Oracle Endeca Commerce

Assembler Application Developer's Guide

Version 3.1.1 • December 2012

Contents

Copyright and disclaimer

Preface

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

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 with results from the MDEX Engine.

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

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

About this guide

This guide provides an overview of the Endeca Assembler and Experience Manager application development architecture. It uses examples from the Discover Electronics reference application to describe the configuration and customization tasks developers can perform to implement features in their own applications.

Who should use this guide

This guide is intended for developers responsible for building applications using the Endeca Assembler and supporting users of Experience Manager. You should familiarize yourself with the concepts in the *Oracle Endeca Commerce Concepts Guide* and the *Oracle Endeca Commerce Getting Started 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*.

Chapter 1

Supporting an Assembler Application

This section covers the supporting constructs that you must create in order to enable an 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 collections 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

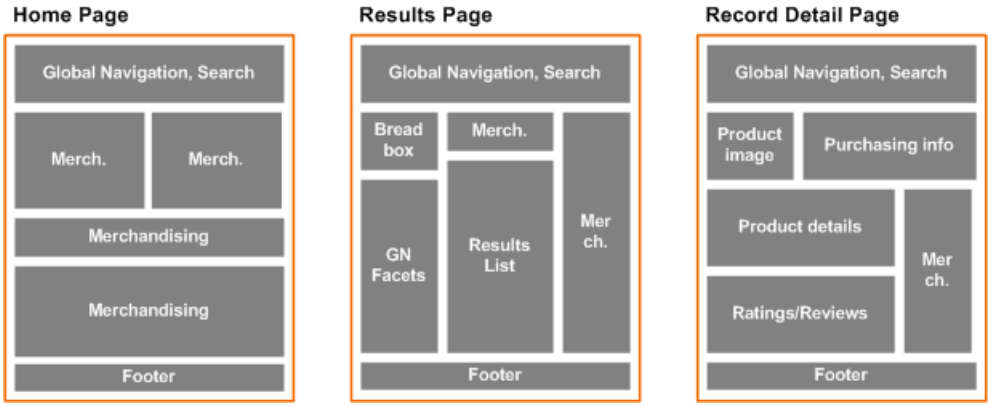
In the example above, the promotional Christmas and Mother's Day pages no longer exist as explicit pages. Instead, the content associated with those promotions is stored as available "browse" page configurations that each trigger during a specified date range.

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

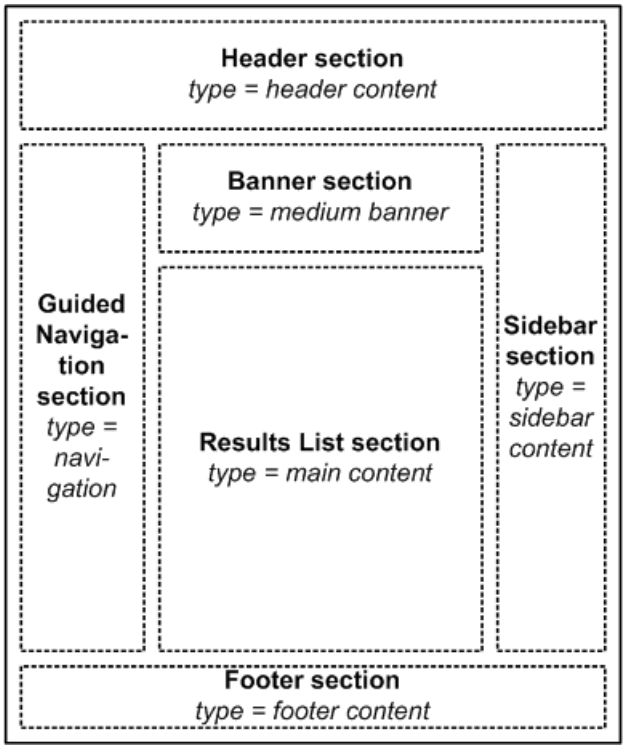
pages may be accessed at a URL that includes the path `/detail/`. These URL mappings are typically achieved by setting up individual services for each page type.

The Discover Electronics reference application includes examples of results pages and a record details page.

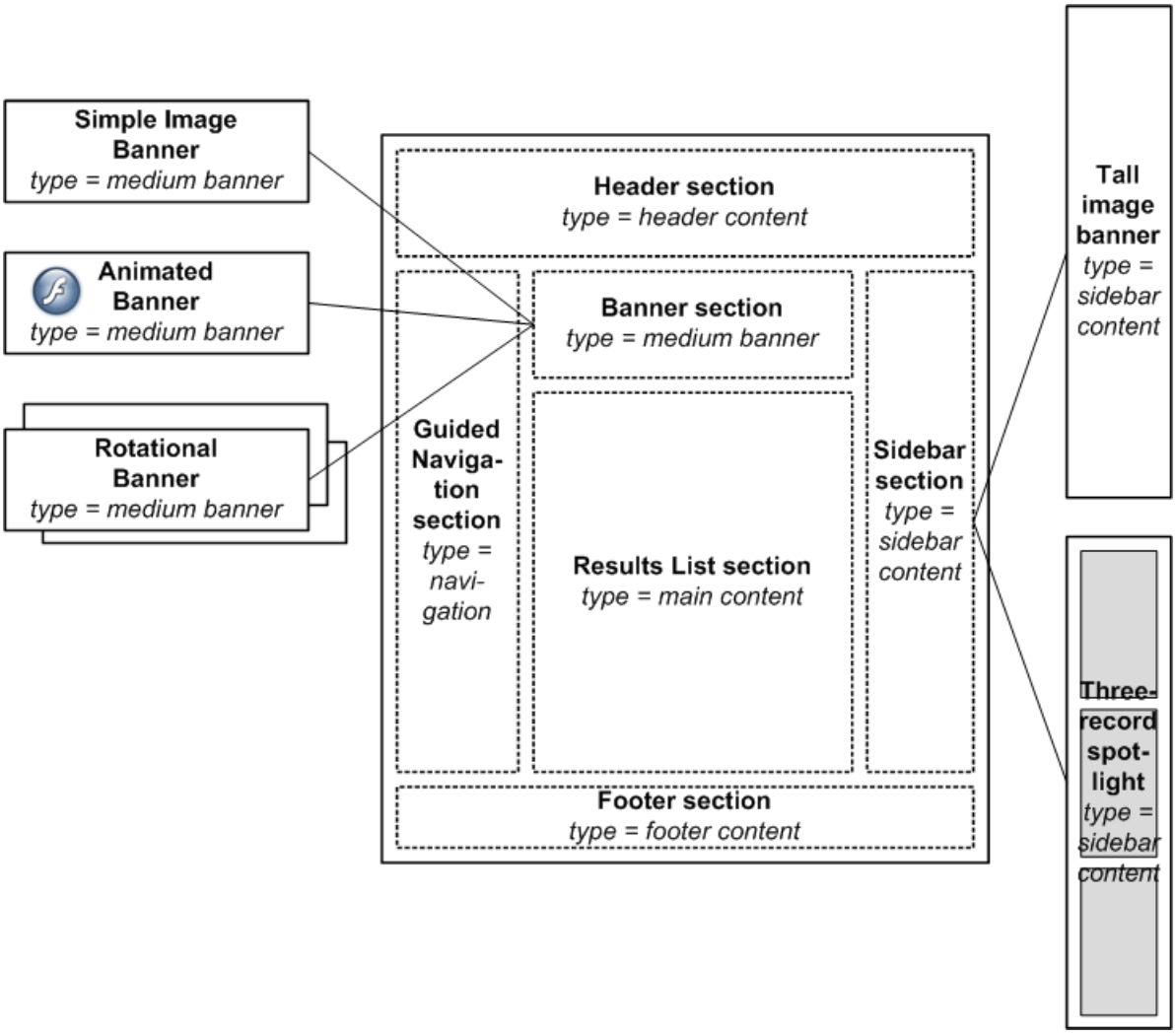
About page structure and content types

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 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."

