

```
workbench.host=localhost
workbench.port=8006

# ... Additional settings removed from this example ...

preview.enabled=true
```

These properties are then included in the Assembler context file, `WEB-INF\assembler-context.xml`:

```
<!--
#####
# ASSEMBLER FACTORY
#
# Required.
#
-->
```

```

<bean id="assemblerFactory"
class="com.endeca.infront.assembler.spring.SpringAssemblerFactory"
scope="singleton">
    <constructor-arg>
        <bean class="com.endeca.infront.assemblerAssemblerSettings">
            <property name="previewEnabled" value="${preview.enabled}" />
            <property name="previewModuleUrl" value="http://${workbench.host}:${workbench.port}/preview" />
        </bean>
    </constructor-arg>
    <constructor-arg>
        <list>
            <bean class="com.endeca.infront.logger.SLF4JAssemblerEventLogger" />
        </list>
    </constructor-arg>
</bean>

```

2. In the constructor arguments for your `ecrStoreFactory`, set `isAuthoring` to `true`.

In the Discover Electronics reference implementation, `isAuthoring` takes the value of the `isAuthoring` property:

```

<bean id="ecrStoreFactory" class="com.endeca.infront.content.source.EcrStoreFactory">
    <init-method>init</init-method>
    <destroy-method>destroy</destroy-method>
    <property name="isAuthoring" value="true"/>
    <property name="appName" value="${workbench.app.name}" />
    <property name="host" value="${workbench.host}" />
    <property name="clientPort" value="${workbench.publishing.clientPort}" />
    <property name="serverPort" value="${workbench.publishing.serverPort}" />
</bean>

```

3. Configure a link service for your application that returns a preview link as a JSONP response.

This service must construct a link to the page selected for preview; for example, if a content administrator previews the Brand - Canon Web Browse page in the reference application, the service returns `"/browse/_/N-25y6"`. Additionally, the response from the service is used to construct the links in the Audit Panel.

In Discover Electronics, the link service is configured as a link servlet that uses the `com.endeca.infront.web.spring.PreviewLinkServlet` class. The servlet is defined in `WEB-INF\web.xml`:

```

<servlet>
    <servlet-name>link</servlet-name>
    <servlet-class>
        com.endeca.infront.assembler.servlet.spring.SpringPreviewLinkServlet
    </servlet-class>
    <init-param>
        <description>
            The ID of the NavigationStateBuilder in the spring
            contextConfig file
        </description>
        <param-name>navigationStateBuilderBeanId</param-name>
        <param-value>navigationStateBuilder</param-value>
    </init-param>
    <init-param>
        <description>
            The ID of the MdexResource in the spring
            contextConfig file
        </description>

```

```

</description>
<param-name>mdexResourceBeanId</param-name>
<param-value>mdexResource</param-value>
</init-param>
</servlet>

```

Changing the preview application in Workbench

You can set the preview application in Workbench by using the **Preview Settings** tool.

Once you have instrumented and enabled your application for preview, you can select it as the preview application in Workbench.

To change the preview application in Workbench:

1. In Workbench, navigate to the **Application Settings > Preview Settings** tool.
2. Click the **Edit** link beside the **Preview URL** field.
3. In the **Preview URL** field, enter the fully qualified URL of your preview application.

If you wish to revert to the default Discover Electronics preview application, recall that the URL is `http://<host>:<port>/discover-electronics-authoring`, or `http://localhost:8006/discover-electronics-authoring` by default.

4. Click **Save**.

The application at the specified URL is now used for preview.

Changing the preview link service

If you have implemented your own link service for use with preview, you can specify the path to the service in your application in the **Preview Settings** tool.

Once you have created your own preview link service, you can specify it for use with preview instead of the default link service included with the Discover Electronics reference application.



Note: For information on the required inputs and outputs for a link service, see the Javadoc for the `AbstractPreviewLinkServlet` class in the `com.endeca.infront.assembler.servlet` package.

To change the preview link service:

1. Stop the Endeca Tools Service.
2. Open your application's deployment descriptor file, `web.xml`.

For the Discover Electronics reference application, this file is located at
`%ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring\WEB-INF\web.xml`.

3. Define the link servlet.

The servlet definition for the Discover Electronics reference application is shown below:

```

<servlet>
    <servlet-name>link</servlet-name>
    <servlet-class>
        com.endeca.infront.assembler.servlet.spring.SpringPreviewLinkServlet
    </servlet-class>
    <init-param>
        <description>
            The ID of the NavigationStateBuilder in the spring
            contextConfig file
        </description>
        <param-name>navigationStateBuilderBeanId</param-name>
    </init-param>

```

```

<param-value>navigationStateBuilder</param-value>
</init-param>
<init-param>
    <description>
        The ID of the ContentSource in the spring
        contextConfig file
    </description>
    <param-name>contentSourceBeanId</param-name>
    <param-value>contentSource</param-value>
</init-param>
</servlet>

```

4. Define the link servlet mapping.

For example:

```

<servlet-mapping>
    <servlet-name>link</servlet-name>
    <url-pattern>/servlet/link.json/*</url-pattern>
</servlet-mapping>

```

5. Save and close the file.
6. Start the Endeca Tools Service.
7. Login to Workbench.
8. In Workbench, navigate to the **Application Settings > Preview Settings** tool.
9. Click the **Edit** link beside the **Link Settings URL** field.
10. In the **Link Settings URL** field, enter the fully qualified URL of your link service within the application.

If you wish to revert to the default Discover Electronics reference application and link servlet, recall that the URL for the link servlet is `http://<host>:<port>/discover-authoring/link.json`.

11. Click **Save**.

The link service at the specified URL is now used for preview.

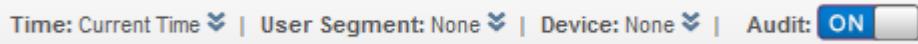
Testing your preview application

After instrumenting and enabling your preview application, you can test the preview and audit functionality in Endeca Workbench.

Your custom preview application must be fully instrumented and enabled in Workbench in order for the preview option to display.

To test your custom preview application:

1. In Workbench, navigate to the **Experience Manager** tool.
2. Navigate to a content item of your choosing.
3. Mouseover the content item.
The Action dropdown button appears.
4. Select **Preview** from the Action dropdown.
The preview application launches in a new browser tab.
5. Optionally, specify the preview time instead of using the default indicated by the system clock for the MDEX Engine:
 - a) Click the arrow beside the selected **Device** in the Preview Toolbar:



Time: Current Time | User Segment: None | Device: None | Audit: **ON**

The Preview Toolbar expands to show configuration options.

- Select a device from the **Device** dropdown and click **Submit**.

Specifying a preview device lets you see how the application renders on that device.

- To test audit functionality:

- Mouseover the cartridge you wish to audit.

The **Audit** button appears.

- Click the **Audit** button.

The **Audit Panel** appears with a list of all content items considered for the specified content slot:

Content Collection: /content/Shared/Guided Navigation						
Status	Name	Location	State	User Segments	Schedule	Priority
● Fired	Category - Cameras	product.category > cameras	<input checked="" type="checkbox"/>			9
Not considered	Category - Cameras - Bags & Accessories	product.category > cameras > camera accessories & supplies	<input checked="" type="checkbox"/>			9
Collection full	Category	product.category	<input checked="" type="checkbox"/>			10
Collection full	Default Guided Navigation	Applies at all locations	<input checked="" type="checkbox"/>			30

- Click any of the listed **Locations** to navigate to that location in the preview application.
- Click the name of any of the listed content items and confirm that you return to that item in Experience Manager.

Configuring Logging for an Assembler Application

By default, the Assembler logs the search and navigation information associated with a request event. However, you can create custom cartridge handlers to collect and act on any information that is important to your application.

About request events

Each invocation of the Assembler creates an associated `RequestEvent` object that tracks request information.

Information on a `RequestEvent` is stored as key/value pairs. You can include arbitrary information on an Assembler request by extending the `RequestEvent` object in a cartridge handler's `process` method. For example:

```
/** 
 * Cartridge Handler process method
 */
public void process(ConfigType pContentType) {

    // Create a new RequestEvent from the global RequestEvent object
    RequestEvent event = RequestEventFactory.getEvent();

    // Store arbitrary information
    event.put("myKey", "my arbitrary value");
}
```

```
    } ...
```

The NavigationEventWrapper class

The `NavigationEventWrapper` class provides convenience methods for getting and setting common search and navigation information on a request event. It modifies the `RequestEvent` object specified in the constructor, as in the example below:

```
/**  
 * Cartridge Handler process method  
 */  
public void process(ConfigType pContentType) {  
  
    // Create a new NavigationEventWrapper around the global RequestEvent object  
    NavigationEventWrapper navigationEvent = new NavigationEventWrapper(RequestEvent-  
    Factory.getEvent());  
  
    // Store navigation event information  
    navigationEvent.setAutocorrectTo("autocorrected term");  
  
    ...  
}
```

For additional information about the `RequestEvent` and `NavigationEventWrapper` classes, including a full list of the convenience methods available for the `NavigationEventWrapper`, see the [Assembler API Reference \(Javadoc\)](#).

About request event adapters

Request event adapter classes perform some action based on information included with a request event.

A request event adapter class implements the `handleAssemblerRequest()` method in the abstract `RequestEventListener` class. This method is invoked at the end of the Assembler's `assemble()` method.

The following is an example of a simple request event adapter:

```
/**  
 * Add log information to root content item  
 */  
public class SampleRequestEventAdapter extends RequestEventListener {  
  
    /**  
     * Constructor  
     * @param sessionIdProvider provides an ID for the current user session  
     */  
    public SampleRequestEventAdapter(SessionIdProvider sessionIdProvider) {  
        super(sessionIdProvider);  
    }  
  
    /**  
     * Prints the request event's session id and search term (if present) to the  
     * console  
     * @param assemblerRequestEvent the event containing all of the  
     * information about the Assembler request  
     * @param rootContentItem the Assembler output  
     */  
    public void handleAssemblerRequestEvent(RequestEvent event, ContentItem  
    rootContentItem) {  
        NavigationEventWrapper navigationEvent = new NavigationEventWrapper(assem-  
    }
```

```

blerRequestEvent);
    // Print Session ID - Note that the session ID has already been determined
    // and set in the event object
    System.out.println("The current session is: "+event.getSessionId());

    // Print Search Term
    if (navigationEvent.getSearchTerms() != null && !navigationEvent.get-
    SearchTerms().trim().isEmpty()) {
        System.out.println("The current search terms are: "+navigationEvent.get-
    SearchTerms());
    } else {
        System.out.println("There were no search terms in the current re-
    quest");
    }
}
}

```

The SessionIdProvider interface

The example request event adapter above registers an implementation of `SessionIdProvider` in the constructor. This enables it to retrieve the server session ID.

The Oracle Endeca Tools and Frameworks installation implements this interface in the included `SpringUtility` class. You can create your own `SessionIdProvider` class by extending the `SessionIdProvider` interface and implementing the `getSessionID()` method.

Request event adapters in the reference application

The Discover Electronics reference application includes the following implementations of the `AssemblerEventListener` interface:

- `AssemblerEventAdapter`
- `ContentItemAugmentAdapter`
- `LogServerAdapter`
- `RequestEventListener`

For additional information on these classes, see the *Assembler API Reference (Javadoc)*.

About registering a request event adapter

To use a request event adapter, you must register it with your `AssemblerFactory`.

You can disable request event adapters by removing them from the `AssemblerFactory` configuration.

Request event adapter configuration in the reference application

In the reference application, the `AssemblerFactory` interface is implemented as `SpringAssemblerFactory`, and the `AssemblerEventListener` objects are specified as constructor arguments in the `Assembler` context file:

```

<!--
#####
# ASSEMBLER FACTORY
#
# Required.
#
-->
<bean id="assemblerFactory" class="com.endeca.infront.assembler.spring.SpringAssemblerFactory"

```

```

        scope="singleton">
        <constructor-arg>
            <bean class="com.endeca.infront.assembler.AssemblerSettings">
                <property name="previewEnabled" value="${preview.enabled}" />
                <property name="previewModuleUrl" value="http://${work-
bench.host}:${workbench.port}/preview" />
            </bean>
        </constructor-arg>
        <constructor-arg>
            <list>
                <bean class="com.endeca.infront.logger.SLF4JAssemblerEventLogger" />

                <bean class="com.endeca.infront.assembler.event.request.ContentItemAug-
mentAdapter">
                    <constructor-arg ref="springUtility"/>
                </bean>
                <bean class="com.endeca.infront.navigation.event.LogServerAdapter">
                    <constructor-arg ref="springUtility"/>
                    <constructor-arg value="${logserver.host}" />
                    <constructor-arg value="${logserver.port}" />
                </bean>
            </list>
        </constructor-arg>
    </bean>

```

Request event adapters in the reference application

The Discover Electronics reference application includes two request event adapters, `ContentItemAugmentAdapter` and `LogServerAdapter`.

Adapter	Description
<code>com.endeca.infront.assem- bler.request.ContentItemAug- mentAdapter</code>	Appends request event information to the Content Item returned by the <code>assemble()</code> method. Information is included as a nested Content Item of type <code>AssemblerRequestEvent</code> , with the key <code>endeca:as- semblerRequestInformation</code> . For a list of attributes that are available out-of-the-box, see Request Event Attributes on page 324.
<code>com.endeca.infront.naviga- tion.event.LogServerAdapter</code>	Formats data from the request event and sends it to the Oracle Endeca Log Server, which allows Workbench users to generate reports using the Report Generator. The adapter must be configured with the host and port of the log server. In the reference application, these values are configured in the <code>WEB-INF\assembler.properties</code> file.

Client side click events

The Oracle Endeca log server tracks the following click events from the client side of an Assembler application:

Attribute Key	Type	Description
<code>IN_DIM_SEARCH</code>	Boolean	Did the user select a dimension search result.

Attribute Key	Type	Description
IN_DYM	Boolean	Did the user select the "did-you-mean" value.
IN_MERCH	Boolean	Did the user select a merch rule (spotlight).
CONVERTED	Boolean	Did an action cause a conversion.

You can include the information collected from these events in your application reports. For more information about the Log Server and Report Generator components, refer to the *Platform Services Log Server and Report Generator Guide*.

Configuring cartridge performance logging

The Assembler tracks performance statistics for registered events; this information is available from the administrative servlet at `http://<workbench host>:<workbench port>/<application>/admin` using the `/admin?op=stats` operation.

For example, you can view the performance statistics for the default Discover Electronics application by navigating to `http://localhost:8006/discover-authoring/admin?op=stats`. For more information on the administrative servlet, see the *Oracle Endeca Commerce Administrator's Guide*.

Performance logging is enabled for the core cartridges included with Tools and Frameworks. If you create a custom cartridge handler and wish to track its processing time, you must use the static `PerfUtil.start()` method to create a corresponding Event.

Example

For example:

```
Event event = PerfUtil.start("com.example.ClassName_MyMethod");
try {
    /* cartridge handler logic */
    event.succeed();
} finally {
    event.failIfNotCompleted();
}
```



Note: A call to `PerfUtil.start` must include a corresponding call to either the `Event.succeed()` or `Event.fail()` method of the returned `Event` instance. Oracle recommends using the `Event.failIfNotCompleted()` helper method within a `finally{}` block to ensure proper resolution.

For more information on the `com.endeca.infront.perf` package, see the *Assembler API Reference (Javadoc)*.

Understanding and Debugging Query Results

The MDEX Engine provides several methods for understanding why certain results were returned for a query so that you can determine how to tune search features to provide the desired results.

Related Links

[Developing an Assembler Application](#) on page 61

This part contains the following sections:

[About the query debugging features](#) on page 102

The MDEX Engine query debugging features include Why Match, Word Interpretation, Why Rank, and Why Precedence Rule Fired. Each feature provides information on a different aspect of search results.

[About enabling query debugging features](#) on page 102

You enable the query debugging features on an Assembler application via the `debugEnabled` constructor argument on your `MdexRequestBroker` object. In the Discover Electronics reference application, this is configured in the MDEX Resource section of the Spring context file for the Assembler.

[URL parameters for query debugging features](#) on page 103

All query debugging features except for Word Interpretation may be enabled on a per-query basis via URL parameters.

[About query debugging results in the reference application](#) on page 103

In Discover Electronics, query debugging results can be returned as part of the response model for the Results List, Search Adjustments, and Refinement Menu cartridges as appropriate. In the Discover Electronics reference application, these results can be enabled by un-commenting the corresponding properties in each cartridge handler.

About the query debugging features

The MDEX Engine query debugging features include Why Match, Word Interpretation, Why Rank, and Why Precedence Rule Fired. Each feature provides information on a different aspect of search results.

Feature	Description
Why Match	Augments record results with information about which record properties were involved in search matching.
Why Rank	Augments record results with information about which relevance ranking modules ordered the results and why a particular record was ranked in the way that it was.
Why Precedence Rule Fired	Augments root dimension values with information about how the precedence rule was triggered (explicitly or implicitly), which dimension ID and name triggered the precedence rule, and the type of precedence rule (standard, leaf, or default).
Word Interpretation	Reports word or phrase substitutions made during text search processing due to stemming, thesaurus expansion, or spelling correction.

About enabling query debugging features

You enable the query debugging features on an Assembler application via the `debugEnabled` constructor argument on your `MdexRequestBroker` object. In the Discover Electronics reference application, this is configured in the MDEX Resource section of the Spring context file for the Assembler.

When `debugEnabled` is set to `true`, it enables query debugging features to be applied to an Assembler request. When set to `false`, debugging features are turned off for every request. Debugging features are disabled by default.

 **Important:** Query debugging features are not optimized for performance and can be very expensive to process. For both performance and security reasons, the debug constructor argument should always be set to false in a production environment.

In addition to the corresponding object configuration, Word Interpretation must be enabled via the `--wordInterp Dgraph` flag.

The following shows the default MDEX resource configuration in the Discover Electronics application:

```
<bean id="mdexRequestBuilder" scope="request" class="com.endeca.infront.navigation.request.MdexRequestBroker">
    <constructor-arg ref="mdexResource" />
    <!-- Debug Enabled Parameter. When set to true, allows debug information to
be returned from the Assembler -->
    <constructor-arg value="false"/>
</bean>
```

URL parameters for query debugging features

All query debugging features except for Word Interpretation may be enabled on a per-query basis via URL parameters.

The following parameters take a value of 1 (for enabled) or 0 (for disabled):

- whymatch
- whyrank
- whyprecedencerulefired

The Word Interpretation feature can only be enabled at the level of an individual cartridge handler.



Note: If the debug constructor argument on the MDEX resource bean is set to `false`, all debugging features are disabled on every request regardless of URL parameters.

About query debugging results in the reference application

In Discover Electronics, query debugging results can be returned as part of the response model for the Results List, Search Adjustments, and Refinement Menu cartridges as appropriate. In the Discover Electronics reference application, these results can be enabled by un-commenting the corresponding properties in each cartridge handler.

The debugging results are returned as properties on returned records:

Feature	Results
Why Match	Returns information about why each record matched the query in a <code>Dgraph.WhyMatch</code> property on the record.
Why Rank	Returns information about why each record was ranked the way it was in a <code>Dgraph.WhyRank</code> property on the record.
Why Precedence Rule Fired	Returns information about precedence rules that fired on a query in a <code>DGraph.WhyPrecedenceRuleFired</code> property on each root dimension value.
Word Interpretation	Returns information about word or phrase substitutions as a map that can be accessed via <code>getInterpretedTerms()</code> on the <code>SearchAdjustments</code> model.

For details about the format of the debugging results, refer to the *MDEX Engine Developer's Guide*.



Note: The renderers in the Discover Electronics application do not include rendering logic to display the query debugging properties, but the information is available from the JSON or XML view.

The relevant configuration for the individual cartridge handlers in the Discover Electronics reference application is shown below:

- Results List — Why Match, Why Rank

```
<bean class="com.endeca.infront.cartridge.ResultsListConfig" scope="singleton">  
    <!-- <property name="whyMatchEnabled" value="true"/> -->  
    <!-- <property name="whyRankEnabled" value="true"/> -->  
    <!-- additional elements omitted from this example -->  
</bean>
```

Enabling these settings overrides the default values specified for the `setWhyMatchEnabled` and `setWhyRankEnabled` methods on the `com.endeca.infront.cartridge.ResultsListConfig` object when the Endeca Tools Service is initialized.

- Refinement Menu — Why Precedence Rule Fired

```
<bean class="com.endeca.infront.cartridge.RefinementMenuConfig" scope="singleton">  
    <property name="moreLinkText" value="More..."/>  
    <!-- <property name="whyPrecedenceRuleFired" value="true"/> -->  
</bean>
```

Enabling this setting overrides the default value specified for the `setWhyPrecedenceRuleFired` method on the `com.endeca.infront.cartridge.RefinementMenuConfig` object when the Endeca Tools Service is initialized.

- Search Adjustments — Word Interpretation

```
<bean class="com.endeca.infront.cartridge.SearchAdjustmentsConfig" scope="singleton">  
    <!-- <property name="showWordInterp" value="true"/> -->  
</bean>
```

Enabling this setting overrides the default value specified for the `setShowWordInterp` method on the `com.endeca.infront.cartridge.SearchAdjustmentsConfig` object when the Endeca Tools Service is initialized.



Important: In order for interpreted word information to be available, you must start your Dgraph with the `--wordInterp` flag. This flag is not enabled in the deployment descriptor file for the Discover Electronics reference application.

Part 4

Optimizing Application URLs

- *About the URL optimization classes*
- *Working with Application URLs*
- *Preparing your application*
- *Building optimized URLs*
- *Configuring URLs*
- *Integrating with the Sitemap Generator*

Chapter 10

About the URL optimization classes

This section provides an introduction to the URL optimization classes in the Assembler API.

Package contents

The `com.endeca.soleng.urlformatter` package within `ToolsAndFrameworks\<version>\Assembler\lib\endeca_assembler-<version>.jar` contains the classes and dependencies necessary for generating optimized URLs and canonical links in your application.

To enable the API for the Discover Electronics reference application, the `endeca_assembler-<version>.jar` file is also included under the `ToolsAndFrameworks\<version>\reference\discover-electronics-authoring\WEB-INF\lib` directory.

Introduction to URL optimization

Dynamically created URLs that are composed of meaningless, randomly generated strings can lower your site's search engine ranking and make it harder for users to recognize your site. The Assembler API includes classes that enable you to create site links using directory-style URLs. These URLs 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.



Note: The examples in this guide assume a sample Web application running on `http://localhost:8888` against a wine data set.

Overview of URL optimization capabilities

The URL optimization classes are designed to increase your search engine rankings by enabling you to create search engine-friendly URLs.

Integration of keywords into the URL string

Many search engines evaluate URL strings 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 classes, you can configure the following strings to appear in the URL:

- Dimensions
- Dimension values
- Dimension ancestors
- Record properties
- Text search queries

For example, the base URL for a Merlot page in a wine application configured to include ancestors in the string could appear as:

```
http://localhost/ContentAssemblerRefApp/Content.aspx/Wine-Red-Merlot/
```

The optimized URL is more comprehensible to users and more search-engine friendly than the traditional URL, which contains no keywords:

```
http://localhost:8888/endeca_jspref/controller.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 lower search engine ranking.

For example, users might be able to reach a 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 `com.endeca.soleng.urlformatter.NavStateCanonicalizer` interface provides options for creating a single "canonical" URL for a given location.

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

To optimize the URL, 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 classes allow 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 can lower the search engine ranking of a page. Canonicalizing URLs reduces the 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. Multiple URLs generated by a dynamic site can lower the ranking of a page because, to the search engine, each version of the URL appears to point to a different page.

For example, users might be able to reach a 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-Napa/_/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 classes prevent 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` object 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 139

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 11

Working with Application URLs

Each of the user-facing pages in an Assembler application exists as a page with a corresponding navigation or record state; the combination of the page and its state results in a specific set of results or a set of record details. The Assembler API includes an `Action` class for storing these URL components and returning them as part of the output model produced by a cartridge handler.

About application URLs

Features in a front-end application can provide one or more links to other locations within a site. The information required for constructing these links is provided on the cartridge response model as an `Action` object that includes the components of a destination URL.

For example, a dimension refinement in a Refinement Menu cartridge has an associated action to select the refinement and add it to the end user's navigation state. A record in a Results List cartridge has an action to view the corresponding record detail page.

The Assembler API includes an `ActionPathProvider` interface that returns components of an application URL. For the Discover Electronics reference application, an implementation of this interface is configured in the `NavigationCartridgeHandler`.

Cartridge handlers in the reference application use this implementation to create `NavigationAction` paths to a certain navigation state (like the modified navigation state created when a user selects a dimension refinement), or `RecordAction` paths to specified records (such as a record select from the results list).

About Actions

An `Action` object allows you to construct a link that represents a decision by an end user. The included fields and values depend on the action that the user wishes to take; they can include the action label, the root site path, and the path to the destination content within the site.

The `Action` class does not include a complete URL to the resulting navigation state or record; instead, the URL resulting from an `Action` is generally created by combining fields. For details, see "Action fields."

The Assembler splits the class into three subclasses:

- `NavigationAction` — An `Action` that represents changing the current navigation state, such as through a search query or the addition of a dimension refinement. For example, the "See All" link on a `RecordSpotlight` object includes a `NavigationAction` for navigating to the refinement state represented by the spotlight.

- **RecordAction** — An Action that represents selecting a record or aggregate record. The individual records in a RecordSpotlight each include a RecordAction for selecting that record.
- **UrlAction** — An Action that represents following an arbitrary URL. The Media Banner cartridge includes a UrlAction for URLs that are manually specified in Experience Manager.



Note: For information on the Actions associated with each output model, refer to the *Assembler API Reference (Javadoc)* for the corresponding class.

Action fields

All Actions include the following fields:

Field	Description
Label	The label that displays to the application end-user for the specified action. For example, you might set this to a product name for a link from a results list to a record detail page, or it you might set it to a dimension refinement name when displaying a breadcrumb with an associated Action to remove the refinement and adjust the user's navigation state.
Site root path	The path that identifies the site associated with the Action.
Content path	The path that identifies the content associated with the Action within the containing site . In the Discover Electronics reference application, this is the servlet that handles the specified content type, such as /browse or /detail.

Additionally, certain types of Actions may include additional fields. A `NavigationAction` includes a field for the navigation state represented by the Action, while a `RecordAction` action includes a field for the corresponding record state.

Using action fields

To construct a useable link from an Action, the UI tier of your application (the cartridge renderers in the Discover Electronics reference application) must include logic for combining the Action fields. A typical use case consists of directly concatenating fields, depending on the type of page you wish to link to.

In the reference application, a link to a navigation state typically combines the content path and the desired navigation state:

```
String href = action.getContentPath() + action.getNavigationState();
```

A link to a record details page combines the content path with the appropriate record state:

```
String href = action.getContentPath() + action.getRecordState();
```

A link to an arbitrary URL does not require combining fields, since the `UrlAction` object includes a method for directly retrieving a configured URL:

```
String href = action.getUrl();
```

Most of the Discover Electronics cartridge renderers use the `<discover:link>` tag, defined in `WEB-INF\tags\discover\link.tag`. The tag makes use of the `getUrlForAction` function declared in `WEB-INF\tlds\functions.tld` and defined in `WEB-INF\classes\com\endeca\infront\refapp\support\FunctionTags.java`.

About using Actions with the packaged services

The packaged services in Oracle Endeca Tools and Frameworks return specific actions for the included cartridges.

The following is an example of the results of a guided search service query for the URI `http://localhost:8006/assembler-authoring/json/services/guidedsearch?Ntt=pink+camera`, serialized to JSON:

```
{
  "@type": "GuidedSearchService",
  "name": "Guided Search Service",
  "navigation": { ... },
  "breadcrumbs": { ... },
  "resultsList": {
    "@type": "ResultsList",
    "totalNumRecs": 213,
    "sortOptions": [
      {
        "@class": "com.endeca.infront.cartridge.model.SortOptionLabel",
        "selected": false,
        "navigationState": "?Ntt=pink+camera",
        "contentPath": "/guidedsearch",
        "siteRootPath": "/services",
        "label": "Relevance"
      },
      {
        "@class": "com.endeca.infront.cartridge.model.SortOptionLabel",
        "selected": false,
        "navigationState": "?Ns=product.price%7C0&Ntt=pink+camera",
        "contentPath": "/guidedsearch",
        "siteRootPath": "/services",
        "label": "Price (Ascending)"
      },
      { ... }
    ],
    "firstRecNum": 1,
    "lastRecNum": 10,
    "pagingActionTemplate": { ... },
    "recsPerPage": 10,
    "records": [
      {
        "@class": "com.endeca.infront.cartridge.model.Record",
        "detailsAction": {
          "@class": "com.endeca.infront.cartridge.model.RecordAction",
          "recordState": "/Kodak/Slim-Camera-Case/_/A-2707821",
          "contentPath": "/recorddetails",
          "siteRootPath": "/services"
        },
        "numRecords": 1,
        "attributes": { ... },
        "records": [ ... ]
      },
      { content removed from this example }
    ],
    { content removed from this example }
  ],
  "content removed from this example"
}
```

Note that the `sortOptions` returned for the Results List cartridge include the Action fields required to create a URL for the navigation state resulting from modifying the sort order. Sorting by Price (Ascending) requires constructing a URL with the appropriate `navigationState`, resulting in `http://localhost:8006/assembler-authoring/json/services/guidedsearch?Ns=product.price|0&Ntt=pink+camera`. Querying this URL returns the JSON response for the re-ordered results.

Similarly, each of the records returned in the Results List includes the Action fields for an associated record details page. Using the `/recorddetails` content root and the `recordState` for the Slim Camera Case results in the URL

`http://localhost:8006/assembler-authoring/json/services/recorddetails/Kodak/Slim-Camera-Case/_/A-2707821`. Querying this URL returns the record details for the Slim Camera Case.

About working with URL parameters

The `navigationStateBuilder` handles both Endeca-specific and non-Endeca URL parameters.

All URL parameters are parsed into the parameters map in the `NavigationState` object that represents the user's current navigation state. Endeca-specific parameters are used in constructing MDEX Engine queries. All other parameters are included in the navigation state or record state fields on the Action object in the output model. You can change this default behavior by specifying which parameters to remove when generating Actions:

Property	Description
<code>removeAlways</code>	A list of URL parameters that should be removed from all Actions.
<code>removeOnUpdateFilterState</code>	A list of URL parameters that should be removed from an Action when the Action represents a change in the filter (search or navigation) state.
<code>removeOnClearFilterState</code>	A list of URL parameters that should be removed from an Action when the user clears the filter state of all search and navigation selections.

About URL configuration in the reference application

URL configuration in the Discover Electronics reference application is located in the Assembler context file, `WEB-INF\assembler-context.xml`. Configuration is divided between the `navigationStateBuilder` and the `NavigationCartridgeHandler`.

The configuration for the `navigationStateBuilder` specifies a `urlFormatter` to use when serializing a `NavigationState`:

```

<!--
 ~ Navigation state (including record state) and related config
-->

<bean id="navigationStateBuilder" scope="request"
      class="com.endeca.infront.navigation.url.UrlNavigationStateBuilder">
  <property name="urlFormatter" ref="seoUrlFormatter" />
  <property name="mdexRequestBroker" ref="mdexRequestBroker" />
  <property name="defaultSearchKey" value="All" />

```

```

<property name="defaultMatchMode" value="ALLPARTIAL" />
<property name="defaultFilterState">
    <!-- Filter state properties removed from this example -->
</property>

```



Note: The seoUrlFormatter bean is defined in the imported endeca-seo-url-config file.

Configuring URL parameters

The configuration for the navigationStateBuilder also lets you specify the URL parameters to remove from the request URL when serializing a NavigationState or RecordState:

```

<property name="removeAlways">
    <list>
        <value>contentText</value>
        <value>Nty</value>
        <value>Dy</value>
        <value>collection</value>
    </list>
</property>
<property name="removeOnUpdateFilterState">
    <list>
        <value>No</value>
    </list>
</property>
<property name="removeOnClearFilterState">
    <list>
        <value>Ns</value>
        <value>Nrpp</value>
        <value>more</value>
    </list>
</property>
</bean>

```

Configuration for navigation and record paths

The content paths that prefix navigation and record states when creating Action URLs are configured in the actionPathProvider of the NavigationCartridgeHandler as sets of key-value pairs:

```

<bean id="NavigationCartridgeHandler" abstract="true">
    <property name="navigationState" ref="navigationState" />
    <property name="mdexRequestBroker" ref="mdexRequestBroker" />
    <property name="actionPathProvider">
        <bean scope="request" class="com.endeca.infront.refapp.navigation.BasicActionPathProvider">
            <constructor-arg index="0" ref="contentSource" />
            <constructor-arg index="1" ref="httpServletRequest" />
            <!-- navigationActionUriMap -->
            <constructor-arg index="2">
                <map>
                    <entry key="^/pages/mobile/detail$" value="/pages/mobile/browse" />
                    <entry key="^/pages/detail$" value="/pages/browse" />
                </map>
            </constructor-arg>
            <!-- recordActionUriMap -->
            <constructor-arg index="3">
                <map>

```

```

        <entry key="^/pages/mobile/.*$" value="/pages/mobile/detail"
/>
        <entry key="^/pages/.*$" value="/pages/detail" />
    </map>
</constructor-arg>
</bean>
</property>
</bean>

```

URL formatter configuration

The Discover Electronics reference application serializes `NavigationState` objects through the use of a `UrlNavigationStateBuilder` configured with a `UrlFormatter`. By default, the application is configured for search engine optimized (SEO) URLs using the `SeoUrlFormatter` class, but it also includes a `BasicUrlFormatter` for creating basic Endeca URLs.

The basic URL formatter

The following properties can be set on the `basicUrlFormatter` bean:

Property	Description
<code>defaultEncoding</code>	Specifies the default query encoding, for example, <code>UTF-8</code> .
<code>prependQuestionMark</code>	Specifies whether a question mark is prepended to the URL parameter portion of the URL, after the servlet path.

The configuration in `WEB-INF\endeca-url-config` is shown below:

```

<!--
#####
# BEAN: basicUrlFormatter
#
# This is an UrlFormatter that generates "classic" Endeca URLs.
#
-->

<bean id="basicUrlFormatter" class="com.endeca.soleng.urlformatter.basic.BasicUrlFormatter">
    <property name="defaultEncoding">
        <value>UTF-8</value>
    </property>

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

```

The SEO URL formatter

The following properties can be set on the `seoUrlFormatter` bean:

Property	Description
<code>defaultEncoding</code>	Specifies the default query encoding, for example, <code>UTF-8</code> .
<code>pathSeparatorToken</code>	The separator token used to separate the path section of the URL from the parameter section.

Property	Description
pathKeyValueSeparator	The character used to separate key/value pairs in the parameter section of the URL.
pathParamKeys	Specifies the URL parameter keys for the following: <ul style="list-style-type: none"> The parameter key used for record detail links. The default value is <code>R</code>. The parameter key used for aggregate record detail links. The default value is <code>A</code>. The parameter key used for navigation state. The default value is <code>N</code>.
navStateFormatter	The <code>NavStateFormatter</code> that maps navigation state information to URL path keywords.
ERecFormatter	The <code>ERecFormatter</code> that maps Endeca record attributes to URL path keywords.
aggrERecFormatter	The <code>AggrERecFormatter</code> that maps aggregate record attributes to URL path keywords.
navStateCanonicalizer	Specifies the canonicalizer used to sort URL parameters to ensure that included parameters are arranged a specific order.
useNavStateCanonicalizer	Determines whether or not the canonicalizer specified in <code>navStateCanonicalizer</code> is used. The default value is <code>true</code> . This value is ignored if the <code>canonicalLinkBuilder</code> enables canonical links.
urlParamEncoders	A list of <code>UrlParamEncoder</code> objects to use for encoding URL parameters.

The configuration in `WEB-INF\endeca-seo-url-config` is shown below:

```

<!--
#####
# BEAN: seoUrlFormatter
#
# This is the SEO URL formatter, which is responsible for
# transforming UrlState objects into URL strings.
#
-->
<bean id="seoUrlFormatter" class="com.endeca.soleng.urlformatter.seo.SeoUrlFormatter">

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

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

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

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

```

```

</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="useNavStateCanonicalizer">
    <value>false</value>
</property>

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

```

About working with canonical links

Configure the Assembler to add canonical link support to the root content item.

The canonical link configuration in the Discover Electronics reference application is located in the Assembler context file, `WEB-INF\Assembler-context.xml`. Configuration is handled by the `canonicalLinkBuilder` which constructs links for navigation state and record state URLs that include the canonical link element.

The Canonical Link Builder

The following properties can be set on the `canonicalLinkBuilder`:

Property	Description
<code>objectLocator</code>	Allows the retrieval of services without explicit injection. In this case, it is used to reference the framework for retrieving the <code>recordState</code> and <code>navigationState</code> for the current request.
<code>recordStateId</code>	The ID of the <code>recordState</code> being retrieved, not the actual <code>recordState</code> .
<code>navigationStateId</code>	The ID of the <code>navigationState</code> being retrieved, not the actual <code>navigationState</code> .
<code>includedParameters</code>	The list of URL parameters that are included in the canonical link.

The configuration for the canonicalLinkBuilder specifies an objectLocator to use when creating canonical links:

```

<bean id="assemblerFactory" class="com.endeca.infront.assembler.spring.SpringAssemblerFactory">
    ...
    <constructor-arg>
        <list>
            ...
            <bean class="com.endeca.infront.navigation.url.event.CanonicalLinkBuilder">
                <property name="objectLocator" ref="springUtility"/>
                <property name="recordStateId" value="recordState"/>
                <property name="navigationStateId" value="navigationState"/>
                <property name="includedParameters">
                    <list>
                        <value>R</value>
                        <value>A</value>
                        <value>N</value>
                        <value>Ntt</value>
                    </list>
                </property>
            </bean>
        </list>
    </constructor-arg>
</bean>

```

Output content items

The Assembler API returns navigation state and record state content items as output from the CanonicalLinkBuilder. The following examples are JSON representations of the output.

NavigationState

```
{
    name: "Static Page Slot",
    ...,
    canonicalLink: {
        @class: "com.endeca.infront.cartridge.model.NavigationAction",
        navigationState: "/Canon/cameras/_/N-1z141xuZ1z141yaZ25y6Zej4?format=json",

        contentPath: "/browse",
        siteRootPath: "/pages",
        label: ""
    }
}
```

RecordState

```
{
    name: "Static Page Slot",
    ...,
    canonicalLink: {
        @class: "com.endeca.infront.cartridge.model.RecordAction",
        recordState: "/_A-1318562?format=json",
        contentPath: "/detail",
        siteRootPath: "/pages",
        label: ""
    }
}
```

```
    }  
}
```

For each of the content items, a JSP file can render output as in this example:

```
<link rel="canonical" href="
```

Chapter 12

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

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 optimized URLs, 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 URLs to optimized URLs, or when you change the configuration of your 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 classes are responsible for transforming URLs into Endeca search and navigation queries, and vice-versa. They do 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 13

Building optimized URLs

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

Core URL optimization classes

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. When 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 more information about these and other classes, refer to the *Assembler API Reference (Javadoc)*.

Overview of building URLs using the URL optimization classes

Building optimized URLs with the Assembler API requires passing in the necessary configuration and instantiating the required objects.

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 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, follow these steps:

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 possibilities, 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`

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

This procedure requires that you have an informed `UrlState` representing the current navigation state of your page.

To create a link URL, follow these steps:

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

For example, the following statement creates 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 *Assembler API Reference (Javadoc)*.

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 14

Configuring URLs

The following sections provide information about creating and using a URL configuration file 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 URL optimization 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 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 appears 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 that are not configured to appear in the path-params section of the URL – such as `N`, `Ne`, `Ntt`, and `R` – appear in the query string.

About the URL configuration file

The example 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 Assembler 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 a `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 include 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 URL configuration file defines a `BasicQueryBuilder` and a top-level `SeoUrlFormatter`.

To create a URL configuration file, follow these steps:

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.Basic-
QueryBuilder">
</bean>
```

2. Add the following properties:

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

For example:

```
<bean id="queryBuilder" class="com.endeca.soleng.urlformatter.basic.Basic-
QueryBuilder">

<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.urlformatter.seo.SeoUrlFormatter` class:

For example:

```

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

```

4. Add the following properties:

Option	Description
<code>defaultEncoding</code>	Specifies the default query encoding. For example: <code><value>UTF-8</value></code>
<code>pathSeparatorToken</code>	Specifies the character used to separate the misc-path from the path-params section in URLs.
<code>pathKeyValueSeparator</code>	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.urlformatter.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.urlformatter.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.

Related Links

[Using the URL configuration file with your application](#) on page 154

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

About optimizing the misc-path

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`

By optimizing the URL, it 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 classes prevent 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, and dimension value names are available for aggregate record detail pages. While the `urlconfig.xml` configuration file uses the same `dimLocationFormatters` for the `aggrERecFormatter` and the `erectFormatter` 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.

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 Assembler API. Formatting may include changing capitalization or applying a regular expression to replace portions of the string.

There are several `StringFormatter` objects in the Assembler 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 *Assembler API Reference (Javadoc)*.

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.RegexStringFormatter">

    <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.RegexStringFormatter">

<property name="pattern">
<value><![CDATA[ [\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.RegexStringFormatter">

<property name="pattern">
<value><![CDATA[ ^-?([\u00C0-\u00FF][\u00C0-\u00FF]*[\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.RegexStringFormatter">

```

```

<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 URL optimization, 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. Open your URL configuration file.
2. Create a `navStateFormatter` bean to invoke the `com.endeca.soleng.urlformatter.seo.SeoNavStateFormatter`:

For example:

```

<bean id="navStateFormatter" class="com.endeca.soleng.urlformatter.seo.SeoNavStateFormatter">
</bean>

```

3. Add a `navStateFormatter` property to your top-level `seoUrlFormatter` bean.

For example:

```

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

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

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

</bean>

```

4. Add a `useDimensionNameAsKey` property on the `navStateFormatter`.

For example:

```
<bean id="navStateFormatter" class="com.endeca.soleng.urlformatter.seo.SeoNavStateFormatter">

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

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

5. Add a `dimLocationFormatters` property and list each `dimLocationFormatter` bean you plan to define.

For example:

```
<bean id="navStateFormatter" class="com.endeca.soleng.urlformatter.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>
```

6. 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.

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

Property	Description
key	In the <code>navStateFormatter</code> bean, the <code>useDimensionNameAsKey</code> property sets the key type. If you set the <code>useDimensionNameAsKey</code> to <code>true</code> , then use the dimension name as the value for this property (for example <code><value>Region</value></code>). If you set the <code>useDimensionNameAsKey</code> to <code>false</code> , use the dimension ID number.
appendRoot	Specifies whether or not to append root dimension values to the URL. Set to <code>true</code> to append root dimension values.
appendAncestors	Specifies whether or not to append ancestor dimension values to the URL. Set to <code>true</code> to append ancestor dimension values.

Property	Description
appendDescriptor	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to <code>true</code> 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 <code>defaultStringFormatterChain</code> bean to invoke the <code>com.endeca.soleng.urlformatter.seo.StringFormatterChain</code> .
dimValStringFormatter	Specifies the bean to format the dimension value names. The reference application uses a <code>defaultStringFormatterChain</code> bean to invoke the <code>com.endeca.soleng.urlformatter.seo.StringFormatterChain</code> . The examples below also use a <code>defaultStringFormatterChain</code> 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>

```

8. Create a `navStateCanonicalizer` bean to invoke the `com.endeca.soleng.urlformatter.seo.SeoNavStateCanonicalizer` class.

For example:

```

<bean name="navStateCanonicalizer" class="com.endeca.soleng.urlformatter.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.

9. Add a `navStateCanonicalizer` property to your top-level `seoUrlFormatter` bean.

For example:

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

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

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

</bean>
```

10. Configure the `navStateCanonicalizer`.

For example, the following configuration creates URLs sorted by dimension ID in descending order:

```
<bean name="navStateCanonicalizer" class="com.endeca.soleng.urlformatter.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 are a number of possible configuration options for canonicalization.

11. Save and close the file.

Related Links

[Preparing your properties](#) on page 121

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

[Preparing your dimensions](#) on page 121

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**.

[About URL canonicalization](#) on page 109

Dynamic sites often produce syntactically different URLs for the same page. Multiple variant URLs can lower the search engine ranking of a page. Canonicalizing URLs reduces the duplicate content and improves search engine ranking.

[Formatting misc-path strings in optimized URLs](#) on page 133

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

Sort by	Configuration	Example base URL (sort direction ascending)
case sensitive	<pre data-bbox="376 291 922 481"><property name="sortByDimension"> <value>false</value> </property> <property name="ignoreCase"> <value>false</value> </property></pre>	
Dimension value name, case insensitive	<pre data-bbox="376 519 922 840"><property name="sortByName"> <value>true</value> </property> <property name="sortByDimension"> <value>false</value> </property> <property name="ignoreCase"> <value>true</value> </property></pre>	http://localhost/urlformatter_jspref/controller/region-Napa/Wine-red/
Dimension value ID	<pre data-bbox="376 882 922 1077"><property name="sortByName"> <value>false</value> </property> <property name="sortByDimension"> <value>false</value> </property></pre>	http://localhost/urlformatter_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.urlformatter.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.urlformatter.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 classes, 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. Open your URL configuration file.
2. 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>
```

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

For example:

```
<bean id="seoUrlFormatter" class="com.endeca.soleng.urlformatter.seo.SeoUrlFormatter">
<!-- additional elements deleted from this example --!>
```

```

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

</bean>

```

4. Add a `useDimensionNameAsKey` property on the `erecFormatter`.

For example:

```

<bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.SeoERec-
Formatter">

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

</bean>

```

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

5. 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.SeoERec-
Formatter">

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

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

</bean>

```

6. 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.SeoERec-
Formatter">

  <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>

```

7. Add a `dimLocationFormatters` property and list each `dimLocationFormatter` bean you plan to define.

For example:

```
<bean id="erecFormatter" class="com.endeca.soleng.urlformatter.seo.SeoERec-
Formatter">

    <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>
```

8. 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.

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

Property	Description
<code>key</code>	In the <code>navStateFormatter</code> bean, the <code>useDimensionNameAsKey</code> property sets the key type. If you set the <code>useDimensionNameAsKey</code> to true, then use the dimension name as the value for this property (for example <code><value>Re- gion</value></code>). If you set the <code>useDimensionNameAsKey</code> to false, use the dimension ID number.
<code>appendRoot</code>	Specifies whether or not to append root dimension values to the URL. Set to true to append root dimension values.
<code>appendAncestors</code>	Specifies whether or not to append ancestor dimension values to the URL. Set to true to append ancestor dimension values.

Property	Description
appendDescriptor	Specifies whether or not to append the selected or descriptor dimension values to the URL. Set to <code>true</code> 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 <code>defaultStringFormatterChain</code> bean to invoke the <code>com.endeca.soleng.urlformatter.seo.StringFormatterChain</code> .
dimValStringFormatter	Specifies the bean to format the dimension value names. The reference application uses a <code>defaultStringFormatterChain</code> bean to invoke the <code>com.endeca.soleng.urlformatter.seo.StringFormatterChain</code> . The examples below also use a <code>defaultStringFormatterChain</code> 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>

```

10. Save and close the file.

Related Links

[Preparing your properties](#) on page 121

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

[Preparing your dimensions](#) on page 121

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**.