Related Links

Working with Application URLs on page 61

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.

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 a selected content collection.

About content collections

Before a content administrator can configure dynamic content items within an application, you must create content collections to contain those items. Content items within the same collection 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 collections define the top-level organizational structure of an application, in which the content administrator can browse for content. Within the MDEX Engine, a collection represents the set of content items that are evaluated against each other to determine which ones are returned for a particular query. If a query satisfies the trigger criteria for multiple content items within a collection, items with higher priority take precedence over those with lower priority. A single application request may trigger content items from multiple collections.

Content collections have the following properties:

- Content type — Specifies the type of content items that can be created in this collection, as defined by the `type` attribute of the content template.
- Evaluation limit — The maximum number of content items in this collection that can be returned for a single query.

Oracle recommends that you create at least one content collection 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.

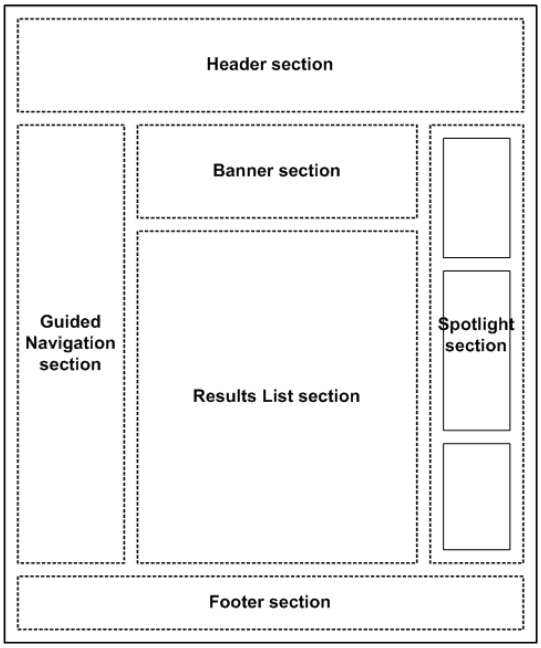
For example, the Discover Electronics reference application includes the following content collections:

- **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.

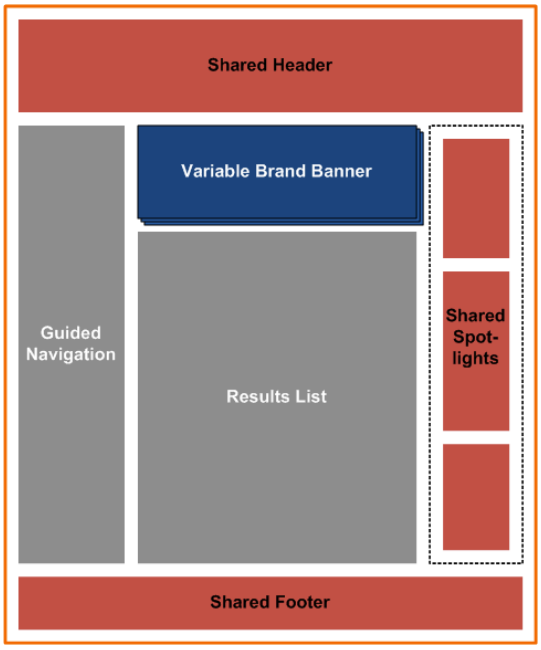
Content collections example

Content collections determine which content items are evaluated and which items are returned for a particular query.

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

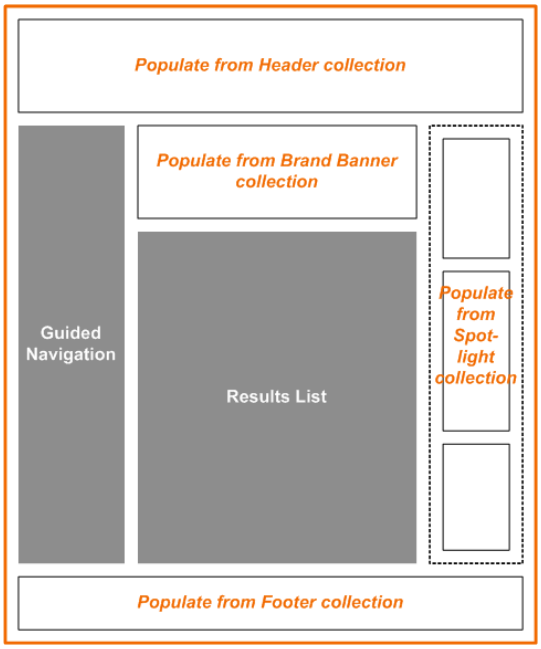


Based on this template, the content administrator wants to configure a page for a specific trigger (for example, Category > Cameras > Digital Cameras) using contextual, shared, and variable content as in this picture:



- The header and footer are populated from a content collection 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.

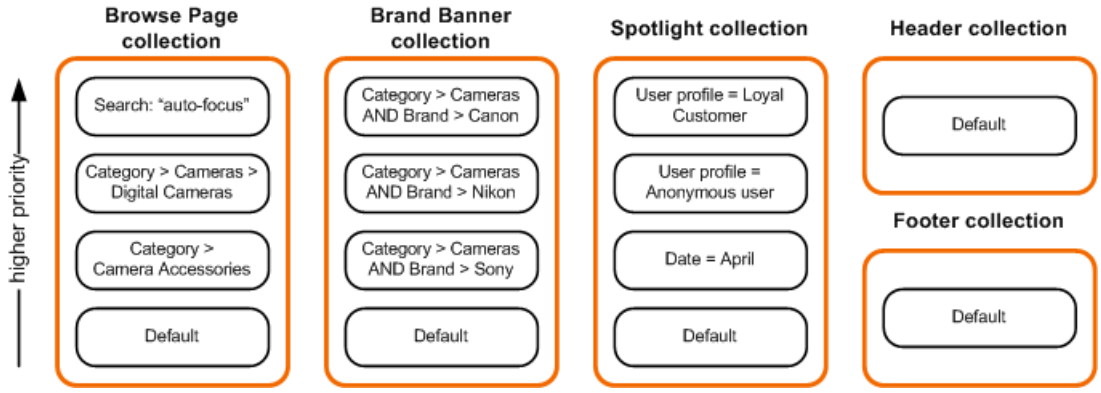
The configuration for the page (as specified in Experience Manager) looks like this:



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 collections.

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 collections:

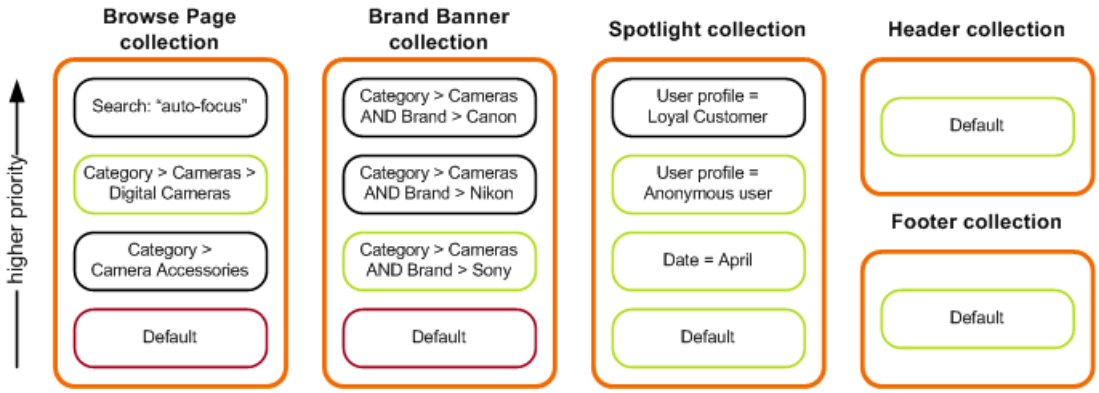


The collections are configured as follows:

- The Browse Page collection contains all the content items representing search and navigation pages in the site. Because only one page can display to a user for any given query, the evaluation limit is `1`.
- The Brand Banner collection contains cartridges of type `MediumBanner` that are appropriate to display in the Banner slot. This collection also has an evaluation limit of `1` because the page is designed to display only one banner at a time.
- The Spotlight collection 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 this collection is 3.
- The Header and Footer collections each contain cartridges of type `FullWidthContent`. The evaluation limit for these collections is also `1`.

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

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



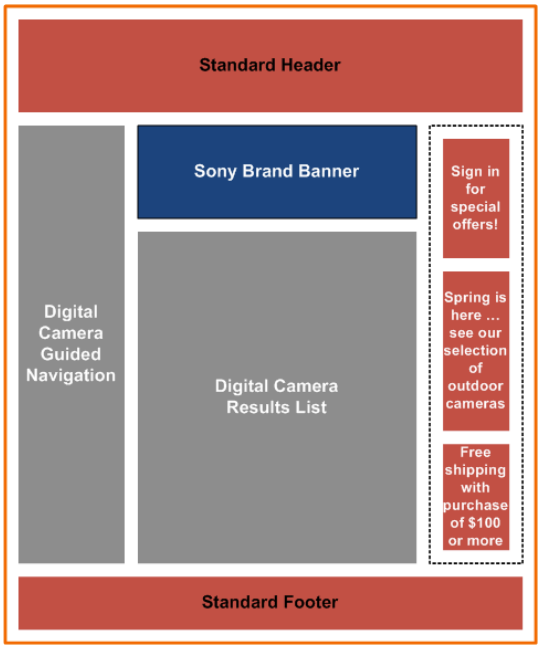
- The Digital Cameras page is returned from the Browse Page collection, which includes the content administrator's configuration for Guided Navigation and for Results List. Note that the Default page (with a trigger of "Applies at all locations") is also eligible to fire, but the Browse Page collection is limited to one content item and 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 that matches the user's navigation state in the Brand Banner collection. 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 that match the user's navigation state in the Spotlight collection. 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 collections 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 collections (including the Browse Page collection). Priority between items is specified within each collection. 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 a content collection has an evaluation limit of 1.

Oracle recommends that you create separate collections 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 collections, or else the same banner (the one with the highest priority for the current navigation state) would be returned for both sections of the page.

Oracle also recommends that you do not mix reusable and variable content within the same collection. For example, if a slot (such as the Spotlight slot) can be populated with either reusable or variable content, create two different collections, Reusable Spotlight collection and Variable Spotlight collection. The content administrator can configure a particular page to populate the Spotlight slot from either collection 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 collection for each type of content.

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



Related Links

About dynamic slots on page 108

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.

Creating a content collection

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

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 Collection**.
   The **Add Content Collection** panel appears.
4. Enter the **Name** of the collection you wish to add.
5. Enter the content type that you wish to use for the collection in the **Content Type Allowed** field.
   You may also use the drop-down arrow to display a list of available content types. The drop-down list is populated based on the available `type` values for each cartridge template you have uploaded to the Endeca Configuration Repository.
   This selection restricts the content collection to items of the specified type.
6. Set the **Evaluation Limit** for the content collection in the combo box.
7. Click **Add**.
   The new content collection is added to the Content Tree in Experience Manager.

8. Optionally, drag the new content collection to a folder within Experience Manager for organizational purposes.

   You can create a new folder using the drop-down menu for the **Content** heading.

About moving content collections

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

If you move a content collection 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 2

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 parameters for queries to an MDEX Engine or another external resource).

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 could contain meta keywords used for search engine optimization, or a cartridge could include embedded code that does not render in the page but enables functionality such as Web analytics beaconing.

The Discover Electronics reference application provides some sample page templates for some standard page types as well as templates to 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 and can be customized 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 sections or content collections in the application. 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 the 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.

Related Links

About the type and ID for a template on page 26

Each template is required to have a `type` and a unique `id`.

Configuring Front-End Application Features on page 75
This section describes the cartridge configuration model for front-end application features, and provides a description of the core cartridges in Oracle Endeca Tools and Frameworks.

Template Property and Editor Reference on page 143
This section describes how to define basic content properties and associated editing interfaces in Experience Manager templates.

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, ID, 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" id="DimensionSearchResults">
  <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.

Template naming conventions

Templates are saved as XML files that are then uploaded to Experience Manager. It is possible to have multiple templates in a single file, however, for ease of maintenance Oracle recommends the following practices.

- Each template should be in a separate file.
- Name each template file using the following format: `<TemplateType>-<TemplateID>.xml`. For example, `ResultsPage-ThreeColumnNavigationPage.xml` or `HorizontalBanner-ImageMap.xml`

Note: Template file names cannot have spaces in them.

Endeca also 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 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://endeca.com/schema/content-template/2008"
  type="PageTemplate" id="ThreeColumnNavigationPage">
```

Before you upload your templates to Experience Manager, Endeca recommends that you validate them against the template schema. A copy of the template schema (`content-template.xsd`) is located for your reference with your Endeca Workbench installation in `%ENDECA_TOOLS_CONF%\conf\schema` (on Windows) or `$ENDECA_TOOLS_CONF/conf/schema` (on UNIX).

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

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://endeca.com/schema/content-template/2008"
  xmlns:xavia="http://endeca.com/schema/xavia/2010"
```

```
  xmlns:editors="editors"
  type="PageTemplate" id="ThreeColumnNavigationPage">
```

About the type and ID for a template

Each template is required to have a `type` and a unique `id`.

The template *type* determines where a template can be applied. There are two general categories of templates. For top-level templates, such as page templates or templates that describe content items that can be triggered independently from a content collection, the type must match the content type of the collection in Experience Manager. Container templates specify the content type for each of their sections, which determines which cartridges can be inserted into that section. For example, if you have a template that includes a "HorizontalBanner" section, only cartridges of type "HorizontalBanner" are available to insert into that section in Experience Manager.

The template *id* is a string that is used to identify the template. It must be unique within your application; templates with non-unique IDs are not available in Experience Manager. The ID 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."

The template `type` and `id` are specified as required attributes on the `<ContentTemplate>` element. For example:

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

Note: The `type` and `id` attributes are 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
`<Description>`	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").

Element	Description
<ThumbnailUrl>	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. **Note:** If your thumbnails are hosted on the same server as Endeca Workbench, you can omit `http://<host>:<port>` from the URL.