[Formatting misc-path strings in optimized URLs](#) on page 133

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 classes, 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. Open your URL configuration file.
2. Create an `aggrERecFormatter` bean to invoke the `com.endeca.soleng.urlformatter.seo.SeoAggrERecFormatter` class:

For example:

```
<bean id="aggrERecFormatter" class="com.endeca.soleng.urlformatter.seo.SeoAggrERecFormatter">
  </bean>
```

3. Add an `aggrERecFormatter` property to your top-level `seoUrlFormatter` bean.

For example:

```
<bean id="seoUrlFormatter" class="com.endeca.soleng.urlformatter.seo.SeoUrlFormatter">
  <!-- additional elements deleted from this example --!>
  <property name="aggrERecFormatter">
    <ref bean="aggrERecFormatter"/>
  </property>
</bean>
```

4. Add a `useDimensionNameAsKey` property on the `aggrERecFormatter`.

For example:

```
<bean id="aggrERecFormatter" class="com.endeca.soleng.urlformatter.seo.SeoAggrERecFormatter">
  <property name="useDimensionNameAsKey">
    <value>true</value>
  </property>
</bean>
```

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

5. Add a `propertyKeys` property to include record properties in the URLs of record details pages.

For example:

```
<bean id="aggrERecFormatter" class="com.endeca.soleng.urlformatter.seo.SeoAggrERecFormatter">

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

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

</bean>
```

6. 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.urlformatter.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>
```

7. Add a `dimLocationFormatters` property and list each `dimLocationFormatter` bean you plan to define.

For example:

```
<bean id="aggrERecFormatter" class="com.endeca.soleng.urlformatter.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.

8. 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>

```

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

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

```

10. Save and close the file.

Related Links

[Preparing your properties](#) on page 121

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

[Preparing your dimensions](#) on page 121

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**.

[Formatting misc-path strings in optimized URLs](#) on page 133

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 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 classes provide 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 Assembler 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.urlformatter.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.urlformatter.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.urlformatter.seo.SeoUrlFormatter">
```

```

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

</bean>

```

Encoding Endeca parameters

You can 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. Open your URL configuration file.
2. Create a paramEncoder bean to invoke the com.endeca.soleng.urlformatter.seo.SeoNavStateEncoder:

For example:

```

<bean name="An-paramEncoder" class="com.endeca.soleng.urlformatter.seo.SeoNavStateEncoder">
</bean>

```



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

3. Add a paramKey property to specify which numeric Endeca parameter to encode.

For example:

```

<bean name="An-paramEncoder" class="com.endeca.soleng.urlformatter.seo.SeoNavStateEncoder">
  <property name="paramKey">
    <value>An</value>
  </property>
</bean>

```

4. Repeat steps one and two for each Endeca parameter you want to encode.

5. 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>

```

6. Add a `urlParamEncoders` property:

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

7. Add a `list` attribute and specify each of the parameter encoder beans.

For example:

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

8. Save and close the file.

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 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\ToolsAndFrameworks\<version>\reference\discover-electronics-authoring\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 129

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

[Creating a URL configuration file](#) on page 129

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

Chapter 15

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 as those configured for the Assembler API.

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 Assembler, configuration in the URL configuration file must correspond to 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 Assembler API, you can copy the `urlconfig.xml` file for sitemap generation.

Using the URL configuration file with the Sitemap Generator

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 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: `ToolsAndFrameworks\<version>\sitemap_generator\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\ToolsAndFrameworks\<version>\reference\discover-electronics-authoring\WEB-INF\urlconfig.xml</URL_FORMAT_FILE>
```

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

Related Links

[Creating a URL configuration file](#) on page 129

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

[About the URL configuration file](#) on page 128

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

Part 5

Extending the Assembler

- *Extending and Developing Cartridges*
- *Developing Editors for Workbench*

Chapter 16

Extending and Developing Cartridges

If your application requires functionality that is not covered by the core cartridges and navigation cartridges included in Tools and Frameworks, you can extend the existing cartridges or develop your own.

Cartridge Basics

This section introduces the basic components of a cartridge by examining how they work together in a "Hello, World" example cartridge.

First steps with a new cartridge

This section describes how to define a new cartridge and use Workbench to configure it to appear on a page.

To create and configure a basic "Hello, World" cartridge, follow these steps:

1. Navigate to the templates directory of your application, and create a subdirectory named "HelloWorld." This directory name is the template ID for your template.
For example: C:\Endeca\apps\Discover\config\cartridge_templates\HelloWorld.
2. Create a cartridge template.
 - a) Open a new plain text or XML file.
 - b) Type or copy the following into the contents of the file:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:editors="editors"
                  type="SecondaryContent">
  <Description>A sample cartridge that can display a simple
  message.</Description>
  <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/sidebar_con-
tent.jpg</ThumbnailUrl>
  <ContentItem>
    <Name>Hello cartridge</Name>
    <Property name="message">
      <String/>
    </Property>
    <Property name="messageColor">
      <String/>
    </Property>
  </ContentItem>
```

```

<EditorPanel>
  <BasicContentItemEditor>
    <editors:StringEditor propertyName="message" label="Message" />
    <editors:StringEditor propertyName="messageColor"
      label="Color"/>
  </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

c) Save the file with the name `template.xml` in the `HelloWorld` directory of your Discover Electronics application, for example:

`C:\Endeca\apps\Discover\config\cartridge_templates\HelloWorld.`

3. Upload the template to Endeca Workbench.

a) Open a command prompt and navigate to the `control` directory of your deployed application, for example, `C:\Endeca\apps\Discover\control`.
 b) Run the `set_templates` command.

```

C:\Endeca\apps\Discover\control>set_templates.bat
Removing existing cartridge templates for Discover
Setting new cartridge templates for Discover
Finished setting templates

```

```
C:\Endeca\apps\Discover\control>
```

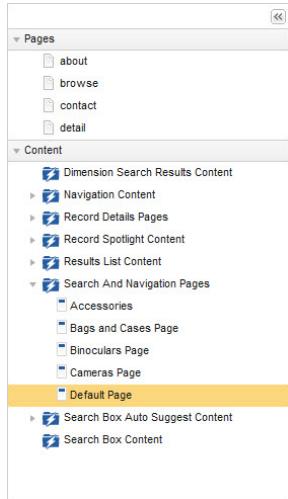
4. Add the cartridge to a page.

a) Open Endeca Workbench in a Web browser.

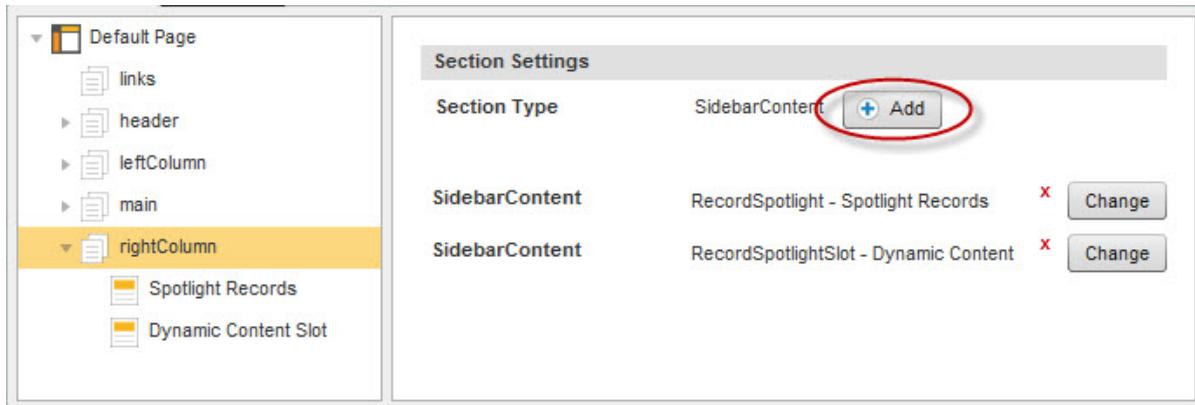
The default URL for Workbench is `http://<workbench-host>:8006`. The default **Username** is `admin` and the default **Password** is `admin`.

b) From the launch page, select **Experience Manager**.

c) In the tree on the left, select **Search and Navigation Pages** under the Content section, then select the **Default Page**.

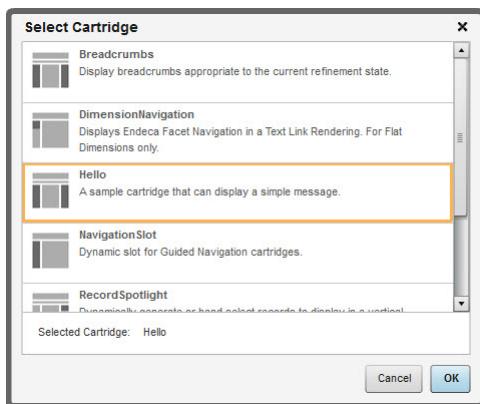


d) In the Edit Pane on the right, select the right column section from the Content Tree in the bottom left.



e) Click **Add**.

The cartridge selector dialog displays.



f) Select the **Hello** cartridge and click **OK**.

g) Select the new Hello cartridge from the Content Tree on the left and configure it as shown:



h) Click **Save Changes** in the upper right of the page.

5. Try to view the cartridge in the Discover Electronics application.

a) In a Web browser, navigate to <http://<workbench-host>:8006/discover-authoring/>.

Showing 1 - 10 of 5616 items [1](#) [2](#) [3](#) [4](#) [5](#) >

Category	Items
Bags & Cases	(665)
Cameras	(4951)

Price Range

- Under 25 (81)
- 25 - 50 (96)
- 50 - 100 (170)
- 100 - 250 (548)
- 250 - 500 (915)
- 500 - 1000 (1912)
- Over 1000 (1934)

Brand Name

- Kodak (285)
- Canon (2306)
- Case Logic (176)
- D-Link (14)
- Fujifilm (352)
- Logitech (175)
- Pentax (186)

The error displays because we have not yet created a renderer for the Hello cartridge.

- Scroll down to the bottom of the page and click the **json** link view the serialized Assembler response model that represents the current page.

Oracle recommends that you use a browser or install a plugin that supports native JSON display. Otherwise, you can download the JSON response as a file.

Alternatively, you can click the **xml** link to view the same response in XML. In this guide, we use the JSON format when examining the Assembler response.

The following shows the JSON representation of the page with most of the tree collapsed, highlighting the data for the cartridge that we just added.

```
{
```

```

  "@type": "ResultsPageSlot",
  "name": "Browse Page",
  "contentCollection": "Search And Navigation Pages",
  "ruleLimit": "1",
  "contents": [
    {
      "@type": "ThreeColumnNavigationPage",
      "name": "Default Page",
      "title": "Discover Electronics",
      "metaKeywords": "camera cameras electronics",
      "metaDescription": "Endeca eBusiness reference application.",
      "links": [ ],
      "header": [ ... ],
      "leftColumn": [ ... ],
      "main": [ ... ],
      "rightColumn": [
        { ... },

```

```

        {
            ...
        },
        {
            "@type": "Hello",
            "name": "Hello cartridge",
            "message": "Hello, World!",
            "messageColor": "#FF0000"
        }
    ],
    ...
}

```

In the next section, we'll create a simple renderer that displays the message based on the values configured in Experience Manager.

Adding a basic renderer

While there is no one way to write rendering code for an application, in this example we'll write a simple JSP renderer for our basic cartridge.

To write a basic "Hello, World" renderer:

1. Create a new JSP page and type or copy the following:

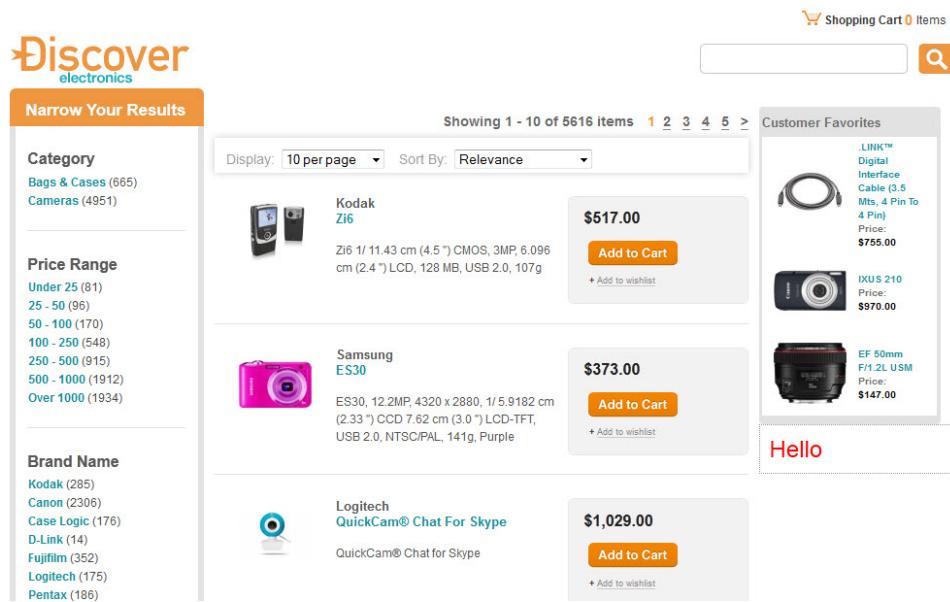
```

<%@page language="java" pageEncoding="UTF-8"
    contentType="text/html; charset=UTF-8"%>

<%@include file="/WEB-INF/views/include.jsp"%>
<div style="border-style: dotted; border-width: 1px;
    border-color: #999999; padding: 10px 10px">
    <div style="font-size: 150%;
        color: ${component.messageColor}">${component.message}</div>
</div>

```

2. Save the renderer to
discover-electronics-authoring/WEB-INF/views/desktop/Hello/Hello.jsp.
3. Refresh the Discover Electronics authoring application at
<http://<workbench-host>:8006/discover-authoring/> to see the result.



Deeper dive on the example cartridge

Now that we have created a basic example cartridge, let's examine each of the cartridge elements more closely.

As we have seen, the high-level workflow for creating a basic cartridge is:

1. Create a cartridge template and upload it to Endeca Workbench.
2. Use Experience Manager to create and configure and instance of the cartridge.
3. Add a renderer to the front-end application.

Step 2 is necessary during development in order to have a cartridge instance with which to test. However, once the cartridge is complete, the business user is typically responsible for creating and maintaining cartridge instances in Experience Manager.

In the following sections, we'll describe each of these elements of the cartridge in greater detail.

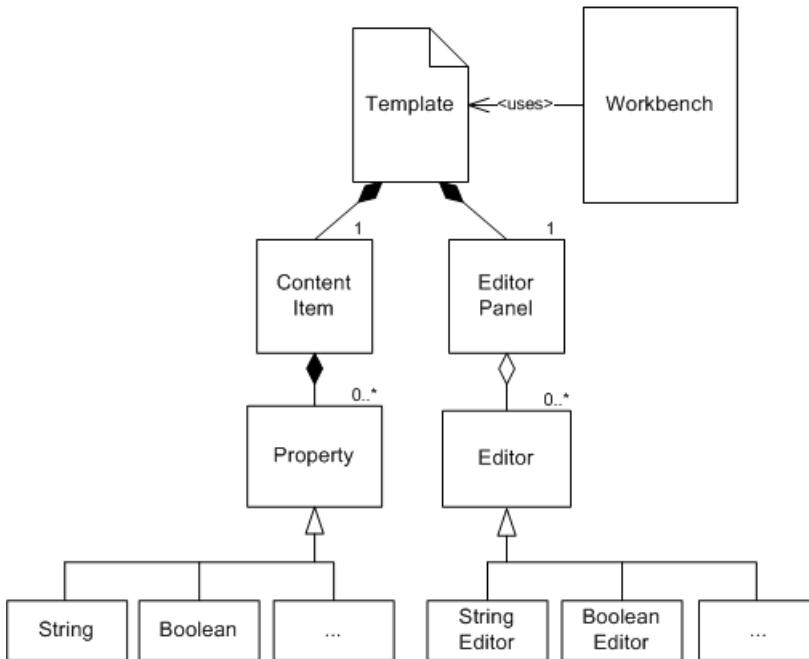
The cartridge template

The template defines the configuration that the business user can specify in Endeca Workbench using Experience Manager.

The template contains two main sections: the `<ContentItem>` element and the `<EditorPanel>` element.

The *content item* is a core concept in Assembler applications that can represent both the configuration model for a cartridge and the response model that the Assembler returns to the client application. A content item is a map of properties, or key-value pairs. The `<ContentItem>` element in the template defines the prototypical content item and its properties, similar to a class or type definition.

The `<EditorPanel>` defines the interface that can be used in Experience Manager to configure the properties of the content item. The *editor panel* is composed of a number of *editors*. The editors provide the UI controls that the business user can use to specify the property values for a particular instance of that cartridge.



In our example template, we defined two string properties named `message` and `messageColor` and attached two simple string editors to those properties. The result looks like this in Experience Manager:

Section Settings	
Section Type	SidebarContent
SidebarContent	Hello - Hello cartridge X Change
Name	Hello cartridge
Message	
Color	

For more information about creating and managing cartridge templates, see [About creating templates](#) on page 45.

The cartridge instance configuration

The business user creates and configures instances of cartridges in Experience Manager based on a template. During cartridge development you need to create at least one instance of a cartridge for testing.

Experience Manager writes this cartridge instance configuration as XML. You can view the XML representation of the configuration using the **XML View** tab in Experience Manager. The following shows the XML that corresponds to the configured instance of our example cartridge:



The screenshot shows the Content Editor interface with the 'Content XML' tab selected. The XML code in the main pane is as follows:

```

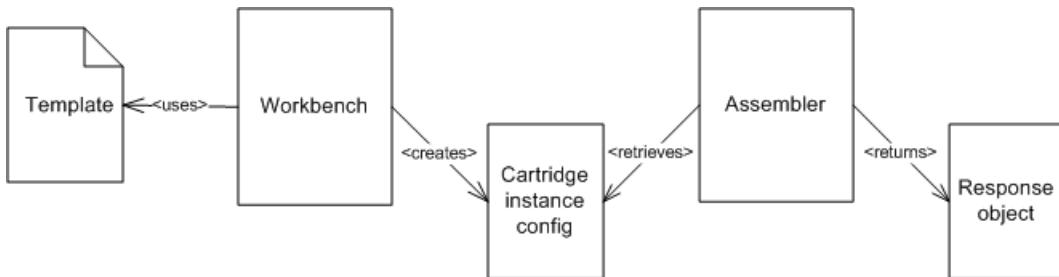
161 <ContentItem type="SecondaryContent">
162   <Name>Hello cartridge</Name>
163   <Property name="message">
164     <String>Hello</String>
165   </Property>
166   <Property name="messageColor">
167     <String>#FF0000</String>
168   </Property>
169 </ContentItem>

```

Below the XML pane are buttons for 'Find', 'Next Match', and 'Previous Match'. At the top of the editor are tabs for 'Content Editor' (selected), 'XML View', 'Content XML' (radio button), 'Rule XML' (radio button), 'Saved Copy' (radio button), and 'Working Copy' (radio button).

Note the similarities to the `<ContentItem>` portion of the template that we created. At this stage, the values of the string properties have been filled in based in the input in the **Content Editor** pane.

The Assembler retrieves this configuration at runtime and uses it to build the response model that it returns to the client application.



For any given cartridge, the default behavior is for the Assembler to do no processing on the configuration and simply return the configuration content item as a map of properties. That is, the response object is the same as the configuration object unless specific processing logic is defined in the Assembler for that cartridge.

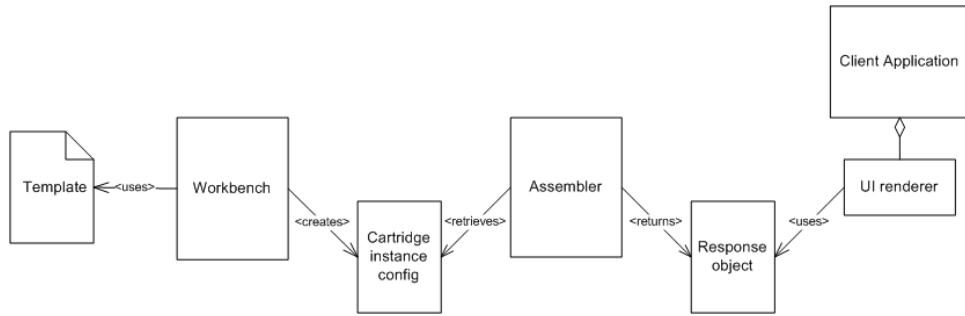
The cartridge renderer

As a best practice, the client application should be composed of modular rendering components, each corresponding to a particular cartridge.

Recall the contents of the Assembler response object corresponding to the example cartridge:

```
{
  "@type": "Hello",
  "name": "Hello cartridge",
  "message": "Hello, World!",
  "messageColor": "#FF0000"
}
```

For each cartridge, the `@type` of the response object corresponds to the `id` of the template that was used to create it. The Discover Electronics application uses this type to identify the appropriate renderer to use for this content item.

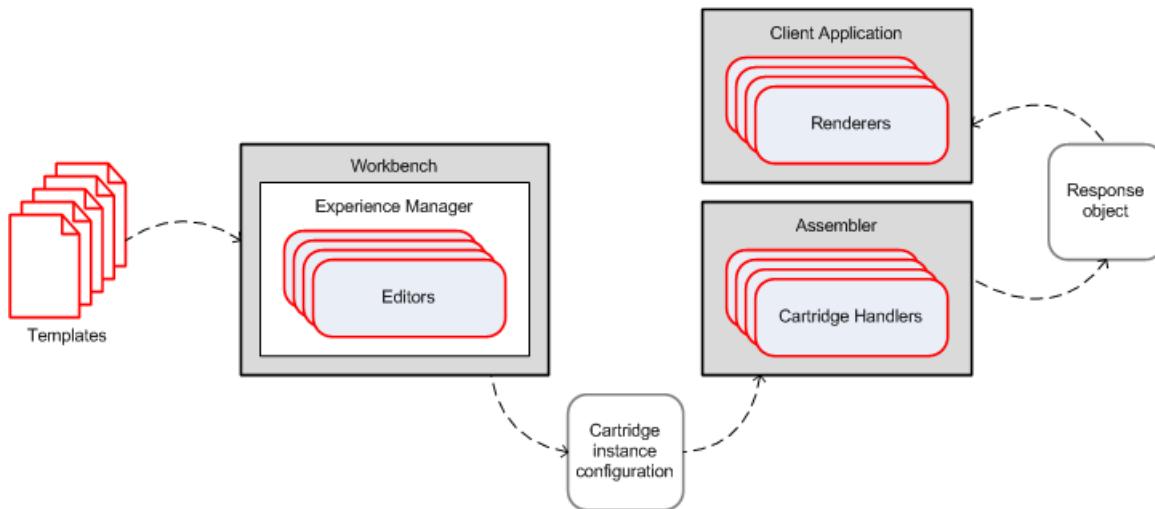


The logic for mapping response objects to the appropriate renderer is contained in `include.tag` in the reference application.

Overview of cartridge extension points

Cartridges are made up of several components that may be customized for specific purposes.

The following diagram shows the parts of a cartridge and where they fit within the overall architecture:



The cartridge *template* defines the configuration options that are available to the business user in Workbench. The Experience Manager interface is composed of *editors*, or Flex components that provide UI controls for specifying property values. Experience Manager produces the *cartridge instance configuration* that is consumed by the Assembler. During the processing of a query, the Assembler may invoke *cartridge handlers* that define specific processing logic for particular cartridges. Using these cartridge handlers, the Assembler produces the *response object* that it returns to the client application. Typically, the client application includes modular *renderers* that are intended to handle a particular cartridge.

We created a basic template and renderer in the example cartridge. We also inspected the cartridge instance configuration generated in Workbench and the response returned by the Assembler. In the example cartridge, both the configuration and the response model were generic content items that are simple maps of properties. Many of the core Endeca cartridges have strongly typed configuration models and response objects associated with them that extend from the basic content item. This makes it easier to understand the expected input to and output from the core cartridge handlers, and also enables reuse of the models for the core cartridges. Strongly typed configuration beans also make it possible to configure default values for cartridge properties.

via Spring. Creating strongly typed model objects for the Assembler configuration and response is not required when developing cartridges.

In the following sections, we discuss how to customize the Experience Manager interface using editors, and how to define custom processing logic in the Assembler using cartridge handlers.

Customizing the Experience Manager interface

Experience Manager provides a set of standard editors that you can use in cartridge templates as well as the ability to develop custom editors.

Adding embedded user assistance to a cartridge

You can provide embedded assistance for the business user in the Experience Manager interface by specifying it in the cartridge template.

In our example cartridge, we provided two simple text fields for the business user to enter a message and the desired color. This user interface makes it unclear what values are allowed or expected for those fields. The template schema for configuring editors allows you to supply a short descriptive label for each field, but sometimes additional context can be helpful. For such cases, you can use the `bottomLabel` attribute to provide further information.

To add additional guidance for the business user to the example cartridge:

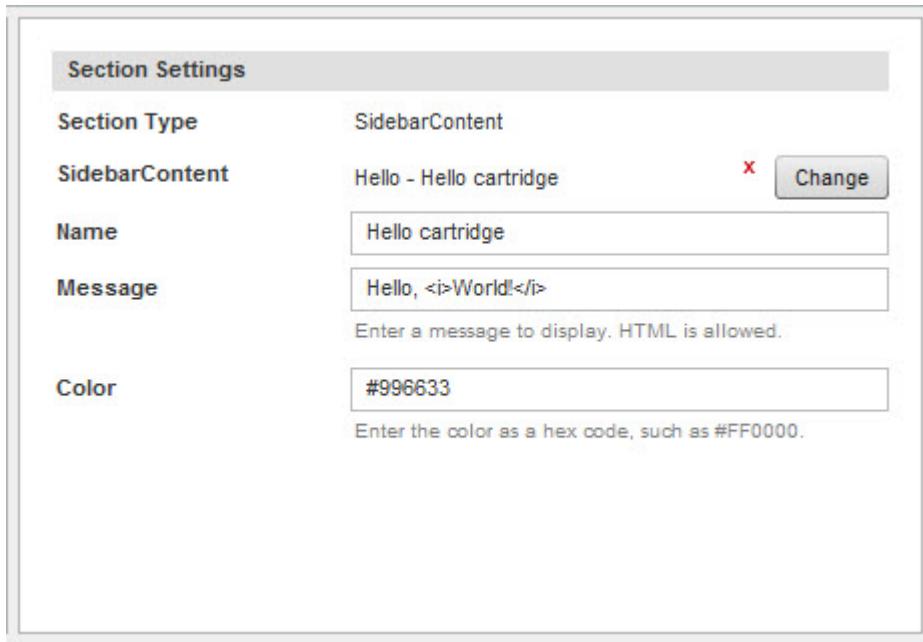
1. Open the template file (`HelloWorld\template.xml`) that you previously created.
2. Add a `bottomLabel` attribute to each editor in the `<EditorPanel>`, as in the example below:

```
<EditorPanel>
  <BasicContentItemEditor>
    <editors:StringEditor propertyName="message" label="Message"
      bottomLabel="Enter a message to display. HTML is allowed." />
    <editors:StringEditor propertyName="messageColor"
      label="Color" bottomLabel="Enter the color as a hex code, such
      as #FF0000." />
  </BasicContentItemEditor>
```

This additional label text can be configured for all editors built using the Experience Manager SDK, including all the standard editors. For the full content of the updated template, see the example below. If your implementation uses multiple locales, see [About multiple locales](#) on page 55 for information about localizing strings.

3. Save and close the template.
4. Upload the template by running the `set_templates` script.

The resulting user interface in Experience Manager looks like the following:



The following shows the complete content of the updated template:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:editors="editors"
                  type="SecondaryContent">
  <Description>A sample cartridge that can display a simple
  message.</Description>
  <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/sidebar_content.jpg</ThumbnailUrl>
  <ContentItem>
    <Name>Hello cartridge</Name>
    <Property name="message">
      <String/>
    </Property>
    <Property name="messageColor">
      <String/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <editors:StringEditor propertyName="message" label="Message"
        bottomLabel="Enter a message to display. HTML is allowed."/>
      <editors:StringEditor propertyName="messageColor"
        label="Color" bottomLabel="Enter the color as a hex code, such as
        #FF0000."/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

For more information about label options for Experience Manager editors, see the [Editor label configuration reference](#) on page 224.

Using the core Experience Manager editors

Experience Manager provides a set of editors that can configure primitive property types as well as Endeca-specific features. You specify which editor to use to configure which properties in the `<EditorPanel>` portion of the template.

Even with additional user assistance text, asking the business user to type a hex code into a text field does not provide a very user-friendly experience. One of the standard editors included with Experience Manager is a combo box that can be used to specify a set of valid values for a string property. In this example, we provide a set of colors from which the business user can choose. This not only relieves the business user from typing in a hex code, but it can also ensure that the selected color matches the site's color scheme.

To update the example cartridge to use a combo box editor:

1. Open the template file, `HelloWorld\template.xml`, that you previously created.
2. Replace the string editor configuration for the `messageColor` property with the following:

```
<EditorPanel>
  <BasicContentItemEditor>
    <editors:StringEditor propertyName="message" label="Message"
      bottomLabel="Enter a message to display. HTML is allowed." />
    <editors:ChoiceEditor propertyName="messageColor" label="Color">
      <choice label="Red" value="#FF0000"/>
      <choice label="Green" value="#00FF00"/>
      <choice label="Blue" value="#0000FF"/>
    </editors:ChoiceEditor>
  </BasicContentItemEditor>
</EditorPanel>
```

For the full content of the updated template, see the example below.

3. Upload the template by running the `set_templates` script.

The resulting user interface in Experience Manager looks like the following:

Section Settings	
Section Type	SidebarContent
SidebarContent	Hello - Hello cartridge X Change
Name	Hello cartridge
Message	Hello, <i>World!</i>
Enter a message to display. HTML is allowed.	
Color	Red ▼

Depending on the option that the business user selects, the value of the property is set to the appropriate hex code. You can change the value and refresh the application to see the change.

The following shows the complete content of the updated template:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:editors="editors"
                  type="SecondaryContent">
  <Description>A sample cartridge that can display a simple
  message.</Description>
  <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/sidebar_content.jpg</ThumbnailUrl>
  <ContentItem>
    <Name>Hello cartridge</Name>
    <Property name="message">
      <String/>
    </Property>
    <Property name="messageColor">
      <String/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <editors:StringEditor propertyName="message" label="Message"
        bottomLabel="Enter a message to display. HTML is allowed."/>
      <editors:ChoiceEditor propertyName="messageColor" label="Color">
        <choice label="Red" value="#FF0000"/>
        <choice label="Green" value="#00FF00"/>
        <choice label="Blue" value="#0000FF"/>
      </editors:ChoiceEditor>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

For more information about the standard Experience Manager editors and their configuration, refer to the [Template Property and Editor Reference](#) on page 221.

About custom editors

If none of the standard editors meet your needs, you can develop your own editors using the Experience Manager Editor SDK.

You may want to develop an editor if:

- You want to allow the business user to configure more advanced properties such as lists or maps of key-value pairs.
- You want to provide a more advanced interface for the business user, such as a list that enables drag-and-drop.
- You want the editor options to be populated dynamically from an external system rather than configured in the template.
- You want the behavior of one editor or UI control to be linked to the state of another.

For more information about the Experience Manager Editor SDK and developing Experience Manager editors, see [Extending and Developing Cartridges](#) on page 159.

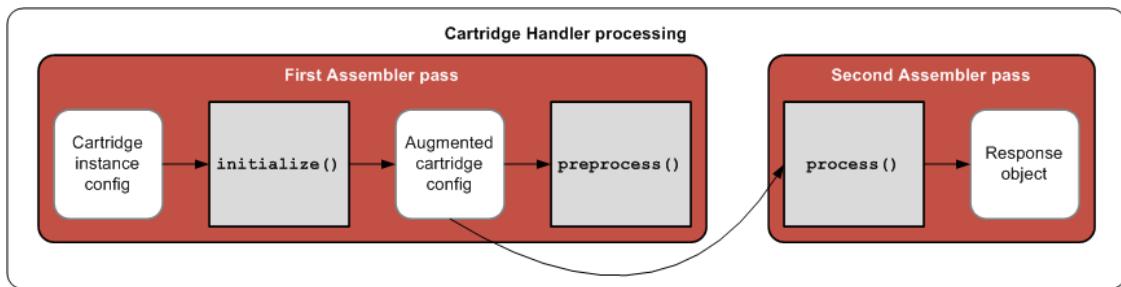
About Cartridge Handlers and the Assembler

This section provides an overview of the Assembler. It describes the Assembler processing model and core interfaces as well as how to implement a cartridge handler.

About the CartridgeHandler interface

A cartridge handler takes a content item representing the cartridge instance configuration as input and is responsible for returning the response as a content item.

The `CartridgeHandler` interface defines three methods: `initialize()`, `preprocess()`, and `process()`.



The `initialize()` method provides an opportunity for the cartridge handler to augment the cartridge instance configuration specified in Experience Manager with configuration from other sources. This can be used to define default behavior for a cartridge in the case where there is no Experience Manager configuration, or to override the Experience Manager configuration for the current query. The `initialize()` method should return a content item containing the complete configuration for the cartridge from all possible configuration sources. This augmented configuration item can either be the mutated input content item or a new instance of `ContentItem`, and is used as input to both the `preprocess()` and `process()` methods.

Because the `preprocess()` method is called on all cartridges before `process()` is called on any cartridges, it provides an opportunity to coordinate processing between cartridges. Many of the core Endeca cartridges make use of this mechanism in order to consolidate queries to an MDEX Engine among several cartridges during the course of a single assembly cycle.

The `process()` method is responsible for returning a `ContentItem` that represents the cartridge response.

A cartridge handler need not define any behavior for `initialize()` or `preprocess()`. The `AbstractCartridgeHandler` class exists to simplify the task of implementing the `CartridgeHandler` interface. It provides empty implementations for `initialize()` and `preprocess()`. Subclasses of `AbstractCartridgeHandler` need only implement the `process()` method to return the response object. They can optionally override the `initialize()` and `preprocess()` methods.

About initializing the cartridge configuration

The `initialize()` phase in the cartridge processing life cycle enables the cartridge handler to synthesize the complete configuration for the cartridge from several sources.

The configuration content item that is passed in to the assembly process is the cartridge instance configuration from Experience Manager, however, any given cartridge may also have other configuration sources.

In a typical scenario, a cartridge has some default behavior that can be specified as a property value in a Spring context file. A business user can specify a value for a specific instance of a cartridge using Experience Manager. The site visitor may also have the ability to override either the default or the cartridge instance setting from the client application. For example, in the Results List cartridge, the default value for records per page is

10. The business user can set this value to 25 in Experience Manager, and the site visitor can choose to display 50 records by selecting the appropriate option on the site.

The Assembler API includes the `ConfigInitializer` utility class with the method `initialize()`. The default implementation of `initialize()` layers the cartridge configuration in the following order (from lowest to highest):

1. **Default configuration**, typically defined in the Spring configuration for the cartridge handler
2. **Cartridge instance configuration**, typically created in Experience Manager and passed in as the configuration content item
3. **Request-based configuration** parsed from the HTTP request parameters, using the `RequestParamMapper` helper class

The `ConfigInitializer` class also provides methods for additional layering of configuration. Subclasses can override `ConfigInitializer` to define custom layering behavior, for example, to incorporate configuration saved in the session state.

About the `NavigationCartridgeHandler` class

The core Endeca cartridges that make queries to an MDEX Engine use cartridge handlers that extend from `NavigationCartridgeHandler`.

The `NavigationCartridgeHandler` makes use of the two-pass Assembler processing model to consolidate MDEX Engine queries across cartridges.

In the `preprocess()` phase, the cartridge handler calls `createMdexRequest()` but does not execute the request. In subsequent calls to `createMdexRequest()` by other handlers, the MDEX resource broker determines whether the new request can be consolidated with an existing request in order to minimize the number of queries to the MDEX Engine for a single assembly cycle.

During the `process()` phase, the handler calls `executeMdexRequest()` to retrieve the results. The actual query to the MDEX Engine is executed when the first handler in the assembly cycle calls `executeMdexRequest()` and the results are cached for all subsequent handlers that try to execute the same request.

You can use a similar approach if you have multiple cartridges that need to make requests to the same external resource and can achieve efficiencies by consolidating requests across cartridges.

For further information about the `NavigationCartridgeHandler` class, refer to the *Assembler API Reference (Javadoc)*.

Implementing a cartridge handler

You add a cartridge handler by writing a Java class that implements the `CartridgeHandler` interface and configuring the Assembler to use the new handler in the Spring context file.

In this example, we update our "Hello, World" cartridge to do some simple string manipulation on the message that was specified in Experience Manager. Because this cartridge does not use any configuration other than the cartridge instance configuration from Experience Manager and does not need to do any preprocessing, we can extend `AbstractCartridgeHandler`.

To add a cartridge handler to the example cartridge:

1. Create a new Java class in the package `com.endeca.sample.cartridges` and type or copy the following:

```
package com.endeca.sample.cartridges;

import com.endeca.infront.assembler.AbstractCartridgeHandler;
import com.endeca.infront.assembler.CartridgeHandlerException;
```

```

import com.endeca.infront.assembler.ContentItem;

public class UppercaseCartridgeHandler extends AbstractCartridgeHandler
{
    //=====
    // The cartridge handler 'process' method
    public ContentItem process(ContentItem pContentItem) throws CartridgeHandlerException
    {
        // Get the message property off of the content item.
        final String message = (String) pContentItem.get("message");
        // If the message is non-null, uppercase it.
        if (null != message) {
            pContentItem.put("message", message.toUpperCase());
        }
        return pContentItem;
    }
}

```

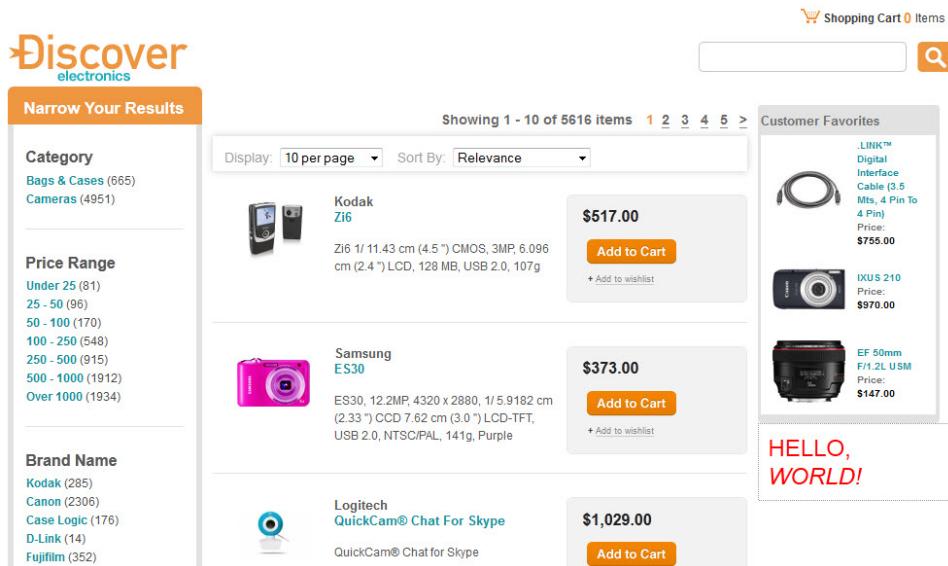
2. Compile the cartridge handler and add the compiled class to your application, for example, by saving it in %ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring\WEB-INF\classes.
3. Configure the Assembler to use the UppercaseCartridgeHandler for the Hello cartridge.
 - a) Navigate to the WEB-INF directory of your application, for example, %ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring\WEB-INF.
 - b) Open the assembler-context.xml file.
 - c) Add the following in the CARTRIDGE HANDLERS section:

```

<!--
~~~~~
~ BEAN: CartridgeHandler_Hello
-->
<bean id="CartridgeHandler_Hello"
      class="com.endeca.sample.cartridges.UppercaseCartridgeHandler"
      scope="prototype" />

```
- d) Save and close the file.
4. Restart the Endeca Tools Service.
5. Refresh the authoring instance of the application.

The message now displays in all-uppercase letters.



Cartridge handler development scenarios

You should write a cartridge handler in cases where you need to perform some processing on the cartridge instance configuration before sending the response to the client application.

It is always possible to do processing in the client application, but encapsulating the business logic in an extension to the Assembler provides several advantages:

- It makes the rendering code cleaner and easier to maintain.
- It centralizes the processing in one place so that the results can be consumed by multiple client applications including across multiple channels such as desktop, mobile, and others.
- It provides an opportunity for coordinating processing across multiple cartridges before returning the response to the client application.

Depending on what the cartridge handler needs to accomplish, your implementation approach may vary. Cartridge handlers must always implement the `process()` method to return the response model.

Scenario	Implementation approach	Example cartridge
Update properties from the cartridge instance configuration in place (data cleansing or manipulation scenario)	Extend <code>AbstractCartridgeHandler</code> and override <code>process()</code> to update the property values in the input content item	"Hello, World" with <code>UppercaseCartridgeHandler</code>
Use information from the cartridge instance configuration to query an external resource for the information to display	Extend <code>AbstractCartridgeHandler</code> and override <code>process()</code> to query the resource and insert the results in the output content item	RSS Feed cartridge
Query an external resource, consolidating queries between cartridges within a single assembly cycle for improved performance	Take advantage of the two-pass assembly model with <code>preprocess()</code> and <code>process()</code> and implement a resource broker that can consolidate queries and manage their execution	<code>NavigationCartridgeHandler</code>

Scenario	Implementation approach	Example cartridge
Augment the results from a core Endeca cartridge with additional information from a non-MDEX resource	Extend the core cartridge and override <code>process()</code> to query the resource and add additional properties to the MDEX query results before returning the response	Custom Record Details with availability information
Customize a core Endeca cartridge to modify the MDEX Engine query parameters	Extend the core cartridge and override either <code>initialize()</code> or <code>preprocess()</code> to modify the query before it is executed	Custom Results List with recommendations
Combine multiple sources of cartridge configuration before processing results	Extend <code>AbstractCartridgeHandler</code> or implement the <code>CartridgeHandler</code> interface and override <code>initialize()</code> , making use of the <code>ConfigInitializer</code> and <code>RequestParamMarshaller</code> helper classes to generate the complete configuration model	"Hello, World" with layered color configuration

About using event listeners to extend the navigation cartridges

You can use the Assembler eventing framework as an extension point for navigation cartridges in cases where extending an existing cartridge handler is insufficient.

If you are making modifications to the navigation cartridges, you can trigger processing logic based on Assembler events instead of subclassing the core cartridge handlers.

Using an event listener instead of extending a cartridge handler introduces the following considerations:

- Unlike extending a cartridge handler, logic included in an event listener is evaluated for every cartridge handler.
- Event listeners do not have access to the current Assembler request or to the navigation state.
- Event listeners must be thread safe.

Related Links

[Assembler event framework reference](#) on page 25

The Assembler includes an `AssemblerEventListener` interface that you can use to create and register event listeners.

Creating an event listener

The Assembler provides an empty implementation of the `AssemblerEventListener`, `AssemblerEventAdapter`. You can extend this implementation to create a listener that triggers on an Assembler event.

To create an event listener:

1. Create a new Java class that extends the `AssemblerEventAdapter`.

For example:

```
public class ResultsListListener extends AssemblerEventAdapter { }
```

2. Override the methods that correspond to the events for which you wish to trigger custom processing logic:

```
public class ResultsListListener extends AssemblerEventAdapter {
    @Override
    public void cartridgePreprocessStarting(AssemblerEvent event){
        ...
    }

    @Override
    public void cartridgeProcessComplete(AssemblerEvent event){
        ...
    }
}
```

For a list of Assembler events, see the [Assembler event framework reference](#) on page 25 or refer to the [Assembler API Reference \(Javadoc\)](#).

3. Add conditional logic to restrict processing to a specific cartridge handler:

```
public class ResultsListListener extends AssemblerEventAdapter {
    ...

    @Override
    public void cartridgeProcessComplete(AssemblerEvent event){
        if(event.getContentItem() != null && "ResultsList".equals(event.getContentItem().getType())){
            ...
        }
    }
}
```

4. Add processing logic.

The example below prefixes the `max_price` property on a record with a dollar sign:

```
public class ResultsListListener extends AssemblerEventAdapter {
    ...

    @Override
    public void cartridgeProcessComplete(AssemblerEvent event){
        if(event.getContentItem() != null && "ResultsList".equals(event.getContentItem().getType())){
            ResultsList resultsList = (ResultsList) event.getContentItem();
            for(Record record : resultsList.getRecords()){
                Attribute price = record.getAttributes().get("product.max_price");
                if(price != null){
                    for(int i = 0 ; i < price.size(); i++){
                        price.set(i, "$" + price.get(i).toString());
                    }
                }
            }
        }
    }
}
```

After creating a new listener, you must register it by including it in the list of listeners for the `AssemblerFactory` object.

About registering an event listener

You must register all event listeners with the `AssemblerFactory` object.

The `AssemblerFactory` takes event listeners as constructor arguments. These listeners are instantiated with each `Assembler` object created by the factory class.

Optionally, you may also choose to use the `Assembler.addAssemblerEventListener()` method to add a listener for a single assembly request.

Example

The example below uses the `ResultsListListener` defined in the previous topic, registered in the Discover Electronics reference application.

The reference application uses the `Assembler` context file to configure global application properties. The configuration bean for the `AssemblerFactory` includes a list of listeners as constructor arguments:

```
<bean id="AssemblerFactory" class="com.endeca.infrontAssembler.spring.SpringAssemblerFactory"
    scope="singleton">
    <constructor-arg>
        ...
    </constructor-arg>
    <constructor-arg>
        <!-- List of listeners registered in the assembler -->
        <list>
            <bean class="com.endeca.infront.ResultsListListener" />
            <bean class="com.endeca.infront.logger.SLF4JAssemblerEventLogger" />
            ...
            <bean class="com.endeca.infrontAssembler.event.request.ContentItemAugmentAdapter">
                <constructor-arg ref="springUtility" />
            </bean>
            ...
        </list>
    </constructor-arg>
    ...
</bean>
```

Sample Cartridges

This section contains sample cartridge customizations that demonstrate how to use the various cartridge extension mechanisms to address different use cases.

About using the sample cartridges

The sample cartridges are intended to demonstrate the cartridge extension mechanisms and provide a model for your own cartridge customizations.

The sample code provided is written to be generic and easy to follow, rather than production-quality code. Oracle recommends that you follow a few best practices when working with the examples:

- Set up a new instance of the Discover Electronics application to use as a sandbox for deploying the sample cartridges. This isolates the samples from the out-of-the-box configuration for Discover Electronics as well as your own application.

- Within your sandbox application, create a separate Spring context file for the custom cartridge handlers described in this guide.
- When copying and pasting examples from this guide, pay attention to the end-of-line marker (~) that indicates that a long line of text has been wrapped. Ensure that any occurrences of the symbol and the corresponding line break are deleted and any remaining space is closed up.

The steps described for creating and deploying the components of the sample cartridges correspond to the steps described in previous sections for the "Hello, World" cartridge. If you need additional information to complete a particular step in deploying one of the sample cartridges, refer to the more detailed procedures for the "Hello, World" example.

Setting up a test application based on Discover Electronics

Oracle recommends that you use a test application to test the sample cartridges instead of deploying them in Discover Electronics or your own application.

Because a test application is for development use only, we do not need to deploy a live instance of the application.

To deploy a copy of Discover Electronics to use as a test for the sample cartridges:

1. Deploy a new test application using the Deployment Template.
 - a) From a command prompt, navigate to %ENDECA_TOOLS_ROOT%\deployment_template\bin (on Windows) or \$ENDECA_TOOLS_ROOT/deployment_template/bin (on UNIX).
 - b) Run the deploy script:
 - On Windows: deploy.bat --app ...\\reference\\discover-data\\deploy.xml
 - On UNIX: deploy.sh --app .../reference/discover-data/deploy.xml
 - c) Specify the application name Test and specify the following ports when prompted:

Port	Recommended value
Live Dgraph	15100
Authoring Dgraph	15102
LogServer	15110

2. Provision the test application.
 - a) Ensure that the Endeca HTTP Service and Endeca Tools Service are running.
 - b) From a command prompt, navigate to <APP-DIR>\control (on Windows) or <APP-DIR>/control (on UNIX).
 - c) Run initialize_services.
 - d) Run load_baseline_test_data.
 - e) Run baseline_update.
3. Deploy a copy of the authoring instance of the Discover Electronics application.
 - a) Navigate to %ENDECA_TOOLS_ROOT%\reference (on Windows) or \$ENDECA_TOOLS_ROOT/reference (on UNIX).
 - b) Make a copy of the directory discover-electronics-authoring and save the copy with the name sandbox in the same parent directory.
 - c) Navigate to the test directory and then to the WEB-INF subdirectory.
 - d) Open assembler-context.xml in a text editor.

e) Locate the CARTRIDGE SUPPORT section:

```
<!--
#####
# CARTRIDGE SUPPORT
#
# The following section configures managers and other supporting objects.
#
-->
```

f) In the mdexResource bean, update the Dgraph port:

```
<bean id="mdexResource" scope="request"
class="com.endeca.infront.navigation.model.MdexResource">
    <property name="host" value="localhost" />
    <property name="port" value="15102" />
    <property name="recordSpecName" value="common.id" />
</bean>
```

g) Locate the Content Sources section:

```
<!--
~~~~~
~ Content Sources
-->
```

h) In the authoringContentSource bean, update the application name:

```
<bean id="authoringContentSource" class="com.endeca.infront.content.source.AuthoringContentSource"
scope="singleton" lazy-init="true">
    <property name="sitePath" value="/sites/Test" />
    <property name="rootUrl" value="/ifcr" />
    <property name="host" value="localhost" />
    <property name="port" value="8006" />
    <property name="serviceUrl" value="/ifcr/system/endeca/contentResolver" />

    <property name="user" value="admin" />
    <property name="password" value="admin" />
</bean>
```

i) In the authoringMediaSources bean, update the application name:

```
<bean id="authoringMediaSources" class="java.util.ArrayList" lazy-init="true">

    <constructor-arg>
        <list>
            <bean class="com.endeca.infront.cartridge.model.MediaSourceConfig">
                <property name="sourceName" value="IFCRSource" />
                <property name="sourceValue" value="http://localhost:8006/ifcr/sites/Test/media/" />
            </bean>
            <bean class="com.endeca.infront.cartridge.model.MediaSourceConfig">
                <property name="sourceName" value="default" />
                <property name="sourceValue" value="http://localhost:8006/ifcr/sites/Test/media/" />
            </bean>
        </list>
    </constructor-arg>
</bean>
```

```

</constructor-arg>
</bean>

```

- j) Save and close the file.
- k) Navigate to %ENDECA_TOOLS_CONF%\conf\Standalone\localhost (on Windows) or \$ENDECA_TOOLS_CONF/conf/Standalone/localhost (on UNIX).
- l) Make a copy of discover-authoring.xml and save the copy with the name test in the same directory.
- m) Open test.xml in a text editor.
- n) Change the value of docBase as follows:

```
docBase="${catalina.base}/../../reference/test"
```

- o) Restart the Endeca Tools Service.

4. Validate your new sandbox application:

- a) Navigate to `http://<WorkbenchHost>:8006/login` and verify that Test displays as an option in the **Application** drop-down.
- b) Select the Test application and verify that the sample page content from Discover Electronics is available in Experience Manager.
- c) In a separate browser window, navigate to the newly deployed sandbox application, at `http://<WorkbenchHost>:8006/test` and verify that it displays.

5. Optionally, update the Workbench configuration to use the test Web application for preview.

- a) Ensure that you are logged in to the Test application in Workbench.
- b) Select **Application Configuration**.
- c) Specify the URL to the sandbox application (for example, `http://<WorkbenchHost>:8006/test`) as the **Preview URL**.
- d) Preview a page from Experience Manager by selecting a page or content item and clicking **Preview** in the upper right.

Creating a Spring context file for sample cartridges

Oracle recommends that you specify the configuration for the sample cartridges in a separate Spring context file from the core Endeca cartridges.

To create a Spring context file for the sample cartridges:

1. Navigate to %ENDECA_TOOLS_ROOT%\reference\sandbox\WEB-INF (on Windows) or \$ENDECA_TOOLS_ROOT/reference/sandbox/WEB-INF (on UNIX).
2. Open assembler-context.xml in a text editor.
3. At the top of the file, add the following import:

```

<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xsi:schemaLocation="
           http://www.springframework.org/schema/beans
           http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
       ">
    <bean class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
        <property name="locations">
            <list>
                <value>WEB-INF/assembler.properties</value>
            </list>
        </property>
    </bean>

```

```

</bean>

<import resource="endeca-url-config.xml"/>
<import resource="perf-logging-config.xml"/>
<import resource="sample-cartridge-config.xml" />

```

4. Delete the configuration for the "Hello, World" sample cartridge that we added in an earlier example.

```

<!--
~~~~~
~ BEAN: CartridgeHandler_Hello
-->
<bean id="CartridgeHandler_Hello"
class="com.endeca.sample.cartridges.UppercaseCartridgeHandler"
scope="prototype" />

```

5. Save and close the file.

6. Create a new file named `sample-cartridge-config.xml` in the same directory with the following contents:

```

<beans xmlns="http://www.springframework.org/schema/beans"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.springframework.org/schema/beans
  http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

  <!--
  ~~~~~
  ~ BEAN: CartridgeHandler_Hello
  -->
  <bean id="CartridgeHandler_Hello"
  class="com.endeca.sample.cartridges.UppercaseCartridgeHandler"
  scope="prototype" />

</beans>

```

7. Save and close the file.

8. Validate the new configuration by adding the "Hello, World" cartridge to your new sandbox application.

- Copy the "Hello, World" directory and its contents (`HelloWorld\`) from the Discover Electronics application (`<APP-DIR>\config\cartridge_templates`) to the sandbox application.
- Upload the template to Workbench using the `set_templates` script.
- Using Experience Manager, add the cartridge to the default page of the sandbox application and save your changes.
- Verify that the `Hello.jsp` renderer and `UppercaseCartridgeHandler` are present in the sandbox Web application. (These should have been included when you copied the Discover Electronics authoring application.)
- Refresh the sandbox application (`http://<WorkbenchHost>:8006/sandbox`) and verify that the text you entered in Experience Manager displays, and has been converted to all-uppercase letters.

RSS Feed cartridge

In this example, we build a cartridge that displays items from an RSS feed.

This cartridge enables a business user to specify some basic information about an existing RSS feed in Experience Manager. The cartridge handler fetches the RSS results and returns an output model to the client suitable for rendering.

It demonstrates the following use cases:

- Using a cartridge handler to fetch information from a source other than an MDEX Engine.
- Using the business user configuration from Experience Manager as input into the assembly process and returning a different output model from the configuration model.

In this cartridge, we create the following components:

Component	Description
cartridge template	Enables the business user to specify the URL to an RSS feed and the number of entries to display.
cartridge handler	Fetches results from the RSS feed and returns a number of entries up to the value specified by the business user or the number of entries in the feed, whichever is lower.
cartridge renderer	Displays the name of the feed with a link to the channel URL, and the title and description of each entry with a link to the entry on the original site.

Creating the cartridge template

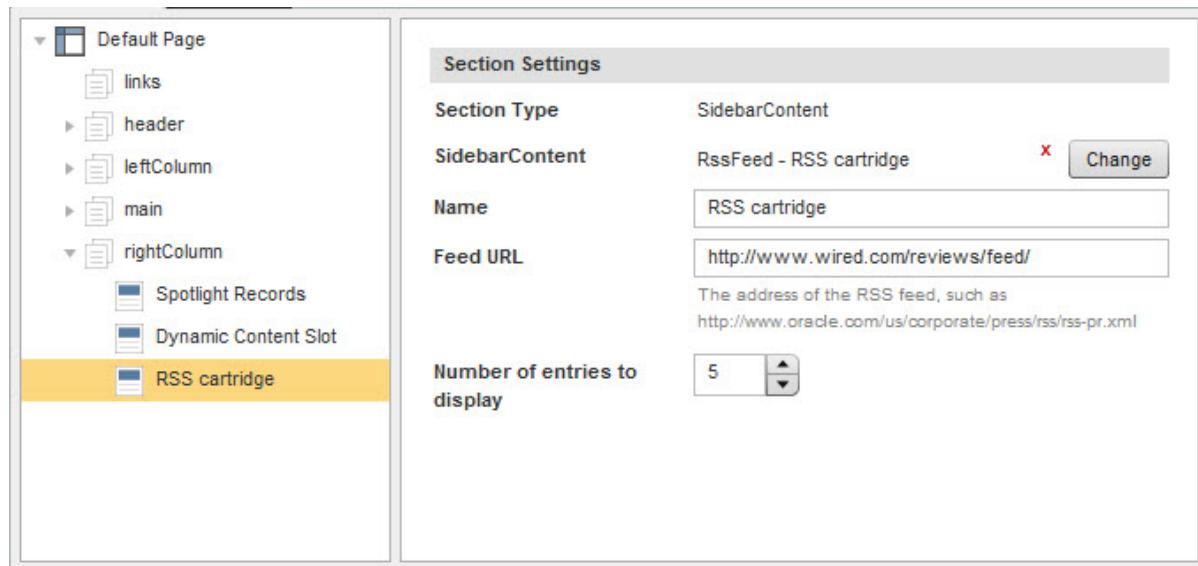
The business user needs to be able to configure the RSS Feed with a URL and the number of entries to display.

To create the RSS Feed template and add it to your application:

1. Create a new template based on the example below.

Since the number of entries is expected to be an integer, the example uses a `NumericStepperEditor` for this property. It could also use a `SliderEditor` — both options guarantee that the value of the string property is an integer. In the example, we specify a default value of 5 for the number of entries.

2. Create a directory with the name `RssFeed` in the templates directory of your application.
3. Save the template with the name `template.xml` to the `RssFeed` directory of your application.
4. Upload the template using the `set_templates` script.
5. Add the cartridge to the default search and navigation page as in the example below.



Note: The sample renderer for this cartridge works best with RSS feeds that have brief descriptions with no images or advertisements in the description field. A possible enhancement to this cartridge would be to make displaying the description configurable.

6. Save your changes to the page.

The cartridge instance configuration is saved as XML. At this point, because there is no cartridge handler specified for this cartridge, the same configuration is passed to the client as the response from the Assembler.

```
<ContentItem type="SecondaryContent">
  <Name>RSS cartridge</Name>
  <Property name="feedUrl">
    <String>http://www.wired.com/reviews/feed/</String>
  </Property>
  <Property name="numEntries">
    <String>5</String>
  </Property>
</ContentItem>
```

The following shows the sample template for the RSS Feed cartridge:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:editors="editors"
                  type="SecondaryContent">
  <Description>A cartridge that displays entries from an RSS feed.</Description>
  <ThumbnailUrl>ifcr/tools/xmgr/img/template_thumbnails/sidebar_content.jpg</ThumbnailUrl>
  <ContentItem>
    <Name>RSS cartridge</Name>
    <Property name="feedUrl">
      <String/>
    </Property>
    <Property name="numEntries">
      <String>5</String>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <editors:StringEditor propertyName="feedUrl" label="Feed URL"
        bottomLabel="The address of the RSS feed, such as http://www.oracle.com/us/corporate/press/rss/rss-pr.xml"/>
      <editors:NumericStepperEditor propertyName="numEntries"
        label="Number of entries to display" minValue="1" maxValue="15"/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Creating the cartridge handler

The cartridge handler fetches the RSS results and returns an output model to the client suitable for rendering.

To create the RSS Feed cartridge handler and add it to the application:

1. Create a new Java class in the package `com.endeca.sample.cartridges` based on the example below, which extends `AbstractCartridgeHandler`.
2. Compile the cartridge handler and add the compiled class to your application.
3. Configure the Assembler to use the `RssFeedHandler` for the RSS Feed cartridge by adding the following to the Spring context file:

```
<!--
~~~~~
~ BEAN: CartridgeHandler_RssFeed
```

```
-->
<bean id="CartridgeHandler_RssFeed"
  class="com.endeca.sample.cartridges.RssFeedHandler"
  scope="prototype" />
```

4. Restart the Endeca Tools Service.

5. Refresh the application.

The RSS feed does not display yet because we have not created the renderer, but you can validate that the response model has been populated with the information that we want to display via the JSON view:

```
{
  "@type": "RssFeed",
  "name": "RSS cartridge",
  "feedUrl": "http://www.wired.com/reviews/feed/",
  "numEntries": "5",
  "chanTitle": "Product Reviews",
  "chanUrl": "http://www.wired.com/reviews",
  "entries": [
    {
      "@type": "rssEntry",
      "itemDesc": "(description text omitted from this example)",
      "itemTitle": "(title text omitted from this example)",
      "itemUrl": "(url omitted from this example)"
    },
    {
      "@type": "rssEntry",
      "itemDesc": "(description text omitted from this example)",
      "itemTitle": "(title text omitted from this example)",
      "itemUrl": "(url omitted from this example)"
    },
    {
      "@type": "rssEntry",
      "itemDesc": "(description text omitted from this example)",
      "itemTitle": "(title text omitted from this example)",
      "itemUrl": "(url omitted from this example)"
    },
    {
      "@type": "rssEntry",
      "itemDesc": "(description text omitted from this example)",
      "itemTitle": "(title text omitted from this example)",
      "itemUrl": "(url omitted from this example)"
    },
    {
      "@type": "rssEntry",
      "itemDesc": "(description text omitted from this example)",
      "itemTitle": "(title text omitted from this example)",
      "itemUrl": "(url omitted from this example)"
    }
  ]
}
```

The following shows the code for the sample RSS Feed cartridge handler:

```
package com.endeca.sample.cartridges;

import com.endeca.infront.assembler.AbstractCartridgeHandler;
import com.endeca.infront.assembler.CartridgeHandlerException;
```

```

import com.endeca.infront.assembler.ContentItem;
import com.endeca.infront.assembler.BasicContentItem;
import java.net.URL;
import java.util.ArrayList;
import javax.xml.parsers.DocumentBuilderFactory;
import javax.xml.parsers.DocumentBuilder;
import org.w3c.dom.CharacterData;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.NodeList;
import org.w3c.dom.Node;

public class RssFeedHandler extends AbstractCartridgeHandler {

    public ContentItem process(ContentItem pContentItem)
        throws CartridgeHandlerException {

        final String urlString = (String) pContentItem.get("feedUrl");
        final int numEntries =
            Integer.parseInt((String)pContentItem.get("numEntries"));

        try {
            URL url = new URL(urlString);
            DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
            DocumentBuilder docBuilder = factory.newDocumentBuilder();
            Document RssContents = docBuilder.parse(url.openStream());

            // get the channel info
            Element channel =
                (Element)RssContents.getElementsByTagName("channel").item(0);
            pContentItem.put("chanTitle", getElementValue(channel, "title"));
            pContentItem.put("chanUrl", getElementValue(channel, "link"));

            // get the entries and add them to a list
            ArrayList<ContentItem> entries = new ArrayList<ContentItem>(numEntries);
            NodeList nodes = RssContents.getElementsByTagName("item");
            for(int i=0; i<numEntries; i++) {
                Element element = (Element)nodes.item(i);
                if (element!=null) {
                    ContentItem entry = new BasicContentItem("rssEntry");
                    entry.put("itemTitle", getElementValue(element, "title"));
                    entry.put("itemUrl", getElementValue(element, "link"));
                    entry.put("itemDesc", getElementValue(element, "description"));
                    entries.add(entry);
                }
            }
            pContentItem.put("entries", entries);
        }
        catch (Exception e) {
            throw new CartridgeHandlerException(e);
        }

        return pContentItem;
    }

    private static String getCharacterDataFromElement(Element e) {
        try {
            Node child = e.getFirstChild();
            if(child instanceof CharacterData) {
                CharacterData cd = (CharacterData) child;

```

```
        return cd.getData();
    }
}
catch(Exception ex) {
}
return "";
}

private static String getElementValue(Element parent, String label) {
    return getCharacterDataFromElement(
        (Element)parent.getElementsByTagName(label).item(0));
}

}
```

Creating the cartridge renderer

The renderer displays a summary of the results with links that take the site visitor to the site that originated the RSS feed.

To create a renderer for the RSS feed:

1. Create a new JSP page based on the example below.
2. Save the renderer to /WEB-INF/views/desktop/RssFeed/RssFeed.jsp.
3. Refresh the application to see the result.

The results from the RSS feed display in the right sidebar.

\$373.00

Add to Cart

[+ Add to wishlist](#)



**EF 50mm
f/1.2L USM**
Price:
\$147.00

Product Reviews

\$1,029.00

Add to Cart

[+ Add to wishlist](#)

SCOR Wedges Tighten Up Your Short Game
A new line of golf clubs from SCOR lets players replace their standard 9 iron and their various wedges with a set of clubs designed for precision from 130 yards and closer.

\$1,113.00

Add to Cart

[+ Add to wishlist](#)

Behold, It Folds
Sony's Tablet P has a unique folding design that works well for games and e-books. But it's a tough sell for most users.

\$914.00

Add to Cart

[+ Add to wishlist](#)

Mamma Mia!
When viewed purely as a monument to two-wheeled speed, the Ducati 1199 Panigale is about as good as production superbikes get.

\$1,496.00

A Mouse Small Enough for a Cat

The following shows the code for the sample RSS Feed renderer in JSP:

```
<%@page language="java" pageEncoding="UTF-8"
contentType="text/html; charset=UTF-8" %>

<%@include file="/WEB-INF/views/include.jsp" %>

<div style="padding:2ex 0">
<b><a href="${component.chanUrl}">${component.chanTitle}</a></b>
<c:forEach var="rssEntry" items="${component.entries}">
    <p><a href="${rssEntry.itemUrl}">${rssEntry.itemTitle}</a><br />
    ${rssEntry.itemDesc}</p>
</c:forEach>
</div>
```

Custom Record Details cartridge with availability information

In this example, we extend the Record Details cartridge to display information about the availability of a product.

It demonstrates the following use cases:

- Extending one of the core Endeca cartridges

- Combining results from the MDEX Engine with information from another source during the `process()` phase of the assembly cycle
- Configuring a third-party service through Spring

In this cartridge, we create the following components:

Component	Description
cartridge handler	Extends the <code>RecordDetailsHandler</code> to add a property to the response model containing availability information.
mock "availability service"	Stands in for a real source of availability information such as an inventory system.

Because this cartridge does not introduce any change in the configuration options for the business user, there are no template changes for this cartridge. To enable the full functionality of this cartridge, the renderer should be updated to display the availability information, however that is not demonstrated in this guide.

Creating the cartridge handler and supporting classes

The `AvailabilityRecordDetailsHandler` extends the core `RecordDetailsHandler` to call a simple mock availability service to retrieve availability information about a particular record.

To create a cartridge handler that calls an availability service:

- Create the following classes: `Availability`, `AvailabilityService`, and `FixedAvailabilityService` based on the examples below.
The `AvailabilityService` defines an interface that returns availability information based on a record identifier, and `FixedAvailabilityService` provides a basic implementation of the interface.
- Create a new Java class in the package `com.endeca.sample.cartridges` based on the example below, which extends `RecordDetailsHandler`.
The handler takes the results of the MDEX Engine query and adds an additional property that represents the product availability.
- Compile the classes and add them to your application.
- Configure the Assembler to use the `AvailabilityRecordDetailsHandler` for the Record Details cartridge by editing the Spring context file as in the following example.



Note: If you have created a `sample-cartridge-config.xml` file for configuring the example cartridges, copy the `CartridgeHandler_ResultsList` bean from `assembler-context.xml` to your sample context file, comment out the version in `assembler-context.xml`, and then modify the version in your sample context file as indicated below.

```

<bean id="CartridgeHandler_RecordDetails"
  class="com.endeca.sample.cartridges.AvailabilityRecordDetailsHandler"
  parent="NavigationCartridgeHandler" scope="prototype" >
  <property name="recordState" ref="recordState" />
  <property name="availabilityService" ref="availabilityService" />
  <property name="recordSpec" value="common.id" />
  <property name="availabilityPropertyName"
    value="product.availability" />
</bean>

<bean id="availabilityService"
  class="com.endeca.sample.cartridges.FixedAvailabilityService"
  scope="singleton" >
  <!-- Implementation-specific configuration for the service
  ...
</bean>

```

```
could be specified here -->
</bean>
```

5. Restart the Endeca Tools Service.
6. Refresh the application and then click on any record to view its details page.

The availability property is now returned as part of the record details information:

```
{
    "@type": "RecordDetailsPageSlot",
    "name": "Record Details Page",
    "contentCollection": "Record Details Pages",
    "ruleLimit": "1",
    "contents": [
        {
            ...
        },
        "recordDetails": {
            "@type": "RecordDetails",
            "record": {
                "@class": "com.endeca.infront.cartridge.model.Record",
                "numRecords": 1,
                "attributes": {
                    ...
                    "product.availability": [
                        "BACKORDER"
                    ],
                    ...
                },
                "records": [ ... ]
            },
            "name": "Record Details"
        }
    ],
    ...
}
```

The renderer can now be updated to display availability information based on the value of this property.

The following shows the code for the availability service and its supporting classes:

```
package com.endeca.sample.cartridges;

public enum Availability {
    IMMEDIATE,
    WEEK,
    DROP_SHIP,
    BACKORDER;
}

package com.endeca.sample.cartridges;

public interface AvailabilityService {
    Availability getAvailabilityFor(String identifier);
}

package com.endeca.sample.cartridges;
```

```

public class FixedAvailabilityService implements AvailabilityService {

    public Availability getAvailabilityFor(String identifier) {
        try {
            return Availability.valueOf(identifier);
        } catch (IllegalArgumentException e) {
            return Availability.BACKORDER;
        }
    }
}

```

The following shows the code for the custom cartridge handler:

```

package com.endeca.sample.cartridges;

import com.endeca.infront.assembler.CartridgeHandlerException;
import com.endeca.infront.cartridge.RecordDetails;
import com.endeca.infront.cartridge.RecordDetailsConfig;
import com.endeca.infront.cartridge.RecordDetailsHandler;
import com.endeca.infront.cartridge.model.Attribute;
import org.springframework.beans.factory.annotation.Required;

public class AvailabilityRecordDetailsHandler extends RecordDetailsHandler {

    private AvailabilityService availabilityService;
    private String recordSpec;
    private String availabilityPropertyName;

    @Required
    public void setAvailabilityService(
        AvailabilityService availabilityService_) {
        availabilityService = availabilityService_;
    }

    @Required
    public void setRecordSpec(String recordSpec_) {
        recordSpec = recordSpec_;
    }

    @Required
    public void setAvailabilityPropertyName(
        String availabilityPropertyName_) {
        availabilityPropertyName = availabilityPropertyName_;
    }

    @Override
    public RecordDetails process(RecordDetailsConfig detailsConfig)
        throws CartridgeHandlerException {
        RecordDetails details = super.process(detailsConfig);
        if (null == details) return null;
        Attribute attr =
            details.getRecord().getAttributes().get(recordSpec);
        if (null == attr || 1 != attr.size()) {
            throw new CartridgeHandlerException("No record spec
                available on record, or spec is multiassign");
        }
        Attribute<Availability> availability =
            new Attribute<Availability>();
        availability.add(
            availabilityService.getAvailabilityFor(attr.toString()));
        details.getRecord().getAttributes().put(availabilityPropertyName,

```

```

        availability);
    return details;
}
}

```

Custom Results List with recommendations

In this example, we extend the Results List cartridge to boost certain products based on information from a recommendation engine.

It demonstrates the following use cases:

- Extending one of the core Endeca cartridges
- Using data from another source to modify the query to the MDEX Engine created during the `preprocess()` phase of the assembly cycle
- Configuring a third-party service through Spring

In this cartridge, we create the following components:

Component	Description
cartridge handler	Extends the <code>ResultsListHandler</code> to retrieve a set of items to boost from a recommendations engine and add a boost stratum to the MDEX Engine query.
mock recommendations service	Stands in for a real source of recommendations.

Because this cartridge does not introduce any change in the configuration options for the business user, there are no template changes for this cartridge. Additionally, the response model for the customized cartridge is the same as the default Results List (only with the records in a different order), so there is no need for changes to the default renderer.

Creating the cartridge handler and supporting classes

The `RecommendationsResultsListHandler` extends the core `ResultsListHandler` to call a simple mock recommendations service and boosts the recommended products.

To create a cartridge handler that boosts recommended records:

- Create the interface `RecommendationService` and the concrete implementation `TestRecommendationService` based on the examples below.

As a proof of concept, the recommendations service always returns the same recommendations from the Discover Electronics data set.

- Create a new Java class in the package `com.endeca.sample.cartridges` based on the example below, which extends `ResultsListHandler`.

The handler retrieves a list of recommended records from the service and adds them to a boost stratum for the MDEX Engine query. If the records are present in the results set, they are boosted to the top of the results list.

- Compile the classes and add them to your application.
- Configure the Assembler to use the `RecommendationsResultsListHandler` for the Results List cartridge by editing the Spring context file as follows:



Note: If you have created a `sample-cartridge-config.xml` file for configuring the example cartridges, copy the `CartridgeHandler_ResultsList` bean from `assembler-context.xml` to your sample context file, comment out the version in `assembler-context.xml`, and then modify the version in your sample context file as indicated below.

```

<bean id="CartridgeHandler_ResultsList"
  class="com.endeca.sample.cartridges.RecommendationsResultsListHandler"
  parent="NavigationCartridgeHandler" scope="prototype">
  <property name="contentItemInitializer">
    <!-- additional elements omitted from this example -->
  </property>
  <property name="sortOptions">
    <!-- additional elements omitted from this example -->
  </property>
  <property name="recommendationService" ref="recommendationService" />
  <property name="recordSpec" value="common.id"/>
</bean>

<bean id="recommendationService"
  class="com.endeca.sample.cartridges.TestRecommendationService"
  scope="singleton" >
  <!-- Implementation-specific configuration for the service
       could be specified here -->
</bean>

```

5. Restart the Endeca Tools Service.
6. Refresh the application.

The recommended records are boosted to the top of the results:

Display: 10 per page ▾		Sort By: Relevance ▾
	Canon PowerShot S95 PowerShot S95, 10MP, 3.8x Optical, 4x Digital, 15x Combined Zoom, 7.62 cm (3.0") LCD, 3648 x 2048, 16:9, Zwart	\$1,000.00 Add to Cart + Add to wishlist
	Trust Cuby Pro Cuby Pro, Titanium, 1.3 Mp, 1280 x 1024, USB 2.0	\$918.00 Add to Cart + Add to wishlist
	Pentax K20D K20D, 14.6 Megapixels	\$907.00 Add to Cart + Add to wishlist
	Fujifilm FinePix F50fd & SD Card 1GB FinePix F50fd Black & SD Card 1GB	\$991.00 Add to Cart + Add to wishlist
	Kodak EasyShare M863 1319094, EasyShare M863 Digital Camera, 8.2 MP for prints up to 30 x 40 in. (76 x 102 cm), 3X optical zoom	\$964.00 Add to Cart + Add to wishlist
	Pentax Optio M40 Optio M40 Digital Camera	\$747.00 Add to Cart + Add to wishlist
	Sony DSC-H50 9.1 Mega Pixel Cyber-shot H50 Series Digital Camera	\$991.00 Add to Cart + Add to wishlist

The following shows the code for the recommendations service interface and concrete implementation:

```
package com.endeca.sample.cartridges;

import java.util.List;

public interface RecommendationService {
    public List<String> getRecommendedProductIds();
}

package com.endeca.sample.cartridges;

import java.util.Arrays;
import java.util.List;

public class TestRecommendationService
    implements RecommendationService {
    public static final List<String> IDS =
        Arrays.asList("5891932", "6001963", "1438066", "1581692",
                      "2708142", "1235424", "3422480");

    public List<String> getRecommendedProductIds() {
        return IDS;
    }
}
```

The following shows the code for the custom cartridge handler:

```
package com.endeca.sample.cartridges;

import java.util.ArrayList;
import java.util.List;

import com.endeca.infront.assembler.CartridgeHandlerException;
import com.endeca.infront.cartridge.ResultsListConfig;
import com.endeca.infront.cartridge.ResultsListHandler;
import com.endeca.infront.navigation.model.CollectionFilter;
import com.endeca.infront.navigation.model.PropertyFilter;

public class RecommendationsResultsListHandler extends ResultsListHandler {
    private RecommendationService recommendationService;
    private String recordSpec;

    public String getRecordSpec() {
        return recordSpec;
    }

    public void setRecordSpec(String recordSpec_) {
        this.recordSpec = recordSpec_;
    }

    public void setRecommendationService(
        RecommendationService recommendationService_) {
        recommendationService = recommendationService_;
    }

    /**
     * This cartridge will get the list of recommended products
     * (by record spec) and explicitly boost each one of them using
     * a PropertyFilter.
    }
```

```

    */
    @Override
    public void preprocess(ResultsListConfig pContentItem)
        throws CartridgeHandlerException {
        List<String> ids =
            recommendationService.getRecommendedProductIds();
        List<CollectionFilter> boostFilters =
            new ArrayList<CollectionFilter>(
                ids.size());
        for (String s : ids) {
            boostFilters.add(new CollectionFilter(new PropertyFilter(
                recordSpec, s)));
        }

        pContentItem.setBoostStrata(boostFilters);
        super.preprocess(pContentItem);
    }
}

```

"Hello, World" cartridge with layered color configuration

In this example, we extend the "Hello, World" example cartridge to demonstrate the layering of configuration from several sources.

In this scenario, we can define a default color for the message in our "Hello, World" cartridge, which the business user can override on a per-instance basis in Experience Manager. The site visitor can also select a preferred color from the client application.

It demonstrates the following use cases:

- Combining the default cartridge configuration, cartridge instance configuration, and request-based configuration using the `ConfigInitializer` and `RequestParamMarshaller` helper classes
- Using a cartridge configuration bean

In this cartridge, we create the following components:

Component	Description
cartridge handler	Uses the <code>ColorConfigInitializer</code> to layer multiple sources of configuration for message color.
cartridge configuration bean	Provides a means of specifying default values for this cartridge via Spring.
cartridge renderer	Provides a drop-down list from which the site visitor can choose a color for the message.

Because this cartridge does not introduce any change in the configuration options for the business user, there are no template changes for this cartridge.

Creating the cartridge handler and supporting classes

The cartridge handler combines the various sources of configuration for message color using the `ConfigInitializer` and `RequestParamMarshaller` helper classes.

To create the "Hello, World" cartridge handler with color configuration and add it to the application:

1. Create a new Java class in the package `com.endeca.sample.cartridges` based on the example below, which extends `AbstractCartridgeHandler`.
2. Create a configuration bean for this cartridge based on the example below. This enables us to define default values for the cartridge properties in the Spring context file.
3. Compile the cartridge handler and configuration bean and add them to your application.
4. Configure the Assembler to use the `ColorConfigHandler` for the "Hello, World" cartridge by editing the Spring context file as follows:

```

<bean id="CartridgeHandler_Hello"
  class="com.endeca.sample.cartridges.ColorConfigHandler"
  scope="prototype">
  <property name="contentItemInitializer">
    <bean class="com.endeca.infront.cartridge.ConfigInitializer"
      scope="singleton">
      <property name="defaults">
        <bean class="com.endeca.sample.cartridges.ColorConfig"
          scope="singleton">
          <property name="messageColor" value="#FF6600"/>
        </bean>
      </property>
    <property name="requestParamMarshaller">
      <bean
        class="com.endeca.infront.cartridge.RequestParamMarshaller"
        scope="singleton">
        <property name="httpServletRequest" ref="httpServletRequest"/>
        <property name="requestMap">
          <map>
            <entry key="color" value="messageColor"/>
          </map>
        </property>
      </bean>
    </property>
  </bean>
  </property>
  <property name="colorOptions">
    <map>
      <entry key="Red" value="#FF0000"/>
      <entry key="Green" value="#00FF00"/>
      <entry key="Blue" value="#0000FF"/>
      <entry key="Black" value="#000000"/>
    </map>
  </property>
</bean>

```

5. Restart the Endeca Tools Service.
6. Refresh the application.

The color options do not display yet because we have not updated the renderer, but you can validate that the response model has been populated with the information that we want the renderer to use via the JSON view:

```
{
  "@type": "Hello",
  "name": "Hello cartridge",
  "message": "Hello, color world!",
  "messageColor": "#0000FF",
  "colorOptions": [
    {
      "@type": "colorOption",

```

```

        "hexCode": "#FF0000",
        "label": "Red"
    },
    {
        "@type": "colorOption",
        "hexCode": "#00FF00",
        "label": "Green"
    },
    {
        "@type": "colorOption",
        "hexCode": "#0000FF",
        "label": "Blue"
    },
    {
        "@type": "colorOption",
        "hexCode": "#000000",
        "label": "Black"
    }
]
}

```

The following shows the code for the sample "Hello, World" cartridge handler with color configuration:

```

package com.endeca.sample.cartridges;

import com.endeca.infront.assembler.AbstractCartridgeHandler;
import com.endeca.infront.assembler.CartridgeHandlerException;
import com.endeca.infront.assembler.ContentItem;
import com.endeca.infront.assembler.BasicContentItem;
import com.endeca.infront.assembler.ContentItemInitializer;
import com.endeca.sample.cartridges.ColorConfig;
import java.util.ArrayList;
import java.util.Map;

public class ColorConfigHandler extends AbstractCartridgeHandler {

    private ContentItemInitializer mInitializer;
    private Map<String, String> mColorOptions;

    public void setContentItemInitializer(ContentItemInitializer initializer) {
        mInitializer = initializer;
    }

    public void setColorOptions(Map<String, String> colorOptions) {
        mColorOptions = colorOptions;
    }

    /**
     * Returns the merged configuration based on Spring defaults,
     * Experience Manager configuration, and request parameters
     */
    @Override
    public ContentItem initialize(ContentItem pContentItem) {
        // If any configuration from Experience Manager is empty, remove
        // that property so we can use the default value
        for (String key: pContentItem.keySet()) {
            if (((String)pContentItem.get(key)).isEmpty())
                pContentItem.remove(key);
        }
    }
}

```

```

        return mInitializer == null ? new ColorConfig(pContentItem) :
            mInitializer.initialize(pContentItem);
    }

    /**
     * Returns the merged configuration and information about the color options
     * available to the site visitor.
     */
    @Override
    public ContentItem process(ContentItem pContentItem)
        throws CartridgeHandlerException {
        int numColors = mColorOptions.size();
        ArrayList<ContentItem> colors =
            new ArrayList<ContentItem>(numColors);
        if (mColorOptions != null && !mColorOptions.isEmpty()) {
            for (String key: mColorOptions.keySet()) {
                ContentItem color = new BasicContentItem("colorOption");
                color.put("label", key);
                color.put("hexCode", mColorOptions.get(key));
                colors.add(color);
            }
            pContentItem.put("colorOptions", colors);
        }
        return pContentItem;
    }
}

```

The following code implements a basic bean that enables us to specify a default value for the message color in the Spring configuration:

```

package com.endeca.sample.cartridges;

import com.endeca.infront.assembler.BasicContentItem;
import com.endeca.infront.assembler.ContentItem;

public class ColorConfig extends BasicContentItem {

    public ColorConfig() {
        super();
    }

    public ColorConfig(final String pType) {
        super(pType);
    }

    public ColorConfig(ContentItem pContentItem) {
        super(pContentItem);
    }

    public String getMessageColor() {
        return getTypedProperty("messageColor");
    }

    public void setMessageColor(String color) {
        this.put("messageColor", color);
    }
}

```

```
    }
}
```

Creating the cartridge renderer

In this example we update the "Hello, World" renderer to add a control for the site visitor to select a color for the message.

To add a drop-down for the site visitor to select a message color based on the options configured for this cartridge:

1. Create a new JSP page based on the example below, or update the renderer you previously created by adding the section in bold.
2. Save the renderer to /WEB-INF/views/desktop/Hello/Hello.jsp.
3. Refresh the application to verify that the drop-down menu displays.



The following shows the code for the sample "Hello, World" renderer with color choice drop-down in JSP:

```
<%@page language="java" pageEncoding="UTF-8"
contentType="text/html; charset=UTF-8"%>

<%@include file="/WEB-INF/views/include.jsp"%>
<div style="border-style: dotted; border-width: 1px;
border-color: #999999; padding: 10px 10px">
    <div style="font-size: 150%;
        color: ${component.messageColor}">${component.message}</div>
    <div style="font-size: 80%; padding: 5px 0px">
        <select onchange="location = this.options[this.selectedIndex].value">
            <option value="">Select a color</option>
            <c:forEach var="colorOption" items="${component.colorOptions}">
                <c:url value="<% request.getPathInfo() %>" var="colorAction">
                    <c:param name="color" value="${colorOption.hexCode}" />
                </c:url>
                <option value="${colorAction}">${colorOption.label}</option>
            </c:forEach>
        </select>
    </div>
</div>
```

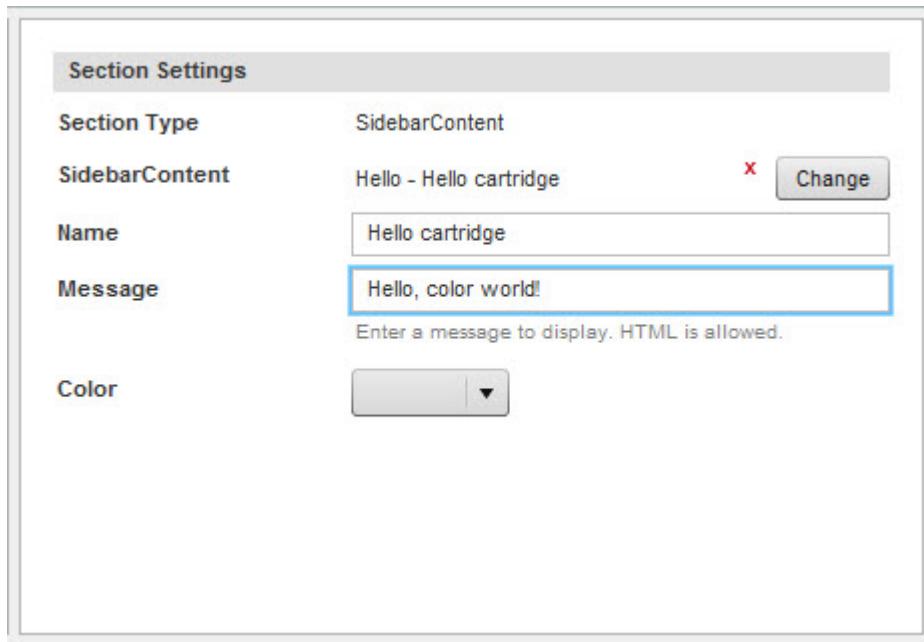
Testing the "Hello, World" cartridge with layered color configuration

We can validate that the cartridge handler applies the different sources of configuration properly by incrementally populating each source of the configuration.

To test the "Hello, World" cartridge:

1. In Experience Manager, remove any previously created instance of the Hello cartridge.

2. Insert a new instance of the cartridge on the default page and specify a message string, but do not select a color.

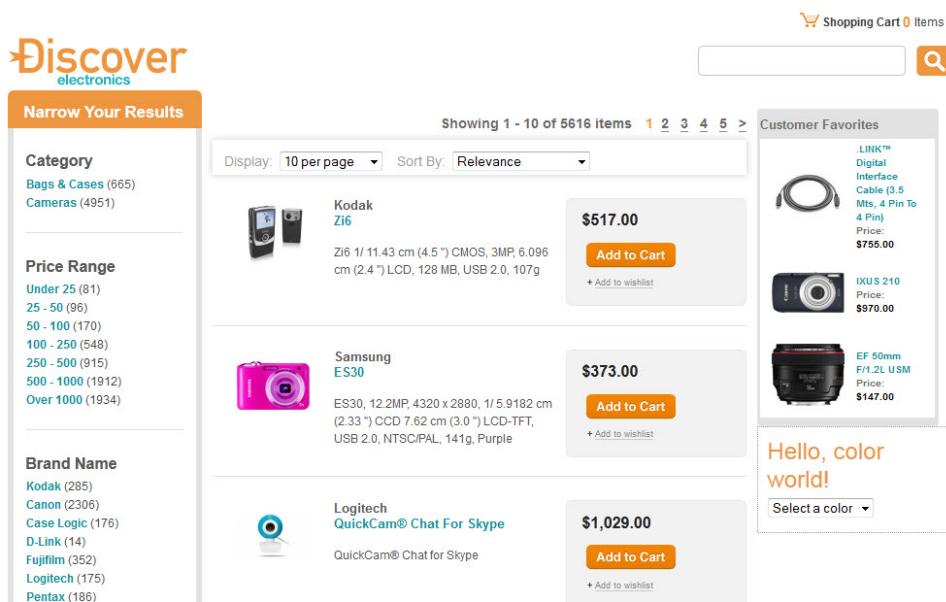


Section Settings

Section Type	SidebarContent
SidebarContent	Hello - Hello cartridge x Change
Name	Hello cartridge
Message	Hello, color world!
Enter a message to display. HTML is allowed.	
Color	 ▾

3. Save the page.
4. Refresh the application.

The message displays using the default color, orange.



Discover electronics

Narrow Your Results

Showing 1 - 10 of 5616 items 1 2 3 4 5 >

Category	Kodak Z16	\$517.00 Add to Cart
Price Range	Samsung ES30	\$373.00 Add to Cart
Brand Name	Logitech QuickCam® Chat For Skype	\$1,029.00 Add to Cart

Customer Favorites

- LINK™ Digital Interface Cable (3.5 Mts, 4 Pin To 4 Pin) Price: \$755.00
- IXUS 210 Price: \$970.00
- EF 50mm F1.2L USM Price: \$147.00

Hello, color world!

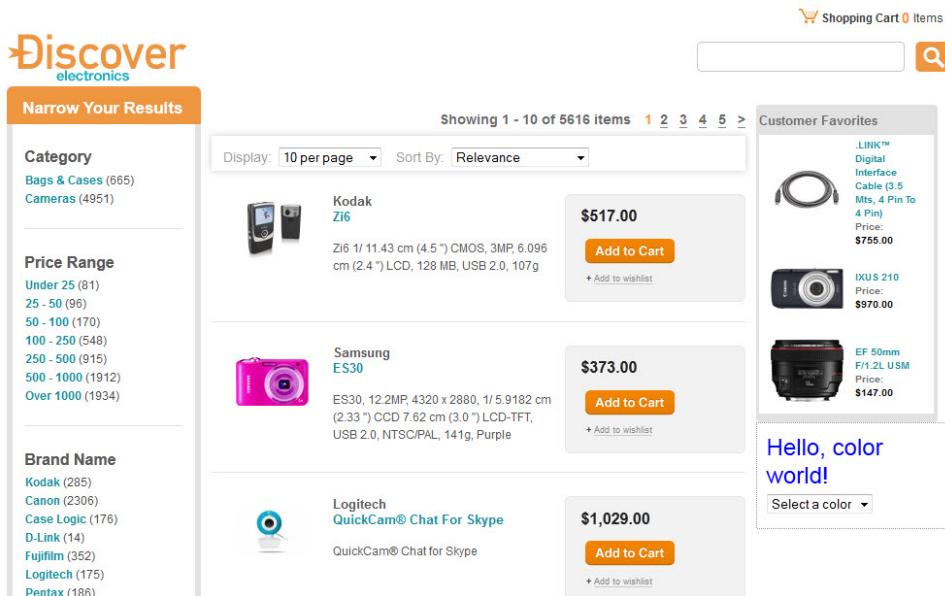
5. Going back to Experience Manager, now select a message color for this instance of the cartridge.

Section Settings

Section Type	SidebarContent
SidebarContent	Hello - Hello cartridge X Change
Name	Hello cartridge
Message	Hello, color world!
Enter a message to display. HTML is allowed.	
Color	Blue ▼

6. Refresh the application.

The message displays using the color configured in Experience Manager.



The screenshot shows a shopping website for 'Discover electronics'. On the left, there's a sidebar with 'Narrow Your Results' filters for Category (Bags & Cases, Cameras), Price Range (Under 25, 25 - 50, 50 - 100, 100 - 250, 250 - 500, 500 - 1000, Over 1000), and Brand Name (Kodak, Canon, Case Logic, D-Link, Fujifilm, Logitech, Pentax). The main content area shows a grid of product cards. In the top right corner of the grid, there is a blue box containing the text 'Hello, color world!'. Below this message is a dropdown menu with the text 'Select a color ▼'.

7. Using the drop-down list on the cartridge, select another color.

The drop-down control adds a `color` parameter to the URL, which is parsed by the `RequestParamMapper` into the `messageColor` property.

Discover Electronics

Narrow Your Results

Category: Bags & Cases (665), Cameras (4951)

Price Range: Under 25 (81), 25 - 50 (96), 50 - 100 (170), 100 - 250 (548), 250 - 500 (915), 500 - 1000 (1912), Over 1000 (1934)

Brand Name: Kodak (285), Canon (2306), Case Logic (176), D-Link (14), Fujifilm (352), Logitech (175), Pentax (186), Samsung (518), Sony (1017), Trust (40)

Showing 1 - 10 of 5616 items

Display: 10 per page, Sort By: Relevance

Kodak Z16: \$517.00

Samsung ES30: \$373.00

Logitech QuickCam® Chat For Skype: \$1,029.00

Customer Favorites:

- LINK™ Digital Interface Cable (3.5 Mts, 4 Pin To 4 Pin) Price: \$755.00
- IXUS 210 Price: \$970.00
- EF 50mm F1.2L USM Price: \$147.00

Hello, color world!

Select a color

Chapter 17

Developing Editors for Workbench

If you wish to expose configuration to your Content Administrator in Workbench, you should first check whether the existing set of editors supports your requirements. If not, you may wish to develop a custom editor to suit your needs.

Setting up the Experience Manager Editor SDK

The Experience Manager Editor SDK is included with your Tools and Frameworks installation. This section provides instructions for setting up a development environment for developing custom editors.



Important: The Experience Manager Editor SDK is deprecated in the Tools and Frameworks 11.0.0 release. Consult your support representative for guidance if you wish to develop custom editors for your applications.

Flex prerequisites and resources

To develop editors with the Experience Manager Editor SDK, you must have the following components:

Flex development requirements

- **Flex SDK 4.5.x or Flash Builder** — Flash Builder is an integrated editing and debugging environment offered by Adobe. It requires a developer license. The raw SDK is open source and also available from Adobe. You can consult the wiki at <http://sourceforge.net/adobe/flexsdk/wiki/About/> for additional information.

Additionally, Oracle recommends using Apache Maven 3.0.4 to build your projects. Apache Maven is an open source project hosted at <http://maven.apache.org/>.

Flex resources

Flex is an open source development framework created and maintained by Adobe. It supports common design patterns and is based on MXML and ActionScript 3. If you are unfamiliar with Flex, you may find the following resources helpful:

- <http://www.adobe.com/products/flex.html> — The Adobe Flex Web site provides an overview of the Flex development framework and includes download links to the Flex SDK.

- http://help.adobe.com/en_US/Flex/4.0/UsingSDK/index.html—The Adobe Flex 4 resources page contains links to Flex documentation, including the *ActionScript 3.0 Reference* and *ActionScript 3.0 Developer's Guide*.

About setting up a Flex development environment

Oracle recommends setting up a Flex development environment for creating your custom editors.

You can use a standalone installation of Flash Builder, or set up your development environment in Eclipse. Setting up an Experience Manager SDK project in Eclipse requires the Flash Builder 4 plugin.

Configuring a Flash Builder 4.0 development environment

If you choose to develop using Flash Builder, you must use Flash Builder 4.0. In addition, the version of the Flex SDK that Flash Builder uses to compile must be 4.5.0 or higher. You should compile your editors using the Halo theme for visual consistency.

To configure a Flash Builder development environment:

1. Confirm that you are running version 4.0 with the Flex 4.5 SDK:
 - a) Start Flash Builder 4.0.
 - b) Navigate to **Window > Preferences**.
 - c) In the tree control on the left, navigate to **Flash Builder > Installed Flex SDKs**.
 - d) Verify that the Flex 4.5 entry appears and is selected in the list of **Installed SDKs**.
2. Update your project to use the Halo visual theme:
 - a) In Flash Builder, navigate to **Project > Properties > Flex Theme**.
 - b) Select **Halo**.
 - c) Click **OK** to save your changes.

Configuring Flex Framework dependencies as Runtime Shared Libraries (RSLs)

Experience Manager includes Flex Framework dependencies as Runtime Shared Libraries. Configure your editors to use these same dependencies in order to avoid version conflicts.