Example

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

```
ColumnNavigationPage.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 adminstrator 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://endeca.com/schema/content-template/2008"
xmlns:editors="editors"
                 type="ResultsPage"
                 id="ThreeColumnNavigationPage">
  <!-- 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 143
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 143
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 29
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" id="ThreeColumnNavigationPage">
  <!-- 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" id="ThreeColumnNavigationPage">
  <!-- 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>
  </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://endeca.com/schema/content-template/2008"
                 xmlns:xavia="http://endeca.com/schema/xavia/2010"
                 xmlns:editors="editors"
                 type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- 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 administratorinserts 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.

Chapter 3

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. Start a command prompt (on Windows) or a shell (on UNIX).
2. Navigate to the `control` directory of your deployed application.

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

3. Run the `set_templates` script.
   For example:

```
C:\Endeca\apps\Discover\control>set_templates.bat
Removing existing cartridge templates for Discover
Setting new cartridge templates for Discover
Finished setting templates
```

Troubleshooting problems with uploading templates

Template errors are returned to the emgr_update command line call and detailed in the `webstudio.log` file.

The `webstudio.log` file is located in:

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

Uploading templates can fail for the following reasons:

Schema validation

Schema validation failure issues an error returned to the emgr_update command line call similar to the following:

```
C:\Endeca\apps\ContentAssemblerRefApp\config\cartridge_templates
ERROR: The template "NavigationPage.xml" is invalid (org.xml.sax.SAXParse¬
Exception: cvc-complex-type.4: Attribute 'id' must appear on element 'Con¬
tentTemplate'.)
ERROR: Failed to set app config.  Make sure you can connect to http://local¬
host:8006.
```

Each template that fails validation appears as a separate component. The error is also written to the `webstudio.log` file at the `WARN` level.

In the case of malformed XML, a similar error is output to both the command line and the `webstudio.log` file. For a file with multiple validation errors, only the first failure is logged.

Duplicate template ID

If you upload two template files with the same ID but different file names, then two separate templates are stored in Experience Manager but neither one displays to content administrators. An error message is returned to the emgr_update command line call similar to the following:

```
ERROR: 2 errors follow:
[ERROR 1] The template "HorizontalBanner-ImageMap.xml" has a non-unique ID
 ("ImageMap").
[ERROR 2] The template "VerticalBanner-ImageMap.xml" has a non-unique ID
("ImageMap").
ERROR: Failed to set app config. Make sure you can connect to http://local¬
host:8006.
```

The error is also written to the `webstudio.log` file at the SEVERE level.

To re-enable the templates, edit the `id` attribute of the `<ContentTemplate>` element so that each template ID is unique, remove the templates from Experience Manager, and re-upload the templates. In general, it is a best practice to remove templates from the Experience Manager and upload a complete set of templates whenever you need to update templates.

Invalid filename or directory path

If the template file or containing directories include a space, the emgr_update command line call issues an error similar to the following (in this case, for a file named `Copy of TestA.xml`):

```
ERROR:  09/22/09 15:54:36.795 UTC (1253634876795)       EMGR_MKPKG
{emgr_mkpkg}: Unknown file type "of" specified for filename "C:\Ende¬
ca\apps\ContentAssemblerRefApp\config\cartridge_templates\Copy". Valid file
 types are:  AENE_OP_CONFIG ANALYTICS_CONFIG CONTENT CRAWLER_DEFAULTS
CRAWLER_GLOBAL_CONFIG CRAWL_PROFILE CRAWL_PROFILES CRAWL_PROFILE_CON¬
FIGCRAWL_PROFILE_URL_LIST DERIVED_PROPS DIMENSIONS DIMENSION_GROUPS DIMEN¬
SION_REFS DIMSEARCH_CONFIG DIMSEARCH_INDEX DVAL_RANKS DVAL_REFS ENEIDX_OP_CON¬
FIG ENE_OP_CONFIG FORGEOUTDIMS FORGE_OP_CONFIG KEYWORD_REDIRECT_GROUP
KEY_PROPS LANGUAGES MERCHSTYLES MERCHZONES MERCH_RULES MERCH_RULE_GROUP
PHRASES PIPELINE PIPELINE_PARTIAL PRECEDENCE_RULES PRECOMPUTE PROFILES
PROP_REFS RECORD_FILTER RECORD_ID_PROP RECORD_SORT_CONFIG RECORD_SPEC REC¬
SEARCH_CONFIG RECSEARCH_INDEXES REFINEMENT_CONFIG RELRANK_STRATEGIES REN¬
DER_CONFIG ROLLUPS SEARCH_CHARS STEMMING STOP_WORDS THESAURUS VIEWS RESOURCE
```

To avoid this error, make sure that your file and directory names do not include spaces.

Empty directory

When uploading templates, if the specified directory does not contain any XML files, the emgr_update command line call displays the following message:

```
There are no templates in the specified directory.
```

If you receive this message, check to make sure that you specified the correct directory to the emgr_update call. The utility script provided with the Deployment Template module assumes that the templates are located in `APP_DIR/config/cartridge_templates`.

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 messsage 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 updating templates

When updating templates in Experience Manager, you should be aware of how conflicts are handled.

Experience Manager uses the most recently uploaded template. If you have an existing template in Experience Manager and upload a template with the same file name, the new template replaces the previously uploaded template.

However, if you upload two template files with the same ID but different file names, then two separate templates are stored in Experience Manager but neither one displays to content administrators. For this reason, you should avoid renaming template files after they have been uploaded to Experience Manager unless you make sure to remove the old template first. In general, it is a best practice to remove templates from Experience Manager and upload a complete set of templates whenever you need to update templates.

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.

About removing templates

If you remove a template that is being used for an existing page, the properties of the corresponding content item are no longer editable in Experience Manager.

When a content administrator attempts to edit an existing content item that uses a missing template, the following occurs:

- The properties defined in the content item itself can no longer be edited in the content details panel. All the existing values of the missing template's properties are preserved unless the content administrator removes the content item and selects a different template.
- The tree of child content items is still active. The content administrator can change or edit child cartridges via the Content Tree, as long as their templates are still available.

The content administrator has the following options:

- Leave the existing content as is. The Assembler continues to evaluate and process configurations as long as the appropriate cartridge handlers are present, regardless of whether the template exists in Experience Manager. Existing pages continue to display in the client application as long as the appropriate rendering code is still in place.
- Replace the missing template or cartridge with another template. This action deletes all configured properties of the template as well as any nested cartridges.
- The existing content can be re-enabled for editing by uploading the missing template.

Note: Changing the ID of a template is equivalent to removing the template with the old ID and creating a new template with the new ID. Avoid changing the ID of templates that are being used for existing pages.

Removing templates from Experience Manager

You can remove all the templates from Experience Manager using the emgr_update utility.

Note: Before removing templates from Experience Manager, be sure you have a backup of the current set of templates. Oracle recommends that you store page and cartridge templates in a version control system.

When removing or updating templates, make sure that all users are logged out of Experience Manager.

The `emgr_update --action remove_templates` command removes all templates from an application, not specific templates. Removing specific templates from Experience Manager consists of the following steps:

1. Retrieving the current set of templates from Experience Manager.
2. Deleting the templates that are no longer needed from your local copy.
3. Removing all templates from Experience Manager using the procedure below.
4. Uploading the remaining templates to Experience Manager.

To remove templates from Experience Manager:

1. Open a command prompt or UNIX shell.
2. Run emgr_update with the `--action` of `remove_templates` and the following parameters:

Parameters	Value
`--host`	The machine name and port for the staging Endeca Workbench environment, in the format *host*:*port*.
`--app_name`	The name of the application from which you want to remove the templates.

The following is a Windows example:

```
emgr_update.bat --action remove_templates --host localhost:8006
--app_name My_application
```

The following is a UNIX example:

```
emgr_update --action remove_templates --host localhost:8006
--app_name My_application
```

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. Start a command prompt (on Windows) or a shell (on UNIX).
2. Navigate to the `control` directory of your deployed application.

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

3. Run the `get_templates` script.

Chapter 4

Building Applications with the Endeca Assembler

This section describes how to build a client application that uses the Endeca Assembler to query the MDEX Engine or other services.

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 6.3.0
- Endeca Logging API 6.3.0
- 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.assem¬
bler.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>
</bean>
```

For details on the `AssemblerFactory` interface and `SpringAssemblerFactory` implementation, see the *Assembler API Reference (Javadoc)*.