The following dependencies ship with Tools and Frameworks and are included within the Endeca Configuration Repository at `ifcr/tools/xmgr`. You should configure the resources below as Runtime Shared Libraries if they are dependencies of your editor module:

- `com.adobe.flex.framework.rpc-4.5.1.21328.swc`
- `com.adobe.flex.framework.framework-4.5.1.21328.swc`
- `com.adobe.flex.framework.textLayout-4.5.1.21328.swc`
- `com.adobe.flex.framework.spark-4.5.1.21328.swc`
- `com.adobe.flex.framework.sparkskins-4.5.1.21328.swc`
- `com.adobe.flex.framework.charts-4.5.1.21328.swc`
- `com.adobe.flex.framework.spark_dmv-4.5.1.21328.swc`
- `com.adobe.flex.framework.osmf-4.5.1.21328.swc`
- `com.adobe.flex.framework.mx-4.5.1.21328.swc`

- com.adobe.flex.framework.advancedgrids-4.5.1.21328.swc
- com.adobe.flex.framework.textLayout-4.5.1.21328.swc
- com.endeca.ExperienceManagerAPI-4.0.0.swf

To configure a dependency as a Runtime Shared Library:

Update your project to use the ExperienceManagerAPI-4.0.0.swc and the Halo theme:

- In Flash Builder, navigate to **Project > Properties > Flex Build Path**.
- Add the dependency..

For example,

ToolsAndFrameworks\4.0.0\editor_sdk\libs\ExperienceManagerAPI-4.0.0.swc.

- Beneath the new SWC in the tree view, select **Link Type:** and click **Edit....**
- Select **Runtime shared library (RSL)**.
- For the **Deployment Path/URL:**, enter the location of the dependency in the Endeca Configuration Repository.

For example, /ifcr/tools/xmgr/com.endeca.ExperienceManagerAPI-4.0.0.swf.

- Click **OK**.

- Click **OK** to save your changes.

Installing the Experience Manager API to a Maven repository

The Experience Manager Editor SDK includes a Maven script for installing the included API package into a Maven repository.

These steps assume you have Maven 3.0.4 installed as part of your development environment.

To install the Experience Manager API package:

1. Navigate to the %ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven directory.
2. Open a command prompt.
3. Run the install-api profile in the Maven POM file using the following command:

```
mvn -Pinstall-api
```

This installs ExperienceManagerAPI-4.0.0.swc to your Maven repository.

Developing Editors With the Experience Manager Editor SDK

This chapter covers steps for developing your own editors using the Experience Manager Editor SDK.

About developing custom editors

Once you have set up your development environment, creating and using new editors consists of the following general steps:

1. Configure external Flex Framework and Experience Manager API dependencies as Runtime Shared Libraries (RSLs).
2. Build your editor module and copy the SWF file to your <app dir>\config\editors_config\modules directory.

3. Register your custom editor module and the included editors within your application's editor configuration file.
4. Upload the editor module and editor configuration file to your deployed application by using the `<app dir>\control\set_editors_config` script.
5. Add your editors to an existing cartridge template or create a new cartridge template that includes them.



Note: If you make changes to an existing cartridge, any saved cartridges that use the old template are unchanged until you access them in Experience Manager. When accessed, they are updated with default values specified in the cartridge template. For this reason, Oracle does not recommend updating cartridges that are in widespread use throughout your application.

6. Upload the cartridge template to your deployed application by using the `<app dir>\control\set_templates` script.



Note: When naming your editors, note that the package names `com.xmgr` and `com.endeca.xmgr` are reserved for the Experience Manager product. Do not use them for custom editors.

Creating an editor module for custom editors

You must create an editor module to contain your custom editors. Oracle Endeca Experience Manager Extensions includes a sample editor module that you can use as a reference.

In order to add your editors to the `sample_editors.swf` file, you must modify the `sample_editors.mxml` editor registry to include them.

To create an editor module for custom editors:

1. Create an MXML registry file for your editor module.

For the sample editor project, this is

`editor_sdk\reference\build\maven\src\main\flex\sample_editors.mxml`.

2. Define the `editor` namespace within a root `<editor:EditorModule>` element:

```
<editor:EditorModule xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:editor="com.endeca.tools.pagebuilder.editor.*">
</editor:EditorModule>
```

3. Within the root `<editor:EditorModule>` element, add a `<editor:registeredEditors>` element that contains an `<mx:Array>`:

```
<editor:EditorModule xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:editor="com.endeca.tools.pagebuilder.editor.*">
    <editor:registeredEditors>
        <mx:Array>
        </mx:Array>
    </editor:registeredEditors>
</editor:EditorModule>
```

4. Within the `<mx:Array>` element, add an `<editor:EditorEntry>` element for each of your custom editors:

```
<editor:EditorModule xmlns:mx="http://www.adobe.com/2006/mxml"
    xmlns:editor="com.endeca.tools.pagebuilder.editor.*">
    <editor:registeredEditors>
        <mx:Array>
```

```

<editor:EditorEntry/>
</mx:Array>
</editor:registeredEditors>
</editor:EditorModule>

```

- Specify the following attributes:

Attribute	Value
uri	The editor namespace. This is used in your cartridge templates and in the editor configuration file.
localName	The name to use for the editor in your cartridge templates.
editor	The fully qualified name of your editor.

```

<editor:EditorModule xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:editor="com.endeca.tools.pagebuilder.editor.*">
  <editor:registeredEditors>
    <mx:Array>
      <editor:EditorEntry
        uri="http://endeca.com/sample/2010"
        localName="MyCustomEditor"
        editor="com.endeca.tools.pagebuilder.samples.editors.MyCustomEditor"
      />
    </mx:Array>
  </editor:registeredEditors>
</editor:EditorModule>

```

- Repeat Steps 4-5 for each editor you wish to register.

- Save and close the file.
- Build the editor module.

For the sample editor module, this consists of the following steps:

- Navigate to the %ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven directory.
- Build the sample editor module using the following command:

```
mvn clean install
```

The sample_editors.swf file is output to the target subdirectory.

- Upload the editor module to your application:

- Navigate to your build output directory.

For example, %ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven\target.

- Copy the editor SWF file to the config\editors_config\modules directory of your deployed application.
- If this directory does not exist, you must create it.
- Navigate to the control directory of your deployed application.
- Run the set_editors_config script.

Registering custom editors

You must modify the editor configuration file to register new editors with Experience Manager. The file, editors.xml, is maintained in the Endeca Configuration Repository and stored locally within the <app dir>\config\editors_config directory.



Note: The steps below assume a default installation, with Workbench running on port 8006 of your local machine.

To register a custom editor:

1. Navigate to the `<app dir>\config\editors_config` directory of your deployed application.

For the default Discover Electronics reference application, this is

`C:\Endeca\apps\Discover\config\editors_config` on Windows, or
`/usr/local/endeca/apps/Discover/config/editors_config` on UNIX.

2. Open the `editors.xml` file.

3. Add an `<EditorModule>` element within the closing `</EditorConfig>` tag, and set the `url` attribute to your custom editor module.

For example:

```
...
<EditorModule url="/ifcr/sites/Discover/configuration/tools/xmgr/modules/sample_editors.swf">
    </EditorModule>
</EditorConfig>
```

4. To add additional editors, insert an `<Editor>` element inside the `<EditorModule>` element for each new editor that you wish to include.

For example:

```
...
<EditorModule url="/ifcr/sites/Discover/configuration/tools/xmgr/modules/sample_editors.swf">
    <Editor name="http://endeca.com/sample/2010:RichTextEditor">
        </Editor>
    </EditorModule>
</EditorConfig>
```

5. To set default editor configuration, add an `<EditorConfig>` element within each `<Editor>` element:

For example:

```
...
<EditorModule url="/ifcr/sites/Discover/configuration/tools/xmgr/modules/sample_editors.swf">
    <Editor name="http://endeca.com/sample/2010:RichTextEditor">
        <EditorConfig sample="customAttribute">
            <ExampleCustomEditorXML foo="bar" size="10" resizable="false" />
        </EditorConfig>
    </Editor>
    </EditorModule>
</EditorConfig>
```

6. Save and close the file.

7. Navigate to the `<app dir>\control` directory of your deployed application.

8. Run the `set_editors_config` batch or shell script.

This script uploads the updated `editors.xml` file and any editor modules in the `<app dir>\config\editors_config\modules` directory to the Endeca Configuration Repository.

About creating and uploading a cartridge template

To use your custom editors in Experience Manager, you need to create and upload a cartridge template that includes the new editors. You can choose to create a new cartridge, or to modify an existing cartridge template.

After creating or modifying a cartridge to include your custom editors, you must upload it to your application.. You can accomplish this by moving the template to your deployed application's `config\cartridge_templates` directory and running the `control\set_templates` batch or shell script.

Example: The sample RichTextEditor

The Experience Manager editor SDK includes a sample `RichTextEditor` and associated cartridge template that you can use as a reference when developing your own editors.

The source code for the `RichTextEditor` is available under `editor_sdk\reference\build\maven\src\main\flex\com\endeca\tools\pagebuilder\samples\editors\RichTextEditor.mxml`.

The sample Rich Text Box cartridge template

The Rich Text Box cartridge is provided as a sample cartridge that makes use of the `RichTextEditor`. The associated cartridge template is included with the Experience Manager Editor SDK.

The cartridge template is available under `editor_sdk\reference\cartridge_templates\SampleEditor\template.xml`. The cartridge uses a basic `StringEditor` for the title box, and the `RichTextEditor` to enter and configure body text:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="SecondaryContent">
  <Description>Displays rich text.</Description>
  <ThumbnailUrl>/thumbnails/PageTemplate/TextBoxSidebar.png</ThumbnailUrl>
  <ContentItem>
    <Name>New Rich Text Box</Name>
    <Property name="title">
      <String/>
    </Property>
    <Property name="rich_text">
      <String/>
    </Property>
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <StringEditor propertyName="title" label="Title" enabled="true" xmlns="editors"/>
      <!-- default value for the optional height attribute for RichTextEditor is 400px -->
      <!-- make sure not to make it any smaller or it will not render well -->
      <RichTextEditor propertyName="rich_text" xmlns="http://endeca.com/samples/2010" label="Custom Editor" />
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

Installing the sample editor module and cartridge template

The Experience Manager Editor SDK includes a sample editor module with a Rich Text editor that you can install in your application.



Note: For example purposes this guide assumes that you are extending the default Discover Electronics reference application, with Workbench running on port 8006 of your local machine.

To install the sample editor module and cartridge template:

1. Create a directory for custom editor modules:

- a) Navigate to the `<app dir>\config\editors_config` directory.

For the default Discover Electronics reference application, this is

`C:\Endeca\apps\Discover\config\editors_config` on Windows, or
`/usr/local/endeca/apps/Discover/config/editors_config` on UNIX.

- b) Create a `modules` directory.

2. Build the sample editor module:

- a) Navigate to the `%ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven` directory.
- b) Build the sample editor module using the following command:

```
mvn clean install
```

The `sample_editors.swf` file is output to the `target` subdirectory.

- c) Navigate to the `%ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven\target` directory.
- d) Copy the `sample_editors.swf` file to the `<app dir>\config\editors_config\modules` directory you created in Step 1.

3. Register the sample editors:

- a) Open the sample editor configuration file,
`editor_sdk\reference\editors_config\sample_editors.xml`.

- b) Open the editor configuration file for your application.

For the default Discover Electronics reference application, this is

`C:\Endeca\apps\Discover\config\editors_config\editors.xml` on Windows, or
`/usr/local/endeca/apps/Discover/config/editors_config/editors.xml` on UNIX..

- c) Copy the `<EditorModule>` element from the `sample_editors.xml` file to the editor registry file:

```
<EditorModule url="/ifcr/sites/[site name]/configuration/tools/xmgr/modules/sample_editors.swf">
    <Editor name="http://endeca.com/sample/2010:RichTextEditor">
        <EditorConfig sample="customAttribute">
            <ExampleCustomEditorXML foo="bar" size="10" resizeable="false" />
        </EditorConfig>
    </Editor>
</EditorModule>
```

The element should be parallel to the existing `<EditorModule>` element.

- d) Replace `[site name]` with the name of your application:

```
<EditorModule url="/ifcr/sites/[site name]/configuration/tools/xmgr/modules/sample_editors.swf">
```

For the default Discover Electronics reference application, this is `Discover`.

- e) Save and close the file.

4. Upload your custom content:

- a) Navigate to the `<app dir>\control` directory.
- b) Run the `set_editors_config` batch or shell script.
This uploads the `sample_editors.swf` and `editors.xml` files to the Endeca Configuration Repository.

5. Upload a template that includes the custom editor:
 - a) Copy `editor_sdk\reference\cartridge_templates\SampleEditor` directory to your `<app dir>\config\cartridge_templates` directory.
 - b) Navigate to the `<app dir>\control` directory.
 - c) Run the `set_templates` batch or shell script.
This uploads the template files to the Endeca Configuration Repository.

About custom editors in multiple locales

If your implementation supports multiple locales, you can localize your custom editors.

You must do the following:

- Modify your editor's `pom.xml` file
- Create resource properties files that contain localized strings
- Modify the editor module
- Add the `getMessage()` function to your custom editors to retrieve the localized strings

Modifying the POM file to support multiple locales

You need to pass a list of locales and a directive to the compiler to retain the declarations of embedded resource bundles to the compiler. The following task uses a POM file for passing this information.

1. Open the `pom.xml` file in an editor.

For the sample editor project, this is under `editor_sdk\reference\build\maven`

2. Within the `<configuration>` element add a `<localesCompiled>` element with a list of locales:

```
<configuration>
  <localesCompiled>
    <locale>en_US</locale>
    <locale>fr_FR</locale>
    <locale>de_DE</locale>
  </localesCompiled>
</configuration>
```

This example includes US, French, and German locales.

3. Specify a `<keepAs3Metadatas>` element to declare embedded resources bundles. This allows the editor container to detect the embedded resources automatically.

```
<configuration>
  <localesCompiled>
    <locale>en_US</locale>
    <locale>fr_FR</locale>
    <locale>de_DE</locale>
  </localesCompiled>
  <keepAs3Metadatas>
    <keepAs3Metadata>ResourceBundle</keepAs3Metadata>
  </keepAs3Metadatas>
</configuration>
```

4. Save and close the file.

Resources properties files

You must create resource property files for each locale for storing localized strings.

Each locale is required to reside in its own directory. The default location is `src/main/locales/<locale>` where `<locale>` is the ISO language and region code combination. For example `src/main/locales/en_US/com/endeca/tools/pagebuilders/samples/SampleResources.properties` indicates that US values are stored in the properties file.

Here is an example of `SampleResources.properties` file contents:

```
editor.sample.message=Enter your message in the following box.
```

This localized content can be obtained by the custom editor using the `getMessage()` function to retrieve the string from the properties file. See the following example:

```
<ext:Editor>
  <!-- ... -->
  <mx:FormItem id="propertyItem"
    label="{info.templateConfig.@label}"
    labelWidth="135"
    labelStyleName="leftColumnLabel"
    required="false">
    <mx:Label text="{locale.getMessage('editor.sample.message')}" />
    <mx:RichTextEditor id="propertyValueTextInput"
      htmlText="{property.string}"
      change="property.string = propertyValueTextInput.htmlText;" 
      borderThickness="3"
      fontFamily="Verdana" />
  </mx:FormItem>
</ext:Editor>
```

Modifying an editor module for multiple locales.

You must modify your editor module for custom editors to add locale support for to your SWF file.

To modify your editor module:

1. Open the MXML registry file for your editor module.

For the sample editor project, open

`editor_sdk\reference\build\maven\src\main\flex\sample_editors.mxml`.

2. Within the `<editor:EditorModule>` element, add a `<mx:Metadata>` declaration of one or more resource bundles to be included in the SWF.

For example:

```
<editor:EditorModule xmlns:mx="http://www.adobe.com/2006/mxml"
  xmlns:editor="com.endeca.tools.pagebuilder.editor.*">
  <mx:Metadata>
    [ResourceBundle("com.endeca.tools.pagebuilder.samples:SampleResources")]
  </mx:Metadata>
  <editor:registeredEditors>
    <mx:Array>
      <editor:EditorEntry
        uri="http://endeca.com/sample/2010"
        localName="RichTextEditor"
        editor="com.endeca.tools.pagebuilder.samples.editors.RichTextEditor"
      />
    </mx:Array>
  </editor:registeredEditors>
</editor:EditorModule>
```

3. Save and close the file.
4. Rebuild the editor module.

For the sample editor module, this consists of the following steps:

- a) Navigate to the %ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven directory.
- b) Build the sample editor module using the following command:

```
mvn clean install
```

The sample_editors.swf file is output to the target subdirectory.

5. Upload the editor module to your application:

- a) Navigate to your build output directory.

For example, %ENDECA_TOOLS_ROOT%\editor_sdk\reference\build\maven\target.

- b) Copy the editor SWF file to the config\editors_config\modules directory of your deployed application.

If this directory does not exist, you must create it.

- c) Navigate to the control directory of your deployed application.
- d) Run the set_editors_config script.

Specifying dependencies between editors

You can set up dependencies between editors in the same cartridge such that the behavior of one editor is dependent upon a property that is bound to another editor in the cartridge.

In a typical editor definition you can specify whether the editor is enabled via the enabled attribute. Editors are enabled by default. You can disable the editor by setting the value of enabled to false as follows:

```
<StringEditor propertyName="moreLinkText" label="Title" enabled="false"/>
```

In this case, the value of the associated property (in this case, moreLinkText) displays in the Content Details Panel but cannot be updated by the user.

The enabled status of an editor can be updated dynamically based on the value of another property by replacing the literal value of the enabled attribute (either true or false) with an expression that is evaluated to determine the editor's behavior. For example:

```
<StringEditor propertyName="moreLinkText" label="Title" enabled="{showMoreLink == true}">
```

In this case, the string editor that enables editing of the text for the "More" link is not enabled unless the property showMoreLink (which is bound to another editor) is set to true. This kind of dependency enables you to assemble complex editing interfaces out of simple property editors, without writing custom editors that contain the dependency logic.

The following example shows a template with two editors, where the enabled state of one editor depends on the value of the property bound to the other editor. In this case, the string editor is disabled by default (because the value of showMoreLink is false by default) and is not enabled until the user selects the "Enable 'More' link" option in the Boolean editor.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:xavia="http://endeca.com/schema/xavia/2010"
                  xmlns:editors="editors"
                  type="SidebarItem" id="FlatDimension">
  <!-- additional elements deleted from this example -->
  <ContentItem>
```

```

<Name>Flat Dimension Example</Name>
<Property name="showMoreLink">
  <Boolean>false</Boolean>
</Property>
<Property name="moreLinkText">
  <String>Show More Refinements...</String>
</Property>
</ContentItem>
<EditorPanel>
  <BasicContentItemEditor>
    <editors:BooleanEditor propertyName="showMoreLink"
      label="Enable 'More' link" enabled="true"/>
    <StringEditor propertyName="moreLinkText"
      label="'More' link text" enabled="{showMoreLink == true}"/>
  </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

The editor dependency expression language

The value of an attribute is treated as an expression if it is contained within {curly braces}, otherwise it is treated as a literal value.

There is no validation for editor dependency expressions within a template, however, if an expression contains syntax errors, an `InvalidExpressionError` is thrown upon initialization of the editor. Currently, the only attribute for which expressions are evaluated is the `enabled` attribute.

Allowed operands

The expression language enables you to evaluate an expression based on the value of a particular property in the same cartridge by comparing it to either a Boolean or string literal.

Operand	Description	Example expression
<i>property name</i>	The name of the property that the editor depends on. The value of this property is used during the evaluation of the expression. The property is treated as a Boolean if it is specified as the <code><Boolean></code> type, otherwise it is treated as a string.	{ sort == 'static'}
<code>true</code>	Treated as a Boolean literal.	{ <code>showMoreLink == true</code> }
<code>false</code>	Treated as a Boolean literal.	{ <code>showMoreLink == false</code> }
'string in single quotes'	Treated as a string literal.	{ <code>sort == 'static'</code> }

Allowed operators

The expression language provides the following comparison operators for use in editor dependency expressions. Order of operations are respected in editor dependency expressions.

Operator	Description	Example expression
<code>==</code>	Equality	<code>{sort == 'static'}</code>
<code>!=</code>	Inequality	<code>{sort != 'static'}</code>
<code> </code>	Logical OR	<code>{sort == 'static' sort == 'default'}</code>
<code>&&</code>	Logical AND	<code>{sort == 'price' && order == 'desc'}</code>
<i>no operator</i>	If the expression is a property name, it evaluates to the value of the property.	<code>{showMoreLink}</code>
<code>!</code>	Logical NOT	<code>{!showMoreLink}</code>
<code>(and)</code>	Groups expressions in order to enforce a particular evaluation order.	<code>{showMoreLink == true && (sort == 'static' sort == 'default')}</code>

Escaping characters when specifying editor dependencies

Because the editor dependency expressions are embedded in XML, it is important to apply the appropriate escaping to special characters within expressions.

The editor dependency language supports two different ways to escape special characters.

XML escaping

Use this style of escaping for handling characters that may lead to invalid XML.

Name	Character representation	Escape sequence	Note
ampersand	<code>&</code>	<code>&amp;</code>	Required, otherwise the XML document is invalid.
less than sign	<code><</code>	<code>&lt;</code>	Required, otherwise the XML document is invalid.
quotation mark	<code>"</code>	<code>&quot;</code>	Required. Quotation marks designate the attribute value in XML. Since the editor dependency expression is defined in an attribute value, the quotation mark must be escaped or the XML document is invalid.
greater than sign	<code>></code>	<code>&gt;</code>	Optional. While escaping this character is not required to ensure that the XML is valid, Oracle recommends that you escape the greater than

Name	Character representation	Escape sequence	Note
			sign as with the less than sign.
apostrophe or single quotation mark	'	'	Optional. Since single quotes are also used to designate string literals, you must escape single quotation marks within a string literal. However, for readability, Oracle recommends that you escape single quotes using the alternative sequence \ ' instead.

Non-XML escaping

Use this style of escaping for an alternate method of escaping the single quotation mark or for handling other special characters.

Character name	Escape sequence
single quotation mark	\ '
backslash	\ \
backspace	\ b
horizontal tab	\ t
line feed (new line)	\ n
vertical tab	\ v
form feed	\ f
carriage return	\ r

The following examples show editor dependency expressions and their escaped forms.

Unescaped expression	Escaped format
{sort == 'price' && order == 'desc'}	{sort == 'price' & order == 'desc'}
{generateMetadata == 'Don't generate'}	{generateMetadata == 'Don\'t generate'}
{welcomeMessage == 'Hello, <name>! '}	{welcomeMessage == 'Hello, <name>! '}

Enforcing a specific value when the editor is disabled

In some cases, when an editor is dynamically enabled based on the value of another property, you want to ensure that the associated property has a specific value when the editor is disabled.

Specifying a "disabled value" for an editor ensures that whenever the enabled expression evaluates to false (that is, the editor is dynamically disabled), the associated property is set to the specified value. This can be useful in the case where the value of the property associated with the editor should be set to a default value whenever the editor is disabled.

The disabled value for an editor is optional. If no value is specified and an editor is disabled, then its property retains its most recently set value (whether this was originally a default value or specified by the content administrator in Experience Manager). The content administrator cannot update the value while the editor is disabled, but it preserves the latest setting in the case that the content administrator re-enables the editor at a later point.

To specify a default value for an editor:

Specify a disabledValue attribute in the editor definition.

```
<editors:RadioGroupEditor propertyName="showDisabledRefinements"
    label="Show 'Disabled Refinements'"
    enabled="{sort == 'static'}"
    disabledValue="false"/>
```

As with default property values, ensure that the disabled value for the property meets the constraints defined by the editor. For a choice editor or radio group editor, it must be one of the options defined for the editor in a `<choice>` element. For a numeric editor, it must be between the minimum and maximum values for the editor.

The following example shows a radio group editor configured with a disabled value.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    xmlns:editors="editors"
    type="SidebarItem" id="FlatDimension">
    <!-- additional elements deleted from this example -->
    <ContentItem>
        <Name>New Text Link Flat Dimension</Name>
        <!-- additional elements deleted from this example -->
        <Property name="sort">
            <String>default</String>
        </Property>
        <Property name="showDisabledRefinements">
            <String>false</String>
        </Property>
        <!-- additional elements deleted from this example -->
    </ContentItem>
    <EditorPanel>
        <BasicContentItemEditor>
            <!-- additional elements deleted from this example -->
            <editors:ChoiceEditor label="Sorting Options" propertyName="sort">
                <choice label="Dimension default" value="default"/>
                <choice label="Alphanumeric" value="static"/>
                <choice label="By frequency" value="dynRank"/>
            </editors:ChoiceEditor>
            <editors:RadioGroupEditor propertyName="showDisabledRefinements"
                label="Show 'Disabled Refinements'"
                enabled="{sort == 'static'}"
                disabledValue="false">
                <choice label="Yes" value="true"/>
                <choice label="No" value="false"/>
            </editors:RadioGroupEditor>
            <!-- additional elements deleted from this example -->
        </BasicContentItemEditor>
```

```
</EditorPanel>
</ContentTemplate>
```

When the Content Details Panel is first instantiated, the selected value for the radio group editor is `false`, which displays with the label "No," and the editor is disabled, because the default value of the `sort` property is not equal to `static`. If the content administrator selects "Alphanumeric" from the choice editor, the radio group editor is enabled, and the content administrator can change the value of `showDisabledRefinements` to `true`. However, if the content administrator later selects a different value from the choice editor (either "Dimension default" or "By frequency"), the radio group editor is once again disabled and the value of `showDisabledRefinements` set to `false`.

Appendix A

Template Property and Editor Reference

This section describes how to define basic content properties and associated editing interfaces in Experience Manager templates.

Editor property mapping reference

This section provides an overview of which property types are associated with the different Oracle Endeca Commerce Suite editors.

Oracle Endeca Commerce Core Editors

The following core editors are included with all installations of Oracle Endeca Commerce:

Editor	Property Type	Functionality
BooleanEditor	<Boolean>	Displays as a checkbox that the content administrator selects or de-selects. Optionally, the editor may be set to a read-only state.
ChoiceEditor	<String>	Displays as a dropdown with an optional default value. The content administrator selects from a set of pre-defined values.
DynamicSlot Editor	<String>	Displays as a drop-down list for specifying a valid content collection, and a numeric stepper for setting the evaluation limit for that collection.
ImagePreview	(None)	Displays an image from a specified URL.
NumericStepperEditor	<String>	Displays as a one-line text field with a pair of arrow buttons for increasing or decreasing the value by a set amount. The content administrator inputs or adjusts the value to any number within the minimum and maximum boundaries defined in the editor.
RadioGroupEditor	<String>	Displays as a series of radio buttons with an optional default value. The content administrator selects from a set of pre-defined values.

Editor	Property Type	Functionality
RecordListEditor	<xavia:List>	 Important: This editor is deprecated. Use the <code>SpotlightSelectionEditor</code> instead. Displays as a button that launches the microbrowser and allows the content administrator to select the list of records that populates a <code><xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection"/></code> record selection property.
SliderEditor	<String>	Displays as a slider bar. The content administrator selects a value by moving the slider along specified intervals within the minimum and maximum boundaries defined in the editor.
SpotlightSelectionEditor	<xavia:Item>	Displays as a button that launches the Select Records dialog and allows the content administrator to select the navigation state or list of records that populates a <code><xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection"/></code> record selection property.
StringEditor	<String>	Displays as a text field or text area. The content administrator enters arbitrary string values. Optionally, the editor may be set to a read-only state to display a fixed, default value.

Oracle Endeca Experience Manager Editors

The following editors are included in the Oracle Endeca Experience Manager package:

Editor	Property Type	Functionality
BoostBuryEditor	<xavia:List>	Displays as a three-pane, drag-and-drop interface consisting of a central pane that lists available dimension refinements, a left pane for boosted refinements, and a right pane for buried refinements. The content administrator can filter the list of available dimensions by searching against a text string. The editor populates two <code><xavia:List></code> properties, one for boosted dimension refinements and one for buried dimension refinements.
BoostBuryRecordEditor	<xavia:List>	Displays as two panes, Boosted Records and Buried Records , each with an Edit List button that launches the Select Records dialog. The content administrator uses the Select Records dialog to populate the lists of boosted and buried records. The editor populates two <code><xavia:List></code> properties, one for boosted records and one for buried records.

Editor	Property Type	Functionality
DimensionListEditor	<xavia:List>	Displays as two panels, one with a list of available dimensions and one with a list of selected dimensions. The content administrator can drag values back and forth between the two lists.
DimensionSelectorEditor	<String>	Displays as a dropdown. The content administrator selects a value from the list of available dimensions retrieved from the MDEX Engine. The editor populates two <xavia:String> properties, one for the dimension name and one for the ID.
DimvalListEditor	<xavia:List>	Displays as two panels, one with a list of available dimension refinements and one with a list of selected refinements. The content administrator can drag values back and forth between the two lists. Additionally, the list of available refinements includes a search box for finding specific refinements in a large data set.
GuidedNavigationEditor	<ContentItemList>	Displays as a button for launching the Generate Guided Navigation wizard, which allows a content administrator to select and order a set of dimensions in order to create multiple Refinement Menu cartridges at once.
LinkBuilderEditor	<xavia:Item>	Displays two radio buttons, one for specifying an External link via a text field, and one for specifying an Internal (Relative) link. The content administrator specifies a relative link by selecting a servlet from a dropdown list, then launching the Select Records dialog to navigate to a specific record or a navigation state. The editor populates a <xavia:Item class=com.endeca.in-front.cartridge.model.LinkBuilder /> item property. For more information, see "Adding a Link Builder."
MediaEditor	<xavia:Item>	Displays as a Media URL field, with an associated preview box and Select and Clear buttons for launching the media editor or clearing the current URL. The content administrator can browse through media in the configured source repository, and generate a link to a selected asset.
RecordStratificationEditor	<xavia:List>	 Important: This editor is deprecated. Use the Boost-BuryRecordEditor instead. Displays as two panes, Boosted Records and Buried Records , each with an Edit List button that launches the microbrowser. The content administrator uses the microbrowser to populate the lists of boosted and buried records.

Editor	Property Type	Functionality
		The editor populates two <code><xavia:List></code> properties, one for boosted records and one for buried records.
RichTextEditor	<code><String></code>	Displays as a text area with a configurable formatting toolbar. The content administrator enters arbitrary string values and can include markup to add text formatting and hyperlinks.
SortEditor	<code><xavia:Item></code>	Displays as a dropdown. The content administrator selects a sort order from those configured in the editor. The editor includes multiple <code><xavia:Item class="com.endeca.infront.navigation.model.SortOption" /></code> item properties that each specify an available sort option. For more information, see "Adding a Sort editor."

Related Links

[Basic content properties](#) on page 224

Content items properties must be one of several basic types. All configuration models are composed of the same primitive property types.

[Complex property editors](#) on page 238

This section describes editors that are designed for specific aspects of Endeca feature configuration.

Editor label configuration reference

All editors share a set of common attributes that can be used to configure the appearance of the editor in Experience Manager.

When adding an editor to a template, you can configure its appearance by setting the following attributes:

Attribute	Description
label	This attribute enables you to specify a more descriptive label for the editor in Experience Manager. If no label is specified, the value of the associated <code>propertyName</code> is used by default.
labelPosition	The position of the label text. Valid values are "left" (the default) and "top".
bottomLabel	This attribute allows you to specify a descriptive label that appears below the editor.
tooltip	This attribute allows you to specify mouseover text for the editor.

Basic content properties

Content items properties must be one of several basic types. All configuration models are composed of the same primitive property types.

The basic content property types are:

- <String>
- <Boolean>
- <xavia:List>
- <xavia:Item>

The following example shows a several properties of various types.

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">

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

  <ContentItem>
    <Name>Results List</Name>
    <Property name="boostStrata">
      <xavia:List/>
    </Property>
    <Property name="buryStrata">
      <xavia:List/>
    </Property>
    <Property name="sortOption">
      <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
        <xavia:Property name="label">Most Sales</xavia:Property>
        <xavia:Property name="sorts">
          <xavia:List>
            <xavia:Item class="com.endeca.infront.navigation.model.Sort-
Spec">
              <xavia:Property name="key">product.analytics.to-
tal_sales</xavia:Property>
              <xavia:Property name="descending">false</xavia:Property>
            </xavia:Item>
          </xavia:List>
        </xavia:Property>
      </xavia:Item>
    </Property>
    <Property name="relRank">
      <!-- Margin Bias -->
      <String>nterms,maxfield,exact,static(product.analytics.conversion_rate,de-
scending)</String>
    </Property>
    <Property name="recordsPerPage">
      <String>10</String>
    </Property>
  </ContentItem>
  <!-- additional elements omitted from this example -->
</ContentTemplate>

```

Adding a string property

String properties are very flexible and can be used to specify information such as text to display on a page, URLs for banner images, or meta keywords for search engine optimization.

To add a string property to a template:

1. Insert a `<String>` element inside a `<Property>` element.
2. Optionally, specify the default value for the property as the content of the `<String>` element.

The following example shows a variety of string properties:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    xmlns:editors="editors"
    type="SidebarContent">
    <!-- additional elements omitted from this example -->
    <ContentItem>
        <Name>Dimension Navigation</Name>
        <Property name="dimensionName">
            <String/>
        </Property>
        <Property name="dimensionId">
            <String/>
        </Property>
        <Property name="sort">
            <String>default</String>
        </Property>
        <Property name="showMoreLink">
            <Boolean>false</Boolean>
        </Property>
        <Property name="moreLinkText">
            <String>Show More Refinements...</String>
        </Property>
        <Property name="numRefinements">
            <String>10</String>
        </Property>
        <Property name="maxNumRefinements">
            <String>200</String>
        </Property>
        <!-- additional elements omitted from this example -->
    </ContentItem>
    <!-- additional elements omitted from this example -->
</ContentTemplate>

```

Adding a string editor

You add a string editor to enable configuration of string properties. The string editor displays in the Experience Manager interface as a text field or text area depending on the configuration.

String editors enable content administrators to supply arbitrary values for a string property. If you want to constrain the input to a specific enumeration of values, use a choice editor.

To add a string editor to a template:

1. Insert an `<StringEditor>` element within `<BasicContentItemEditor>`.
2. Specify label attributes and additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the string property that this editor is associated with. This property must be declared in the same template as the string editor.
<code>enabled</code>	If set to <code>false</code> , this attribute makes the property read-only so that the value of the property displays in the Content Details Panel in Experience Manager, but cannot be

Attribute	Description
	edited. Set this to <code>false</code> only if you specify a default value in the definition of the string property. Editors are enabled by default.
<code>width</code>	The width in pixels of the text field presented in the Experience Manager interface. The default width is 100% and scales with the screen width.
	 Note: You cannot specify a percent value in your editor configuration. You must specify the editor width in pixels.
<code>height</code>	The height in pixels of the text field presented in the Experience Manager interface. The default height for a single-row field is 24 pixels. Setting the value to 34 pixels or higher creates a multiline text area with word wrap enabled.

The following example shows a variety of editing options for string properties:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
    type="ResultsPage">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Three-Column Navigation Page</Name>
    <Property name="title">
      <String>Discover Electronics</String>
    </Property>
    <Property name="metaKeywords">
      <String>camera cameras electronics</String>
    </Property>
    <Property name="metaDescription">
      <String>Endeca eBusiness reference application.</String>
    </Property>
    <!-- additional elements omitted from this example -->
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <GroupLabel label="Page metadata"/>
      <editors:StringEditor propertyName="title" label="Title" enabled="true" />
      <editors:StringEditor propertyName="metaKeywords" label="Meta keywords" enabled="true" height="72" />
      <editors:StringEditor propertyName="metaDescription" label="Meta description" enabled="true" height="72" />
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

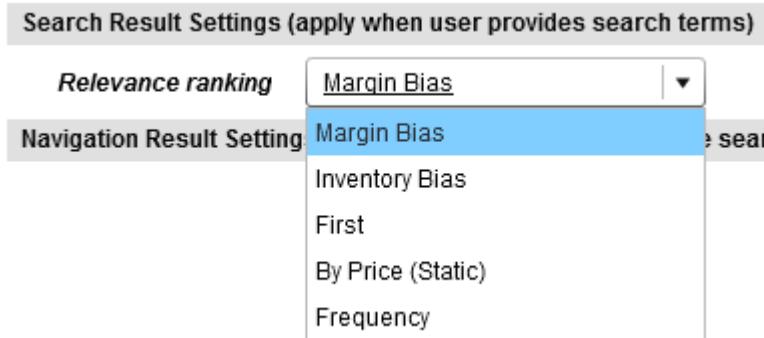
```

 **Note:** Neither Experience Manager nor the Assembler applies HTML escaping to strings. This enables content administrators to specify HTML formatted text in Experience Manager and have it rendered appropriately. If you intend to treat a string property as plain text, be sure to add HTML escaping to your application logic in order to avoid invalid characters and non-standards-compliant HTML.

Adding a choice editor

A choice editor enables the user to select from predefined string values for a property that are presented in a drop-down list.

Choice editors affect the value of a string property. For example, you might use a choice editor to provide sorting options for dimension values in a Guided Navigation cartridge:



To add a choice editor:

1. Insert an `<editors:ChoiceEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the string property that this editor is associated with. This property must be declared in the same template as the choice editor.
<code>editable</code>	If set to <code>true</code> , this attribute allows Experience Manager users to specify custom string values. By default, choice editors are not editable.
<code>enabled</code>	If set to <code>false</code> , the choice editor displays in Experience Manager but the value cannot be changed by the user. By default, choice editors are enabled.
<code>prompt</code>	Specifies a custom prompt. The default prompt is an empty string.
<code>tooltip</code>	If present, specifies optional help text to display in a tool tip window. The default behavior is no tool tip.
<code>width</code>	The width, in pixels, of the choice editor. By default, the width of the editor adjusts to fit the longest choice in the editor. Use this attribute if you want to set a fixed width for the editor.

3. Specify one or more menu options for the choice editor by adding `<choice>` elements. `<choice>` takes the following attributes:

Attribute	Description
<code>value</code>	Required. The string value to assign to the associated property if this <code><choice></code> is selected.

Attribute	Description
label	This attribute allows you to specify a more descriptive label for this option in the drop down list. If no label is specified, the value is used by default. You must either specify a <code>label</code> for all of the choices or none of them. You cannot have labels for some choices and not others.



Note: If you choose to make a choice editor editable (so that users can enter arbitrary strings), you should not use the `label` attribute for choices. Instead, the choice editor should display the raw value of the string so that users entering custom values can see the expected format of the string property.

4. Optionally, set a default value in the corresponding `<ContentItem>` property.

For example, to specify the default sort order for a dimension as the default choice for a choice editor with `propertyName="sort"`:

```
<Property name="relrank">
  <!-- Margin Bias -->
  <String>nterms,maxfield,exact,static(product.analytics.conversion_rate,de-
scending)</String>
</Property>
```



Note: Ensure that the default value for the property is one of the options defined for the choice editor in a `<choice>` element.

The following example shows a choice editor configured with a default value. The selected value when the editor is first instantiated is `nterms,maxfield,exact,static(product.analytics.conver-` `sion_rate,descending)`, which displays with the label "Margin Bias" in the drop-down menu. Content administrators can select a different sort order.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
<ContentItem>
  <Name>Results List</Name>
  <!-- additional elements omitted from this example -->
  <Property name="relrank">
    <!-- Margin Bias -->
    <String>nterms,maxfield,exact,static(product.analytics.conversion_rate,de-
scending)</String>
  </Property>
  <!-- additional elements omitted from this example -->
</ContentItem>

<EditorPanel>
  <BasicContentItemEditor>
    <!-- additional elements omitted from this example -->
    <GroupLabel label="Search Result Settings (apply when user provides search
terms)" />
    <editors:ChoiceEditor propertyName="relrank" label="Relevance ranking">
      <choice label="Margin Bias" value="nterms,maxfield,exact,static(prod-
uct.analytics.conversion_rate,descending)" />
      <choice label="Inventory Bias" value="nterms,maxfield,exact,static(prod-
```

```

uct.inventory.count,descending)" />
    <choice label="First" value="first" />
    <choice label="By Price (Static)" value="static(product.price)" />
    <choice label="Frequency" value="freq" />
</editors:ChoiceEditor>
<!-- additional elements omitted from this example -->
</BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

Adding a radio group editor

A radio group editor is similar to the choice editor in that it enables the user to select from predefined string values for a property. The choices are presented as a set of radio button controls.

Although radio buttons are often used for binary choices such as yes/no, the radio group editor can be used for any scenario where the user must specify exactly one value out of a number of options. In order to enable the more general use case, the radio group editor affects the value of a string property.

To add a radio group editor:

1. Insert an `<editors:RadioGroupEditor>` element within `<BasicContentItemEditor>`.
2. Specify label attributes and the additional attributes for the editor:

Attribute	Description
-----------	-------------

<code>propertyName</code>	Required. The name of the string property that this editor is associated with. This property must be declared in the same template as the choice editor.
---------------------------	--

<code>enabled</code>	If set to <code>false</code> , the radio group editor displays in Experience Manager but the value cannot be changed by the user. By default, radio group editors are enabled.
----------------------	--

3. Specify one or more radio button options by adding `<choice>` elements. `<choice>` takes the following attributes:

Attribute	Description
-----------	-------------

<code>value</code>	Required. The string value to assign to the associated property if this <code><choice></code> is selected.
--------------------	--

<code>label</code>	This attribute allows you to specify a more descriptive label for the radio button associated with this option. If no label is specified, the <code>value</code> is used by default.
--------------------	--

4. Optionally, set a default value in the corresponding `<ContentItem>` property.

For example, to specify the default value for a radio group editor with `propertyName="showDisabledRefinements"`:

```

<Property name="showDisabledRefinements">
    <String>false</String>
</Property>

```



Note: Ensure that the default value for the property is one of the options defined for the editor in a `<choice>` element.

The following example shows a radio group editor configured with a default value. The selected value when the editor is first instantiated is `false`, which displays with the label "No."

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:xavia="http://endeca.com/schema/xavia/2010"
                  xmlns:editors="editors"
                  type="SidebarItem">
    <!-- additional elements omitted from this example -->
    <ContentItem>
        <!-- additional elements omitted from this example -->
        <Property name="showDisabledRefinements">
            <String>false</String>
        </Property>
        <!-- additional elements omitted from this example -->
    </ContentItem>
    <EditorPanel>
        <BasicContentItemEditor>
            <!-- additional elements omitted from this example -->
            <editors:RadioGroupEditor propertyName="showDisabledRefinements"
                label="Show 'Disabled Refinements'" enabled="true">
                <choice label="Yes" value="true"/>
                <choice label="No" value="false"/>
            </editors:RadioGroupEditor>
            <!-- additional elements omitted from this example -->
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

About numeric properties

Numeric properties should be specified as string properties in the template.

Properties that are expected to have numeric values can be associated with editors that are designed to work with numbers. These editors guarantee that the property is assigned a numeric value.

Adding a numeric stepper

A numeric stepper enables content administrators to select a numeric value from a set of possible values by stepping through values or typing into an input field.

The numeric stepper provides a single-line input text field and a pair of arrow buttons for stepping through values. If a user enters number that is not a multiple of the `stepSize` property or is not in the range between the `maximum` and `minimum` properties, this property is set to the nearest valid value.

To add a numeric stepper to a template:

1. Insert an `<editors:NumericStepperEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the string property that this editor is associated with. This property must be declared in the same template as the string editor.
<code>width</code>	The width, in pixels, of the editor. The default width is 60.
<code>height</code>	The height, in pixels, of the editor. The default height is 24.

Attribute	Description
minValue	The minimum value of the property bound to this editor. The <code>minValue</code> can be any number, including a fractional value. The default minimum value is 0.
maxValue	The maximum value of the property bound to this editor. The <code>maxValue</code> can be any number, including a fractional value. The default maximum value is 10.
stepSize	The increment by which the property value is increased or decreased when a user clicks on the up or down arrows. The value must be a multiple of this number. The default step size is 1.

The following example shows the configuration for a numeric stepper:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    xmlns:editors="editors"
    type="SidebarContent">
    <!-- additional elements omitted from this example -->
    <ContentItem>
        <Name>Dimension Navigation</Name>
        <!-- additional elements omitted from this example -->
        <Property name="numRefinements">
            <String>10</String>
        </Property>
        <Property name="maxNumRefinements">
            <String>200</String>
        </Property>
        <!-- additional elements omitted from this example -->
    </ContentItem>
    <EditorPanel>
        <BasicContentItemEditor>
            <!-- additional elements omitted from this example -->
            <editors:NumericStepperEditor propertyName="numRefinements"
                label="Max. Refinements" maxValue="10000" enabled="true" />
            <!-- additional elements omitted from this example -->
            <editors:NumericStepperEditor propertyName="maxNumRefinements"
                label="'More' Max. Refinements" maxValue="100000" enabled="true" />
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

Adding a slider

A slider enables content administrators to select a numeric value by moving a slider between predefined endpoint values.

The current value of the slider is determined by the relative location of the thumb between the end points of the slider, corresponding to the slider's minimum and maximum values.

To add a slider to a template:

1. Insert an `<editors:SliderEditor>` element within `<BasicContentItemEditor>`.
2. Specify label attributes and additional attributes for the editor:

Attribute	Description
propertyName	Required. The name of the string property that this editor is associated with. This property must be declared in the same template as the string editor.
width	The width, in pixels, of the editor. The default width is 160.
height	The height, in pixels, of the editor. The default height is 36.
default	The default position of the slider thumb. By default, the thumb is set to 0.
minValue	The minimum value of the property bound to this editor. The minValue can be any number, including a fractional value. The default minimum value is 0.
maxValue	The maximum value of the property bound to this editor. The maxValue can be any number, including a fractional value. The default maximum value is 10.
snapInterval	Specifies the increment value of the slider thumb as the user moves the thumb. A value of 0 means that the slider moves continuously between the minimum and maximum values. The default value is 0.
tickInterval	The spacing of the tick marks. A value of 0 displays no tick marks. The default value is 0.
precision	Number of decimal places to use for the property value and data tip text. A value of 0 means all values are rounded to the nearest integer. The default value is 0.
labels	An array of strings to use for the slider labels. These labels display at the beginning and end of the track and, if there are more than two values, spaced evenly between the two ends. By default, the beginning and end of the slider track are labeled in Experience Manager with the minimum and maximum values.

The following example shows the configuration for a slider:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:editors="editors" type="SidebarItem">
    <!-- additional elements omitted from this example -->
    <!-- Define the content properties -->
    <ContentItem>
        <!-- additional elements omitted from this example -->
        <!-- define numeric properties as simple string properties -->
        <Property name="numRefinements">
            <String>10</String>
        </Property>
    </ContentItem>
    <!-- Define editors for numeric properties -->
    <EditorPanel>
        <BasicContentItemEditor>
            <!-- additional elements omitted from this example -->
            <editors:SliderEditor propertyName="numRefinements"
                label="Number of refinements" minValue="10" maxValue="30"
                snapInterval="5" tickInterval="5" labels="10,15,20,25,30"/>
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

Adding a Boolean property

Boolean properties represent a true or false value and can be used to enable or disable features in your application.

To add a Boolean property to a template:

1. Insert a `<Boolean>` element inside a `<Property>` element.
2. Optionally, you can specify the default value for the property.

```
<Property name="eligibleFreeShipping">
  <Boolean>true</Boolean>
</Property>
```

Any value other than the string "true" (case insensitive) defaults to a value of false.