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 `UrlFormatter` object to use for parsing the request URL into a `NavigationState` object and for generating Endeca URLs based on a `NavigationState`. **Note:** In the Discover Electronics application, this bean is configured in `endeca-url-config.xml`.
`mdexRequestBuilder`	The `MdexRequestBuilder` implementation to use for forming MDEX Engine requests. For more information, see "About 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, `BasicContentPath¬`

Navigation state builder property	Description
	`Provider`, 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 *MDEX Engine Advanced Developer's Guide*.
`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 *MDEX Engine Advanced Developer's Guide*.

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 `mdexRequest¬ Builder` 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 `NavigationCartridge¬ Handler` 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 need to retrieve content dynamically from content collections 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 contents from a content collection should extend from this handler.

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

Related Links

Setting up the Preview Application for Workbench on page 111

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 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`.

Invoking the Assembler in Java

You invoke the Assembler by passing in a content item object to be assembled.

The content item that is passed as input to the Assembler is known as the configuration model. The Assembler invokes the appropriate cartridge handler to process that content item, which may in turn invoke other cartridge handlers to process child content items. The end result of the processing cycle is another content item representing the response model.

```
// Get the AssemblerFactory from the Spring context and
// create an Assembler
WebApplicationContext webappCtx =
  WebApplicationContextUtils.getRequiredWebApplicationContext(application);
AssemblerFactory assemblerFactory =
  (AssemblerFactory)webappCtx.getBean("assemblerFactory");
Assembler assembler = assemblerFactory.createAssembler();
assembler.addAssemblerEventListener(new MyLogger());

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

You can instantiate any configuration model programmatically and pass it to the Assembler, but typically an assembly cycle begins with a `ContentInclude` or `ContentSlotConfig` item. Both of these

methods retrieve additional content configuration specified in Workbench, the former by URI, and the latter by triggering content from a collection populated either in Experience Manager or Rule Manager.

Note: If you have only 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 servlet.

Related Links

About retrieving Assembler results using the packaged services on page 50
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

The content include mechanism retrieves content configuration that a content administrator has specified in the sitemap section of Experience Manager.

In an authoring instance the content configuration is stored in the Endeca Configuration Repository. In a live instance, the Assembler retrieves the content configuration from the live content source. This is specified in the `CartridgeHandler_ContentInclude` bean in `assembler-context.xml`.

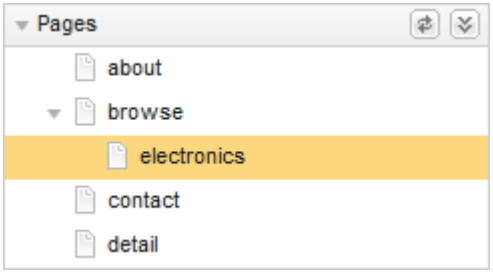
The `ContentInclude` item specifies the URI from which to retrieve the content.

- In Oracle Endeca Experience Manager implementations, the URI typically begins with `/pages` (a hard-coded path to distinguish the sitemap from the content collections 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 Assembler packaged services.

The following shows an example of a content include query that retrieves the 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());
```

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/Sony/Cameras`, the URI passed to the Assembler is `/pages/browse/electronics/Sony/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.

Invoking the Assembler with a ContentSlotConfig item

The content slot mechanism retrieves configuration from a content collection based on a set of trigger criteria.

The `ContentSlotConfig` specifies a content collection and the number of content items to return from that collection.

The following shows an example of a content slot query that drives the autosuggest feature of the Search Box cartridge:

```
// Construct a content slot config to query content
// in the SearchBoxAutoSuggestContent collection
ContentItem contentItem = new ContentSlotConfig(request.getParameter("col¬
lection"), 1);
```

The `ContentSlotHandler` uses these parameters to form a content trigger request to fetch the top configuration content item (or items) from the collection by priority. The Assembler then processes the content items from the collection and returns them as part of the response for rendering.

Content slots within other content items

You can construct a `ContentSlotConfig` as the root content item of an Assembler query, but other content items can also contain content slots that are configured to retrieve content dynamically from collections.

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 collection. Most of the pages within the Web Browse Pages collection contain a mixture of content items that are configured directly in the page, and content slots that retrieve content from other collections. 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 `Result¬sListHandler` to process it.

f. It invokes the `RecordSpotlightHandler` to process the spotlight records.

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 the content collections 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 from a content collection 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 content collection.

Related Links

Invoking the Assembler with a ContentInclude item on page 47

The content include mechanism retrieves content configuration that a content administrator has specified in the sitemap section of Experience Manager.

Content slots within other content items on page 48
You can construct a `ContentSlotConfig` as the root content item of an Assembler query, but other content items can also contain content slots that are configured to retrieve content dynamically from collections.

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 reponse 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.

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

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 in the Spring context file for the Assembler. The exception is the dynamic content that can be configured in content collections 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.

Related Links

Default cartridge configuration on page 76
You can specify default configuration settings for a cartridge as part of the cartridge handler configuration in the Spring context file for the Assembler.

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_Dimension¬SearchResults` bean in the Spring context file for the Assembler. For information about the configuration options for the Dimension Search Results cartridge, refer to the *Endeca 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/dimension¬search?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.Dimension¬
SearchGroup",
                "dimensionSearchValues": [ … ],
                "dimensionName": "camera.flash"
            },
            {
                "@class": "com.endeca.infront.cartridge.model.Dimension¬
SearchGroup",
                "dimensionSearchValues": [ … ],
                "dimensionName": "product.features"
            },
            {
                "@class": "com.endeca.infront.cartridge.model.Dimension¬
SearchGroup",
                "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_RecordDe¬tails` bean in the Spring context file for the Assembler. For information about the configuration options for the Record Details cartridge, refer to the *Endeca 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://lo¬calhost: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 from content collections.

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	CartridgeHan¬ dler_GuidedNaviga¬ tion	GuidedNavigation¬ Config
breadcrumbs	Breadcrumbs	CartridgeHan¬ dler_Breadcrumbs	BreadcrumbsConfig
resultsList	ResultsList	CartridgeHan¬ dler_ResultsList	ResultsListConfig
searchAdjustments	SearchAdjustments	CartridgeHan¬ dler_SearchAdjust¬ ments	SearchAdjust¬ mentsConfig
dimensionSearchRe¬ sults	DimensionSearchRe¬ sults	CartridgeHan¬ dler_Dimension¬ SearchResults	DimensionSearchRe¬ sultsConfig
zones	Depends on contents of referenced content collections	CartridgeHan¬ dler_Con¬ tentSlotList	ContentSlotConfig

The following is an example of the results of a guided search service query for the URI `http://lo¬ calhost: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=%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"
}
```

Note:

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

About dynamic content and the Guided Search Service

In order to retrieve results from content collections as part of the response from the Guided Search Service, you must configure the default content slot list in the Spring context file for the Assembler.

The content slot list is configured in the `CartridgeHandler_ContentSlotList` bean in `assembler-context.xml`.

For each content collection that you want to enable for the Guided Search Service, add a `ContentSlot¬Config` bean to the `contentSlotList` as in the example below:

```
<bean id="CartridgeHandler_ContentSlotList" class="com.endeca.infront.car¬
tridge.ContentSlotListHandler"
 scope="prototype">
    <property name="contentItemInitializer">
        <bean class="com.endeca.infront.cartridge.ConfigInitializer"
scope="request">
            <property name="defaults">
                <bean class="com.endeca.infront.cartridge.ContentSlotList¬
Config" scope="singleton">
                    <property name="contentSlotList">
                        <list>
                         <bean class="com.endeca.infront.cartridge.ContentSlot¬
Config" scope="singleton">
                                <property name="contentCollection" val¬
ue="/content/Right Column Spotlights"/>
                                <property name="ruleLimit" value="3"/>
                            </bean>
                        </list>
                    </property>
                </bean>
            </property>
```

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

For each `ContentSlotConfig`, specify the following properties:

Property name	Value
`contentCollection`	The path to the collection from which you want to return results. The example above specifies a collection named Record Spotlight Content in a folder named Web.
`ruleLimit`	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.

Recall the `zones` property returned by the guided search service query for the URI `http://in¬front.eng.endeca.com:8006/assembler-authoring/json/services/guided¬search?Ntt=pink+camera`:

```
{
    [Additional items removed from this response]
    "zones": {
        "@type": "ContentSlotList"
    },
    "endeca:contentPath": "/services/guidedsearch",
    "previewModuleUrl": "http://localhost:8006/preview"
}
```

The response model, determined by the configuration above, is the following:

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-au¬thoring/json/services/guidedsearch?Ntt=pink+camera`:

```
{
    "@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=%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.assembler.Assembler"%>
<%@page import="com.endeca.infront.assembler.AssemblerFactory"%>
<%-- 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.getRe¬
quiredWebApplicationContext(application);

    // Get the AssemblerFactory from the Spring context file
    AssemblerFactory assemblerFactory = (AssemblerFactory)webappCtx.get¬
Bean("assemblerFactory");
```

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 `id` 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 *Endeca 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.

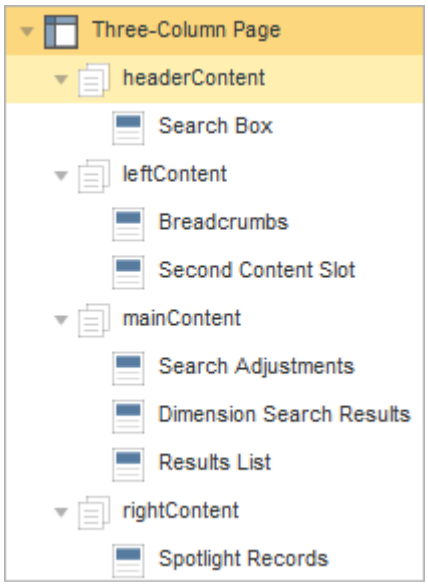
Recall that each of the `<div>` elements that make up the page uses a custom `<discover:include>` tag, defined in `WEB-INF\tags\discover\include.jsp`, to include the rendering code for the associated page component:

```
<!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>
        </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, refer to the *Cartridge Development Guide*.

About Assembler error handling

The Assembler API defines two kinds of exceptions: `AssemblerException` and `CartridgeHandlerException`.

The exceptions are distinguished as follows:

Exception	Description
`AssemblerException`	Indicates that an exception occurred while creating or processing an Assembler request. Exceptions of this type indicate that the entire assembly process was terminated.
`CartridgeHandlerException`	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 servlet

The Assembler servlet returns an HTTP status code of 200 (OK) regardless of whether any exceptions occurred during Assembler processing. The Assembler communicates error conditions as exceptions that are serialized as part of the response, as in the following example.

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

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

Chapter 5

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"
}
```

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
`removeAlways`	A list of URL parameters that should be removed from all Actions.
`removeOnUpdateFilterState`	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.
`removeOnClearFilterState`	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 `navigationState¬Builder` 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.Ba¬
sicActionPathProvider">
            <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/mo¬
bile/browse" />
```

```
                    <entry key="^/pages/detail$" value="/pages/browse" />
                </map>
            </constructor-arg>
            <!-- recordActionUriMap -->
            <constructor-arg index="3">
                <map>
                  <entry key="^/pages/mobile/.*$" value="/pages/mobile/de¬
tail" />
                    <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
`defaultEncoding`	Specifies the default query encoding, for example, `UTF-8`.
`prependQuestionMark`	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.Ba¬
sicUrlFormatter">
    <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
defaultEncoding	Specifies the default query encoding, for example, UTF-8.
pathSeparatorToken	The separator token used to separate the path section of the URL from the parameter section.
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: • The parameter key used for record detail links. The default value is R. • The parameter key used for aggregate record detail links. The default value is A. • The parameter key used for navigation state. The default value is N.
navStateFormatter	The NavStateFormatter that maps navigation state information to URL path keywords.
ERecFormatter	The ERecFormatterthat maps Endeca record attributes to URL path keywords.
aggrERecFormatter	The AggrERecFormatter 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.
useNavStateCanonicaliz¬er	Determines whether or not the canonicalizer specified in navState¬Canonicalizer is used. The default value is true. This value is ignored if the canonicalLinkBuilder enables canonical links.
urlParamEncoders	A list of UrlParamEncoder 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.urlformat¬
ter.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 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
`objectLocator`	Allows the retrieval of services without explicit injection. In this case, it is used to reference the

Property	Description
	framework for retrieving the `recordState` and `navigationState` for the current request.
`recordStateId`	The ID of the `recordState` being retrieved, not the actual `recordState`.
`navigationStateId`	The ID of the `navigationState` being retrieved, not the actual `navigationState`.
`includedParameters`	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.assem¬
bler.spring.SpringAssemblerFactory">
...
    <constructor-arg>
        <list>
            ...
          <bean class="com.endeca.infront.navigation.url.event.CanonicalL¬
inkBuilder">
              <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 CanaonicalLinkBuilder. 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?for¬
mat=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="<c:url value='${util:getUrlForAction(rootCompo¬
nent.canonicalLink)}'/>"/>
```

Related Links

About application URLs on page 61
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.

About Actions on page 61
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.

About working with URL parameters on page 64
The `navigationStateBuilder` handles both Endeca-specific and non-Endeca URL parameters.

About URL configuration in the reference application on page 64
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`.

Chapter 6

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 Endeca 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

The set of templates in a multichannel application should give content administrators the flexibility to manage channel-specific content in Experience Manager. At the same time, it should enable them to share configuration across channels whenever possible in order to provide the maximum consistency throughout a site.

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 "pages" that differ from those intended for mobile Web browsers.

- Use dynamic slots for configuration that should be shared across channels, similar to how dynamic slots 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 this in a separate content collection for navigation 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 collection via the content type mechanism in Experience Manager. 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 collection 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, all the Dimension Navigation cartridges in an application should be identical as the configuration of data-driven cartridges should be the same regardless of channel. Each of these identical cartridges should be mapped to the same cartridge handler in the Endeca Assembler. For more information about configuring cartridge handlers, refer to the *Endeca Assembler Application Developer's Guide*.

About device detection in the reference application

The reference application uses a `DeviceManager` class in the `com.endeca.mobile.services.detection` package to parse user agent strings and determine the appropriate content path and rendering channel for a request.

Each end user request is sent to the application controller, `WEB-INF\services\assemble.jsp`. The controller determines the device channel, dispatches the request to the Assembler, then sends the resulting response model to the appropriate renderer:

```
<%
    //The channel to prefix map. See the DeviceManager class for more info

    //Note: LinkedHashMap will preserve the order of entries
    Map<String,String> channelToPagePrefixMap = new Linked¬
HashMap<String,String>();
    channelToPagePrefixMap.put("Endeca.Infront.Channel.Mobile","/mobile");

    //This channel is the default one. This entry should be the last one
in the map
    //because all strings start with the empty string.
    channelToPagePrefixMap.put("Endeca.Infront.Channel.Desktop","");

    //The resources map. See the DeviceManager class for more info
    Map<String,String> resourcesMap = new LinkedHashMap<String,String>();
    resourcesMap.put("/desktop", "/WEB-INF/views/desktop/");
```

```
    resourcesMap.put("/mobile", "/WEB-INF/views/mobile/");

   //The list of the pages that will get redirected. See the DeviceManager
 class for more info
    List<String> validMobilePages = new ArrayList<String>();
    validMobilePages.add("/browse");
    validMobilePages.add("/detail");

   DeviceManager deviceManager = new DeviceManager(channelToPagePrefixMap,
 validMobilePages, resourcesMap);

    //AssemblerFactory, spring context, and user state creation removed
from this example

    String userAgent =  userState.getUserAgent();

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

   String contentUri = deviceManager.getContentPath(request.getPathInfo(),
 userAgent);

    if(!contentUri.equals(request.getPathInfo())){
        String query = request.getQueryString() == null ? "" : "?" + re¬
quest.getQueryString();
       response.sendRedirect(request.getContextPath() + contentUri + query);

    }else{
        // Assembler and ContentInclude creation removed from this example


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


        // If the response contains an error, send an error
        // error handling removed from this example

        else
        {
            // Determine the output format and write to response
            String format = request.getParameter("format");
            if("json".equals(format)) {
                response.setHeader("content-type", "application/json");
               new JsonSerializer(response.getWriter()).write(responseCon¬
tentItem);
            } else if ("xml".equals(format)) {
                response.setHeader("content-type", "application/xml");
                new XmlSerializer(response.getWriter()).write(responseCon¬
tentItem);
            } else {
                request.setAttribute("component", responseContentItem);
               request.setAttribute("rootComponent", responseContentItem);

                // This is used by the discover:include tag
              request.setAttribute("Endeca.InFront.ComponentResourcePath",
 deviceManager.getResourcePath(contentUri));
```

```
                %>
                <discover:include component="${component}"/>
                <%
            }
        }
    }
%>
```

For more information about content paths and application URLs, see "Working with Application URLs."

Related Links

Working with Application URLs on page 61

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.

Chapter 7

Configuring Front-End Application Features

This section describes the cartridge configuration model for front-end application features, and provides a description of the core cartridges in Oracle Endeca Tools and Frameworks.

About configuring application features

Features in an Assembler application are implemented as cartridges. In the Discover Electronics reference application, the behavior of cartridges depends on multiple sources of configuration. These are combined into a configuration model.

Cartridge-specific configuration falls into the following categories, in ascending order of priority:

- **MDEX Engine configuration**, which may be specified as part of the data integration process or by a configuration update
- **Default cartridge configuration**, which is typically specified in the Spring context file for the Assembler
- **Cartridge instance configuration**, which is typically specified by the content administrator in Workbench
- **Request-based configuration**, which is typically specified by the end-user in the client application

Request-based configuration overrides the cartridge instance configuration, which overrides the cartridge-level defaults, which override the application-wide defaults 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

Class name	Description
	cartridge configuration (such as introducing additional sources of configuration), refer to the *Experience Manager Cartridge Developer's Guide*.
`<CartridgeName>`	The response model produced by the cartridge handler. Cartridge response models may include objects that are reused among cartridges. For example, the `ResultsList` and `RecordSpotlight` both contain `Record` objects.

For details about the implementations of these classes for specific cartridges, refer to the *Endeca Assembler API Reference* (Javadoc).

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 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 Dgidix flags, refer to the *Oracle Endeca Commerce Administrator's Guide*.

Default cartridge configuration

You can specify default configuration settings for a cartridge as part of the cartridge handler configuration in the Spring context file for the Assembler.

Cartridge handlers are responsible for managing the various sources of configuration that apply to a specific cartridge. The 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 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 *Endeca 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.RecordSpotlight¬
Config" 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>
```

Note: Any property of the configuration model can be specified in the Spring context file. However, it is not required to specify a value for every property, only those for which you want to specify a default.

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 143
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 *Endeca 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 features

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 Developer's 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.
`minAutoSuggestInput¬ Length`	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 80

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 82

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 78
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 `img_thumbnail_url`. **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 `false`. It is enabled via URL request by setting the `Dy` URL parameter to `1`.
`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.
`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 `img_thumbnail_url`. **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`	`Dy`	Enables or disables the display of returned dimension refinements. Setting `Dy=1` sets the property to `true`.

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 `SearchAdjustemntsConfig`. It includes the following properties:

Property name	Description
`phraseSuggestio¬nEnabled`	Specifies whether to enable automatic phrasing. Defaults to `true`. Set via URL request by setting the `Ntp` URL parameter to `1`.
`spellSuggestionEn¬abled`	Specifies whether to enable automatic spelling correction. Defaults to `false`. Set via URL request by setting the `Nty` URL parameter to `1`.
`showWordInterp`	If query debugging features are enabled, this property enables debugging information about word or phrase subsitutions as a map that can be accessed via `SearchAdjustments.getInterpretedTerms()`. For additional information, see "About query debugging results in the reference application."

Related Links

About query debugging results in the reference application on page 134

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.

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_hthresh</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 87

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
`phraseSugges¬ tionEnabled`	`Ntp`	Specifies whether to enable automatic phrasing.

Property name	URL Parameter	Description
spellSuggestio¬nEnabled	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 Advanced 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 true, 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.mod¬
el.FilterState">
            <property name="rollupKey" value="product.code" />
            <property name="autoPhraseEnabled" value="true" />
            <!-- <property name="securityFilter" value="" /> -->
            <!-- <property name="languageId" value="en" /> -->
        </bean>
    </property>
    <!-- additional elements removed from this example -->
</bean>
```

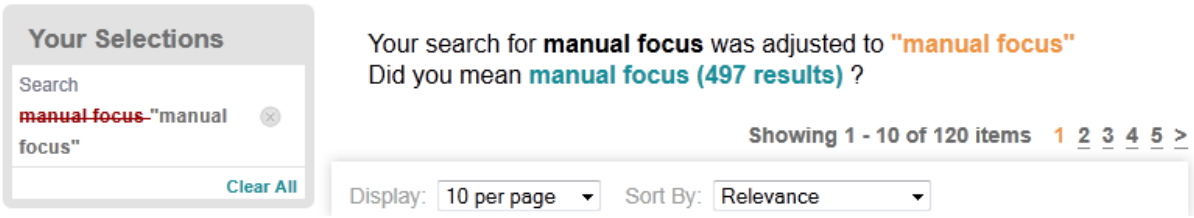
Interaction with the Did You Mean feature

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.



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:

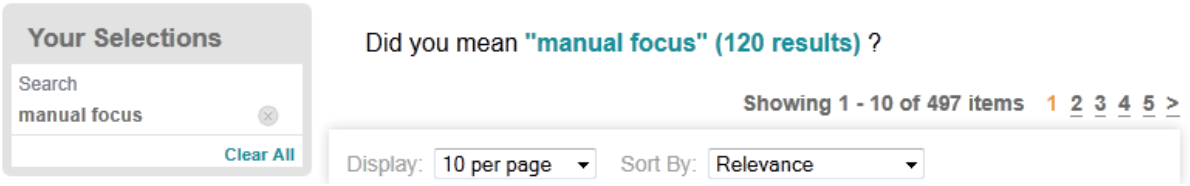
- Select the Did You Mean suggestion to search for the keywords separately, rather than as a phrase. This link sends the same query with the URL parameter `Ntp=0` to override the Filter State configuration, and also sets `Nty=0` since we do not need to suggest the phrased version of the query after the 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 `au¬toPhraseEnabled=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 (`Ntp`)	Did You Mean setting (`Nty`)	Result
Initial search	`true`	`Nty=1`	Phrase is automatically applied to the text search. A Did You Mean suggestion is offered for the unphrased version.
Select Did You Mean suggestion	`Ntp=0`	`Nty=0`	Phrase is not applied to the search. No suggestion is offered.
Make another follow-on selection	`true`	`Nty=0`	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 Endeca 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 `RedirectAwareCon¬tentIncludeHandler`.