The following example shows the configuration of a Boolean property:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  type="HeaderContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Search Box</Name>
    <Property name="autoSuggestEnabled">
      <Boolean>false</Boolean>
    </Property>
    <!-- additional elements omitted from this example -->
  </ContentItem>
  <!-- additional elements omitted from this example -->
</ContentTemplate>
```

Adding a Boolean editor

A Boolean editor provides a checkbox for Experience Manager users to specify the value of a Boolean property.

To add a Boolean editor:

1. Insert a `<editors:BooleanEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the Boolean property that this editor is associated with. This property must be declared in the same template as the Boolean editor.
<code>enabled</code>	If set to <code>false</code> , the checkbox displays in Experience Manager but the value cannot be changed by the user. By default, checkboxes are enabled.

The following example illustrates a checkbox for specifying whether auto-suggest search results should be enabled, with a default value of false:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  type="HeaderContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Search Box</Name>
```

```

<Property name="autoSuggestEnabled">
    <Boolean>false</Boolean>
</Property>
<!-- additional elements omitted from this example -->
</ContentItem>
<EditorPanel>
    <BasicContentItemEditor>
        <GroupLabel label="Auto-Suggest Configuration"/>
        <editors:BooleanEditor propertyName="autoSuggestEnabled"
            label="Enable Auto-Suggest"
            enabled="true"/>
        <!-- additional elements omitted from this example -->
    </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

Adding a list property

A property can consist of an ordered list of strings, Booleans, items, or other lists.

Because lists can be used for a variety of purposes, Oracle Endeca Guided Search does not include any generic editors for working with lists. However, editors intended for specific purposes may store their values in list properties.

To add a list property to a template:

1. Insert a `<xavia>List>` element inside a `<Property>` element.
2. Optionally, specify a default value by inserting either `<String>`, `<Boolean>`, `<xavia>List>`, or `<xavia:Item>` elements.

Following is an example of a template that uses lists both with and without default values:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:editors="editors"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    type="MainContent">
    <!-- additional elements omitted from this example -->
    <ContentItem>
        <Name>Results List</Name>
        <Property name="boostStrata">
            <xavia>List/>
        </Property>
        <Property name="buryStrata">
            <xavia>List/>
        </Property>
        <Property name="sortOption">
            <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
                <xavia:Property name="label">Most Sales</xavia:Property>
                <xavia:Property name="sorts">
                    <xavia>List>
                        <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
                            <xavia:Property name="key">product.analytics.total_sales</xavia:Property>
                        </xavia:Item>
                    </xavia>List>
                </xavia:Property>
            </xavia:Item>
        </Property>
    </ContentItem>
</ContentTemplate>

```

```

</xavia:Item>
</Property>
<!-- additional elements omitted from this example -->
</ContentItem>
<!-- additional elements omitted from this example -->
</ContentTemplate>

```

Adding an item property

A property can consist of a collection of properties (key-value pairs) of any valid type.

Because item properties can be used for a variety of purposes, InFront does not include any generic editors for working with items. However, editors intended for specific purposes may store their values in item properties.

To add an item property to a template:

1. Insert a `<xavia:Item>` element inside a `<Property>` element.
2. Specify the `class` attribute with the fully qualified class name of the configuration model class that corresponds to this item property.
3. Optionally, specify a default value by inserting a `<xavia:Property>` of type `<String>`, `<Boolean>`, `<xavia>List>`, or `<xavia:Item>`. (A `<Property>` with no type specified is treated as a string by default.)



Note: Properties defined within `<xavia:Item>` must declare the Xavia namespace (i.e., `<xavia:Property>` instead of `<Property>`).

Following is an example of a template that uses an item with a default value:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
<!-- additional elements omitted from this example -->
<ContentItem>
  <Name>Results List</Name>
  <!-- additional elements omitted from this example -->
  <Property name="sortOption">
    <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
      <xavia:Property name="label">Most Sales</xavia:Property>
      <xavia:Property name="sorts">
        <xavia>List>
          <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
            <xavia:Property name="key">product.analytics.total_sales</xavia:Property>
          <xavia:Item>
            <xavia:Property name="ascending">true</xavia:Property>
          </xavia:Item>
        </xavia>List>
      </xavia:Property>
    </xavia:Item>
  </Property>
  <!-- additional elements omitted from this example -->
</ContentItem>

```

```
<!-- additional elements omitted from this example -->
</ContentTemplate>
```

Following is an example of a template that uses an item without a default value:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:editors="editors"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    type="SidebarContent">
    <!-- additional elements omitted from this example -->
    <ContentItem>
        <!-- additional elements omitted from this example -->
        <Property name="recordSelection">
            <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection" />
        </Property>
        <!-- additional elements omitted from this example -->
    </ContentItem>
    <!-- additional elements omitted from this example -->
</ContentTemplate>
```

Adding a group label

In the Experience Manager interface, group labels can serve as a visual cue that several properties are related.

Group labels are only used to provide additional context in the editing interface of Experience Manager and do not affect rendering in the front-end application. Group labels are optional.

One use of group labels is to give the content administrator information about properties that they need to configure the cartridge. For example, if a template defines properties that are required in order to render the content properly, you can indicate these with a descriptive group label so that the content administrator can easily identify the required fields in Experience Manager.

The editor panel in Experience Manager includes a default heading of "Section settings." This heading includes the required Name field and the read-only type of a template, as well as any properties that are defined before the first group label.

To add a group label to the editor panel:

Insert the `<GroupLabel>` element inside `<BasicContentItemEditor>` as in the following example:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:editors="editors"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    type="SidebarContent">
    <!-- additional elements omitted from this example -->
    <EditorPanel>
        <BasicContentItemEditor>
            <GroupLabel label="Define Spotlight"/>
            <editors:StringEditor propertyName="title" label="Spotlight Title"
enabled="true"/>
            <editors:StringEditor propertyName="maxNumRecords" label="Max Number
Of Records" enabled="false"/>
            <editors:RecordListEditor propertyName="recordSelection" label="Spot-
light Records">
                <PreviewProperty name="product.name" />
            </editors:RecordListEditor>
            <editors:StringEditor propertyName="seeAllLink" label="See All
Link" />
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>
```

```

        Link" enabled="true"/>
    </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

<GroupLabel> is an empty tag that allows you to specify the label text with the `label` attribute.

Complex property editors

This section describes editors that are designed for specific aspects of Endeca feature configuration.

About the microbrowser

The microbrowser is used in several editors in the core cartridges to enable a content administrator to specify a set of records. It is deprecated in this release; use the **Select Records** dialog instead.



Important: The microbrowser is deprecated. Use the Select Records dialog instead.

The microbrowser is a lightweight search and Guided Navigation application that enables a content administrator to browse to a particular location in the data set (which may include search terms, dimension refinements, or a combination of both). The content administrator can then do one of two things:

- Save the current filter state to designate a dynamic set of records.
- Select specific records from that filter state (or other filter states) to designate a set of specific featured records.

An instance of a microbrowser is usually bound to a list property, which contains items that represent either refinements or record IDs.

The microbrowser communicates with the MDEX Engine to retrieve search and navigation results.



Note: In order to enable the microbrowser, ensure that you have enabled communication between Experience Manager and the MDEX Engine. For instructions, see "Communicating with the MDEX Engine" in the *Tools and Frameworks Installation Guide*.

Data service configuration reference

The microbrowser uses a data service to access MDEX Engine information. By default, the service is configured to provide relevant record properties for the Discover Electronics reference application.



Important: The microbrowser is deprecated. Use the Select Records dialog instead.

The data service configuration file, `<app dir>\config\editors_config\services\dataservice.json`, is shown below:

```
{
    "jcr:primaryType": "endeca:unstructured",
    "host": "myhost.mydomain.com",
    "port": "15002",
    "recordSpecName": "common.id",
    "aggregationKey": "product.code",
    "recordFilter": "",
    "wildcardSearchEnabled": false,
```

```

"recordNameField": "",
"fields": {
    "product.id": "",
    "product.name": "plain",
    "product.price": "currency",
    "product.short_desc": ""
}
}

```

It specifies the following:

Key	Value
host	The hostname or IP address of your MDEX Engine server. By default, this is populated with the same host as the authoring MDEX Engine when you deploy the Discover Electronics reference application and run the <code>initialize_services</code> script.
port	The port that the MDEX Engine server listens on. By default, this is populated with the same port as the authoring MDEX Engine.
recordSpecName	The dimension used as the record specifier. This must be a unique identifier.
aggregationKey	Optional. Enables aggregated records mode in the microbrowser, using the specified property or dimension as the aggregation key when displaying and sorting records. All records with the same value in the selected dimension or property are treated as a single record.
recordFilter	Optional. The property used to filter records for record boost and bury.
wildcardSearchEnabled	Optional. Wildcard search is enabled by default. If your configuration does not index dimensions by wildcard index, you must explicitly set this property to <code>false</code> .
recordNameField	Optional. The property that should be used to represent the name of a record.
fields	Each key in the array of key/value pairs specifies a property or dimension to display as a column in the microbrowser. Optionally, you may specify a formatting value from among the following: <ul style="list-style-type: none"> • plain — no formatting. Used as the default if no format value is present. • currency — adds a dollar (\$) symbol before the value. • integer — removes the decimal point and any trailing digits, if present. This setting does not round the integer value. • html — attempts to handle markup tags within the content returned from the MDEX Engine.

Running `<app dir>\control\set_editors_config` pushes changes to the Discover Electronics reference application.

About the Select Records dialog

The **Select Records** dialog is used in several editors in the core cartridges to enable a content administrator to specify a set of records.

The **Select Records** dialog is a lightweight search and Guided Navigation application that enables a content administrator to browse to a particular location in the data set (which may include search terms, dimension refinements, or a combination of both). The content administrator can then do one of two things:

- Save the current filter state to designate a dynamic set of records.
- Select specific records from that filter state (or other filter states) to designate a set of specific featured records.

An instance of a **Select Records** dialog is usually bound to a `<List>` property in a cartridge template, which contains `<Item>` properties that represent either dimension refinements or record IDs. The dialog communicates with the MDEX Engine to retrieve search and navigation results.



Note: In order to enable the **Select Records** dialog, ensure that you have enabled communication between Experience Manager and the MDEX Engine. For instructions, see "Communicating with the MDEX Engine" in the *Tools and Frameworks Installation Guide*.

The following editors launch the **Select Records** dialog:

- Link Builder editor
- Boost-Bury Record editor
- Spotlight Selection editor

Select Records data service configuration reference

The Select Records dialog in Experience Manager communicates with the MDEX Engine through a configurable data service. By default, the service is configured to provide relevant record properties for the Discover Electronics reference application.

The configuration file, `<app dir>\config\editors_config\services\endecaBrowserService.json`, is shown below:

```
{
  "host": "myhost.mydomain.com",
  "port": "15002",
  "recDisplayNameProp" : "product.name",
  "recSpecProp": "common.id",
  "recAggregationKey": "product.code",
  "recFilter": "",
  "recImgUrlProp" : "product.img_url_thumbnail",
  "recDisplayProps": [ "product.name", "product.price", "product.short_desc"
],
  "textSearchKey": "All",
  "textSearchMatchMode" : "ALLPARTIAL"
}
```

It specifies the following:

Key	Value
host	The hostname or IP address of your MDEX Engine server. By default, this is populated with the same host as the authoring MDEX Engine when you deploy the Discover Electronics reference application and run the <code>initialize_services</code> script.

Key	Value
port	The port that the MDEX Engine server listens on. By default, this is populated with the same port as the authoring MDEX Engine.
recDisplayNameProp	The dimension used as the record display name in the editor that launches the dialog.
recSpecProp	The dimension used as the record specifier. This must be a unique identifier.
recAggregationKey	Optional. Enables aggregated records mode in the Select Records dialog using the specified property or dimension as the aggregation key when displaying and sorting records. All records with the same value in the selected dimension or property are treated as a single record.
recFilter	Optional. The property used to filter records for record boost and bury.
recImgUrlProp	Optional. The property used to retrieve the URL for the record thumbnail image.
recDisplayProps	An array of record properties to display in the dialog.
textSearchKey	Optional. Specifies the search key to apply to text searches in the Select Records dialog.
textSearchMatchMode	Optional. Specifies the match mode to apply to text searches in the Select Records dialog.

You can modify these values as necessary for your own application. Running `<app dir>\control\set_editors_config` pushes changes to the Discover Electronics reference application.

About the Dynamic Slot editor

The Dynamic Slot editor enables the content administrator to configure a section of an application page at query time by specifying one or more content folders from which to return content.

The editor has no associated template configuration, although it launches a configuration dialog in Experience Manager. When the content administrator edits the cartridge in Experience Manager, the editor queries the Endeca Configuration Repository for a list of content folders. These results are refined based on the template type or template ID restrictions entered by the content administrator.

Creating a cartridge template with a dynamic slot

You should configure a separate cartridge template for each template type that requires dynamic slot functionality.

To create a cartridge template with a dynamic slot:

1. Insert a `ContentItem` that includes the following properties:

- `ruleLimit`

- **contentPaths** — Include a nested `<xavia>List/>` element.
- **templateTypes** — Include a nested `<xavia>List/>` element.
- **templateIds** — Include a nested `<xavia>List/>` element.

For example:

```
<ContentItem>
  <Name>Sidebar Slot</Name>
  <Property name="ruleLimit"/>
  <Property name="contentPaths"><xavia>List/></Property>
  <Property name="templateTypes"><xavia>List/></Property>
  <Property name="templateIds"><xavia>List/></Property>
</ContentItem>
```

These properties are sent in as configuration to a `ContentSlotConfig` object that dynamically populates the page with a suitable content item. For more information, see [About ContentInclude and ContentSlotConfig objects](#) on page 23.

2. Add any default values.

For example:

```
<ContentItem>
  <Name>Sidebar Slot</Name>
  <Property name="ruleLimit"><String>1</String></Property>
  <Property name="contentPaths"><xavia>List/></Property>
  <Property name="templateTypes">
    <xavia>List>
      <xavia:String>SecondaryContent</xavia:String>
    </xavia>List>
  <Property name="templateIds"><xavia>List/></Property>
</ContentItem>
```

3. In the `EditorPanel`, insert an `<editors:DynamicSlotEditor>` element within a `<BasicContentItemEditor>`:

```
<EditorPanel>
  <BasicContentItemEditor>
    <editors:DynamicSlotEditor/>
  </BasicContentItemEditor>
</EditorPanel>
```

4. Save and close the template.

5. Upload the template to your application:

- a) Navigate to your `<app dir>\control` directory.

For the Discover Electronics reference application, this is `C:\Endeca\apps\Discover\control` on Windows, or `/usr/local/endeca/apps/discover/control` on UNIX.

- b) Run the `set_templates` batch or shell script.



Note: You must configure a cartridge handler for your template in order to use it in Experience Manager.

The following shows the sample template in the Discover Electronics application for a sidebar dynamic slot cartridge. The slot is restricted to cartridges of type `SecondaryContent`.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
```

```

xmlns:editors="editors"
type="SecondaryContent">

    <Description>${template.description}</Description>
    <ThumbnailUrl>thumbnail.png</ThumbnailUrl>

    <ContentItem>
        <Name>Secondary Content Slot</Name>
        <Property name="contentPaths"><xavia>List/></Property>
        <Property name="templateTypes">
            <xavia>List>
                <xavia:String>SecondaryContent</xavia:String>
            </xavia>List>
        </Property>
        <Property name="templateIds"><xavia>List/></Property>
        <Property name="ruleLimit"><String>1</String></Property>
    </ContentItem>

    <EditorPanel>
        <BasicContentItemEditor>
            <editors:DynamicSlotEditor/>
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

You must specify a cartridge handler for each cartridge template that you configure as a dynamic slot.

Specifying a cartridge handler for a dynamic slot template

All dynamic slot cartridges can share the same cartridge handler, but each unique cartridge must be explicitly configured to do so.

Once you have created a cartridge template that uses a dynamic slot, you must register a cartridge handler for that template. This cartridge handler should inherit the `CartridgeHandler_ContentSlot` handler.

To specify a cartridge handler for a dynamic slot template:

1. Open the configuration file for your application framework.

In the Discover Electronics reference application, this is the Spring context configuration file located in `%ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring\WEB-INF\assembler-context.xml`.

2. Configure a cartridge handler for your template that inherits or extends the `ContentSlotHandler`.

In the Spring implementation of the Assembler, this consists of adding a new `CartridgeHandler` bean for your dynamic slot cartridge:

- a) Set the `id` attribute to `CartridgeHandler_<template_id>`.
- b) Set the `parent` attribute to the `CartridgeHandler_ContentSlot` handler.
- c) Set the `scope` attribute to `prototype` to instantiate a new handler each time one is required.

This results in configuration similar to the following:

```
<bean id="CartridgeHandler_MyPageSlot" parent="CartridgeHandler_ContentSlot"
      scope="prototype" />
```

3. Repeat as necessary for any other dynamic slot templates in your application.
4. Save and close the file.

Adding a Link Builder

The Link Builder editor allows the content administrator to specify a link to a static page, a single selected record, or a navigation state.

The Link Builder uses the **Select Records** dialog to enable the content administrator to browse to a single record or a particular navigation state in the data set (which may include search terms, dimension refinements, or a combination of both). Alternately, the Link Builder also supports entering an absolute URL to a static resource.

To add a Link Builder to a template:

1. Insert an `Item` property named `link`, of class `com.endeca.infront.cartridge.model.LinkBuilder`, as in the following example:

```
<Property name="link">
  <Item class="com.endeca.infront.cartridge.model.LinkBuilder"
  xmlns="http://endeca.com/schema/xavia/2010">
    </Item>
  </Property>
```

2. Within the `Item` property, insert three empty `Property` elements named `path`, `linkType`, and `queryString`:

```
<Property name="link">
  <Item class="com.endeca.infront.cartridge.model.LinkBuilder"
  xmlns="http://endeca.com/schema/xavia/2010">
    <Property name="path"></Property>
    <Property name="linkType"></Property>
    <Property name="queryString"></Property>
  </Item>
</Property>
```

These properties are populated by the Select Records dialog and processed by the cartridge handler into an action string.

3. Insert a corresponding `<editors:LinkBuilderEditor>` element within `<BasicContentItemEditor>`.
4. Specify the `propertyName` attribute for the editor:

```
<editors:LinkBuilderEditor propertyName="link" enabled="true" />
```

5. Specify any additional label attributes for the editor:

```
<editors:LinkBuilderEditor propertyName="link" label="Link Destination" enabled="true" />
```

The following shows an example of a template that includes a link builder editor:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Media banner</Name>
    <!-- additional elements omitted from this example -->
    <Property name="link">
      <Item class="com.endeca.infront.cartridge.model.LinkBuilder"
      xmlns="http://endeca.com/schema/xavia/2010">
        <Property name="path"></Property>
        <Property name="linkType"></Property>
```

```

<Property name="queryString"></Property>
</Item>
</Property>
<!-- additional elements omitted from this example -->
</ContentItem>

<EditorPanel>
  <BasicContentItemEditor>
    <!-- additional elements omitted from this example -->
    <GroupLabel label="Link Settings"/>
    <editors:LinkBuilderEditor propertyName="link" label="Link Destination"
enabled="true"/>
    <!-- additional elements omitted from this example -->
  </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

About configuring the Link Builder

The Link Builder must be configured with a path to a data service in order to display the **Select Records** dialog.

Below is the configuration for the Link Builder in the editor configuration file for the Discover Electronics reference application, located at <app dir>\config\editors_config\editors.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- additional elements removed from this example -->
<EditorConfig xmlns="http://endeca.com/schema/editor-config/2010">
  <EditorModule url="/ifcr/tools/pbx/modules/editors.swf">
    <!-- additional elements removed from this example -->
    <Editor name="editors:LinkBuilderEditor">
      <EditorConfig resourcePath="/configuration/tools/xmgr/services/endecaBrowserService.json" />
    </Editor>
    <!-- additional elements removed from this example -->
  </EditorModule>
  <GlobalEditorConfig></GlobalEditorConfig>
</EditorConfig>

```

To publish and view changes to the editor configuration file, run the <app dir>\control\set_editors_config script and clear your browser cache.

Deprecated configuration

The Link Builder formerly supported multiple nested configuration properties that applied to all instances of the editor in an application. This configuration model is deprecated in the current release:

Property	Description
host	The hostname or IP address of the MDEX Engine server to use for the Select Records dialog.
port	The port on which the specified MDEX Engine server listens.
spec	The name of the property that serves as the record spec in the data set. This must be a unique identifier.
searchKey	The name of a property, dimension, or search interface against which searches are performed.

Property	Description
rollupKey	The rollup key (used for aggregated records) to apply to all queries made via this MDEX.
matchMode	The match mode to use for text searches. Valid values for this property follow the syntax of URL parameters for search mode, without the mode+match prefix.
imgUrlProperty	The property that specifies the location of the thumbnail image for a record.
properties	A comma separated list of record properties that display for each record returned by the content administrator's search and navigation state in the link builder Assembler application.

Specifying a path to a data service overrides these settings.

Related Links

[Select Records data service configuration reference](#) on page 240

The Select Records dialog in Experience Manager communicates with the MDEX Engine through a configurable data service. By default, the service is configured to provide relevant record properties for the Discover Electronics reference application.

[About the Select Records dialog](#) on page 240

The **Select Records** dialog is used in several editors in the core cartridges to enable a content administrator to specify a set of records.

About the Media editor

The Media editor allows the content administrator to select and link to media assets stored in a content repository.

The media editor consists of an Experience Manager editor and a lightweight Web application that enables the content administrator to browse and navigate across a set of media assets in order to more easily find specific files.

The default Discover Electronics reference application stores media directly in the Endeca Configuration Repository and uses a built-in asset browser to present these assets to the content administrator. You may also initialize an MDEX Engine to index media asset metadata and URIs as records, making them available for Guided Navigation in an enhanced Media Browser.



Note: The configuration repository provides an acceptable store for media files when used for preview purposes in an authoring environment, but Oracle recommends serving media assets from a media or content delivery server for production environments.

About the Media Browser

The default asset browser for the Media editor can only be configured to browse media assets in the Endeca Configuration Repository. If you are using another system for managing media assets, you must stand up a corresponding media MDEX Engine and enable the Media Browser in the editor configuration file.

Adding a Media editor

A Media editor allows a content administrator to link media into a cartridge. It can be combined with the Link Builder in order to create images that link to destinations in your application, such as those used in site banners.

To add a Media editor to a template:

1. Insert an `Item` property named `media`, of class `com.endeca.infront.cartridge.model.MediaObject`, as in the following example:

```
<Property name="media">
  <Item class="com.endeca.infront.cartridge.model.MediaObject"
  xmlns="http://endeca.com/schema/xavia/2010">
    </Item>
</Property>
```

2. Within the `Item` property, insert six empty `Property` elements:

- `uri`
- `contentWidth`
- `contentHeight`
- `contentBytes`
- `contentType`
- `contentSrcKey`

```
<Property name="media">
  <Item class="com.endeca.infront.cartridge.model.MediaObject"
  xmlns="http://endeca.com/schema/xavia/2010">
    <Property name="uri"></Property>
    <Property name="contentWidth"></Property>
    <Property name="contentHeight"></Property>
    <Property name="contentBytes"></Property>
    <Property name="contentType"></Property>
    <Property name="contentSrcKey"></Property>
  </Item>
</Property>
```

These properties are populated by the **Select Records** dialog and processed by the cartridge handler.

3. Insert a corresponding `<editors:MediaEditor>` element within `<BasicContentItemEditor>`.
4. Specify the `propertyName` attribute for the editor:

```
<editors:MediaEditor propertyName="media" enabled="true"/>
```

5. Specify any additional label attributes for the editor:

```
<editors:MediaEditor propertyName="media" label="Media Url" enabled="true"/>
```

The following shows an example of a template that includes a media editor as part of a media banner cartridge:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Media banner</Name>
    <!-- additional elements omitted from this example -->
    <Property name="media">
      <Item class="com.endeca.infront.cartridge.model.MediaObject"
      xmlns="http://endeca.com/schema/xavia/2010">
        <Property name="uri"></Property>
        <Property name="contentWidth"></Property>
        <Property name="contentHeight"></Property>
        <Property name="contentBytes"></Property>
```

```

<Property name="contentType"></Property>
<Property name="contentSrcKey"></Property>
</Item>
</Property>
</ContentItem>

<EditorPanel>
  <BasicContentItemEditor>
    <!-- additional elements omitted from this example -->
    <GroupLabel label="Media"/>
    <editors:MediaEditor propertyName="media" label="Media Url" enabled="true"/>
  </BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

In order to use the Media editor, if you are using the Endeca Configuration Repository as your media store, you must upload any media files to the repository. If you are using an external digital asset management system with a corresponding MDEX Engine, the matching Endeca application must be configured and running and the Media Browser must be enabled.

Related Links

[Uploading media to the Endeca Configuration Repository](#) on page 249

If you wish to use the ECR as your media content source, you can upload assets directly to Experience Manager. This is required if you aren't using the Media Browser and are instead using the default Flex-based asset navigator for the media editor. It can also be useful in a development environment, where a separate media server may not be worth the effort of maintaining.

About Media editor configuration

You can specify allowable media formats in the editor configuration file. You can also enable or disable the Media Browser, and specify the MDEX Engine that it should query for media records.

The Discover Electronics reference application uses the Endeca Configuration Repository to store media and accesses these resources through a default asset browser, rather than relying on the Media Browser and an accompanying media MDEX Engine.

Below is the configuration for the Media editor in the editor configuration file, located at <app dir>\config\editors_config\editors.xml:

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- additional text removed from this example -->
<EditorConfig xmlns="http://endeca.com/schema/editor-config/2010">
  <EditorModule url="/ifcr/tools/pbx/modules/editors.swf">
    <!-- additional elements removed from this example -->
    <Editor name="editors:MediaEditor">
      <EditorConfig>
        <useMediaBrowser>false</useMediaBrowser>
        <mediaRoots>
          <default>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</default>
          <mediaSource>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</mediaSource>
        </mediaRoots>
        <mdexPort>17000</mdexPort>
        <mdexHost>myhost.mydomain.com</mdexHost>
        <videoFormats>flv|f4v|3gp|mov|mp4</videoFormats>
        <imageFormats>jpg|jpeg|png|gif</imageFormats>
        <mediaURI>/ifcr/sites/Discover/media/</mediaURI>
      </EditorConfig>
    </Editor>
  </EditorModule>
</EditorConfig>

```

```

</Editor>
<!-- additional elements removed from this example -->
</EditorModule>
<GlobalEditorConfig></GlobalEditorConfig>
</EditorConfig>

```

This sets the following properties across all instances of the media editor in the application:

Property	Description
useMediaBrowser	This property enables or disables the media browser. By default, it is set to <code>false</code> .
mediaRoot	This property specifies the absolute URLs to available media repositories. It includes a nested <code>default</code> property that points to the Endeca Configuration Repository, and an additional property for each repository indexed by the media MDEX Engine. For more information, see "About resolving media paths in content items."
default	The absolute URL to the Endeca Configuration Repository, used by the default asset browser. The specified host and port should match those used by Endeca Workbench.
content source (mediaSource)	An absolute URL to a media content source. In the reference data application, records are assigned a <code>media.repository_id</code> property with a value of <code>mediaSource</code> . Your own data ingest process may assign different values for media served from varying locations. In this case, each <code>media.repository_id</code> value should have a corresponding element in the editor configuration file that identifies the URL for that content source.
mdexPort	For applications using the Media Browser, this is the hostname or IP address of the media MDEX Engine server.
mdexHost	For applications using the Media Browser, this is the port on which the specified media MDEX Engine server listens.
videoFormats	A pipe-delimited list of valid video formats. Any videos not matching a listed format do not display in either the default asset browser or Media Browser.
imageFormats	A pipe-delimited list of valid image formats. Any images not matching a listed format do not display in either the default asset browser or Media Browser.
mediaURI	The location of the media node within the Endeca Configuration Repository. This is only used by the default asset browser.



Note: The default list of video and image formats includes only those that are supported by the included renderers for the Discover Electronics reference application. If you wish to extend this list for your own application, ensure that your cartridge renderers can handle additional formats, and that your application includes logic for displaying them.

Uploading media to the Endeca Configuration Repository

If you wish to use the ECR as your media content source, you can upload assets directly to Experience Manager. This is required if you aren't using the Media Browser and are instead using the default Flex-based asset navigator for the media editor. It can also be useful in a development environment, where a separate media server may not be worth the effort of maintaining.

All applications created using the Deployment Template include a `set_media` script in the `<app dir>\control` directory. This script uploads media content from the `<app dir>\config\media` directory

to the Endeca Configuration Repository. After uploading content, it becomes available for use in Experience Manager.

In general, you can store moderate amounts of media content in the Endeca Configuration Repository. Very roughly speaking, a moderate amount of media content is approximately thousands of media files but not tens of thousands of media files. This storage mechanism is intended as a convenience when you build an application in a development environment.

If you have larger amounts of media content, Oracle recommends employing a digital asset management system rather than uploading the media content into the Endeca Configuration Repository.

Here are a few specific guidelines to keep in mind before you upload media content to the Endeca Configuration Repository:

- Do not upload more than approximately 1 GB of media content per transaction. In this context, a transaction is one run of `set_media`.
- Do not upload more than approximately 5000 files in one transaction. This guideline essentially means you should not have more than approximately 5000 files stored in `<app dir>\config\media` and its subdirectories.
- If you have more than approximately 1000 files to upload, create subdirectories under `<app dir>\config\media` and distribute the media files among the subdirectories. (One run of `set_media` uploads all content in all subdirectories.)

To upload media content for use in Experience Manager:

1. Ensure any new media content is stored locally in `<app dir>\config\media`.
This may include image files, video files, and so on.
2. In a command prompt, navigate to the `<app dir>\control` directory of your deployed application.
This is located under your application directory. For example: `C:\Endeca\apps\Discover\control`.
3. Run the `set_media` script.
4. To verify that your media assets are available:
 - a) Log in to Workbench.
 - b) Open Experience Manager.
 - c) Select a cartridge that includes the Media editor.
 - d) Click the **Select** button to launch the Media editor and confirm that your media assets display.

About resolving media paths in content items

Links to media assets are resolved in the Media editor by combining configuration in the editor configuration file with the `media.path` property on the selected record. At runtime, these links are resolved against the media sources specified in the Assembler context file.

About media root elements

You identify authoring content sources as nested elements within the `<mediaRoots>` element in the editor configuration file. The name of each such element corresponds to the value of the `media.repository_id` property assigned to each record in your media MDEX Engine. The value of each element identifies the root location of the authoring content source.

When a content administrator opens the Media Browser in Experience Manager, media assets are retrieved for preview by appending the value of the `media.path` property on the record to the corresponding content source element within `<mediaRoots>`. The `media.path` is then saved to the content item when the content administrator saves the cartridge configuration.

By keeping the relative location of your media assets consistent across environments, you can maintain separate content sources for authoring and live environments without requiring content administrators to reconfigure content items.

For example, assume the following element within `<mediaRoots>` in the editor configuration file:

```
<myMediaSource>http://myhost.mydomain.com:8006/myCMS/Discover/media/</myMediaSource>
```

A media record with a `media.repository_id` value of "myMediaSource" and a `media.path` value of "images/foo.jpg" would resolve to:

```
http://myhost.mydomain.com:8006/myCMS/Discover/media/images/foo.jpg
```

At runtime, the value of the `media.path` property is instead appended to the appropriate media source configured in `assembler-context.xml`:

```
<!--
 ~~~~~
 ~ Media Sources
-->

<bean id="authoringMediaSources" class="java.util.ArrayList" lazy-init="true">
    <constructor-arg>
        <list>
            <bean class="com.endeca.infront.cartridge.model.MediaSourceConfig">
                <property name="sourceName" value="MyMediaSource" />
                <property name="sourceValue" value="http://myhost.mydomain.com:8006/myCMS/Discover/media/" />
            </bean>
            <bean class="com.endeca.infront.cartridge.model.MediaSourceConfig">
                <property name="sourceName" value="default" />
                <property name="sourceValue" value="http://myhost.mydomain.com:8006/myCMS/Discover/media/" />
            </bean>
        </list>
    </constructor-arg>
</bean>

<bean id="liveMediaSources" class="java.util.ArrayList" lazy-init="true">
    <constructor-arg>
        <list>
            <bean class="com.endeca.infront.cartridge.model.MediaSourceConfig">
                <property name="sourceName" value="MyMediaSource" />
                <property name="sourceValue" value="http://myhost.mydomain.com:8006/myBiggerFasterCMS/Discover/media/assets/" />
            </bean>
            <bean class="com.endeca.infront.cartridge.model.MediaSourceConfig">
                <property name="sourceName" value="default" />
                <property name="sourceValue" value="http://myhost.mydomain.com:8006/myBiggerFasterCMS/Discover/media/assets/" />
            </bean>
        </list>
    </constructor-arg>
</bean>
```

In a live environment, the aforementioned media record would resolve to:

```
http://myhost.mydomain.com:8006/myBiggerFasterCMS/Discover/media/assets/images/foo.jpg
```



Note: While the tooling, authoring, and live content sources can all differ, Oracle recommends configuring the Media Browser to use the authoring content source.

Enabling the Media Browser

The default browser for the Media editor can only be configured to browse media assets in the Endeca Configuration Repository. If you are serving media assets from an external content source, you must enable the Media Browser and configure it to use your media MDEX Engine.

You can enable and configure the Media Browser by modifying the editor configuration file for your application.

To enable the Media Browser in the Media editor:

1. Navigate to the editor configuration file at `<app dir>\config\editors_config\editors.xml`.
2. Locate the `<EditorConfig>` element for the Media editor:

```
<Editor name="editors:MediaEditor">
  <EditorConfig>
    <useMediaBrowser>false</useMediaBrowser>
    <mediaRoots>
      <default>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</default>
      <mediaSource>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</mediaSource>
    </mediaRoots>
    <mdexPort>17000</mdexPort>
    <mdexHost>mymediahost.mydomain.com</mdexHost>
    <videoFormats>flv|f4v|3gp|mov|mp4</videoFormats>
    <imageFormats>jpg|jpeg|png|gif</imageFormats>
    <mediaURI>/ifcr/sites/Discover/media/</mediaURI>
  </EditorConfig>
</Editor>
```

3. Within the `<EditorConfig>` element, change the value of the `<useMediaBrowser>` property to `true`:

```
<useMediaBrowser>true</useMediaBrowser>
```

4. Include a content source element under `<mediaRoots>` that points to your media host.

The element name is a unique key that identifies a media host. Each record has a corresponding `media.repository_id` property that identifies its content source. The relevant content source property maps that source to a URL.

For example, in the CAS crawl configuration for the reference data application, each record is assigned a `media.repository_id` property with a value of `mediaSource`. The `mediaSource` property in the editor configuration file specifies the URL:

```
<mediaRoots>
  <default>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</default>

  <mediaSource>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</mediaSource>
</mediaRoots>
```



Note: The `<default>` value is only used by the default asset browser. For more information, see "About Media editor configuration" and "Media MDEX Engine schema definition."

5. Modify the `<mdexPort>` and `<mdexHost>` elements to point to the host and port of the MDEX Engine backing your media host.
6. Save and close the file.
7. Navigate to the `<app dir>\control` directory.
8. Run the `set_editors_config` script to publish your changes to the Endeca Configuration Repository.

Related Links

[Using an MDEX Engine to index media assets](#) on page 253

If you are storing media resources in an independent content store, you can set up an MDEX Engine where records represent media assets and include asset metadata and URIs. Storing this information as records allows content administrators to navigate assets based on image size, modification date, or other attributes when selecting media assets for a content item.

Using an MDEX Engine to index media assets

If you are storing media resources in an independent content store, you can set up an MDEX Engine where records represent media assets and include asset metadata and URIs. Storing this information as records allows content administrators to navigate assets based on image size, modification date, or other attributes when selecting media assets for a content item.

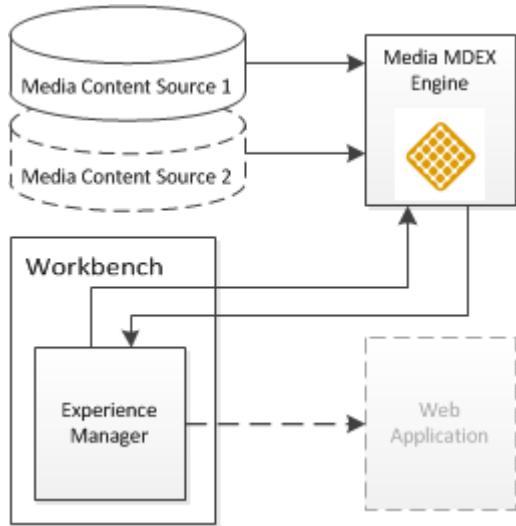
Tools and Frameworks includes a reference media MDEX application, including a CAS pipeline and Deployment Template configuration.

Interaction between Experience Manager and the media MDEX Engine

The interactions between a media MDEX Engine, Experience Manager, and an Assembler application are summarized below.

Interaction between a media MDEX Engine and Experience Manager

Experience Manager retrieves media asset information as follows:

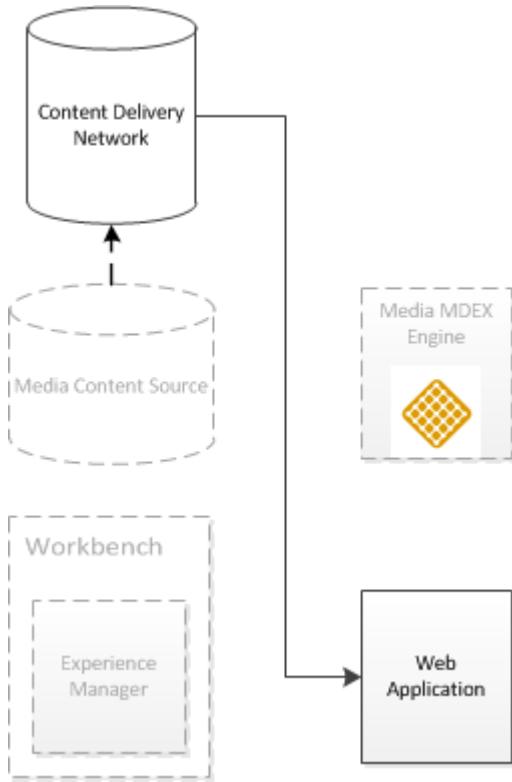


Assuming that an MDEX Engine exists with media records that adhere to the required data schema:

1. In Experience Manager, the Media Browser queries the media MDEX Engine for media records. This allows the content administrator to select media assets by navigating across them with Guided Navigation.
2. The content administrator's configuration changes are published to the Endeca application each time a content item is saved.

Interaction between the media content source and an Assembler application

In a production environment, the Assembler application can be configured to retrieve media assets from a content delivery network or another media delivery server:



1. Media assets are uploaded from the media content source to the runtime content delivery network.
2. The application retrieves media from this content delivery network.



Note: The server hosting media assets can differ between authoring and live environments, as long as the media path relative to the media root is consistent. In the case of the reference pipeline, the authoring Discover Electronics Web application serves as the content source. For more information on configuring content sources, see [About Media editor configuration](#) on page 248.

Overview of the reference data application

The Tools and Frameworks package includes a reference implementation of a media MDEX Engine that includes a CAS crawl and Forge pipeline for crawling resources on the file system and indexing the corresponding metadata and URIs. The Experience Manager can then query the MDEX Engine for record information.

The reference media application illustrates the schema requirements and configuration that you should use when building your own media pipeline.

Endeca software requirements

In addition to the hardware and software required for Oracle Endeca Tools and Frameworks, the data ingest process for the reference data application requires the Oracle Endeca Content Acquisition System. You must have CAS installed on the machine on which you are running the ITL process for the data application.

Reference CAS crawl

The crawl uses the following manipulators:

1. **Directory Filter:** Filters out directory records, so that only media files are output to the MDEX Engine.
2. **Image Property Generator:** Analyzes image binaries to determine their width and height. It adds corresponding `image.width` and `image.height` properties to each record.
3. **Application Property Generator:** Assigns a `media.application` property based on the application specified when running the Deployment Template. This allows the Media Browser to display only those media assets that are relevant to the application that the content administrator is currently modifying in Workbench.
4. **Path Manipulator:** Creates a `media.path` property that contains the path to given asset with respect to the media root.

Media MDEX Forge pipeline

The Forge pipeline for the reference data application reads data from the Endeca Record Store populated by the CAS crawl and runs manipulators against the data to generate the required MDEX Engine schema.



Important: The reference media MDEX Engine and data application are provided as example implementations. If you wish to connect to an external data source, you must configure a CAS crawl specific to your data set, including customizing the record manipulators as necessary. Additionally, your MDEX Engine configuration must include certain properties in order to function correctly with the Media editor in Experience Manager. See [Pipeline configuration for a media crawl](#) on page 256 for details.

Deploying the reference data application for Discover Electronics

The reference media MDEX Engine data application assumes an environment where all required Oracle Endeca components are running on the same machine.

You must have the Oracle Endeca Content Acquisition System and Oracle Endeca Tools and Frameworks installed on the machine onto which you wish to deploy the media MDEX Engine.

The reference data application runs an MDEX Engine with indexed media resources, and integrates with the Discover Electronics reference application to expose the media records to a business user in the Media editor in Experience Manager. The records include properties for metadata, such as image dimensions, making it easier to narrow down a large quantity of media assets to those that fit the requirements for a cartridge in the front-end application.

To deploy the reference data application:

1. Include the CAS manipulators for the reference data application as server plugins:
 - a) Navigate to the `%CAS_ROOT%\lib\cas-server-plugins`.
 - b) Create a directory named `mediaMDEX`.
 - c) Navigate to the `%ENDECA_TOOLS_ROOT%\reference\media-mdex-cas\cas\media-mdex-manipulators` directory.
 - d) Copy the following JAR files to the `%CAS_ROOT%\lib\cas-server-plugins\mediaMDEX` directory you created in step 1b:
 - `media-mdex-manipulators-<version>.jar`
 - `guava-14.0.jar`
2. Restart the Endeca CAS Service.
3. Deploy the reference data application:
 - a) Open a command prompt or command shell.



Note: If you are running the Tools and Frameworks from the included batch files, you must run `ToolsAndFrameworks/<version>/server/bin/setenv.bat` to set the environment variables for the current command window.

- b) Navigate to the `ToolsAndFrameworks/<version>/deployment_template/bin` directory.
- c) Run `deploy.bat` or `deploy.sh` with the following options:

```
deploy --app <Endeca Directory>/ToolsAndFrameworks/<version>/reference/media-  
mindex-cas/deploy.xml
```

- d) Confirm the Platform Services installation directory.

- e) Enter `n` to skip installation of a base application.

- f) Specify `media` as the application name.

- g) Specify the Endeca application directory.

Typically, this is `C:\Endeca\apps` on Windows, or `/usr/local/endeca/apps` on UNIX.

- h) Specify the EAC port previously used for the Discover Electronics reference application.

By default, this is port 8888.

- i) Specify the port that Workbench runs on.

By default, this is port 8006.

- j) Specify a Dgraph port.



Note: This must be a different port from any other Dgraphs used for other applications.

By default, this is port 17000. If you change this value, you must also update the configuration for the `MediaEditor` in the `config\editors_config\editors.xml` file after deploying the application.

- k) Specify the CAS installation directory.

- l) Specify the CAS version.

- m) Enter the port that CAS runs on.

By default, this is port 8500.

- n) Enter the name of the application in which you wish to enable media browsing.

For the Discover Electronics reference application this should be `Discover`.

- o) Enter the absolute path to the location on disk where media assets are stored.

This is the file path that the Content Acquisition System crawls to index the files. In a default Discover Electronics deployment it is `C:\Endeca\apps\Discover\config\media` on Windows, or `/usr/local/endeca/apps/Discover/config/media` on Unix.

4. Provision the application with the EAC and run a baseline update:

- a) Navigate to the `control` directory of the deployed media application.

- b) Run the `initialize_services` script to provision the application in the EAC.

- c) Run the `baseline_update` script to crawl the directory specified in Step 4 and index the assets in the MDEX Engine.

Pipeline configuration for a media crawl

In order for the Media Browser in Experience Manager to have sufficient information for forming content XML, any pipeline that you configure for a media MDEX Engine must define specific properties and dimensions.

Required properties

The following properties are required for the Media Browser to function correctly:

Field	Description
record.id	A unique identifier for each of the media items.
media.name	The filename of the media asset.
media.path	The file path, relative to the root of the content source.
media.repository_id	The logical host of the content source. The value of this property is mapped to configuration elements for the Media editor in the editor configuration file, which in turn contain the path to the content source. For additional information, see "About Media editor configuration."
media.application	The EAC application that the specified media asset is associated with. The Media editor in Experience Manager filters entries in the Media Browser based on which application the content administrator is currently editing.
media.size	The binary size of the media asset, in bytes.
image.height	The height of the media asset, if it is an image. The renderer for the Media editor uses this information to scale images appropriately.
image.width	The width of the media asset, if it is an image. The renderer for the Media editor uses this information to scale images appropriately.

Properties and dimensions provided in the reference data application

Optionally, additional properties and dimensions can be displayed in the Media Browser. The reference implementation of a media MDEX Engine includes the following such fields:

Field	Type	Description
media.file_type	Property	The MIME type of the media asset. This enables filtering by media type and file extension in the Media Browser.
media.last_modification_date	Property	The date and time that the file was last modified prior to being crawled by the Content Acquisition System.
fileType	Dimension	A searchable dimension based on media.file_type values.
height, width	Dimension	Searchable dimensions based on image.height and image.width values.
application	Dimension	A searchable dimension based on media.application values.

If you configure your own media MDEX Engine that includes properties or dimensions not listed above, they become available for Guided Navigation in the Media Browser. However, any such properties are not saved to content XML once a media asset has been selected.

Search interface requirements

The Media Browser requires a defined search interface named "All" that includes all searchable properties and dimensions in the data set. Additionally, the Media Browser in the reference application uses the "MatchAllPartial" search mode.

Adding a Boost-Bury Record editor

The Boost-Bury Record editor enables a content administrator to specify certain records to display either at the top or bottom of the list of results for a page.

The Boost-Bury Record editor uses the **Select Records** dialog to enable the content administrator to specify either an ordered list of record IDs or a set of refinements that define the set of records to be boosted or buried.



Note: The Boost-Bury Record editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

To add a Boost-Bury Record editor:

1. Insert an `<editors:BoostBuryRecordEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the item property that represents the records to be boosted to the top of the results. This property must be declared in the same template as the Record Stratification editor.
<code>buryProperty</code>	Required. The name of the list property that represents the records to be buried at the bottom of the results. This property must be declared in the same template as the Record Stratification editor.

The following shows an example of a template that includes a Boost-Bury Record editor:

```

<ContentTemplate xmlns="http://endeeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Results List</Name>
    <Property name="boostStrata">
      <xavia>List/>
    </Property>
    <Property name="buryStrata">
      <xavia>List/>
    </Property>
    <!-- additional elements omitted from this example -->
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <!-- additional elements omitted from this example -->
      <editors:editors:BoostBuryRecordEditor propertyName="boostStrata"
        buryProperty="buryStrata" label="Customize Results List" />
      <!-- additional elements omitted from this example -->
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

Related Links

[Select Records data service configuration reference](#) on page 240

The Select Records dialog in Experience Manager communicates with the MDEX Engine through a configurable data service. By default, the service is configured to provide relevant record properties for the Discover Electronics reference application.

[About the Select Records dialog](#) on page 240

The **Select Records** dialog is used in several editors in the core cartridges to enable a content administrator to specify a set of records.

Adding a Guided Navigation editor

The Guided Navigation editor enables a content administrator to quickly create a navigation menu through the use of the Generate Guided Navigation wizard.



Note: The Guided Navigation editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

A content administrator can use the **Generate Guided Navigation** button to trigger the Generate Guided Navigation wizard. The wizard allows them to select and order a set of dimensions to add as Refinement Menu cartridges. Alternately, they can choose to add, order, and configure the cartridges manually.

To add a Guided Navigation editor:

1. Insert an `<editors:GuidedNavigationEditor>` element within `<BasicContentItemEditor>`.
2. Set a `propertyName` attribute on the `<editors:GuidedNavigationEditor>` element.

This must be set to the name of the `ContentItemList` property that represents the list of Refinement Menu content items. The property must be declared in the same template.

3. Insert an `<editors:ContentItemMapping>` element within the editor.
4. Map the content item name to the dimension property that should populate it.

This determines the name of the Refinement Menu content items created by the Generate Guided Navigation wizard.

- a) Include an `<endeca:Name>` element within `</endeca:ContentItemMapping>`:

```
<endeca:ContentItemMapping>
  <endeca:Name />
</endeca:ContentItemMapping>
```

- b) Specify the dimension property to use for the content item name in a `dimensionProperty` attribute:

```
<endeca:ContentItemMapping>
  <endeca:Name dimensionProperty="display_name" />
</endeca:ContentItemMapping>
```

- c) Specify the dimension name as a fallback value.

The Generate Guided Navigation wizard uses the first non-null value when naming a newly-created content item.

```
<endeca:ContentItemMapping>
  <endeca:Name dimensionProperty="display_name" />
  <endeca:Name dimensionProperty="endeca:name" />
</endeca:ContentItemMapping>
```

5. Map the `dimensionName` and `dimensionID` properties to the dimension properties that populate them:

```
<endeca:ContentItemMapping>
  <endeca:Name dimensionProperty="display_name" />
  <endeca:Name dimensionProperty="endeca:name" />
```

```

        <endeca:Property name="dimensionName" dimensionProperty="endeca:name" />
        <endeca:Property name="dimensionId" dimensionProperty="endeca:identifier" />
    </endeca:ContentItemMapping>

```

The following shows an example of a template that includes a guided navigation editor:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:endeca="editors" type="SecondaryContent">
    <Description>Creates a container for navigation cartridges.</Description>
    <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/Secondary_Guided-
    Nav.png</ThumbnailUrl>

    <ContentItem>
        <Name>Navigation Container</Name>
        <Property name="navigation">
            <ContentItemList type="Navigation" />
        </Property>
    </ContentItem>

    <EditorPanel>
        <BasicContentItemEditor>
            <endeca:GuidedNavigationEditor propertyName="navigation"
                label="">
                <endeca:ContentItemMapping>
                    <!-- additional elements omitted from this example -->
                    <endeca:Name dimensionProperty="display_name" />
                    <endeca:Name dimensionProperty="endeca:name" />
                    <endeca:Property name="dimensionName" dimensionProperty="en-
                    deca:name" />
                    <endeca:Property name="dimensionId" dimensionProperty="ende-
                    ca:identifier" />
                </endeca:ContentItemMapping>
            </endeca:GuidedNavigationEditor>
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

Adding a Dimension Selector

A Dimension Selector enables a content administrator to specify a dimension by name.



Note: The Dimension Selector communicates with the MDEX Engine. In order to enable the Dimension Selector, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

To add a Dimension Selector:

1. Insert an `<editors:DimensionSelectorEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the string property that represents the dimension name. This property must be declared in the same template as the Dimension Selector.

Attribute	Description
idProperty	Required. The name of the string property that represents the dimension id. This property must be declared in the same template as the Dimension Selector.
enabled	If set to <code>false</code> , this attribute makes the property read-only so that the value of the property displays in the Content Details Panel in Experience Manager, but cannot be edited. Use this option only if you specify a default value in the definition of the dimension name and dimension ID properties. Editors are enabled by default.

The following shows an example of a template that includes a dimension selector:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    xmlns:editors="editors"
    type="Navigation">
    <Description>Displays Endeca Facet Navigation in a Text Link Rendering. For
    Flat Dimensions only.</Description>
    <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/dimension_naviga-
    tion.jpg</ThumbnailUrl>
    <ContentItem>
        <Name>Dimension Navigation</Name>
        <Property name="dimensionName">
            <String/>
        </Property>
        <!-- additional elements omitted from this example -->
    </ContentItem>
    <EditorPanel>
        <BasicContentItemEditor>
            <editors:DimensionSelectorEditor propertyName="dimensionName" idProp-
            erty="dimensionId"
                label="Dimension Name" enabled="true"/>
            <!-- additional elements omitted from this example -->
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

Related Links

[Select Records data service configuration reference](#) on page 240

The Select Records dialog in Experience Manager communicates with the MDEX Engine through a configurable data service. By default, the service is configured to provide relevant record properties for the Discover Electronics reference application.

[About the Select Records dialog](#) on page 240

The **Select Records** dialog is used in several editors in the core cartridges to enable a content administrator to specify a set of records.

Adding a Dimension List editor

The Dimension List editor enables a content administrator to select a list of dimensions from the application data set. The templates included with the reference application use this editor to specify which dimensions should be available for display in a dimension search auto-suggest panel or a dimension search results panel.



Note: The Dimension List editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

To add a Dimension List editor:

1. Insert an `<editors:DimensionListEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the List property that represents the selected dimension values. The property must be declared in the same template.

The following shows an example of a template that includes a dimension list editor:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  xmlns:editors="editors" type="MainContent">
  <Description>Displays dimension search results.</Description>
  <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/Main_DimensionSearchResults.png</ThumbnailUrl>
  <ContentItem>
    <Name>Dimension Search Results</Name>
    <!-- additional elements omitted from this example -->
    <Property name="dimensionList">
      <xavia>List/>
    </Property>
    <!-- additional elements omitted from this example -->
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <!-- additional elements omitted from this example -->
      <editors:DimensionListEditor propertyName="dimensionList"
        label="Dimensions Searched"
        enabled="true"/>
      <!-- additional elements omitted from this example -->
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Adding a Dimension Value Boost-Bury editor

The boost-bury editor enables a content administrator to specify certain dimension values to display either at the top or bottom of the list of refinements for a particular dimension.

In order to enable a Dimension Value Boost-Bury editor, the cartridge template must include a `dimensionId` property with an associated editor or a default value. This specifies the dimension to which the boost-bury editor applies.



Note: The Dimension Value Boost-Bury editor makes use of an auto-suggest dimension search component to enable the content administrator to quickly find the relevant dimension values. In order for this component to display partial matches as the user types in the search box, ensure that wildcard search is enabled for dimension searches in your MDEX Engine configuration.

To add a Dimension Value Boost-Bury editor:

1. Insert an `<editors:BoostBuryEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
propertyName	Required. The name of the list property that represents the list of dimension values to be boosted to the top of the list of refinements. This property must be declared in the same template as the boost-bury editor.
dimensionId	Required. The ID of the dimension that contains the dimension refinements to boost or bury.
boostProperty	Required. The name of the list property that represents the list of dimension values to be boosted to the top of the refinement list. This property must be declared in the same template as the boost-bury editor.
buryProperty	Required. The name of the list property that represents the list of dimension values to be buried at the bottom of the list of refinements. This property must be declared in the same template as the boost-bury editor.
enabled	If set to <code>false</code> , this attribute makes the property read-only so that the value of the property displays in the Content Details Panel in Experience Manager, but cannot be edited. Use this option only if you specify a default value for the <code>boostList</code> and <code>buryList</code> properties. Editors are enabled by default.

The following shows an example of a template that includes a dimension value boost-bury editor:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    xmlns:editors="editors"
    type="Navigation">
    <Description>Displays Endeca Facet Navigation in a Text Link Rendering. For
    Flat Dimensions only.</Description>
    <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/dimension_naviga-
    tion.jpg</ThumbnailUrl>
    <ContentItem>
        <Name>Dimension Navigation</Name>
        <Property name="dimensionName">
            <String/>
        </Property>
        <Property name="dimensionId">
            <String/>
        </Property>
        <!-- additional elements omitted from this example -->
        <Property name="boostRefinements">
            <xavia>List/>
        </Property>
        <Property name="buryRefinements">
            <xavia>List/>
        </Property>
    </ContentItem>
    <EditorPanel>
        <BasicContentItemEditor>
            <!-- additional elements omitted from this example -->
            <editors:BoostBuryEditor propertyName="boostRefinements" buryProp-
            erty="buryRefinements"
                label="Boost and Bury" dimensionIdProperty="dimensionId" en-
            abled="true"/>
            <!-- additional elements omitted from this example -->
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>

```

Adding a Dimension Value List editor

The Dimension Value List editor enables a content administrator to select a list of dimension values from the application data set.



Note: The Dimension Value List editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

To add a Dimension Value List editor:

1. Insert an `<editors:DimvalListEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the List property that represents the selected dimension values. The property must be declared in the same template.
<code>dimensionId</code>	Required. The ID of the dimension that the editor applies to.

The following shows an example of a Refinement Menu template that uses two Dimension Value List editors to specify boosted and buried refinements, instead of a Dimension Value Boost-Bury editor:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
                  xmlns:xavia="http://endeca.com/schema/xavia/2010"
                  xmlns:editors="editors"
                  type="Navigation">
  <Description>Displays Endeca Facet Navigation in a Text Link Rendering. For
  Flat Dimensions only.</Description>
  <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/dimension_naviga-
  tion.jpg</ThumbnailUrl>
  <ContentItem>
    <Name>Dimension Navigation</Name>
    <!-- additional elements omitted from this example -->
    <Property name="dimensionId">
      <String/>
    </Property>
    <!-- additional elements omitted from this example -->
    <Property name="boostRefinements">
      <xavia:List/>
    </Property>
    <Property name="buryRefinements">
      <xavia:List/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <!-- additional elements omitted from this example -->

      <GroupLabel label="Boost and Bury Dimension Refinements"/>
      <editors:DimvalListEditor dimensionIdProperty="dimensionId"
        propertyName="boostRefinements" label = "Boost Records"/>
      <editors:DimvalListEditor dimensionIdProperty="dimensionId"
        propertyName="buryRefinements" label = "Bury Records"/>

      <!-- additional elements omitted from this example -->
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

```
</EditorPanel>
</ContentTemplate>
```

Adding an Image Preview

An image preview retrieves an image from a URL and displays it in the Experience Manager interface.

You can construct an image preview URL from a hard-coded value, or from any number of String properties. Image preview supports JPEG, GIF, and PNG image formats.



Note: If images are hosted on a different server from Workbench, you may have to configure a cross-domain policy file to enable Flash player to access those resources.

To add an image preview to a template:

1. Insert an `<ImagePreview>` element within `<BasicContentItemEditor>`.
2. Specify attributes for the image preview:

Attribute	Description
<code>urlExpression</code>	Required. The source of the image URL. You can construct <code>urlExpression</code> from any number of string properties, or you can enter a static value.
<code>maxHeight</code>	The height in pixels of the image preview presented in the Experience Manager interface. The default value is 100.
<code>maxWidth</code>	The width in pixels of the image preview presented in the Experience Manager interface. The default value is 300.
<code>displayUrl</code>	A Boolean indicating whether to display the resolved URL. The default value is <code>true</code> .

If you are using more than one string property to compose the URL, you may want to use a `<GroupLabel>` to indicate to Experience Manager users that these properties are related.

The following examples show options for constructing an image preview.

Example: Specifying the URL by using a configurable String property

Add an `<ImagePreview>` to the cartridge template:

```
<ContentTemplate ... >
  <ContentItem>
    <Name>Dimension Navigation</Name>
    ...
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      ...
      <ImagePreview
        urlExpression=""
        label="Banner Image"
        maxWidth="200"
        maxHeight="100" />
      ...
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Add an `image_src` String property to the template:

```
<ContentTemplate ... >
  <ContentItem>
    <Name>Dimension Navigation</Name>
    <Property name="image_src">
      <String/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      ...
      <ImagePreview
        urlExpression=""
        label="Banner Image"
        maxWidth="200"
        maxHeight="100" />
      ...
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Add a corresponding `<StringEditor>` to the `<EditorPanel>`, and set the value of `urlExpression` to the `image_src` property:

```
<ContentTemplate ... >
  <ContentItem>
    <Name>Dimension Navigation</Name>
    <Property name="image_src">
      <String/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      ...
      <StringEditor propertyName="image_src" label="Image name" enabled="true"/>
      <ImagePreview
        urlExpression="http://localhost:8006/Discover/{image_src}"
        label="Banner Image"
        maxWidth="200"
        maxHeight="100" />
      ...
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Adding a Record List editor

The Record List editor uses the microbrowser to enable a content administrator to designate specific records to spotlight in a section, or to specify a query to return a dynamic list of records. This editor is deprecated.

 **Important:** This editor is deprecated. Use the `SpotlightSelectionEditor` instead.

 **Note:** The Record List editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

A Record List editor is bound to a `RecordSpotlightSelection` property, which can contain either a list of record IDs (for featured records) or a set of refinements (for dynamic records).

To add a Record List editor to a template:

1. Insert an `Item` property of class `com.endeca.infront.cartridge.RecordSpotlightSelection` named `recordSelection` as in the following example:

```
<Property name="recordSelection">
  <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection" />
</Property>
```

2. Insert an `<editors:RecordListEditor>` element within `<BasicContentItemEditor>`.

3. Specify label attributes and the additional attributes for the editor:

Attribute	Description
-----------	-------------

<code>propertyName</code>	Required. The name of the record selection property that represents the records to be spotlighted. This property must be declared in the same template as the Record Stratification editor.
---------------------------	---

4. Insert a `<PreviewProperty>` element within `<editors:RecordStratificationEditor>` with the following attribute:

Attribute	Description
-----------	-------------

<code>name</code>	The name of the record property to display in the Content Details Panel indicating which records have been selected for boost or bury.
-------------------	--

The following shows an example of a template that includes a record list editor:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="SidebarContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Spotlight records</Name>
    <!-- additional elements omitted from this example -->
    <Property name="recordSelection">
      <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection" />
    </Property>
    <!-- additional elements omitted from this example -->
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <GroupLabel label="Define Spotlight" />
      <!-- additional elements omitted from this example -->
      <editors:RecordListEditor propertyName="recordSelection"
        label="Spotlight Records">
        <PreviewProperty name="product.name"/>
      </editors:RecordListEditor>
      <!-- additional elements omitted from this example -->
    </BasicContentItemEditor>
```

```
</EditorPanel>
</ContentTemplate>
```

Specifying record sort options

The `sortOption` property of the results list cartridge enables the content administrator to specify a default sort order to apply to the results list when a site visitor reaches the page via navigation.

The available default sort options are defined in the Sort editor definition, which enables the content administrator to select from a predefined set of sort orders. The sort options that are available to the site visitor to apply to the results list are configured in the cartridge handler for this cartridge.

To specify sort options for the record list:

1. Insert an item property of class `com.endeca.infront.navigation.model.SortOption` named `sortOption`:

```
<Property name="sortOption">
  <xavia:Item class="com.endeca.infront.navigation.model.SortOption"/>
</Property>
```

2. Optionally, specify a default value for the property. The `SortOption` model includes the following properties:

Property Name	Description
<code>label</code>	A descriptive label that displays in a drop-down menu in Experience Manager.
<code>sorts</code>	A list of one or more items of class <code>com.endeca.infront.navigation.model.SortSpec</code> . Each <code>SortSpec</code> has two properties: <ul style="list-style-type: none"> • <code>key</code> — A string representing the name of an Endeca property or dimension on which to sort • <code>descending</code> — A Boolean value representing the sort direction

The following shows an example of a template that specifies a default sort option:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Results List</Name>
    <!-- additional elements omitted from this example -->
    <Property name="sortOption">
      <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
        <xavia:Property name="label">Most Sales</xavia:Property>
        <xavia:Property name="sorts">
          <xavia:List>
            <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
              <xavia:Property name="key">product.analytics.total_sales</xavia:Property>
              <xavia:Property name="ascending">true</xavia:Property>
            </xavia:Item>
          </xavia:List>
        </xavia:Property>
      </xavia:Item>
    </Property>
  <!-- additional elements omitted from this example -->
```

```

</ContentItem>
<!-- additional elements omitted from this example -->
</ContentTemplate>

```

Adding a Record Stratification editor

The Record Stratification editor enables a content administrator to specify certain records to display either at the top or bottom of the list of results for a page.



Important: This editor is deprecated. Use the `BoostBuryRecordEditor` instead.

The Record Stratification editor uses the microbrowser to enable the content administrator to specify either an ordered list of record IDs or a set of refinements that define the set of records to be boosted or buried.



Note: The Record Stratification editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

To add a Record Stratification editor:

1. Insert an `<editors:RecordStratificationEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
<code>propertyName</code>	Required. The name of the item property that represents the records to be boosted to the top of the results. This property must be declared in the same template as the Record Stratification editor.
<code>buryProperty</code>	Required. The name of the list property that represents the records to be buried at the bottom of the results. This property must be declared in the same template as the Record Stratification editor.

The following shows an example of a template that includes a record stratification editor:

```

<ContentTemplate xmlns="http://endeeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeeca.com/schema/xavia/2010"
  type="MainContent">
<!-- additional elements omitted from this example -->
<ContentItem>
  <Name>Results List</Name>
  <Property name="boostStrata">
    <xavia>List/>
  </Property>
  <Property name="buryStrata">
    <xavia>List/>
  </Property>
  <!-- additional elements omitted from this example -->
</ContentItem>

<EditorPanel>
  <BasicContentItemEditor>
    <!-- additional elements omitted from this example -->
    <editors:RecordStratificationEditor propertyName="boostStrata">

```

```

buryProperty="buryStrata" label="Customize Results List" />
<!-- additional elements omitted from this example -->
</BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

Adding a Rich Text editor

The Rich Text editor provides a text field and formatting toolbar that allows a content administrator to include formatted text and hyperlinks in a content item.



Important: The Rich Text editor UI includes a **Source** button that allows you to view editor content as HTML. This view is for reference purposes only; the editor does not support full HTML editing, and any unsupported elements added in this view are not saved in the content item XML.

To add a Rich Text editor to a template:

1. Insert a `<String>` element inside a `<Property>` element.
2. Optionally, specify the default value for the property as the content of the `<String>` element.
3. Insert a corresponding `<editors:RichTextEditor>` element within `<BasicContentItemEditor>`.
4. Specify the `propertyName` attribute for the editor:

```
<editors:RichTextEditor propertyName="description" enabled="true" />
```

5. Specify any additional label attributes for the editor:

```
<editors:RichTextEditor propertyName="description" label="Description" enabled="true" />
```

6. Specify the height and toolbar configuration for the editor:

```
<editors:RichTextEditor propertyName="description" label="Description" enabled="true" height="200" toolbar="["
  { name: 'document', items : [ 'Source' ] },
  { name: 'clipboard', items : [ 'Cut','Copy','Paste','PasteText','PasteFromWord','--','Undo','Redo' ] },
  { name: 'insert', items : [ 'Image','Table','HorizontalRule','SpecialChar' ] },
  { name: 'paragraph', items : [ 'NumberedList','BulletedList','--','Outdent','Indent','--','JustifyLeft',
                                'JustifyCenter','JustifyRight','JustifyBlock' ] },
  { name: 'links', items : [ 'Link','Unlink','Anchor' ] },
  '/',
  { name: 'basicstyles', items : [ 'Bold','Italic','Underline','Strike','Subscript','Superscript' ] },
  { name: 'styles', items : [ 'Styles','Format','Font','FontSize' ] },
  { name: 'colors', items : [ 'TextColor' ] }
]"]"/>
```



Note: The Rich Text editor is an implementation of the open source CKEditor WYSIWYG Rich Text editor. For a full list of toolbar buttons and their functionality, see the documentation for version 3.x of the CKEditor at http://docs.cksource.com/CKEditor_3.x/Developers_Guide/Toolbar.

The following shows an example of a template that includes a rich text editor:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>CategoryDescription</Name>
    <Property name="description">
      <String></String>
    </Property>
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <GroupLabel label="Contents"/>
      <editors:RichTextEditor propertyName="description" label="Description" enabled="true" height="200"
        toolbar="[
          { name: 'document', items: [ 'Source' ] },
          { name: 'clipboard', items: [ 'Cut', 'Copy', 'Paste', 'PasteText', 'PasteFromWord', '-', 'Undo', 'Redo' ] },
          { name: 'insert', items: [ 'Image', 'Table', 'HorizontalRule', 'SpecialChar' ] },
          { name: 'paragraph', items: [ 'NumberedList', 'BulletedList', '-', 'Outdent', 'Indent', '-' ],
              'JustifyLeft', 'JustifyCenter', 'JustifyRight', 'JustifyBlock' ],
          { name: 'links', items: [ 'Link', 'Unlink', 'Anchor' ] },
          '/',
          { name: 'basicstyles', items: [ 'Bold', 'Italic', 'Underline', 'Strike', 'Subscript', 'Superscript' ] },
          { name: 'styles', items: [ 'Styles', 'Format', 'Font', 'FontSize' ] },
          { name: 'colors', items: [ 'TextColor' ] }
        ]"/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>

```

Adding a Sort editor

A Sort editor enables the content administrator to choose a sort order (sort key and direction) to apply to a list of records.

Within the results list cartridge, this sort order (along with any boost/bury that is configured for the page) is applied to the results list by default when the end user first arrives at a page. If additional sort options are specified for this cartridge, the end user can select an alternate sort order and later return to the default ordering as specified by the content administrator.

To add a Sort editor:

1. Insert an `<editors:SortEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the editor:

Attribute	Description
propertyName	Required. The name of the item property that represents the default sort option. This property must be declared in the same template as the Sort editor.

3. Specify one or more items of class `com.endeca.infront.navigation.model.SortOption` from which the content administrator can select.

The following shows an example of a template that includes a sort editor:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="MainContent">
  <!-- additional elements omitted from this example -->
  <ContentItem>
    <Name>Results List</Name>
    <!-- additional elements omitted from this example -->
    <Property name="sortOption">
      <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
        <xavia:Property name="label">Most Sales</xavia:Property>
        <xavia:Property name="sorts">
          <xavia>List>
            <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
              <xavia:Property name="key">product.analytics.total_sales</xavia:Prop-
erty>
              <xavia:Property name="ascending">true</xavia:Property>
            </xavia:Item>
          </xavia>List>
        </xavia:Property>
      </xavia:Item>
    </Property>
    <!-- additional elements omitted from this example -->
  </ContentItem>

  <EditorPanel>
    <BasicContentItemEditor>
      <!-- additional elements omitted from this example -->
      <GroupLabel label="Navigation Result Settings (apply when user does not
      provide search terms)" />
      <editors:SortEditor propertyName="sortOption" label="Default Sort">
        <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
          <xavia:Property name="label">Default</xavia:Property>
          <xavia:Property name="sorts">
            <xavia>List />
          </xavia:Property>
        </xavia:Item>
        <xavia:Item class="com.endeca.infront.navigation.model.SortOption">
          <xavia:Property name="label">Most Sales</xavia:Property>
          <xavia:Property name="sorts">
            <xavia>List>
              <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
                <xavia:Property name="key">product.analytics.to-
tal_sales</xavia:Property>
                <xavia:Property name="ascending">true</xavia:Property>
              </xavia:Item>
            </xavia>List>
          </xavia:Property>
        </xavia:Item>
      </editors:SortEditor>
    </BasicContentItemEditor>
  </EditorPanel>

```

```

</xavia:Item>
<xavia:Item class="com.endeca.infront.navigation.model.SortOption">
  <xavia:Property name="label">Best Conversion Rate</xavia:Property>
  <xavia:Property name="sorts">
    <xavia:List>
      <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
        <xavia:Property name="key">product.analytics.conver-
sion_rate</xavia:Property>
        <xavia:Property name="ascending">true</xavia:Property>
      </xavia:Item>
    </xavia:List>
  </xavia:Property>
</xavia:Item>
<xavia:Item class="com.endeca.infront.navigation.model.SortOption">
  <xavia:Property name="label">Price (Ascending)</xavia:Property>
  <xavia:Property name="sorts">
    <xavia:List>
      <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
        <xavia:Property name="key">product.price</xavia:Property>
        <xavia:Property name="ascending">true</xavia:Property>
      </xavia:Item>
    </xavia:List>
  </xavia:Property>
</xavia:Item>
<xavia:Item class="com.endeca.infront.navigation.model.SortOption">
  <xavia:Property name="label">Price (Descending)</xavia:Property>
  <xavia:Property name="sorts">
    <xavia:List>
      <xavia:Item class="com.endeca.infront.navigation.model.SortSpec">
        <xavia:Property name="key">product.price</xavia:Property>
        <xavia:Property name="ascending">false</xavia:Property>
      </xavia:Item>
    </xavia:List>
  </xavia:Property>
</xavia:Item>
</editors:SortEditor>
</BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

Adding a Spotlight Selection editor

The Spotlight Selection editor uses the **Select Records** dialog to enable a content administrator to designate specific records to spotlight in a section, or to specify a query to return a dynamic list of records.



Note: The Spotlight Selection editor communicates with the MDEX Engine. In order to enable the editor, ensure that you have enabled communication between Experience Manager and the MDEX Engine.

A Spotlight Selection editor is bound to a `RecordSpotlightSelection` property, which can contain either a list of record IDs (for featured records) or a set of dimension refinements (for dynamic records).

To add a Spotlight Selection editor to a template:

1. Insert an `Item` property of class `com.endeca.infront.cartridge.RecordSpotlightSelection`. In the following example, this is the `recordSelection` property:

```

<Property name="recordSelection">
  <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection">

```

```
  />
</Property>
```

2. Insert a String property that stores the maximum number of records to display in the spotlight. In the following example, this is the `maxNumRecords` property:

```
<Property name="maxNumRecords">
  <String>10</String>
</Property>
```

3. Insert a Boolean property that controls the display of the "See All" link. In the following example, this is the `showSeeAllLink` property:

```
<Property name="showSeeAllLink">
  <Boolean>false</Boolean>
</Property>
```

4. Insert a String property to contain the text for the "See All" link. In the following example, this is the `seeAllLinkText` property:

```
<Property name="seeAllLinkText">
  <String />
</Property>
```

5. Insert an `<editors:RecordSpotlightSelectionEditor>` element within `<BasicContentItemEditor>`.
6. Specify label attributes and map the editor to the associated properties:

Attribute	Description
<code>propertyName</code>	Required. The name of the record selection property that represents the selected records or navigation state. This property must be declared in the same template as the record selection editor.
<code>maxNumRecords</code>	Required. Specifies the maximum number of records to display in the spotlight.
<code>showSeeAllLink</code>	Required. Controls the display of the "See All" link.
<code>seeAllLinkText</code>	Required. Specifies the text for the "See All" link.

The following shows an example of a template that includes a spotlight selection editor:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
  type="SecondaryContent">
  <Description>Displays selected records in secondary content area.</Description>
  <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/Secondary_RecordSpotlight.png</ThumbnailUrl>
  <ContentItem>
    <Name>Spotlight Records</Name>
    <Property name="title">
      <String>Featured Cameras</String>
    </Property>
    <Property name="maxNumRecords">
      <String>10</String>
    </Property>
    <Property name="recordSelection">
      <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelect">
```

```

tion" />
</Property>
<Property name="showSeeAllLink">
<Boolean>false</Boolean>
</Property>
<Property name="seeAllLinkText">
<String />
</Property>
</ContentItem>

<EditorPanel>
<BasicContentItemEditor>
<GroupLabel label="Define Spotlight" />
<editors:StringEditor propertyName="title" label="Spotlight Title" enabled="true" />
<editors:SpotlightSelectionEditor propertyName="recordSelection" label="Spotlight Records" maxNumRecords="maxNumRecords" showSeeAllLink="showSeeAllLink" seeAllLinkText="seeAllLinkText" />
</BasicContentItemEditor>
</EditorPanel>
</ContentTemplate>

```

Related Links

[Select Records data service configuration reference](#) on page 240

The Select Records dialog in Experience Manager communicates with the MDEX Engine through a configurable data service. By default, the service is configured to provide relevant record properties for the Discover Electronics reference application.

[About the Select Records dialog](#) on page 240

The **Select Records** dialog is used in several editors in the core cartridges to enable a content administrator to specify a set of records.

Application feature property reference

This is an overview of the mappings between features in a front-end application and their associated configuration properties.

Query configuration mappings

Global configuration for the features below is typically set in the Assembler context file on the class and property specified in the table.

Feature	URL Parameter	Global Configuration <class>.<property>	Cartridge Handler(s)
Navigation query	N	FilterState.navigationFilters	UrlNavigationStateBuilder
Refinement display in menu	Nrmc	RefinementMenuConfig.refinementsShown	RefinementMenu
Enable "Show More Refinements" link		RefinementMenuConfig.showMore	RefinementMenu

Feature	URL Parameter	Global Configuration <class>.<property>	Cartridge Handler(s)
"Show More" dimension IDs		NavigationContainer.showMoreIds	NavigationContainer
Record details	R	DefaultResultsListConfig	UrlNavigationStateBuilder
Record offset	No	ResultsListConfig.offset	ResultsList
Records to show per aggregate record	--	ResultsListConfig.subRecordsPerAggregateRecord	ResultsList
Record filter	Nr	FilterState.recordFilters	UrlNavigationStateBuilder
Records per page	Nrpp	ResultsListConfig.recordsPerPage	ResultsList
Record search key	Ntk	FilterState.SearchFilters.key	ResultsList, DimensionSearchResult
Aggregate record selection	A	--	UrlNavigationStateBuilder
Aggregate record offset	Nao	ResultsListConfig.offset	ResultsList
Aggregate record rollup key	--	FilterState.rollupKey	UrlNavigationStateBuilder
Why Rank	whyrank	ResultsListConfig.whyRankEnabled	ResultsList
Why Match	whymatch	ResultsListConfig.whyMatchEnabled	ResultsList
Why Precedence Rule Fired	whyprecedencerulefired	RefinementMenu.whyPrecedenceRuleFired, NavigationContainer.whyPrecedenceRuleFired	RefinementMenu, NavigationContainer
Range filter	Nf	FilterState.rangeFilters	UrlNavigationStateBuilder
Geocode range filter	Nfg	FilterState.rangeFilters	UrlNavigationStateBuilder
Set preview time	Endeca_Time	UserState.date	--
Relevance ranking Match Mode	Nrm	FilterState.SearchFilters.MatchMode	ResultsList
Relevance ranking strategy	--	ResultsListConfig.rankStrategy, DimensionSearchResultsConfig.rankStrategy	ResultsList
Relevance ranking search terms	Nrt	--	--
Relevance ranking search key	Nrk	--	ResultsList

Feature	URL Parameter	Global Configuration <class>.<property>	Cartridge Handler(s)
EQL filter	Nrs	FilterState.eqlFilter	UrlNavigationStateBuilder
Sort key	Ns	ResultsListConfig.sortOption, RefinementMenu.sort	ResultsList, RefinementMenu
Sort order			
Compute phrasings	Ntp	SearchAdjustmentsConfig.phraseSuggestionEnabled	UrlNavigationStateBuilder
Rewrite query with alternate phrasing			
Search terms	Ntt	FilterState.SearchFilters.terms	UrlNavigationStateBuilder
Search mode	Ntx	FilterState.SearchFilters.matchMode	UrlNavigationStateBuilder
"Did You Mean"	Nty	SearchAdjustmentsConfig.spellSuggestionEnabled	UrlNavigationStateBuilder
Signal dimension search	Dy		
Dimension search term	Ntt with Dy=1	See Ntt	See Ntt
Dimension search range filter	Nf with Dy=1	See Nf	See Nf
Enable dimension search relevance ranking	--	DimensionSearchResultConfig.relRank	DimensionSearchResultHandler
Dimension search scope	N with Dy=1	See N	See N
Dimension search result offset	--	--	--
Dimension search dimVal count	--	DimensionSearchResultConfig.maxResultsPerDimension	DimensionSearchResultHandler
Dimension search record filter	Nr with Dy=1	See Nr	See Nr
Dimension search refinement configuration	--	DimensionSearchResultConfig.showCountsEnabled	DimensionSearchResultHandler
Dimension search EQL filter	Nrs with Dy=1	See Nrs	See Nrs
Dimension search options	--	--	--

Appendix B

Navigation Cartridge Configuration Reference

This appendix provides an overview of the configuration models for the included navigation cartridges. You should review this information if you use these cartridges in your Assembler application to communicate with an MDEX Engine.

Navigation cartridge URL parameter reference

This section provides a reference to URL parameters in the navigation cartridges. The documented parameter names are configured in the Assembler, and your application can include additional parameters if you choose to extend the `RequestParamMarshallle` class or its cartridge-specific subclasses.

Related Links

[Navigation Cartridge Configuration Reference](#) on page 279

This appendix provides an overview of the configuration models for the included navigation cartridges. You should review this information if you use these cartridges in your Assembler application to communicate with an MDEX Engine.

[About this section](#) on page 279

The tables in this section describe the Endeca navigation cartridge query parameters. They include the following information:

[Core URL query parameters](#) on page 280

The URL query parameters that define the search and navigation objects passed into the MDEX Engine Navigation API are configured on the `UrlNavigationStateBuilder` object. By default, the Assembler is configured to use the following parameters:

[Cartridge-specific URL query parameters](#) on page 287

For some cartridges, it is appropriate for aspects of their configuration to be overridden at query time. Typically, request-based configuration is specified as URL query parameters. This section covers the URL query parameters for the core cartridges included with Tools and Frameworks.

About this section

The tables in this section describe the Endeca navigation cartridge query parameters. They include the following information:

URL parameter description format

Parameter	The query parameter, which is case-sensitive.
Name	The common name for the query parameter.
Type and format	The valid value type for the query parameter, as well as the format for listing multiple parameters, if applicable.
Object	The associated object in the Assembler API.
Description	A description of the parameter's functionality.
Dependencies	Additional query parameters that are required to give this parameter context.

Core URL query parameters

The URL query parameters that define the search and navigation objects passed into the MDEX Engine Navigation API are configured on the `UrlNavigationStateBuilder` object. By default, the Assembler is configured to use the following parameters:

URL Parameter	Feature
N	Navigation filter
Ntt	Record search terms
Ntk	Record search key
Ntx	Record search match mode
Nf	Range filter
Nfg	Geocode filter
Nr	Record filter
Nrs	EQL filter
Rsel	Featured Records selector
R	Record
A	Aggregate record
Ntp	Auto-phrasing
Ntl	Language ID



Note: To execute an aggregate record query using the `A` parameter, you must specify an aggregated record rollup key. Oracle recommends setting this key in your global application configuration; for example, in the Discover Electronics reference application, it is configured in the Assembler context file.

These parameters are described in detail in the following sections. The examples provided are for the Discover Electronics reference application.

N (Navigation)

The `N` parameter sets the navigation field for a query.

Parameter	N
Name	Navigation
Type and format	<dimension value id>+<dimension value id>+<dimension value id>...
Object	FilterState
Description	A unique combination of dimension value IDs. A value of zero indicates the root navigation object.
Dependencies	(none)

Examples

The following example is for an all-inclusive search, as it does not refine the results by any dimension value:

N=0

The following example returns products with an average review rating of 5:

N=100021



Note: The Discover Electronics reference application has Search Engine Optimization enabled by default, which encodes the above URL value to N-256d. For more information on creating optimized URLs, see [Building optimized URLs](#) on page 123 and the *Sitemap Generator Developer's Guide*.

Ntt (Record Search Terms)

The Ntt parameter sets the actual terms of a record search for a navigation query.

Parameter	Ntt
Name	Record Search Terms
Type and format	<string>+<string> <string> <string>+<string>+<string>...
Object	FilterState.SearchFilter
Description	Sets the terms of the record search for a navigation query. Each term is delimited by a plus sign (+). Each set of terms is delimited by a pipe ().
	Note: There is no explicit text search descriptor API object, so your application logic must extract search terms from the query if you wish to display them in Breadcrumbs or a similar search tracker.
Dependencies	N,Ntk; Ntt should have the same number of terms as Ntk has keys.

Examples

The following example returns records with a match for the term "zoom":

N=0&Ntt=zoom

The following example returns records with a match for the terms "cameras" and "silver" in the `product.description` record property. Note that the combined terms count as a single "search term" for the purposes of query syntax:

```
N=0&Ntk=product.description&Ntt=cameras+silver
```



Note: The Discover Electronics reference application is configured to use a default search key of "All" in the Spring context definition file for the Assembler, so it will accept a Record Search terms URL parameter (`Ntt`) without an accompanying Record Search key (`Ntk`) parameter.

Ntk (Record Search Key)

The `Ntk` parameter sets which dimension, property, or search interface is evaluated for a record search query.

Parameter	<code>Ntk</code>
Name	Record Search Key
Type and format	<code><search key> <search key>...</code>
Object	<code>FilterState.SearchFilter</code>
Description	Sets the keys of the record search for the navigation query. Multiple keys are delimited by a vertical pipe (). A search key can be a search interface defined in the MDEX Engine, a valid dimension name, or the name of a property enabled for record search in the data set.
Dependencies	<code>N, Ntt; Ntk</code> should have the same number of keys as <code>Ntt</code> has terms.

Examples

The following example returns records with a match for the terms "cameras" and "silver" in the `product.description` record property. Note that the combined terms count as a single "search term" for the purposes of query syntax:

```
N=0&Ntk=product.description&Ntt=cameras+silver
```

The following example returns records with a match for the term "cameras" in the `product.description` record property OR a match for the term "silver" in the `camera.color` record property. Note that these are evaluated as separate terms, and that each search term is associated with the search key that appears in the same location in the sequence:

```
N=0&Ntk=product.description|camera.color&Ntt=cameras|silver
```



Note: The Discover Electronics reference application is configured to use a default search key of "All" in the Spring context definition file for the Assembler, so it will accept a Record Search terms URL parameter (`Ntt`) without an accompanying Record Search key (`Ntk`) parameter.

Ntx (Record Search Match Mode)

The `Ntx` parameter sets the options for record search in the navigation query.

Parameter	<code>Ntx</code>
Name	Record Search Mode

Type and format	<string> <string>...
Object	FilterState.SearchFilter
Description	Sets the options for record search in the navigation query. Multiple values are separated with a vertical pipe () character.
Dependencies	N, Ntt, Ntk

Examples

The following example returns records with a match for the terms "cameras" and "silver" in the `product.description` record property. It overrides the default match mode with "MatchAllAny":

```
N=0&Ntk=product.description&Ntt=cameras+silver&Ntx=matchallany
```

Nf (Range Filter)

The `Nf` parameter sets the range filters for the navigation query.

Parameter	Nf
Name	Range Filter
Type and format	<p><search key> [LT LTEQ GT GTEQ]+<numeric value> [Another range filter]...</p> <p><search key> BTWN+<numeric value>+<numeric value>...</p>
Object	FilterState.RangeFilter
Description	<p>Sets the range filters for the navigation query on properties or on dimensions. Multiple range filters are separated with a double vertical pipe () delimiter.</p> <p>Accepts property and dimension values of Numeric type (Integer, Floating point, DateTime). For values of type Floating point, you can specify values using both decimal (0.00...68), and scientific notation (6.8e-10).</p>
Dependencies	N

Examples

The following example returns products with a price below \$25:

```
N=0?Nf=product.price|LT+25
```

The following example returns products with a price between \$50 and \$100 (inclusive):

```
N=0?Nf=product.price|BTWN+50+100
```

It is equivalent to specifying a "greater than or equal to" filter in combination with a "less than or equal to" filter:

```
N=0?Nf=product.price|GTEQ+50||product.price|LTEQ+100
```

Nfg (Geocode Filter)

The `Nfg` parameter sets a geocode filter for the navigation query, with radius in kilometers.

Parameter	Nfg
Name	Geocode Filter
Type and format	<key> <latitude> <longitude> <radius>
Object	FilterState.GeoFilter
Description	<p>Filters records by evaluating the geocode location contained in the <code>key</code> property to see if it falls within the circular area defined by a central point at <code>latitude</code>, <code>longitude</code> with the specified <code>radius</code> in kilometers.</p> <p>Positive <code>latitude</code> values are interpreted as °N of the equator, and positive <code>longitude</code> values are interpreted as °E of the Prime Meridian.</p>
Dependencies	N

Examples

The following example checks store geocodes within 10 km of the Statue of Liberty in NYC, NY:

```
N=0&Nfg=store.geocode|40.6893|-74.0446|10
```

Nr (Record Filter)

The `Nr` parameter sets a record filter on a navigation query.

Parameter	Nr
Name	Record Filter
Type and format	<string>
Object	FilterState.RangeFilter
Description	This parameter can be used to specify a record filter expression that restricts the results of a navigation query. Record filter syntax is described in the <i>MDEX Engine Advanced Development Guide</i> .
Dependencies	N

Examples

A general syntax example is given below:

```
N=0&Nr=AND(132831,propertyA:valueX,OR(propertyB:valueY,propertyC:valueZ))
```

The following example only includes records that are tagged as products, and it excludes any products that are not in stock:

```
N=0&Nr=AND( (common.record_type:product),NOT(product.inventory.count:0))
```

Nrs (Endeca Query Language Filter)

The `Nrs` parameter sets an EQL record filter on a navigation query. Using EQL enables you to specify multiple filters (such as a geocode range filter, a dimension value filter, and a record search filter) as part of the same query parameter.

Parameter	Nrs
-----------	-----

Name	Endeca Query Language Filter
Type and format	<string>
Object	FilterState.RangeFilter
Description	<p>Sets the Endeca Query Language expression for the navigation query. The expression acts as a filter to restrict the results of the query. Endeca Query Language syntax is documented in the <i>MDEX Engine Advanced Development Guide</i>.</p> <p> Note: The <code>Nrs</code> parameter must be URL-encoded. For clarity's sake, however, the examples below are not URL-encoded.</p>
Dependencies	N

Examples

Consider the sample Geocode Filter discussed earlier, which matches records at stores within 10 km of the Statue of Liberty in NYC, NY:

```
N=0&Nfg=store.geocode|40.6893|-74.0446|10
```

Combining the above with a record filter that excludes out-of-stock records results in the following:

```
N=0&Nfg=store.geocode|40.6893|-74.0446|10&Nr=NOT(product.inventory.count:0))
```

The above functionality can be duplicated with a single EQL query parameter by using the following expression:

```
N=0&Nrs=collection()/record[product.inventory.count!=0 and endeca:distance(store.geocode,endeca:geocode(40.6893,-74.0446))<10]
```

R (Record)

The `R` parameter specifies a single Endeca record to return from the MDEX Engine.

Parameter	R
Name	Record
Type and format	<record id>
Object	RecordState
Description	Query to obtain a single record from the MDEX Engine.
Dependencies	(none)

Examples

The following example specifies the IXUS 85 IS camera in the Discover Electronics data set; however, because the application is configured with a global aggregate record rollup key, all records are treated as aggregated records, so the `R` URL query parameter has no effect:

```
R=1469273
```

Rsel (Featured Records Selector)

The `Rsel` parameter restricts the search results list to a set of records specified by record ID.

Parameter	Rsel
Name	Featured Records Selector
Type and format	<record ID>,<record ID>,<record ID>...
Object	FilterState
Description	A comma-delimited list of record IDs. Search results are restricted to only those records specified as values for this query parameter.
Dependencies	R

Examples

The following example restricts the results list to the Z980 and Digital IXUS 85 IS cameras:

```
R=0?Rsel=1469273,1980692
```

A (Aggregated Record)

The A parameter specifies a single aggregated record to return from the MDEX Engine.

Parameter	A
Name	Aggregated Record
Type and format	<aggregated record id>
Object	RecordState
Description	Query to obtain a single aggregated record from the MDEX Engine.
Dependencies	(none)

Example

The following example specifies the IXUS 85 IS camera in the Discover Electronics data set; however, because the application serves record detail pages using the /detail servlet with a record-specific path, it has no effect:

```
A=1469273
```

Ntp (Auto-Phrasing)

The Ntp parameter sets whether the MDEX Engine applies computed alternative phrasings for the current query.

Parameter	Ntp
Name	Auto-Phrasing
Type and format	[0 1]
Object	FilterState
Description	Set to 1 to enable auto-phrasing, or 0 to disable it. If enabled, the MDEX Engine both computes and applies alternate query phrasings. If disabled, the MDEX Engine does not apply alternate query phrasings, but may compute them if SearchSuggestion-MdexQuery.phraseSuggestionEnabled=true.

Dependencies

N ,Ntt,Ntk

Examples

The following example searches the product description field for "auto focus" as a phrase, rather than searching the terms "auto" and "focus":

```
N=0?Ntk=product.description&Ntt=auto+focus&Ntp=1
```

Ntl (Language ID)

The `Ntl` parameter sets the language ID to pass in to the MDEX Engine.

Parameter	Ntl
Name	Language ID
Type and format	<ISO-639 language code>
Object	FilterState
Description	Specifies a language to cause the MDEX Engine to perform language-specific operations, such as invoking the correct stemming and phrasing dictionaries. For a list of supported languages, see the <i>MDEX Engine Advanced Development Guide</i> .
Dependencies	N

Examples

The following example specifies British English:

```
Ntl=en-GB
```

Cartridge-specific URL query parameters

For some cartridges, it is appropriate for aspects of their configuration to be overridden at query time. Typically, request-based configuration is specified as URL query parameters. This section covers the URL query parameters for the core cartridges included with Tools and Frameworks.

By default, the Assembler is configured to use the following parameters:

URL Parameter	Cartridge	Feature
Dy	Dimension Search Results	Enables or disables the display of returned dimension refinements.
Ntp	Search Adjustments	Specifies whether to display automatic phrasing; core parameter, see Ntp (Auto-Phrasing) on page 286.
Nty	Search Adjustments	Specifies whether to display automatic spelling correction / "Did You Mean"
Nrmc	Refinement Menu, Navigation Container	The <code>Nrmc</code> parameter takes multiple arguments allow you to configure dimension refinement behavior in a cartridge.
Nrpp	Results List	Records per page
Ns	Results List	Sort key and sort order
No	Results List	Record offset (used for paging)

URL Parameter	Cartridge	Feature
Nrt	Results List	Relevance Ranking search terms
Nrk	Results List	Relevance Ranking search key
Nrm	Results List	Relevance Ranking strategy
whymatch	Results List	Includes record matching information if query debugging is enabled
whyrank	Results List	Includes record ranking information if query debugging is enabled

These parameters are described in detail in the following sections. For additional information about the URL query parameters for the core cartridges, refer to the *Assembler API Reference (Javadoc)* for the relevant `RequestParamMarshaller` subclass. These classes define the URL parameters that each cartridge accepts, and their mappings to properties on the cartridge configuration model.

Dy (Dimension Search)

The `Dy` parameter controls the display of the Dimension Search Results cartridge.

Parameter	Dy
Name	Dimension Search
Type and format	[0 1]
Object	DimensionSearchResultsConfig
Description	Set to 1 to enable cartridge display, or 0 to disable it.
Dependencies	N,Ntt

Examples

The following example returns records with a match for the term "Silver," with the Dimension Search Results cartridge enabled:

```
N=0&Ntt=Silver&Dy=1
```

Nty (Auto-Correct / DYM)

The `Nty` parameter controls the display of auto-correct and "Did You Mean" results in the Search Adjustments cartridge.

Parameter	Nty
Name	Auto-Correct / "Did You Mean"
Type and format	[0 1]
Object	SearchAdjustmentsConfig
Description	Set to 1 to enable display, or 0 to disable it.
Dependencies	N,Ntt

Examples

The following example returns records with a match for the term "Sliver," with auto-correct enabled to correct the query to "silver":

```
N=0&Ntt=Sliver&Nty=1
```

Nrmc (Refinement Menu Config)

The `Nrmc` parameter takes multiple arguments that configure dimension refinement behavior in the Refinement Menu cartridge.

Because the Navigation Container cartridge returns a list of `RefinementMenu` objects, it takes the same `Nrmc` URL parameter as the Refinement Menu cartridge.

Parameter	<code>Nrmc</code>
Name	Refinement Menu Config
Type and format	<code><Dimension ID>+show: [all some none] <dimension ID>+show: [all some none]...</code>
Object	<code>RefinementMenuConfig</code>
Description	<p>The <code>Nrmc</code> parameter takes the following values:</p> <ul style="list-style-type: none"> • <code><Dimension ID></code> — Required. The ID of the dimension you wish to configure. • <code>+show: [all some none]</code> — Required; the value is passed to the <code>refinementsShown</code> property on the <code>RefinementMenuConfig</code> object, and controls how many dimension refinements to display. <p>Configuration for multiple dimensions is separated with a vertical pipe () character.</p>
Dependencies	<code>N</code>

Examples

The following modifies the Refinement Menu to display all of the dimension refinements for the "Features" dimension, and hides all refinements for the "Color" dimension:

```
N=0?Nrmc=100031+show:all|101908+show:none
```

Results List cartridge URL query parameters

The following URL query parameters determine the display of search results in the Results List cartridge. They are typically set in the front-end application by the end user.

Nrpp (Records Per Page)

The `Nrpp` parameter limits the records returned from the MDEX Engine.

Parameter	<code>Nrpp</code>
Name	Records Per Page
Type and format	<code><integer></code>
Object	<code>ResultsListConfig</code>
Description	Sets the maximum number of records to include in the <code>ResultsList</code> object.

Dependencies

N

Examples

The following example shows ten records per page:

```
N=0&Nrpp=10
```

Ns (Sort Key and Sort Order)

The `Ns` parameter controls sorting options for the current query. It enables the end user to override default sorting behavior on a per-query basis.

Parameter	Ns
Name	Sort Key and Sort Order
Type and format	<code><sort key>[<geocode reference>] [0 1] <sort key> [0 1]...</code>
Object	ResultsListConfig
Description	<p>The <code><sort key></code> specifies the property or dimension to sort by. Optionally, each key can be followed by a suffix of " 1" to indicate descending sort order, or " 0" to indicate ascending order (the default).</p> <p>Multiple entries are separated with a double vertical pipe (), and each entry after the first applies its sorting within the strata created by preceding entries.</p> <p>To sort records by a geocode property, add the optional <code>geocode</code> argument to the parameter (the <code><sort key></code> must be a geocode property). Records are sorted by the distance from the geocode reference point to the geocode point indicated by the <code><sort key></code> value.</p>
Dependencies	N

Examples

The following settings sort query results by product rating in descending order (higher rated products first). For each rating, it then sorts by price in ascending order (cheaper products first):

```
N=0&Ns=product.rating|0||product.price
```

The following example sorts records with a `store.geocode` property based on proximity to the Statue of Liberty in NYC, NY:

```
N=0&Ns=store.geocode|40.6893,-74.0446
```

No (Record Offset)

The `No` parameter sets the record offset in the query results list.

Parameter	No
Name	Record Offset
Type and format	<code><integer></code>
Object	ResultsListConfig
Description	Offsets the results set by the number of records specified. The offset is applied to a zero-based index; If <code>No=20</code> , the list of records starts at record 21. If an offset is greater than the number of items in a navigation object's record list, an empty record list is returned.

Dependencies

N,Nrpp

Examples

The following example displays the second page of a results set (since the results list is configured to display 36 records per page, and is offset by that amount to start at the 37th record):

```
N=0&Nrpp=36&No=36
```

Nrt (Relevance Ranking Search Terms)

The `Nrt` parameter optionally sets search terms for a Relevance Ranking enabled record search query.

You can apply Relevance Ranking to a subset of your MDEX Engine query by specifying the desired terms in the `Nrt` parameter.



Note: If you specify a Relevance Ranking strategy on the cartridge without specifying Relevance Ranking search terms and a search key (`Nrt` and `Nrk`), the MDEX Engine evaluates the query using the Record Search Terms and Record Search Key (`Ntt` and `Ntk`) parameters. For additional information on relevance ranking strategies, see the *MDEX Engine Advanced Development Guide*.

Parameter	<code>Nrt</code>
Name	Relevance Ranking Search Terms
Type and format	<code><string>+<string>+<string>...</code>
Object	<code>ResultsListConfig</code>
Description	<p>Sets the terms of the record search for a navigation query with relevance ranking. Each term is delimited by a plus sign (+).</p> <p> Note: Unlike the <code>Ntt</code> parameter, <code>Nrt</code> does not support using multiple sets of terms.</p>
Dependencies	<code>N, Nrk</code> . Additionally, you must set the <code>relRankStrategy</code> on the cartridge.

Examples

Because the Discover Electronics application uses Spring as a configuration mechanism, the cartridge-wide default values for Relevance Ranking in the Results List cartridge are specified in the `reference\discover-electronics-authoring\WEB-INF\assembler-context.xml` file:

```
<bean id="CartridgeHandler_ResultsList" class="com.endeca.infront.cartridge.ResultsListHandler"
    parent="NavigationCartridgeHandler" scope="prototype">
    <property name="contentItemInitializer">
        <bean class="com.endeca.infront.cartridge.ConfigInitializer" scope="request">
            <property name="defaults">
                <bean class="com.endeca.infront.cartridge.ResultsListConfig"
scope="singleton">
                    <!-- additional configuration omitted from this example -->
                    <property name="relRankKey" value="All" />
                    <property name="relRankMatchMode" value="ALLPARTIAL" />
                    <property name="relRankStrategy" value="nterms,maxfield,exact,static(product.analytics.conversion_rate,descending)" />
                    <!-- additional configuration omitted from this example -->
                </bean>
            </property>
        </bean>
    </property>
</bean>
```

```

        </property>
        <!-- additional configuration omitted from this example -->
    </bean>
</property>
<!-- additional configuration omitted from this example -->
</bean>

```

The following example returns records with a match for the terms "cameras" and "silver" in the `product.description` record property, and applies the Relevance Ranking strategy specified at the cartridge level:

```
N=0&Ntk=product.description&Ntt=cameras+silver
```

Nrk (Relevance Ranking Search Key)

The `Nrk` parameter sets which dimension, property, or search interface is evaluated for a Relevance Ranking enabled record search query.



Note: If you specify a Relevance Ranking strategy on the cartridge without specifying Relevance Ranking search terms and a search key (`Nrt` and `Nrk`), the MDEX Engine evaluates the query using the Record Search Terms and Record Search Key (`Ntt` and `Ntk`) parameters. For additional information on relevance ranking strategies, see the *MDEX Engine Advanced Development Guide*.

Parameter	<code>Nrk</code>
Name	Relevance Ranking Search Key
Type and format	<code><search key></code>
Object	<code>ResultsListConfig</code>
Description	Sets the search key for the record search query. This must be a navigable dimension, property name, or search interface defined in the MDEX Engine.
Dependencies	<code>N</code> , <code>Nrt</code> . Additionally, you must set the <code>relRankStrategy</code> on the cartridge.

Examples

The following example returns records with a match for the terms "cameras" and "silver" in the `product.description` record property, and applies the Relevance Ranking strategy specified at the cartridge level:

```
N=0&Ntk=product.description&Ntt=cameras+silver
```

Nrm (Relevance Ranking Match Mode)

The `Nrm` parameter sets the relevance ranking strategy for ranking the results of the record search.

You can override the default Relevance Ranking strategy on a per-query basis by using the `Nrm` parameter. For additional information on match modes, see the *MDEX Engine Basic Development Guide*.

Parameter	<code>Nrm</code>
Name	Relevance Ranking Strategy
Type and format	<code><string></code>
Object	<code>ResultsListConfig</code>
Description	Sets the options for record search in a relevance ranking enabled query.
	Note: Unlike the <code>Ntx</code> parameter, <code>Nrm</code> does not support using multiple match modes.

Dependencies	N, both Nrt and Nrk, OR both Ntt and Ntk. Additionally, you must set the rel-RankStrategy on the cartridge.
--------------	---

Examples

The following example returns records with a match for the terms "cameras" and "silver" in the `product.description` record property, and applies the Relevance Ranking strategy specified at the cartridge level. It overrides the default "MatchAllPartial" match mode with "MatchAllAny":

```
N=0&Ntk=product.description&Ntt=cameras+silver&Nrm=matchallany
```

whymatch (Record Match Info)

The `whymatch` parameter controls the logging of record match information on a per-query basis.

This property enables you to include record matching information on a per-query basis, rather than at the cartridge handler level.

Parameter	whymatch
Name	Record match debugging information
Type and format	[0 1]
Object	ResultsListConfig
Description	Set to 1 to include record matching information, or 0 to disable it.
Dependencies	N, as well as either Ntt and Ntk or Nrt and Nrk. Additionally, you must have query debugging enabled in your application.

Examples

The following example returns record matching information for a search against "silver cameras":

```
N=0&Ntk=product.description&Ntt=silver+cameras&whymatch=1
```

A portion of the response (serialized to JSON) is shown below. The `DGraph.WhyDidItMatch` key contains the relevant debugging information:

```
"DGraph.WhyDidItMatch": [
  "product.long_desc: <b>The high-quality 10.0 Megapixel Digital IXUS 870 IS
- finished in gold or silver -
  commands attention.
  ...
  Advanced compression technologies reduce file size, to free up valuable extra
  memory. (Stemming)"
],
```

whyrank (Record Rank Info)

The `whyrank` parameter controls the logging of relevance ranking information on a per-query basis.

This property enables you to include record relevance ranking information on a per-query basis, rather than at the cartridge handler level.

Parameter	whyrank
Name	Record ranking debugging information
Type and format	[0 1]
Object	ResultsListConfig

Description	Set to 1 to include record ranking information, or 0 to disable it.
Dependencies	N, as well as either Ntt and Ntk or Nrt and Nrk. Additionally, you must have query debugging enabled in your application.

Examples

The following example returns record ranking information for a search against "silver cameras.":

```
N=0&Ntk=product.description&Ntt=silver+cameras&whyrank=1
```

A portion of the response (serialized to JSON) is shown below. The DGraph.WhyRank key contains the relevant debugging information:

```
"DGraph.WhyRank": [
  "stratify": [
    evaluationTime: "0.00048828125"
    stratumRank: "3"
    stratumDesc: "no match"
  ]
],
```

About the navigation cartridge configuration models

This section describes the configuration models for the navigation cartridges.

You can use these models as a reference when developing your own cartridges and cartridge handlers. Generally, Oracle recommends adhering to a similar approach and dividing configuration inputs to a cartridge across the following categories (ordered from lowest to highest priority):

- **Application-wide default configuration** — For the navigation cartridges, these values are configured in the Spring context file.
- **Template-specific default configuration** — These values are included in the cartridge template XML.
- **Instance configuration** — These values are configured by the business user in Experience Manager or Rule Manager.
- **End user inputs** — For the navigation cartridges, these values are passed in as URL parameters.

Overview of the navigation cartridge configuration models

The behavior of the navigation cartridges depends on multiple sources of configuration. The data from these source is combined into a configuration model within the `initialize()` method of each associated cartridge handler in the Assembler.

Navigation cartridge configuration falls into the following categories, in ascending order of priority:

- **Default cartridge configuration**, which is specified in the Spring context file for the Assembler application
- **Cartridge instance configuration**, which is specified by the content administrator in Experience Manager or Rule Manager
- **Request-based configuration**, which is specified by the end user in the client application

Additionally, while it is not represented in the cartridge configuration model, configuration in the MDEX Engine impacts the behavior of the navigation cartridges.

Request-based configuration overrides the cartridge instance configuration, which overrides the cartridge-level defaults, which override default feature behavior configured in the MDEX Engine.

The core cartridges typically consist of a strongly typed configuration model, a response model, and a cartridge handler that processes the configuration model into the response model. By convention, they are named as follows:

Class name	Description
<CartridgeName>Config	The configuration model for the cartridge. For the core cartridges, the properties of this class represent all the configuration parameters that the cartridge handler needs to do its processing. It does not include configuration that can only be specified in the MDEX Engine or pass-through properties that are used by the reference application renderers without any modification by the cartridge handler.
<CartridgeName>Handler	The handler that processes a cartridge. The core cartridge handlers are responsible for layering the default configuration, instance configuration, and request-based configuration during processing. For more information about how to customize the handling of cartridge configuration (such as introducing additional sources of configuration), see Developing Cartridges <i>This part contains the following sections:</i> .
<CartridgeName>	The response model produced by the cartridge handler. Cartridge response models may include objects that are reused among cartridges. For example, the <code>ResultsList</code> and <code>RecordSpotlight</code> both contain <code>Record</code> objects.

For details about the implementations of these classes for specific cartridges, refer to the [Assembler API Reference \(Javadoc\)](#).

Default cartridge configuration

You can specify default configuration settings for the navigation cartridges in the reference implementation by adding values to the cartridge handler configuration in the Spring context file.

Cartridge handler configuration (including default configuration values) is specified as part of the Spring context file for the Assembler. In the Discover Electronics application, this is defined in `WEB-INF/assembler-context.xml`.

You specify the cartridge handler for a specific cartridge by defining a bean whose ID follows the format `CartridgeHandler_<CartridgeType>`, where the `<CartridgeType>` is the `id` of the corresponding cartridge template. For example, the cartridge handler for the Breadcrumbs cartridge is defined in the `CartridgeHandler_Breadcrumbs` bean. You can map more than one cartridge to the same cartridge handler.

Typically, you specify the default configuration for a cartridge by defining a `contentItemInitializer` property within the cartridge handler. The value of this property is a bean whose class implements the `ContentItemInitializer` interface. The core cartridges use the `ConfigInitializer` class, which provides a default implementation for merging the default, instance, and request-based configuration for a cartridge. Within the `contentItemInitializer` bean, the `defaults` property (if defined) must be a bean whose class is a `ContentItem` representing the cartridge configuration model to use as a default.

For information about the properties available in the configuration model for the core cartridges, refer to the [Assembler API Reference \(Javadoc\)](#) for the relevant configuration model class.

The following shows an example of default configuration for a Record Spotlight cartridge. The `defaults` property of the `ConfigInitializer` bean is an instance of `RecordSpotlightConfig` that has been initialized with a set of default values for the `fieldNames` property.

```

<bean id="CartridgeHandler_RecordSpotlight"
  class="com.endeca.infront.cartridge.RecordSpotlightHandler"
  parent="NavigationCartridgeHandler"
  scope="prototype">
  <property name="contentItemInitializer">
    <bean class="com.endeca.infront.cartridge.ConfigInitializer" scope="request">
      <property name="defaults">
        <bean class="com.endeca.infront.cartridge.RecordSpotlightConfig" scope="singleton">
          <property name="fieldNames">
            <list>
              <value>product.name</value>
              <value>product.brand.name</value>
              <value>product.price</value>
              <value>product.min_price</value>
              <value>product.max_price</value>
              <value>product.img_url_thumbnail</value>
              <value>product.review.avg_rating</value>
            </list>
          </property>
        </bean>
      </property>
    </bean>
  </property>
</bean>

```

Feature configuration in the MDEX Engine

There are two subcategories of MDEX Engine-level feature configuration: dynamic configuration that can be updated in a running MDEX Engine without re-indexing, and static configuration that must be specified at index time.

Dynamic configuration includes search interfaces, thesaurus, and automatic phrasing. Static configuration includes features such as stop words or precedence rules. Updating static configuration requires that you re-run the data ingest process before the changes can take effect. For detailed information on feature configuration in the MDEX Engine, refer to the *MDEX Engine Basic Development Guide* and the *MDEX Engine Advanced Development Guide*.

In addition, some features depend on certain Dgraph and Dgidx flags to enable or configure their functionality. For information about Dgraph and Dgidx flags, refer to the *Oracle Endeca Commerce Administrator's Guide*.

Cartridge instance configuration

The content administrator can configure each instance of a cartridge using Experience Manager in Endeca Workbench. The cartridge instance configuration is passed in as the argument to the `initialize()` method of the cartridge handler.

You define which aspects of a cartridge are configurable in Workbench via the cartridge template. Typically this is a subset of the properties in the configuration model. The sample templates provided as part of the Discover Electronics application are intended to cover the majority of use cases.

Cartridge templates for the reference application are included in the `reference\discover-data\cartridge_templates` directory, or `<app dir>\config\cartridge_templates` directory for a deployed application.

You can customize the templates for the core cartridges by adding properties to a template in addition to those required by the configuration model. These additional properties can either be processed by a custom cartridge handler implementation or passed through directly to the response model. Some of the templates in the Discover Electronics application define pass-through properties; these are described in the sections on the specific cartridges.

For details about configuring properties and editors in a cartridge template, refer to the "Template Property and Editor Reference" appendix in this guide.



Note: If you have purchased Oracle Endeca Guided Search only and do not have Oracle Endeca Experience Manager, most of the core cartridges are not available for configuration in Workbench. Of the core cartridges, only the Record Spotlight cartridge is available in Rule Manager. Custom cartridges that use primitive properties only (typically as pass-through properties) can also be configured in Rule Manager. The remaining cartridges can be configured with application-wide default values in the Spring context file for the Assembler.

Related Links

[Template Property and Editor Reference](#) on page 221

This section describes how to define basic content properties and associated editing interfaces in Experience Manager templates.

Request-based configuration

For some cartridges, it is appropriate for aspects of their configuration to be overridden at query time. Typically, request-based configuration is specified as URL query parameters.

To enable per-request configuration based on URL parameters, the `contentItemInitializer` bean of the cartridge handler can specify a `requestParamMarshaller` bean whose class is `RequestParamMarshaller` or a subclass. `RequestParamMarshaller` is a helper class that parses request parameters into properties of the cartridge configuration model.

For information about the URL query parameters that apply to the core cartridges, refer to the [Assembler API Reference \(Javadoc\)](#) for the relevant `RequestParamMarshaller` subclass. These classes define the URL parameters that the cartridge accepts and their mappings to properties on the configuration model.

Search cartridges

The Discover Electronics application includes reference implementations of several commonly-used search features. The configuration models for these features are described in the following section.

Search box

The Search Box cartridge enables the site visitor to enter search terms and view record results. If dimension search is enabled, dimension search results may also be displayed. A content administrator can configure Search Box behavior such as whether to apply search adjustments or display auto-suggest search results.

The response model for this cartridge is `SearchBox`.

The Search Box cartridge does not make use of a configuration model or a cartridge handler; properties specified in the cartridge template and in the end user's search request are passed through to the renderer.

The renderer for this cartridge makes use of a JavaScript module, `endeca-auto-suggest.js`, to display the auto-suggest panel for search suggestions.

MDEX Engine configuration for the Search Box cartridge

Because the Search Box enables keyword search for records and dimension values, most search configuration affects the behavior of this cartridge. This section focuses on record search configuration.

Dynamic configuration

The main aspects of search-related configuration that can be updated without re-indexing are the search interfaces for an application. Search interfaces specify a collection of properties and dimensions against which text searches are performed, and may also specify a default relevance ranking strategy. For information about creating search interfaces, refer to the *MDEX Engine Basic Development Guide*.

The properties and dimensions within a search interface must be enabled for record search as part of the data ingest process. For information about enabling properties and dimensions for search, refer to the *Developer Studio Help*.

Search results are also affected by thesaurus configuration that a content administrator can specify in Workbench.

Static configuration

Aspects of search behavior that must be specified at index time include stop words, stemming, and search characters.

- *stop words* are commonly occurring words (like "the") that are ignored for keyword search.
- *stemming* broadens search results to include root words and variants of root words.
- *search characters* configuration enables you to designate certain non-alphanumeric characters as significant for search.

For information about configuring these features, refer to the *MDEX Engine Basic Development Guide*.

Template configuration for the Search Box cartridge

The Search Box cartridge does not include a configuration model or a cartridge handler; instead, template configuration is passed through to the cartridge renderer.

The Search Box cartridge template includes properties that impact auto-suggest behavior. The auto-suggest panel itself is implemented as a configurable dynamic slot, and is configured separately.

The Search Box cartridge template includes the following configurable pass-through properties:

Property name	Description
contentCollection	This property specifies the content collection that should be used to populate the dynamic slot for the auto-suggest panel.
minAutoSuggestInputLength	This property specifies how many characters a user must type before the auto-suggest panel displays.
ruleLimit	This property sets the number of content items to return when populating the auto-suggest panel dynamic slot. It is limited by the evaluation limit of the specified contentCollection. The actual number of auto-suggest content items displayed is also limited by the rendering code, which only supports rendering a single auto-suggest panel by default.



Note: If you do not want to provide the option of enabling auto-suggest search results in Experience Manager, remove the properties and editors from the template, and remove the JavaScript module from the component.

Related Links

[Auto-suggest search results](#) on page 299

Auto-suggest search results display as the site visitor types in the search box, rather than displaying after the visitor has completed the search. In the Discover Electronics reference application, the auto-suggest panel is implemented as a content item that populates a dynamic slot in the Search Box cartridge.

Auto-suggest search results

Auto-suggest search results display as the site visitor types in the search box, rather than displaying after the visitor has completed the search. In the Discover Electronics reference application, the auto-suggest panel is implemented as a content item that populates a dynamic slot in the Search Box cartridge.

In addition to configuring the auto-suggest feature on the Search Box cartridge, a content administrator must also configure the display of different types of search suggestions. This section describes the cartridges that can be configured within the auto-suggest panel.

Currently, the only auto-suggest cartridge in the Discover Electronics reference application is one that displays dimension search results. It returns the same response model as the Dimension Search cartridge. Other possible uses for auto-suggest cartridges include those for Popular Searches, Featured Categories, and Product Search.

MDEX Engine configuration that impacts search results also applies to auto-suggest results. For example, enabling or disabling wildcard search on dimension search will affect the dimensions returned for a dimension search auto-suggest panel.

The JavaScript component of the Search Box in the Discover Electronics application acts as the renderer for the auto-suggest panel.

Template configuration for the auto-suggest panel

The cartridge template for the auto-suggest panel itself includes a dynamic content slot, with no other configuration.

Configuration model for the Auto-Suggest Dimension Search Results cartridge

The Auto-Suggest Dimension Search Results cartridge uses the same configuration model as the Dimension Search Results cartridge.

The configuration model for this cartridge is `DimensionSearchResultsConfig`. For an overview of this model, see "Configuration model for the Dimension Search Results cartridge" or refer to the Assembler API documentation (Javadoc).

Related Links

[Configuration model for the Dimension Search Results cartridge](#) on page 300

The Dimension Search Results cartridge configuration model controls the number, ranking, and display of returned results.

Cartridge handler configuration for the Auto-Suggest Search Results cartridge

Because the Auto-Suggest Dimension Search Results cartridge uses the same configuration model as the Dimension Search Results cartridge, it also shares the same cartridge handler.

The cartridge handler configuration maps the Dimension Search Auto-Suggest cartridge to the `DimensionSearchResultsHandler`. There are no application-wide default values set for this cartridge.

Related Links

[Search box](#) on page 297

The Search Box cartridge enables the site visitor to enter search terms and view record results. If dimension search is enabled, dimension search results may also be displayed. A content administrator

can configure Search Box behavior such as whether to apply search adjustments or display auto-suggest search results.

Template configuration for the Auto-Suggest Dimension Search Results cartridge

The Auto-Suggest Dimension Search Results cartridge populates the dynamic slot in the Auto-Suggest panel. The cartridge template is similar to the Dimension Search Results template.

The Auto-Suggest Dimension Search Results cartridge template allows a content administrator to configure the following properties on the configuration model:

- maxResults
- dimensionList
- maxResultsPerDimension
- showCountsEnabled

In addition, the cartridge template includes the following pass-through properties:

Property name	Description
title	Optional. A header that displays above the dimension search results.
displayImage	If set to true, a thumbnail image displays next to each dimension value. The URL of the image must be the value of a dimension value property named <code>img_thumbnail_url</code> .  Note: If there is no such property on dimension values in the data set, remove this option and its associated editor from the template to disable this feature.

Dimension search results

The Dimension Search Results cartridge displays refinement links based on the names of dimension values that match the search keywords entered by the site visitor.

The dimension search results display in a panel after the site visitor performs the search. These results provide suggestions for additional navigation refinements based on the search terms.

The response model for this cartridge is `DimensionSearchResults`. It contains a list of `DimensionSearchGroup` objects that in turn contain `dimensionSearchValues` that provide refinement links.

Configuration model for the Dimension Search Results cartridge

The Dimension Search Results cartridge configuration model controls the number, ranking, and display of returned results.

The configuration model for this cartridge is `DimensionSearchResultsConfig`. It includes the following properties:

Property name	Description
enabled	Enables or disables the display of returned dimension refinements. By default, this property is <code>false</code> . It is enabled via URL request by setting the <code>Dy</code> URL parameter to <code>1</code> .
maxResults	Specifies the maximum number of dimension value results across all dimensions to display.
maxResultsPerDimension	Specifies the maximum number of dimension values to display per dimension.

Property name	Description
dimensionList	Specifies the dimensions on which to perform dimension search. The results display based on the order in which the dimensions are specified, up to the maximum number of suggestions.
showCountsEnabled	Specifies whether to display refinement counts in dimension search results.
relRank	Optional. Specifies a relevance ranking string to use for dimension search, such as "first,static(nbins,desc)". If you do not set this property, dimension value relevance ranking is set to the default (alpha, numeric, or manual) defined in Developer Studio.

MDEX Engine configuration for dimension search results

Different aspects of dimension search can be configured on a global or per-dimension basis.

Dynamic configuration

You can specify global dimension search behavior in the Dimension Search Configuration editor in Developer Studio. Oracle recommends enabling wildcard search for dimensions, especially if you are using the Auto-Suggest Dimension Search cartridge or the Dimension Value Boost-Bury editor. Wildcard search enables partial matches to be returned for searches in addition to full word matches (for example, a search for "pink" would also return "gray/pink") which is useful for displaying suggestions while the user is typing search terms.

Additional options include whether to return only the highest ancestor dimension value, and whether to return inert dimension values in dimension search results. For more information about global dimension configuration, refer to the *Developer Studio Help*.

Static configuration

You can configure dimension-specific search behavior in the Dimension editor in Developer Studio. This includes whether to search across the entire dimension hierarchy rather than only individual dimension values and also enables you to specify dimension value synonyms to be used for search. For more information about per-dimension configuration, refer to the *Developer Studio Help*.

Cartridge handler configuration for Dimension Search Results

The Dimension Search Results cartridge handler extends the `NavigationCartridgeHandler`.

The cartridge handler uses the `DimensionSearchResultsConfigInitializer` to merge the layered configuration. The included `requestParamMarshaller` bean enables URL request-based configuration for the cartridge, which is required for dynamically enabling the feature.

Template configuration for the Dimension Search Results cartridge

The Dimension Search Results cartridge template allows a content administrator to configure how many results should be displayed to the end user, and how they should display. The cartridge template also includes two pass-through properties that are passed directly to the cartridge renderer.

The Dimension Search Results cartridge template allows a content administrator to configure the following properties on the configuration model:

- `maxResults`
- `dimensionList`
- `maxResultsPerDimension`
- `showCountsEnabled`

In addition, the cartridge template includes the following pass-through properties:

Property name	Description
title	Optional. A header that displays above the dimension search results.
displayImage	If set to true, a thumbnail image displays next to each dimension value. The URL of the image must be the value of a dimension value property named <code>img_thumbnail_url</code> .  Note: If there is no such property on dimension values in the data set, remove this option and its associated editor from the template to disable this feature.

URL request parameters for the Dimension Search Results cartridge

The display of the Dimension Search Results cartridge on a page is controlled by setting the value of the `enabled` property on the cartridge configuration model at runtime via the `Dy` URL parameter.

The cartridge renderer in the reference implementation sets the `Dy` parameter to 1 in all cases. While this is equivalent to setting the property to `true` in the cartridge handler configuration, or as a non-editable property in the cartridge template, the intent is to demonstrate where the logic belongs in the application.

Property name	URL Parameter	Description
enabled	<code>Dy</code>	Enables or disables the display of returned dimension refinements. Setting <code>Dy=1</code> sets the property to <code>true</code> .

Search adjustments

Search adjustments include automatic spelling correction, automatic phrasing, and Did You Mean functionality.

The response model for this cartridge is `SearchAdjustments`.

The behavior of the spelling correction and Did You Mean features are configured at the MDEX Engine level. The Search Adjustments cartridge enables content administrators to specify whether or not search adjustments messaging displays on a page; it does not have any configuration options in Experience Manager.

Configuration model for the Search Adjustments cartridge

The Search Adjustments cartridge configuration model enables you to enable or disable automatic phrasing and automatic spelling correction. If query debugging features are enabled in your application, you can also enable or disable debugging information about Word Interpretation.

The configuration model for this cartridge is `SearchAdjustmentsConfig`. It includes the following properties:

Property name	Description
<code>phraseSuggestionEnabled</code>	Specifies whether to enable automatic phrasing. Defaults to <code>true</code> . Set via URL request by setting the <code>Ntp</code> URL parameter to 1.
<code>spellSuggestionEnabled</code>	Specifies whether to enable automatic spelling correction. Defaults to <code>false</code> . Set via URL request by setting the <code>Nty</code> URL parameter to 1.
<code>showWordInterp</code>	If query debugging features are enabled, this property enables debugging information about word or phrase substitutions as a map that can be accessed via <code>SearchAdjustments.getInterpretedTerms()</code> . For additional information, see "About query debugging results in the reference application."

MDEX Engine configuration for the Search Adjustments cartridge

Search adjustments features are configured at indexing and at Dgraph startup.

Dynamic configuration

You can specify a list of phrases to be automatically applied to text search queries in Developer Studio. For more information about configuring automatic phrasing, refer to the *MDEX Engine Advanced Development Guide*.

Static configuration

You can configure the constraints on the spelling dictionaries for record search and dimension search in the Spelling editor in Developer Studio. These settings determine the size of the spelling dictionary that is generated at indexing time. Larger spelling dictionaries lead to slower performance of spelling correction at query time; setting more restrictive constraints on the contents of the spelling dictionary can lead to improved query performance. For more information about tuning the size of the spelling dictionary, refer to the *MDEX Engine Performance Tuning Guide*.

Dgidx flags

You specify the spelling mode as a flag to Dgidx. Generally, applications that only need to correct normal English words can enable just the default Aspell module. Applications that need to correct international words, or other non-English/non-word terms (such as part numbers) should enable the Espell module. For more information on spelling modes and the associated Dgidx flags, refer to the *MDEX Engine Advanced Development Guide*.

The Deployment Template application configuration for the Discover Electronics reference application has spelling correction and Did You Mean enabled as in the following example:

```
<!--
#####
# Dgidx
#
-->
<dgidx id="Dgidx" host-id="ITLHost">
  <properties>
    <property name="numLogBackups" value="10" />
    <property name="numIndexBackups" value="3" />
  </properties>
  <args>
    <arg>-v</arg>
    <arg>--compoundDimSearch</arg>
  </args>
  <log-dir>./logs/dgidxs/Dgidx</log-dir>
  <input-dir>./data/forge_output</input-dir>
  <output-dir>./data/dgidx_output</output-dir>
  <temp-dir>./data/temp</temp-dir>
  <run-aspell>true</run-aspell>
</dgidx>
```

Dgraph flags

You enable spelling correction and Did You Mean through Dgraph flags. Additional Dgraph flags provide advanced tuning options for the spelling adjustment features that affect performance and behavioral characteristics, such as the threshold for the number of hits at or above which spelling corrections or Did You Mean suggestions are not generated. For more information on Dgraph flags for search adjustment tuning, refer to the *MDEX Engine Advanced Development Guide*.



Note: Auto-correct should be relatively conservative. You only want the engine to complete the correction when there is a high degree of confidence. For more aggressive suggestions, it is best to use Did You Mean.

The Deployment Template application configuration for the Discover Electronics reference application has spelling correction and Did You Mean enabled as in the following example:

```
<!--
#####
# Global Dgraph Settings - inherited by all dgraph components.
#
-->
<dgraph-defaults>
  <properties>
    <!-- additional elements removed from this example -->
  </properties>
  <directories>
    <!-- additional elements removed from this example -->
  </directories>
  <args>
    <arg>--threads</arg>
    <arg>2</arg>
    <arg>--whymatch</arg>
    <arg>--spl</arg>
    <arg>--dym</arg>
    <arg>--dym_lthresh</arg>
    <arg>5</arg>
    <arg>--dym_nsug</arg>
    <arg>3</arg>
    <arg>--stat-abins</arg>
  </args>
  <startup-timeout>120</startup-timeout>
</dgraph-defaults>
```

Cartridge handler configuration for Search Adjustments

The Search Adjustments cartridge handler extends the `NavigationCartridgeHandler`. The application-wide default configuration in the Assembler context file allows you to enable or disable the word interpretation debugging feature.

The cartridge handler uses a `contentItemInitializer` to merge the layered configuration. The included `requestParamMarshaller` bean enables URL request-based configuration for the cartridge, which is required for dynamically disabling or enabling automatic phrase suggestions and spelling correction.

Related Links

[About implementing automatic phrasing](#) on page 305

You can configure the MDEX Engine to consider certain combinations of words in a text search as a phrase search and specify whether to apply phrasing automatically to a site visitor's text search queries.

Template configuration for the Search Adjustments cartridge

The cartridge template for the Search Adjustments cartridge does not include any configurable properties. A content administrator can add the cartridge to a page in order to enable the display of Search Adjustments, but cannot otherwise configure cartridge behavior.

URL request parameters for the Search Adjustments cartridge

Automatic phrasing and spelling correction are controlled by setting the value of their respective properties on the cartridge configuration model at runtime via the `Ntp` and `Nty` URL parameters.

The cartridge renderer in the reference implementation sets both parameters to 1 in all cases. While this is equivalent to setting the properties in the cartridge handler configuration, or in the cartridge template, the intent is to demonstrate where the logic belongs in the application.

Property name	URL Parameter	Description
phraseSuggestionEnabled	Ntp	Specifies whether to enable automatic phrasing.
spellSuggestionEnabled	Nty	Specifies whether to enable automatic spelling correction.

About implementing automatic phrasing

You can configure the MDEX Engine to consider certain combinations of words in a text search as a phrase search and specify whether to apply phrasing automatically to a site visitor's text search queries.

The high level steps for enabling automatic phrasing are:

- Enabling the MDEX Engine to compute phrases
- Configuring the default behavior of the Assembler application as to whether or not to automatically apply computed phrases
- Adding application logic to enable Did You Mean suggestions or override the default automatic phrasing behavior in certain situations

You enable the MDEX Engine to compute phrases that can be applied to a site visitor's text search by creating a phrase dictionary. For information about creating a phrase dictionary, refer to the section on Automatic Phrasing in the *MDEX Engine Developer's Guide*.

You can configure the default behavior of the Assembler application as to whether to automatically rewrite a text search as a phrase search or keep it as a search for individual keywords using the following property on the Filter State object:

Property	Description
autoPhraseEnabled	If set to <code>true</code> , instructs the MDEX Engine to compute phrases that can be applied to a text search and automatically rewrite the query using the phrased version. Automatic phrasing is enabled by default.

The `autoPhraseEnabled` setting on the default Filter State can be overridden at query time using the URL parameter `autophrase`. If the value of `autophrase` is 1, then computed phrases are automatically applied to the query. If the value is 0 then phrases may still be computed, but are not automatically applied to the query.

The Filter State configuration in the Assembler context file for the Discover Electronics reference application is shown below:

```

<bean id="navigationStateBuilder" scope="request"
  class="com.endeca.infront.navigation.url.UrlNavigationStateBuilder">
  <!-- additional elements removed from this example -->
  <property name="defaultFilterState">
    <bean scope="singleton" class="com.endeca.infront.navigation.model.FilterState">
      <property name="rollupKey" value="product.code" />
      <property name="autoPhraseEnabled" value="true" />
      <!-- <property name="securityFilter" value="" /> -->
      <!-- <property name="languageId" value="en" /> -->
    </bean>
  </property>
</bean>

```

```
<!-- additional elements removed from this example -->
</bean>
```

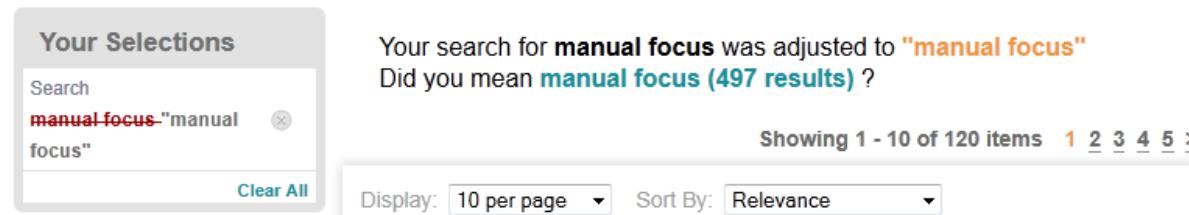
Interaction with the Did You Mean feature

Whether automatic phrasing is applied or not, you can specify whether to return a "Did You Mean" link for the alternate version using the `Nty` URL parameter. For example, if phrasing was automatically applied, the Did You Mean suggestion would provide a link to the unphrased version of the query, and vice versa. If the value of `Nty` is 1, then the Assembler returns suggestions for the alternate form of the query. If the value is 0, no suggestions are returned.

 **Note:** The `Nty` parameter controls Did You Mean suggestions for regular text search as well as for automatic phrasing.

Phrase search scenario: Automatically applying phrases

In the Discover Electronics application, the default behavior is to automatically apply phrases to text search queries and to return the unphrased version as a search suggestion.



The screenshot shows a search interface with the following details:

- Your Selections:** A search bar containing the query `manual focus` with a red underline, followed by the phrase `"manual focus"`.
- Search Results:** A message stating "Your search for **manual focus** was adjusted to **"manual focus"**". Below this, a link "Did you mean **manual focus (497 results)**?" is shown in blue.
- Pagination:** "Showing 1 - 10 of 120 items" with links for pages 1 through 5.
- Display Options:** "Display: 10 per page" and "Sort By: Relevance".

In this scenario, `autoPhraseEnabled` is set to `true` on the default Filter State object, and the Search Box cartridge sets `Nty=1` on the text search query. The user has two choices:

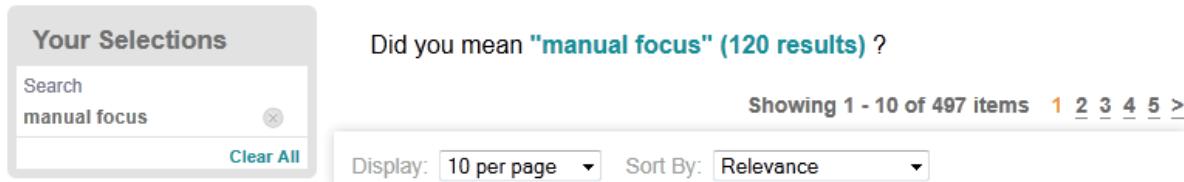
- Select the Did You Mean suggestion to search for the keywords separately, rather than as a phrase. This link sends the same query with the URL parameter `Ntp=0` to override the Filter State configuration, and also sets `Nty=0` since we do not need to suggest the phrased version of the query after the user has decided to use the unphrased version.
- Make another selection on the page, such as clicking on a refinement or advancing to the next page of results. This signifies acceptance of the automatically applied phrase, so we keep `autoPhraseEnabled=true` from the Default Filter State and suppress further suggestions by setting `Nty=0`.

These outcomes are summarized in the following table:

User action	Autophrase setting (<code>Ntp</code>)	Did You Mean setting (<code>Nty</code>)	Result
Initial search	<code>true</code>	<code>Nty=1</code>	Phrase is automatically applied to the text search. A Did You Mean suggestion is offered for the unphrased version.
Select Did You Mean suggestion	<code>Ntp=0</code>	<code>Nty=0</code>	Phrase is not applied to the search. No suggestion is offered.
Make another follow-on selection	<code>true</code>	<code>Nty=0</code>	Phrase continues to be automatically applied. Suggestions are no longer offered.

Phrase search scenario: Phrases as a search suggestion

You can configure the application not to apply phrases by default, but to return phrases as a search suggestion.



In this scenario, `autoPhraseEnabled` is set to `false` on the default Filter State object, and the Search Box cartridge sets `Nty=1` on the text search query. The user has two choices:

- Select the Did You Mean suggestion to consider the text search as a phrase. This link sends the same query with the URL parameter `Ntp=1` to override the default Filter State configuration, and also sets `Nty=0` since we do not need to suggest the unphrased version of the query after the user has decided to use the phrased version.
- Make another selection on the page, such as clicking on a refinement or advancing to the next page of results. This signifies acceptance of the unphrased query, so we keep `autoPhraseEnabled` set to `false` and suppress further suggestions by setting `Nty=0`.

These outcomes are summarized in the following table:

User action	Autophrase setting (Ntp)	Did You Mean setting (Nty)	Result
Initial search	false	Nty=1	Phrase is not applied to the text search. A Did You Mean suggestion is offered for the phrased version.
Select Did You Mean suggestion	Ntp=1	Nty=0	Phrase is automatically applied to the search. No suggestion is offered.
Make another follow-on selection	false	Nty=0	Text search continues to be treated as individual keywords instead of as a phrase. Suggestions are no longer offered.

Keyword redirects

Content administrators can configure keyword redirects that redirect a front-end user to a new page if the user's search terms match the set keyword.

When an end user enters a search term that matches a keyword redirect, the Assembler returns the redirect URI with the response model. The Assembler response can be limited to the redirect URI, or it can also return the results for the user's search term.

The content administrator specifies a search term, match mode, and redirect URI on the **Keyword Redirects** page in Workbench.

Cartridge handler configuration for keyword redirects

The Assembler API includes a `RedirectAwareContentIncludeHandler` that implements keyword redirect functionality.

The cartridge handler takes the following two properties:

- `defaultFullAssembleOnRedirect` — A Boolean that specifies whether to return search results in addition to the redirect URI when making an `assemble()` call. Defaults to `false`. If you do not necessarily wish to execute a redirect (for cases where the redirect URI is displayed as a link, or may be skipped entirely if the user is not on a specific device), you must set this property to `true`.

- `defaultRedirectCollection` — A string that contains the name of the keyword redirect collection in the Endeca Configuration Repository. Setting a null or empty value for this property disables keyword redirect functionality.

The cartridge handler configuration in the Assembler context file for Discover Electronics is shown below:

```
<!--
~~~~~
~ BEAN: CartridgeHandler_ContentInclude
~ Used by the assembler service when keyword redirects are not enabled
-->
<bean id="CartridgeHandler_ContentInclude"
  class="com.endeca.infront.cartridge.ContentIncludeHandler"
  scope="prototype">
  <property name="contentSource" ref="contentSource" />
</bean>

<!--
~~~~~
~ BEAN: CartridgeHandler_RedirectAwareContentInclude
~ For root calls to the assembler when keyword redirects are desired
-->
<bean id="CartridgeHandler_RedirectAwareContentInclude"
  class="com.endeca.infront.cartridge.RedirectAwareContentIncludeHandler"
  scope="prototype">
  <property name="contentSource" ref="contentSource" />
  <property name="contentBroker" ref="contentRequestBroker" />
  <property name="navigationState" ref="navigationState" />
  <property name="defaultFullAssembleOnRedirect" value="false"/>
</bean>
```

Content XML for keyword redirects

You can override the default settings for the `fullAssembleOnRedirect` or `redirectCollection` properties by setting new values in the content XML that is retrieved by the `RedirectAwareContentIncludeHandler`.

The primary use case for setting these properties on content XML is for deployments running the Assembler service. Keyword redirects are programmatically enabled in the service, so by default the feature is explicitly disabled for services where it does not apply (Dimension Search and Record Details) by including an element in the content XML that sets `redirectCollection` to a null value.



Note: If you are creating your Assembler application in Java, you can disable keyword redirects by using the `ContentInclude` class instead of `RedirectAwareContentInclude` for those services where you wish to disable the feature.

About using keyword redirects with the Assembler service

The Assembler service in the Discover Electronics application implements the `com.endeca.infront.assembler.servlet.AbstractAssemblerServlet` abstract class. Keyword redirect configuration is configured in the application's `web.xml` file.

The JSON and XML servlets in the Discover Electronics reference application are configured in `reference\discover-service\WEB-INF\web.xml`:

```
<servlet>
  <servlet-name>JsonAssemblerServiceServlet</servlet-name>
  <servlet-class>com.endeca.infront.assembler.servlet.spring.SpringAssemblerServlet</servlet-class>
  <init-param>
    <param-name>assemblerFactoryID</param-name>
    <param-value>assemblerFactory</param-value>
  </init-param>
```

```

<init-param>
    <param-name>responseWriterID</param-name>
    <param-value>jsonResponseWriter</param-value>
</init-param>
<init-param>
    <param-name>enableKeywordRedirects</param-name>
    <param-value>true</param-value>
</init-param>
</servlet>

```

When the application queries the Assembler service, the redirect URI is returned as part of the response.

About handling keyword redirects in an application

In order to execute a redirect, an application must include logic for handling the URI components returned from the Assembler. You must use the `RedirectAwareContentInclude` class for any content items that require keyword redirect functionality.

The `assemble.jsp` service uses the `RedirectAwareContentInclude` class to enable keyword redirects, as shown below:

```

<%@page import="com.endeca.infront.cartridge.RedirectAwareContentInclude"%>

...
AssemblerFactory assemblerFactory = (AssemblerFactory)webappCtx.getBean("assemblerFactory");
Assembler assembler = assemblerFactory.createAssembler();

// Retrieve the content for the given content uri
ContentItem contentItem = new RedirectAwareContentInclude("/pages" + contentUri);

// Assemble the content
ContentItem responseContentItem = assembler.assemble(contentItem);

```

The Assembler response

When an end user enters a search term that matches a keyword redirect configured in Workbench, the Assembler response includes a `ContentItem` with the necessary information for creating a destination URI.

The following example shows a JSON response from the Guided Search service when `fullAssembleOnRedirect` is `false`:

```

{
    endeca:siteRootPath: "/services",
    endeca:contentPath: "/guidedsearch",
    endeca:assemblerRequestInformation:
    {
        @type: "AssemblerRequestEvent",
        endeca:assemblyStartTimeStamp: 1341943119538,
        endeca:assemblyFinishTimeStamp: 1341943119546,
        endeca:contentPath: "/guidedsearch",
        endeca:sessionId: "FF9D21355A3CBB9DFF75614DD7D2948D",
        endeca:siteRootPath: "/services"
    },
    endeca:redirect:
    {
        @type: "Redirect",
        link:
        {
            @class: "com.endeca.infront.cartridge.model.UrlAction",
            url: "/browse/cameras/_/N-25y6"
        }
    }
}

```

```

    }
}
```

The keyword redirect information is included in the `ContentItem` with the key `endeca:redirect`. The value specifies an `Action` object with the destination URI, which may be either relative or absolute.

Using the Assembler response

You must retrieve and use the information from the Assembler response in your application to execute a keyword redirect. In the Discover Electronics reference application, this is accomplished in the `assemble.jsp` service:

```

<%@ taglib prefix="util" uri="/WEB-INF/tlds/functions.tld" %>
<%@ taglib prefix="c" uri="http://java.sun.com/jsp/jstl/core" %>

...
// Assemble the content
ContentItem responseContentItem = assembler.assemble(contentItem);

request.setAttribute("component", responseContentItem);
request.setAttribute("rootComponent", responseContentItem);

Map map = (Map) request.getAttribute("component");
if (map.containsKey("endeca:redirect")) {
    request.setAttribute("action", ((ContentItem) map.get("endeca:redirect")).get("link"));
    %>
    <c:redirect url="${util:getUrlForAction(action)}" />
    <%
}
...
}
```

For more information about `Action` objects in an Assembler application, see "Working with Application URLs," or consult the *Assembler API Reference (Javadoc)*.

Guided Navigation cartridges

The following sections provide an overview of the configuration models for Guided Navigation features included with Tools and Frameworks and implemented in Discover Electronics.

Refinement menu

The Refinement Menu cartridge displays dimension values within a single dimension for Guided Navigation. It supports dimension value boost and bury.

The response model for this cartridge is `RefinementMenu`, which contains a list of `Refinement` objects.

Dimension value boost and bury

Dimension value boost and bury is a feature that enables re-ordering of dimension values within a particular dimension for Guided Navigation. With dimension value boost, you can assign specific dimension values to ranked strata, with those in the highest stratum being shown first, those in the second-ranked stratum shown next, and so on. With dimension value bury, you can assign specific dimension values to strata that are ranked much lower relative to others. This boost/bury mechanism therefore lets you manipulate ranking of returned dimension values in order to promote or push certain refinements to the top or bottom of the navigation menu.

The Refinement Menu cartridge enables the content administrator to specify an ordered list of dimension values to boost and an ordered list of dimension values to bury. Each dimension value is translated into its own stratum in the query that returns refinements so as to preserve the exact order of refinements specified by the content administrator.

For more information about dimension value boost and bury, refer to the *MDEX Engine Basic Development Guide*.

Configuration model for the Refinement Menu cartridge

The Refinement Menu cartridge configuration model allows you to configure sorting, "Show More..." link behavior, and boosted and buried refinements. Additionally, it includes a `whyPrecedenceRuleFired` property that can be used for debugging precedence rule behavior in your application.

The configuration model for this cartridge is `RefinementMenuConfig`. It includes the following properties:

Property name	Description
<code>dimensionId</code>	A string representing the id of the dimension being configured.
<code>boostRefinements</code>	An ordered list of dimension value refinements to display at the top of the list.
<code>buryRefinements</code>	An ordered list of dimension value refinements to display at the bottom of the list.
<code>sort</code>	The base sort order of dimension values within this dimension. This property should have one of the following values: <ul style="list-style-type: none"> • <code>default</code> — Sort dimension values according to the application configuration for this dimension. • <code>static</code> — Sort dimension values in alphabetic or numeric order, depending on the dimension configuration. • <code>dynRank</code> — Sort dimension values so that the refinements with the highest number of records display first.
<code>showMoreLink</code>	A Boolean indicating whether to enable a link to show more refinements than are displayed by default.
<code>moreLinkText</code>	A string representing the text to use for the "show more refinements" link.
<code>lessLinkText</code>	A string representing the text to use for the "show fewer refinements" link.
<code>numRefinements</code>	A string representing the number of refinements to display by default, or when a user clicks the "show fewer refinements" link.
<code>maxNumRefinements</code>	A string representing the maximum number of refinements to display when a user clicks the "show more refinements" link.
<code>refinementsShown</code>	A string that sets the amount of refinements to return, from the following values: <ul style="list-style-type: none"> • <code>none</code> — returns no refinements. • <code>some</code> — returns <code>numRefinements</code> refinements. • <code>all</code> — returns <code>maxNumRefinements</code> refinements.
<code>showMore</code>	(Deprecated) A Boolean indicating whether to display the <code>maxNumRefinements</code> number of menu items. When this value is <code>false</code> , the number of menu items generated is limited by <code>numRefinements</code> , and a "show more refinements" link is generated. This value should be set using <code>showMoreIds</code> URL parameter when the "show more refinements" link is selected.

Property name	Description
useShowMoreIdsParam	(Deprecated) A Boolean that sets whether to use the <code>showMoreIds</code> URL parameter when determining how many refinements to display. If <code>false</code> , the <code>showMore</code> property on the <code>RefinementMenuConfig</code> object is used instead. If this property is set to <code>true</code> , refinements cannot be collapsed. Defaults to <code>true</code> .
whyPrecedenceRuleFired	If query debugging features are enabled, this property enables debugging information about why precedence rules fired on a query in a <code>DGraph</code> . <code>WhyPrecedenceRuleFired</code> property for each root dimension value. For additional information, see "About query debugging results in the reference application."



Important: The `useShowMoreIdsParam` property and associated `showMoreIds` URL parameter are included in this release for backwards compatibility. Use the `refinementsShown` property if you are refactoring your code or developing a new application.

Notes on sorting

The `static` sort option is described as "Alphanumeric" sorting in the Experience Manager user interface for the default Refinement Menu cartridge. Dimension values are ordered alphanumerically within a dimension by default, however it is possible to manually specify a dimension order (for example, using the Dimension Values editor in Developer Studio). This custom dimension value order is used when `static` sorting is specified. To ensure alphanumeric sorting of dimension values, do not specify a custom dimension value order.

Dynamic refinement ranking is incompatible with displaying disabled refinements for a dimension. In the default Refinement Menu cartridge, the option to show disabled refinements is not available unless the content administrator has explicitly selected `static` sorting.

MDEX Engine configuration for Guided Navigation

No special configuration is necessary to enable Guided Navigation, however, there is some static configuration that affects the display of refinements.

Static configuration

In the Dimension editor in Developer Studio, you can configure dimensions to be:

- *multiselect*—A multiselect dimension enables a user to select more than one refinement at the same time. You can specify whether the navigation results when multiple refinements are selected are treated as a Boolean AND or Boolean OR on a per-dimension basis.
- *hidden*—A hidden dimension does not display in Guided Navigation; however, users can still search for records based on their dimension values in a hidden dimension.

You can also configure the following refinement behavior on a per-dimension basis:

- *dynamic refinement ranking*—Dynamic ranking returns refinements based on their popularity (number of associated record results for each refinement). This is a default setting that can be overridden by the content administrator in Experience Manager.
- *refinement statistics*—Enabling refinement statistics returns the number records (or aggregated records) are associated with each refinement so that this information can be displayed in the application.

Additionally, you can designate specific dimension values as inert. For more information about these configuration options, refer to the *MDEX Engine Basic Development Guide*.

Cartridge handler configuration for the Refinement Menu cartridge

The Refinement Menu cartridge handler extends the `NavigationCartridgeHandler`. The application-wide default configuration in the Assembler context file determines the behavior of collapsed dimensions and "show

more" and "show less" links, and can be set to enable or disable the precedence rule debugging feature if query debugging features are enabled.

The cartridge handler uses a `contentItemInitializer` to merge the layered configuration. The included `requestParamMarshaller` bean enables URL request-based configuration for the cartridge, which is required for disabling or enabling the full list of refinement results returned when the end user clicks the "show more refinements" link.

Template configuration for the Refinement Menu cartridge

The Refinement Menu cartridge template allows a content administrator to configure which dimension to query for the cartridge and how many results should display. It also allows control over boosted and buried dimension refinements, in order to modify the order in which dimensions display to the end user.

The Refinement Menu cartridge template allows a content administrator to configure the following properties on the configuration model:

- `dimensionId`
- `sort`
- `showMoreLink`
- `moreLinkText`
- `lessLinkText`
- `numRefinements`
- `maxNumRefinements`
- `boostRefinements`
- `buryRefinements`

In addition, the cartridge template includes the following pass-through property:

Property name	Description
<code>dimensionName</code>	The name of the string property that represents the dimension name. This is required by the Dimension Selector editor to enable a content administrator to select a dimension by name, rather than by ID.

URL request parameters for the Refinement Menu cartridge

You can configure the Refinement Menu cartridge at runtime by setting the value of the `DYNAMIC_REFINEMENT_MENU_CONFIG` property on the `RefinementMenuRequestParamMarshaller` via the `Nrmc` URL parameter.

The sample cartridge renderer includes logic for displaying the `maxNumRefinements` number of results when a user clicks on the "show more refinements" link.

Property name	URL parameter	Description
<code>DYNAMIC_REFINEMENT_MENU_CONFIG</code>	<code>Nrmc</code>	The <code>Nrmc</code> parameter takes multiple arguments allow you to configure dimension refinement behavior in the cartridge.
<code>showMore</code>	<code>ShowMoreIds</code>	(Deprecated) A Boolean indicating whether to display the <code>maxNumRefinements</code> number of menu items. Use the <code>refinementsShown</code> property if you are refactoring your code or developing a new application.

About `Nrmc` URL parameter syntax

The `Nrmc` parameter takes the following values:

- Dimension ID — Required. The ID of the dimension you wish to configure.

- `+show:<value>` — Required; `<value>` is the value to pass to the `refinementsShown` property on the configuration object.

The configuration for each dimension is separated by a vertical pipe, as in the example below:

```
20001+show:all|20002+show:some
```



Note: You can also use the notation used with the Presentation API, for example:

`Nrc=id+10074+expand+true+more+true`. For more information about this notation, see the *MDEX Engine Basic Development Guide*.

Navigation Container

The Navigation Container is provided as an alternative the refinement menu cartridge for implementations using Oracle Endeca Guided Search with the packaged services. It enables you to retrieve the full list of available dimension refinements for a dimension query.

The response model for the Navigation Container includes a list of `RefinementMenu` objects that each include the records within a dimension refinement. The `NavigationContainerHandler` handles the "show more refinements" link and associated link Action for each of these refinements, and also controls whether to display debugging information.

Configuration model for the Navigation Container

The Navigation Container configuration model includes the `List<String>` property of dimension IDs that are returned with the response model. Since it is a dimension navigation feature, it includes a `whyPrecedenceRuleFired` property that can be used for debugging precedence rule behavior in your application.

The configuration model for this cartridge is `NavigationContainerConfig`. It includes the following properties:

Property name	Description
<code>showMoreIds</code>	A List of dimension IDs to return as expanded lists of available refinements. Any dimension refinements not included in this List are returned in the default, shorter form output by the MDEX Engine.
<code>moreLinkText</code>	A string representing the text to use for the "show more refinements" link. The same string is used for each of the included dimension refinements.
<code>lessLinkText</code>	A string representing the text to use for the "show fewer refinements" link. The same string is used for each of the included dimension refinements.
<code>refinementsShownByDefault</code>	A Boolean indicating whether the refinement menus should be fully expanded. Defaults to <code>true</code> . When using a dataset that includes dimensions with a large number of refinements, you should set this to <code>false</code> .
<code>refinementsShown</code>	A string that sets the amount of refinements to return on each refinement menu, from the following values: <ul style="list-style-type: none"> • <code>none</code> — returns no refinements. • <code>some</code> — returns <code>numRefinements</code> refinements.
<code>useShowMoreIdsParam</code>	(Deprecated) A Boolean that sets whether to use the <code>showMoreIds</code> URL parameter when determining how many refinements to display. If <code>false</code> , the <code>showMore</code> property on the <code>RefinementMenuConfig</code> object is used instead. If this property is set to <code>true</code> , refinements cannot be collapsed. Defaults to <code>true</code> .

Property name	Description
whyPrecedenceRuleFired	If query debugging features are enabled, this property enables debugging information about why precedence rules fired on a query in a DGraph. WhyPrecedenceRuleFired property for each root dimension value. For additional information, see "About query debugging results in the reference application."

Cartridge handler configuration for the Navigation Container

The Navigation Container handler extends the `NavigationCartridgeHandler`. The application-wide default configuration in the Assembler context file determines the behavior of collapsed dimensions and "show more" and "show less" links, and can be set to enable or disable the precedence rule debugging feature if query debugging features are enabled.

The cartridge handler uses a `contentItemInitializer` to merge the layered configuration. The included `requestParamMarshallers` bean enables URL request-based configuration for the cartridge, which is required for modifying the properties on the response model through URL parameters.

URL request parameters for the Navigation Container

Because the Navigation Container returns a list of `RefinementMenu` objects, it takes the same `Nrmc` URL parameter as the Refinement Menu cartridge.

Property name	URL parameter	Description
DYNAMIC_REFINEMENT_MENU_CONFIG	Nrmc	The <code>Nrmc</code> parameter takes multiple arguments allow you to configure dimension refinement behavior in the cartridge.
whyPrecedenceRuleFired	whyPrecedenceRuleFired	If query debugging is enabled for the reference application, this property allows you to include debugging information about why precedence rules fired on a query in a DGraph. WhyPrecedenceRuleFired property for each dimension value.

For details on configuring the `Nrmc` parameter, see "URL request parameters for the Refinement Menu cartridge."

Breadcrumbs

The Breadcrumbs cartridge displays the parameters defining the search or navigation state for the current set of search results.

The response model for this cartridge is `Breadcrumbs`, which may contain `SearchBreadcrumb`, `RefinementBreadcrumb`, `RangeFilterBreadcrumb`, and `GeoFilterBreadcrumb` objects as appropriate. Each breadcrumb contains information about search or navigation selections that the end user has made, and provides links to remove that selection from the filter state.

The Breadcrumbs cartridge does not have any associated Experience Manager configuration options or MDEX Engine configuration.

Cartridge handler configuration for Breadcrumbs

The Breadcrumbs cartridge handler extends the `NavigationCartridgeHandler`, but otherwise does not require any additional configuration.

Results cartridges

The following sections provide an overview of the configuration models for features that display search results in the reference implementation.

Results list

The Results List cartridge displays search and navigation results in a list view.

The response model for this cartridge is `ResultsList`, which contains a list of `Record` objects and `SortOptionLabel` objects that enable the end user to select from a set of pre-defined sort orders.

About the order of records in the record list

The order of records returned by the MDEX Engine is determined by a sort key or relevance ranking strategy depending on the type of query that returns the results.

Relevance ranking is applied when the query includes a text search. *Record sorting* is applied to all other queries including navigation queries. The sort options that are available to the end user in the application represent static sort orders that are not based on relevance to any search terms.

Record boost and bury

Record boost and bury is a feature that enables fine-grained re-ordering of records within search or navigation results. With record boost, you can assign records to ranked strata, with those in the highest stratum being shown first, those in the second-ranked stratum shown next, and so on. With record bury, you can assign records to strata that are ranked much lower relative to others. This boost/bury mechanism therefore lets you manipulate ranking of returned record results in order to promote or push certain records to the top or bottom of the results list. The records in each stratum are defined as a set of specific records or a navigation state that the records must satisfy. A record is assigned to the highest stratum whose definition it matches, so boosting takes precedence over burying. Record boost and bury apply regardless of whether the records returned are the results of a search or navigation query.

The core Results List cartridge enables the content administrator to specify one set of records to boost and one set of records to bury. Boost and bury are applied to the result list before any additional sorting or relevance ranking modules. For more information about record boost and bury, refer to the *MDEX Engine Basic Development Guide*.

Configuration model for the Results List cartridge

The Results List configuration model allows you to configure the number and sorting of records returned by a search or navigation query. Additionally, it includes `whyMatchEnabled` and `whyRankEnabled` properties that can be used for debugging the set of records returned for a query.

The configuration model for this cartridge is `ResultsListConfig`. It includes the following properties:

Property name	Description
<code>recordsPerPage</code>	An integer that controls the number of results to display per page. This value can be set using <code>Nrpp</code> URL parameter.
<code>recordDisplayName</code>	A String that specifies the field that stores the record's logical name.
<code>sortOption</code>	An enumerated list of sort options on the results list available to the site visitor. Each item in this list is a <code>SortOptionConfig</code> with the following properties: <ul style="list-style-type: none"> • <code>label</code> — A descriptive label that displays to the site visitor in the client application

Property name	Description
	<ul style="list-style-type: none"> value — A sort order specified in the format <key> <direction>, where key is the name of the property or dimension on which to sort, and the direction is 0 for ascending and 1 for descending. An empty string represents the default sort order specified by the content administrator in Experience Manager. <p>You can set this value via the <code>Ns</code> URL parameter.</p>
sortRequestParameter	A String that specifies the selected Sort.
includePrecomputedSorts	A Boolean that specifies whether to return precomputed sorts. Defaults to <code>false</code> . If you do not set this to <code>true</code> , any calls to the <code>getPrecomputedSorts()</code> method return an empty list.
relRankStrategy	(Optional) The Relevance Ranking Strategy. If you specify a Relevance Ranking Strategy without setting <code>relRankTerms</code> , <code>relRankKey</code> , or <code>relRankMatchMode</code> , your Relevance Ranking strategy will apply to the results from the current search filter. This setting is ignored if an end user explicitly selects a sort.
relRankKey	(Optional) The Relevance Ranking key to use with the selected Relevance Ranking strategy. This can be a search interface, dimension, or property set in the MDEX Engine. You must set a <code>relRankStrategy</code> and <code>relRankTerms</code> if you specify a value for this property.
relRankTerms	(Optional) Relevance Ranking terms, delimited by a + sign. These can be different from the terms in the search filter. You must set a <code>relRankStrategy</code> and <code>relRankKey</code> if you specify a value for this property.
relRankMatchMode	(Optional) The match mode that determines the subset of results to apply Relevance Ranking to. You must set a <code>relRankStrategy</code> if you specify a value for this property.
boostStrata	An ordered list of <code>CollectionFilters</code> that enable items to be boosted to the top of the results list. This setting is ignored if an end user explicitly selects a sort.
buryStrata	An ordered list of <code>CollectionFilters</code> that enable items to be buried at the bottom of the results list. This setting is ignored if an end user explicitly selects a sort.
subRecordsPerAggregateRecord	<p>The number of sub-records to return for any aggregate records in the results list. This property should have one of the following values:</p> <ul style="list-style-type: none"> ZERO — Sub-records are not returned. ONE — A single representative record is returned. ALL — All sub-records are returned. <p>The default value is <code>ONE</code>. For best performance, Oracle recommends that you use <code>ZERO</code> or <code>ONE</code>.</p>
offset	An integer record offset for the result list. This property defaults to <code>0</code> and is used for paging. This value can be set using <code>No</code> URL parameter.
fieldNames	A list of record fields to pass through from each record to the <code>Record</code> output model of the <code>ResultsListHandler</code> .
subRecordFieldNames	For aggregate records, a list of sub-record fields to pass through from each sub-record to the <code>Record</code> output model of the <code>ResultsListHandler</code> .
whyMatchEnabled	If query debugging features are enabled, this property enables debugging information about why each record matched the search and navigation state. For additional information, see "About query debugging results in the reference application."

Property name	Description
whyRankEnabled	If query debugging features are enabled, this property enables debugging information about why each record was ranked in the given order. For additional information, see "About query debugging results in the reference application."



Note: You only need to set the `relRankKey`, `relRankTerms` and `relRankMatchMode` properties if you wish to apply relevance ranking to values other than those specified in the search filter, or to the results of an EQL expression.

MDEX Engine configuration for the Results List cartridge

Your MDEX Engine configuration for your application allows you to configure which properties and dimensions should display in the results list view, optimize certain properties to use for sorting records, and specify a default sort order.

Dynamic configuration

In the Property and Dimension editors in Developer Studio, you can specify which properties and dimensions are returned for the record with the record list. This configuration can be overridden in the cartridge handler configuration. For more information about configuring the display of properties and dimensions for the record list, refer to the *Developer Studio Help*.

Static configuration

Although you can sort on any property or dimension at query time, it is also possible to optimize a property or dimension for sorting in Developer Studio. This controls the generation of a precomputed sort, which you can retrieve on the `ResultsListConfig` object by using the `getPrecomputedSorts()` method. For more information about precomputed sorts, refer to the *MDEX Engine Basic Development Guide*.

Dgidx flags

You can specify the default sort order for records as a flag in Dgidx. For more information about Dgidx flags and sorting, refer to the *MDEX Engine Basic Development Guide*.

The Deployment Template configuration for the Discover Electronics reference application does not specify a default sort key.

Cartridge handler configuration for the Results List cartridge

The Results List cartridge handler extends the `NavigationCartridgeHandler`. The application-wide default configuration in the Assembler context file specifies default sort options, relevance ranking strategy, and record and sub-record properties to pass through to the cartridge handler response model. It also allows you to enable or disable debugging features if query debugging features are enabled.

The cartridge handler uses a `contentItemInitializer` to merge the layered configuration. The included `requestParamMarshallers` bean enables URL request-based configuration for the cartridge.

Template configuration for the Results List cartridge

The Results List template allows a content administrator to configure the main results of a search or navigation query based on the site visitor's filter state. Configuration options include sort order, boost/bury, and number of records to display per page.

The Results List cartridge template allows a content administrator to configure the following properties on the configuration model:

- `recordsPerPage`
- `sortOption`
- `relRank`

- boostStrata
- buryStrata

URL request parameters for the Results List cartridge

End user configuration is passed to the configuration model as URL parameters. This allows application end users to specify how records should be displayed and sorted in order to customize their navigation experience.

For most of the properties on the configuration model, the cartridge renderer in the reference implementation respects the values set at the cartridge handler or template level. The `offset` value is used to control paging display.

Property	URL Parameter	Description
<code>recordsPerPage</code>	<code>Nrpp</code>	The cartridge renderer uses this property to enable an application end user to set their own limit on records to display per page.
<code>sortOption</code>	<code>Ns</code>	This parameter enables you to override sort options on a per-query basis.
<code>offset</code>	<code>No</code>	This parameter enables you to control record display when paging.
<code>relRankKey</code>	<code>Nrk</code>	(Optional) The Relevance Ranking key. You must set a <code>relRankStrategy</code> on the cartridge to use this parameter. You must also specify <code>relRankTerms</code> .
<code>relRankTerms</code>	<code>Nrt</code>	(Optional) Relevance Ranking terms, delimited by a + sign. You must set a <code>relRankStrategy</code> on the cartridge to use this parameter. You must also specify a <code>relRankKey</code> .
<code>relRankMatchMode</code>	<code>Nrm</code>	(Optional) The match mode that determines the subset of results to apply Relevance Ranking to. You must set a <code>relRankStrategy</code> , <code>relRankKey</code> , and <code>relRankTerms</code> if you specify a value for this property.
<code>whyMatchEnabled</code>	<code>whymatch</code>	If query debugging is enabled for the reference application, this property enables you to include record matching information on a per-query basis, rather than at the cartridge handler level.
<code>whyRankEnabled</code>	<code>whyrank</code>	If query debugging is enabled for the reference application, this property enables you to include record ranking information on a per-query basis, rather than at the cartridge handler level.



Note: The `Nrk`, `Nrt`, and `Nrm` parameters take precedence over any relevance ranking declaration in the `Ntk`, `NTT`, and `NTX` parameters.

Enabling snippetting in record results

The Assembler can return snippets (an excerpt from a record property that contains the user's search terms and the surrounding context) for display in results lists.

Snippetting is configured as part of a search interface. You can enable snippetting on one or more properties in a search interface, typically properties that contain multiple lines of text.

To enable snippetting:

1. Enable snippetting on one or more properties in the relevant search interface.
For more information about configuring snippetting, refer to the *MDEX Engine Basic Development Guide*.
2. In the Results List cartridge handler configuration, specify the relevant snippet property in the list of `fieldNames`.

For example, if you enabled the property `product.short_desc` for snippetting, you would specify the property `product.short_desc.Snippet`, as in the following example:

```

<bean id="CartridgeHandler_ResultsList" class="com.endeca.infront.cartridge.ResultsListHandler"
    parent="NavigationCartridgeHandler" scope="prototype">
<property name="fieldNames">
    <list>
        <value>product.id</value>
        <value>product.code</value>
        <value>product.name</value>
        <value>product.brand.name</value>
        <value>product.short_desc</value>
        <value>product.short_desc.Snippet</value>
        <value>product.price</value>
        <value>product.min_price</value>
        <value>product.max_price</value>
        <value>product.img_url_thumbnail</value>
    </list>
</property>
<!-- additional elements omitted from this example -->
</bean>

```

The snippet is returned as a string property on the response model for each record for display by the renderer.

Record details cartridges

The following section provides an overview of the configuration model for record detail features in the reference implementation.

Record details page

The Record Details page displays detailed information about a specific record.

The response model for this cartridge is `RecordDetails`, which contains a single `Record`.

The rendering logic for a record details page is expected to be highly customized for each site, in order to display the relevant record information and provide additional functionality such as bookmarking or initiating a purchase transaction.

Configuration model for the Record Details cartridge

The Record Details configuration model allows you to configure which properties on the record should be passed through to the output model of the cartridge handler, so that the renderer can display them.

The configuration model for this cartridge is `RecordDetailsConfig`. It includes the following properties:

Property name	Description
<code>fieldNames</code>	A list of record fields to pass through from the record to the <code>Record</code> output model of the <code>RecordDetailsHandler</code> .
<code>subRecordFieldNames</code>	For aggregate records, a list of sub-record fields to pass through from each sub-record to the <code>Record</code> output model of the <code>RecordDetailsHandler</code> .

MDEX Engine configuration for the Record Details page

No special configuration is required for the display of record details, but you can specify what information you want to display on the record page.

Dynamic configuration

You can specify which properties and dimensions are returned with the record for a record details page in Developer Studio. For more information about configuring the display of properties and dimensions for record details, refer to the *Developer Studio Help*.

Cartridge handler configuration for the Record Details cartridge

The Record Details cartridge handler extends the `NavigationCartridgeHandler`, but otherwise does not require any additional configuration.

Template configuration for the Record Details cartridge

The Record Details cartridge in the Discover Electronics application does not require any configuration in Experience Manager. The cartridge can be placed on a Record Details page to display detailed information about a record.

Content and spotlighting cartridges

The following sections provide an overview of the configuration models for features that enable content spotlighting in the reference implementation.

Record Spotlight

The Record Spotlight cartridge can promote either specific featured records or a set of dynamic records based on a navigation state.

The response model for this cartridge is `RecordSpotlight`, which includes a list of `Record` objects and an optional action to show all records (in the case of a dynamic record spotlight).

Configuration model for the Record Spotlight cartridge

The Record Spotlight configuration model allows you to configure the selected records and "See All" link within a record spotlight, as well as the record fields to pass through to the cartridge response model.

The configuration model for this cartridge is `RecordSpotlightConfig`. It includes the following properties:

Property name	Description
<code>maxNumRecords</code>	A string representing the maximum number of records that this spotlight can contain. If the content administrator designates specific records in the Experience Manager, the number of records cannot exceed the value of <code>maxNumRecords</code> . If the content administrator specifies a query, the Assembler returns no more than this number of records.
<code>recordSelection</code>	A <code>RecordSpotlightSelection</code> object that represents the records selected for spotlighting. This includes the specified filter state, sort options, and result limit.
<code>showSeeAllLink</code>	A Boolean that determines whether to display the "See All" link. The link requires a value for <code>seeAllLinkText</code> in order to display.
<code>seeAllLinkText</code>	A string representing the display text for a link that represents the navigation state of a dynamic record spotlight. If this string is not configured, no link is generated for the client application.
<code>fieldNames</code>	A list of record fields to pass through from the record to the <code>Record</code> output model of the <code>RecordSpotlightHandler</code> .
<code>subRecordFieldNames</code>	For aggregate records, a list of sub-record fields to pass through from each sub-record to the <code>Record</code> output model of the <code>RecordSpotlightHandler</code> .

MDEX Engine configuration for a spotlight

You can configure which properties and dimensions can be displayed in a spotlight.

Dynamic configuration

Although the content administrator can designate the records for a spotlight either by specifying a search and navigation query or by specifying individual record IDs, the Assembler query that fetches the spotlighted records is always a navigation query (using records in the specific record case). Therefore, the configuration that determines which properties and dimensions are returned with the record for spotlighting is "show with record list." This configuration can be overridden in the cartridge handler configuration. For more information about configuring the display of properties and dimensions for the record list, refer to the *Developer Studio Help*.

Related Links

[MDEX Engine configuration for the Results List cartridge](#) on page 318

Your MDEX Engine configuration for your application allows you to configure which properties and dimensions should display in the results list view, optimize certain properties to use for sorting records, and specify a default sort order.

Cartridge handler configuration for the Record Spotlight cartridge

The Record Spotlight cartridge handler extends the `NavigationCartridgeHandler`. The application-wide default configuration in the Assembler context file specifies record properties to pass through to the cartridge handler response model.

Template configuration for a record spotlight

A Record Spotlight cartridge enables a content administrator to specify a set of contextually relevant records to spotlight on a particular page.

The Record Spotlight cartridge template allows a content administrator to configure the following properties on the configuration model:

- `maxNumRecords`
- `recordSelection`
- `showSeeAllLink`
- `seeAllLinkText`

These properties are configured using the Spotlight Selection editor.

In addition, the cartridge template includes the following pass-through property:

Property name	Description
<code>title</code>	A title that the content administrator can specify to display for this cartridge in the front-end application.

Media Banner

The Media Banner cartridge displays video or images to the site user and can be configured to link to a static page, a single record, or a specified navigation state.

The response model for this cartridge is `MediaBanner`, which includes a `MediaObject` and an `ActionLabel` that contains a destination link.

Configuration model for the Media Banner cartridge

The configuration model for the Media Banner cartridge includes a media object and an associated link.

The configuration model for this cartridge is `MediaBannerConfig`. It includes the following properties:

Property name	Description
media	The <code>MediaObject</code> representing the image or video asset to display in the application.
link	The <code>LinkBuilder</code> object used to construct a link to a navigation state or a static page within the application.

MDEX Engine configuration for a media banner

No special configuration is required for the media banner, but your MDEX Engine configuration will affect the display of records in the link selector when setting a navigation state or choosing a specified record.

Dynamic configuration

You can specify how records are sorted and which properties and dimensions are returned with a record in Developer Studio. For more information about configuring record sorting and display, refer to the *Developer Studio Help*.

Cartridge handler configuration for the Media Banner cartridge

The Media Banner cartridge handler extends the `NavigationCartridgeHandler`, but otherwise does not require any additional configuration.

Template configuration for the Media Banner cartridge

The Media Banner enables the content administrator to use the media selector and link editor to create a media banner that links to a specified page, selected record, or dynamic navigation state.

The Media Banner cartridge template allows a content administrator to configure the following properties on the configuration model:

- media
- link

In addition, the cartridge template includes the following pass-through property:

Property name	Description
imageAlt	(Optional) The alt-text to display when the end user hovers over the media asset in the application.

For detailed information on the properties within the `media` and `link` properties, consult the Javadoc for the `MediaObject` and `LinkBuilder` classes.

Dynamic triggering cartridges

The following sections contain information about features related to triggering content items based on the user's context.

The dynamic slot feature is typically used to trigger a cartridge independently from the page that contains it, although the Discover Electronics application uses the same mechanism to trigger entire pages by programmatically creating a content slot configuration and passing it to the Assembler.

About dynamic slots

A dynamic slot is a generic mechanism that enables content administrators to manage the content for specific sections of an Experience Manager-driven page independently from the overall page.

There two main scenarios for using dynamic slots:

- **To share content across different pages.** In this case, the triggers on the content items that populate the slot are more general or orthogonal to the trigger criteria for the page. For example, a header cartridge may be shared across an entire site if it is referenced from every page and has an "Applies at all locations" trigger. A promotion may be configured with a user segment trigger and display when a site visitor who belongs to the specified user segment browses to any of the pages that references the collection that contains the promotion.
- **To create variants of a page.** In this case, the triggers on the content items that populate the slot are more specific than the trigger criteria for the page. (Typically, they would "inherit" the parent content item's triggers and add additional criteria for the variable content.)

Following are some specific use cases for dynamic slots:

- A brand manager needs to control the banner images that display throughout the site. This is a different person from the merchandiser who typically manages pages in Experience Manager.
- A brand manager needs to be able to specify the images that display at a particular navigation state (for example, Digital Cameras > Samsung) even if there is no specific landing page for that navigation state.
- A merchandiser wishes to display promotions in the menu area based on more specific trigger criteria than those that apply to the page as a whole. For example, one could create a page to use as a base for all "Digital Cameras" pages, and populate the menu sections with more specific content based on the brand, price range, or other dimensions. This model enables content reuse for most of the content within a page with page-specific overrides for subsections as needed. It removes the need to create many individual pages for each specific combination of triggers.
- A merchandiser wishes to display promotions in the menu area based on trigger criteria that are simply different from those on the page as a whole. For example, there might be a "Back to School" special for a particular time frame that applies to all pages within a category or even the entire site. This model enables content reuse for individual sections across a variety of pages. The reusable sections are managed in a central location so that updates immediately take effect across all the pages that include the reused content, rather than having to edit each one manually.

Dynamic slot prerequisites

The dynamic slot feature enables content administrators to populate a section of a content item with content from a different collection in Experience Manager. As a prerequisite, your application must include a collection with the appropriate content type for populating an administrator's dynamic slot cartridge.



Note: If a content administrator attempts to populate a dynamic slot in a given collection with a content item from the same collection and creates a circular reference, the Assembler detects the conflict during preprocessing and returns the content item with an `@error` property.

Request Event Attributes

The `RequestEvent` and `NavigationEventWrapper` classes support getting and setting common search and navigation information on a request event. This Appendix provides a reference table of out-of-the-box attributes that you can retrieve or set on a `RequestEvent` object.

Related Links

[Navigation Cartridge Configuration Reference](#) on page 279

This appendix provides an overview of the configuration models for the included navigation cartridges. You should review this information if you use these cartridges in your Assembler application to communicate with an MDEX Engine.

[Base request event attributes](#) on page 325

The following describes the base schema for an Assembler request event.

[Navigation cartridge request event attributes](#) on page 325

The following describes the schema for an Assembler navigation cartridge request event. These fields are in addition to those described for the base request.

Base request event attributes

The following describes the base schema for an Assembler request event.

The `RequestEvent` class includes getter and setter methods for each of these attributes.

Attribute	Type	Description
<code>endeca:sessionId</code>	<code>String</code>	The unique identifier for a browser session. To retrieve this information, you must register an implementation of <code>SessionIdProvider</code> in the request event adapter constructor.
<code>endeca:assemblyStartTimestamp</code>	<code>long</code>	The time (in milliseconds from POSIX Epoch) that the <code>assemble()</code> method started
<code>endeca:assemblyFinishTimestamp</code>	<code>long</code>	The time (in milliseconds from POSIX Epoch) that the <code>assemble()</code> method finished

Navigation cartridge request event attributes

The following describes the schema for an Assembler navigation cartridge request event. These fields are in addition to those described for the base request.

The `NavigationEventWrapper` class includes getter and setter methods for each of these attributes.

Attribute	Type	Description
<code>endeca:autocorrectTo</code>	<code>String</code>	The suggested auto-correct term, if it triggers for the request.
<code>endeca:contentPath</code>	<code>String</code>	The content path of the page corresponding to the request.
<code>endeca:didYouMeanTo</code>	<code>List<String></code>	The suggested "Did You Mean" term, if it triggers for the request.
<code>endeca:dimensions</code>	<code>List<String></code>	The dimension names selected for navigation.
<code>endeca:dimensionValues</code>	<code>List<String></code>	The dimension value names selected for navigation.
<code>endeca:eneTime</code>	<code>Long</code>	The time, in milliseconds, that it takes the MDEX Engine to run the query.
<code>endeca:numRecords</code>	<code>Long</code>	The number of records returned for the request.

Attribute	Type	Description
endeCA:numRefinements	Integer	The number of selected refinements.
endeCA:recordNames	List <String>	The names of the records returned by the request. To populate this attribute, the recordDisplayFieldName property on the ResultsListConfig object must be set to the name of the field that contains record names.
endeCA:recordSpec	String	The record specifier for a selected record.
endeCA:requestType	Request-Type	The type of request. Possible values are: <ul style="list-style-type: none"> • T - Root navigation • N - Navigation only • S - Search only • SN - Search, then navigation • R - Record detail • UNKNOWN - Unknown
endeCA:searchKey	String	The search key for the current navigation state.
endeCA:searchMode	String	The search mode for the request.
endeCA:searchTerms	String	The search terms for the request.
endeCA:siteRootPath	String	The site root path of the page corresponding to the request.
endeCA:sortKey	List <String>	The sort keys for the request. Each key is a String with the format <code>fieldName <Descending / Ascending></code> .
endeCA:spotlights	List <String>	The list of spotlights triggered for the request.

Index

301 redirects 122

A

A (Aggregated Record) parameter 286

Actions

- about 111
- Action fields 112, 113
- using 112, 113
- using with packaged services 113

adding help to a cartridge 168

Adobe Flex

- verifying version 206

aggERecFormatter 146

aggregate record detail pages 146

aggrERecFormatter 132

Apache Maven 207

application URLs

- about 111
- Actions 111
- configuration 114

Assembler

- about 13
- Assembler Service 73
- configuration 65
- content include query 72
- content slot query 73
- dependencies 64
- deploying 64
- dynamic content query 73
- error handling 26
- event listeners 25, 176
- eventing framework 25
- input 21
- introduced 13
- invoking in Java 71
- libraries 64
- output 21
- overview 21
- processing content by URI 72
- processing model 21
- rendering the response 84
- requirements 64
- Spring configuration 65

Assembler API

- ActionPathProvider interface 111

Assembler libraries 64

Assembler service

- about 68
- configuring 68

Assembler Service 64

- querying 73

Assembler services 36

Assembler servlet

- error handling 26
- response format 74

authoring environment 63

Auto-suggest cartridges 299

Auto-Suggest Dimension Search Results cartridge

- cartridge handler configuration 299
- configuration model 299
- template configuration 300

Auto-Suggest panel

- template configuration 299

automatic phrasing

- example scenarios 306, 307
- implementing 305

B

baseUrlState 153

basic query builder 129

boost and bury 316

Breadcrumbs cartridge 315

- cartridge handler configuration 316

C

canonical link 118

canonicalization 109, 135, 139

cartridge

- handler 173
- handler interface 28, 172
- Hello World example 159
- help 168
- instance configuration 296
- introduced 159
- renderer 163
- request configuration 297
- samples 178
- template 164
- testing 179

cartridge extension points 167

cartridge handlers 27, 68

- about 21

- configuring 66

cartridge template 211

cartridge templates

- adding dynamic slots 241

cartridges 27

- about 15

- cartridge handlers 16

- cartridge templates 15, 16

- instance configuration 27

- navigation cartridges 17

- Results List cartridge 18

cartridges (*continued*)

response model 27

checkbox editor 234

choice editor 228

combo box editor 228

configuring

cartridge instance 165

configuring URLs

content collections

example 39

content folders

creating 42

moving 43

overview 38

content item

property type 52

type 47

content item list 53

content items 27

about 15, 23

configuring 16

configuring in Workbench 16

container 27, 52

ContentInclude 23

ContentSlotConfig 23

dynamic slots 23, 24

instance configuration 16

leaf 27

nesting 24

content properties

editor mappings 221

primitive types 224

content type 35, 47

content XML 16

core cartridges

Auto-Suggest Dimension Search Results cartridge 299

Auto-Suggest panel 299

Breadcrumbs 315

Dimension Search Results 300

Media Banner 322

Record Details 320

Record Spotlight 321

Refinement Menu 310

Results List 316

Search Adjustments 302

Search Box 297

custom editors

introduced 171

D

data application 253, 254, 255, 256

interaction with Experience Manager 253

development environment 206

Dimension List editor 261

Dimension Search Results cartridge 300

cartridge handler configuration 301

configuration model 300

MDEX Engine configuration 301

Dimension Search Results cartridge (*continued*)

template configuration 301

URL parameters 302

dimension value list editor 264

dimensions

preparing 121

dimLocationFormatters 132

directory-style URLs 150, 151

Discover Electronics

handling of renderers 83

sample templates 45

duplicate content 109

Dy (Dimension Search) parameter 288

dynamic content 79

dynamic slot editor 243

dynamic slots 68, 241, 323

about 323

adding 241

Assembler configuration 243

configuration 243

editor 243

use cases 323

E

editor configuration

data service 238, 240

editor configuration file 210

editor modules

building 208

editors

building 208

configuring 51, 52

dependencies 206

developing 206, 207, 208

dimension list editor 261

dimension selector 260

dimension value boost-bury 262

dimension value list editor 264

dynamic slots 241

editor-specific configuration 52

editors.xml 52

guided navigation editor 259

image preview 265

label configuration 224

Link Builder 244

media editor 246, 247, 248, 249, 252

record list editor 266, 273

record stratification 258, 269

registering 210

rich text editor 270

RSLs 206

sort editor 271

workflow 207

Endeca Assembler

default cartridge processing 27

Endeca Configuration Repository 16

Endeca parameters

base-36 encoding 152

Endeca parameters (*continued*)
 encoding 108, 150
 moving 150, 151
 moving out of the query string 108
Endeca Sitemap Generator
 URL configuration file 155
 urlconfig.xml 155
Endeca URL Optimization API
 using 154
erecFormatter 132, 142
event listeners
 creating 176
 registering 178
examples
 cartridges 16, 17, 18
Experience Manager
 interaction with media MDEX Engine 253
Experience Manager API
 installing 207

G

group labels 237
 guided navigation editor 259
Guided Search Service
 dynamic content 81
 handling the response 81
Guided Search services 75, 113
 Dimension Search Service 76
 Guided Search Service 77, 79
 Record Details Service 77

H

handler implementation cases 175
Hello World
 renderer 163
HTTP servlet request 66

I

initializing a cartridge 172
 internationalization 55, 213
 introduction
 URL optimization 107

J

JSP
 example 81, 84

K

keyword redirects
 about 307
 Assembler response 309
 Assembler service 308
 cartridge handler configuration 307

keyword redirects (*continued*)
 content XML 308
keywords
 adding 132
 canonicalizing 108
 integrating into URLs 108

L

language support 55, 213
Link Builder
 adding 244
 configuration 245
locales
 custom editors 214
 editors 213
 templates 55
Log Server
 client side click events 101
logging
 events 101
 performance 101
 request event adapters 98, 99, 100
 request events 97

M

MDEX Engine
 configuration for Experience Manager editors 238, 240
 data service configuration 238, 240
 media MDEX Engine 254, 256
 using to index media 253
MDEX request builder 66
MDEX resource 66
Media Banner cartridge 322
 cartridge handler configuration 323
 configuration model 322
 MDEX Engine configuration 323
 template configuration 323
media browser
 enabling 252
Media Browser 246, 248
media editor 248
Media editor 246, 247, 250
 media browser 252
 uploading media 249
media MDEX Engine 252, 253
 interaction with Experience Manager 253
media paths 250
microbrowser 238
misc-path
 optimizing 132
multichannel applications
 about 87

N

N (Navigation) parameter 281

- navigation cartridges
 - cartridge handlers 295
 - configuration 294
 - default configuration 295
 - MDEX Engine configuration 296
- Navigation Container
 - about 314
 - cartridge handler configuration 315
 - configuration model 314
 - URL parameters 315
- navigation pages 135
- navigation state builder 66
- NavigationCartridgeHandler
 - and navigation state 68
 - and subclasses 68
- navStateCanonicalizer 132
- navStateFormatter 132, 135
- Nf (Range Filter) parameter 283
- Nfg (Geocode Filter) parameter 284
- No (Record Offset) parameter 290
- non-Endeca parameters
 - passing 150, 153
- Nr (Record Filter) parameter 284
- Nrk (Relevance Ranking Search Key) parameter 292
- Nrm (Relevance Ranking Match Mode) parameter 292
- Nrmc (Refinement Menu Config) parameter 289
- Nrpp (Records Per Page) parameter 289
- Nrs (Endeca Query Language Filter) parameter 284
- Nrt (Relevance Ranking Search Terms) parameter 291
- Ns (Sort Key and Sort Order) parameter 290
- Ntk (Record Search Key) parameter 282
- Ntl (Language ID) parameter 287
- Ntp (Auto-Phrasing) parameter 286
- Ntt (Record Search Terms) parameter 281
- Ntx (Record Search Mode) parameter 282
- Nty (Auto-Correct / DYM) parameter 288

O

- optimized URLs
 - overview 127

P

- packaged services
 - Actions 113
- page types
 - content structure 35
 - overview 34
 - with services 36
- pages
 - creating 37
- paramEncoder 152
- parameters
 - encoding 150, 152
 - Endeca 150, 151, 152
 - non-Endeca 150, 153
 - session-scope 150, 153

- path-param-separator
 - optimizing 149
- path-params
 - optimizing 150
- pathParamKeys 151
- pathSeparatorToken 149
- ports 63
- prerequisites
 - Adobe Flex 205
 - Apache Maven 205
- preview application
 - about 89
 - about previewing mobile devices 90
 - adding a device 90
 - auditing 90
 - changing 95
 - changing the link service 95
 - enabling 93
 - instrumenting 92
 - modifying a device 91
 - removing a device 91
 - testing 96
- preview link service 95
- production environment 63
- properties
 - and configuration in Experience Manager 50
 - Boolean 234
 - content item 52
 - content item list 53
 - defining 49
 - editor mappings 221
 - item 236
 - list 235
 - numeric 231
 - overview 50
 - preparing 121
 - sort options 268
 - string 225
 - template section 52, 53
- property editors
 - cartridge selector 54
 - checkbox 234
 - choice 228
 - default value 226
 - grouping 237
 - introduced 50
 - numeric stepper 231
 - radio group 230
 - section 54
 - slider 232
 - string 226

Q

- query debugging
 - about 102
 - enabling 102, 103
 - results 103
 - URL parameters 103

query parameters 75
 query string 75
 optimizing 150

R

R (Record) parameter 285
 radio button editor 230
 record detail pages 142
 Record Details cartridge 188, 320
 cartridge handler configuration 321
 configuration model 320
 MDEX Engine configuration 321
 template configuration 321
 Record Spotlight cartridge 321
 cartridge handler configuration 322
 configuration model 321
 MDEX Engine configuration 322
 template configuration 322
 RecordDetailsHandler
 and record state 68
 reference application
 request event adapters 100
 reference data application
 configuring 255
 overview 254
 requirements 254
 Refinement Menu cartridge 310
 cartridge handler configuration 313
 configuration model 311
 MDEX Engine configuration 312
 template configuration 313
 URL parameters 313
 relevance ranking 316
 renderers 27, 83
 rendering a cartridge 163
 request event adapters
 registering 99
 request events
 attributes 325
 navigation attributes 325
 response format 74
 Results List cartridge 192, 316
 cartridge handler configuration 318
 configuration model 316
 MDEX Engine configuration 318
 template configuration 318
 URL parameters 319
 Rich Text editor 270
 Rsel (Featured Records Selector) parameter 286
 RSS cartridge 182
 rule groups, See content folders

S

sample editor
 building 212
 registering 212
 RichTextEditor 211

sample editor (*continued*)
 sample cartridge 211
 sample editor module 212
 Search Adjustments cartridge 302
 cartridge handler configuration 304
 configuration model 302
 MDEX Engine configuration 303
 template configuration 304
 URL parameters 305
 Search Box cartridge 297
 MDEX Engine configuration 298
 template configuration 298
 Select Records dialog 240
 SeoNavStateEncoder 152
 serialization
 JSON 24
 XML 24
 services, See Assembler services
 session objects 153
 session-scope parameters
 removing 150, 153
 show with record 121
 show with record list 121
 sitemap
 planning 33
 Sitemap Generator, See Endeca Sitemap Generator
 snippetting 319
 sorting 316
 structural properties 52

T

templates 27
 and Discover Electronics 45
 creating 45
 defining editors 50
 See also property editors
 defining properties 49
 defining sections 52, 53
 description, specifying 48
 id 47
 mobile channel templates 87
 Name property 49
 naming conventions 47
 properties 50
 See also properties
 saving 47
 schema 46
 structure 46
 troubleshooting 56
 troubleshooting default values 57
 type 47
 updating 58
 validation 46, 56
 thumbnail images
 specifying 48
 using 49

U

URL

- anatomy 127
- components 127

URL canonicalization, See canonicalization

URL configuration

- aggregate record detail pages 146
- canonicalization 109, 135, 139
- keywords 132
- misc-path 132, 135, 142, 146
- navigation pages 135
- path-param-separator 149
- path-params 150
- query string 150
- record detail pages 142

URL configuration file

- creating 129
- using 154
- using with Endeca Sitemap Generator 155

URL formatter 129

URL formatter class

- basic URLs 116
- optimized URLs 116

URL formatting 129

See also URL configuration

URL optimization

- application recommendations
- basic application requirements
- classes 107
- configuring URLs
- integrating with Endeca Sitemap Generator
- introduction
- package 107
- preparing your application
- URL configuration

URL Optimization

- CSS 122
- external resources 122
- images 122
- Javascript files 122

URL parameters 114

- A (Aggregated Record) 286
- Dy (Dimension Search) 288
- N (Navigation) 281
- Nf (Range Filter) 283
- Nfg (Geocode Filter) 284

URL parameters (*continued*)

- No (Record Offset) 290
- Nr (Record Filter) 284
- Nrk (Relevance Ranking Search Key) 292
- Nrm (Relevance Ranking Match Mode) 292
- Nrmc (Refinement Menu Config) 289
- Nrpp (Records Per Page) 289
- Nrs (Endeca Query Language Filter) 284
- Nrt (Relevance Ranking Search Terms) 291
- Ns (Sort Key and Sort Order) 290
- Ntk (Record Search Key) 282
- Ntl (Language ID) 287
- Ntp (Auto-Phrasing) 286
- Ntt (Record Search Terms) 281
- Ntx (Record Search Mode) 282
- Nty (Auto-Correct / DYM) 288
- R (Record) 285
- Rsel (Featured Records Selector) 286
- whymatch (Record Match Info) 293
- whyrank (Record Rank Info) 293

URL query parameters

- cartridge specific parameters 287
- core parameters 280
- default parameters 280

URL transitioning 122

urlconfig.xml 129

- See also URL configuration file
- Endeca Sitemap Generator 155
- using 154
- using for URL optimization 155
- using with Endeca Sitemap Generator 155
- See also URL configuration file

URLs

- directory-style 150, 151

W

- whymatch (Record Match Info) parameter 293
- whyrank (Record Rank Info) parameter 293
- word separator
 - configuring 108

Z

zones, See content folders