The primary use case for setting these properties on content XML is for deployments running the Assembler service. Keyword redirects are programatically 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.SpringAssembler
Servlet</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("as
semblerFactory");
Assembler assembler = assemblerFactory.createAssembler();

//Retrieve the content for the given content uri
ContentItem contentItem = new RedirectAwareContentInclude("/pages" + conten
tUri);

// 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 `fullAssem¬ bleOnRedirect` 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:redi¬
rect")).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 *Endeca Assembler API Reference (Javadoc)*.

Related Links

About Actions on page 61

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.

Guided Navigation features

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 `whyPrecedenceRule¬Fired` 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
`dimensionId`	A string representing the id of the dimension being configured.
`boostRefine¬ments`	An ordered list of dimension value refinements to display at the top of the list.
`buryRefine¬ments`	An ordered list of dimension value refinements to display at the bottom of the list.

Property name	Description
sort	The base sort order of dimension values within this dimension. This property should have one of the following values: • default — Sort dimension values according to the application configuration for this dimension. • static — Sort dimension values in alphabetic or numeric order, depending on the dimension configuration. • dynRank — Sort dimension values so that the refinements with the highest number of records display first.
showMoreLink	A Boolean indicating whether to enable a link to show more refinements than are displayed by default.
moreLinkText	A string representing the text to use for the "show more refinements" link.
lessLinkText	A string representing the text to use for the "show fewer refinements" link.
numRefine¬ments	A string representing the number of refinements to display by default, or when a user clicks the "show fewer refinements" link.
maxNumRefine¬ments	A string representing the maximum number of refinements to display when a user clicks the "show more refinements" link.
refine¬mentsShown	A string that sets the amount of refinements to return, from the following values: • none — returns no refinements. • some — returns numRefinements refinements. • all — returns maxNumRefinements refinements.
showMore	(Deprecated) A Boolean indicating whether to display the maxNumRefinements number of menu items. When this value is false, the number of menu items generated is limited by numRefinements, and a "show more refinements" link is generated. This value should be set using showMoreIds URL parameter when the "show more refinements" link is selected.
useShowMoreI¬dsParam	(Deprecated) A Boolean that sets whether to use the showMoreIds URL parameter when determining how many refinements to display. If false, the showMore property on the RefinementMenuConfig object is used instead. If this property is set to true, refinements cannot be collapsed. Defaults to true.
whyPrece¬denceRule¬Fired	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."

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.

Related Links

About query debugging results in the reference application on page 134

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.

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
`dimensionName`	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_RE¬ FINEMENT_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
`DYNAMIC_RE¬ FINE¬ MENT_MENU_CON¬ FIG`	`Nrmc`	The `Nrmc` parameter takes multiple arguments allow you to configure dimension refinement behavior in the cartridge.
`showMore`	`ShowMoreIds`	(Deprecated) A Boolean indicating whether to display the `maxNumRefinements` number of menu items. Use the `re¬ finementsShown` 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 Developer's 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.

Related Links

About Actions on page 61

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.

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
`showMoreIds`	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.
`moreLinkText`	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.
`lessLinkText`	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.
`refinementsShownBy¬Default`	A Boolean indicating whether the refinement menus should be fully expanded. Defaults to `true`. When using a dataset that includes dimensions with a large number of refinements, you should set this to `false`.
`refinementsShown`	A string that sets the amount of refinements to return on each refinement menu, from the following values: • none — returns no refinements.

Property name	Description
	• some — returns numRefinements refinements.
useShowMoreI¬ dsParam	(Deprecated) A Boolean that sets whether to use the showMoreIds URL parameter when determining how many refinements to display. If false, the showMore property on the RefinementMenuConfig object is used instead. If this property is set to true, refinements cannot be collapsed. Defaults to true.
whyPrecedenceRule¬ Fired	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 requestParamMarshaller 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_REFINE¬ MENT_MENU_CONFIG	Nrmc	The Nrmc parameter takes multiple arguments allow you to configure dimension refinement behavior in the cartridge.
whyPrecedenceRule¬ Fired	whyPrece¬ denceRuleFired	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.WhyPrecedenceRule¬ Fired 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 features

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 *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
`recordsPerPage`	An integer that controls the number of results to display per page. This value can be set using `Nrpp` URL parameter.
`recordDisplay¬FieldName`	A String that specifies the field that stores the record's logical name.
`sortOption`	An enumerated list of sort options on the results list available to the site visitor. Each item in this list is a `SortOptionConfig` with the following properties: • `label` — A descriptive label that displays to the site visitor in the client application • `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. You can set this value via the `Ns` URL parameter.
`sortRequestPa¬rameter`	A String that specifies the selected Sort.
`includePrecom¬putedSorts`	A Boolean that specifies whether to return precomputed sorts. Defaults to `false`. If you do not set this to `true`, any calls to the `getPrecomputedSorts()` method return an empty list.
`relRankStrate¬gy`	(Optional) The Relevance Ranking Strategy. If you specify a Relevance Ranking Strategy without setting `relRankTerms`, `relRankKey`, or `relRankMatchMode`, 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 `relRankStrategy` and `relRankTerms` 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 `relRankStrategy` and `relRankKey` if you specify a value for this property.
`relRankMatch¬Mode`	(Optional) The match mode that determines the subset of results to apply Relevance Ranking to. You must set a `relRankStrategy` if you specify a value for this property.
`boostStrata`	An ordered list of `CollectionFilters` 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 `CollectionFilters` 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.

Property name	Description
subRecordsPer¬AggregateRe¬cord	The number of sub-records to return for any aggregate records in the results list. This property should have one of the following values: • ZERO — Sub-records are not returned. • ONE — A single representative record is returned. • ALL — All sub-records are returned. The default value is ONE. For best performance, Oracle recommends that you use ZERO or ONE.
offset	An integer record offset for the result list. This property defaults to 0 and is used for paging. This value can be set using No URL parameter.
fieldNames	A list of record fields to pass through from each record to the Record output model of the ResultsListHandler.
subRecordField¬Names	For aggregate records, a list of sub-record fields to pass through from each sub-record to the Record output model of the ResultsListHandler.
whyMatchEn¬abled	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."
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 getPrecomputed¬Sorts() 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 `requestParamMarshaller` 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
`recordsPerPage`	`Nrpp`	The cartridge renderer uses this property to enable an application end user to set their own limit on records to display per page.
`sortOption`	`Ns`	This parameter enables you to override sort options on a per-query basis.
`offset`	`No`	This parameter enables you to control record display when paging.

Property	URL Parameter	Description
relRankKey	Nrk	(Optional) The Relevance Ranking key. You must set a relRankStrategy on the cartridge to use this parameter. You must also specify relRankTerms.
relRankTerms	Nrt	(Optional) Relevance Ranking terms, delimited by a + sign. You must set a relRankStrategy on the cartridge to use this parameter. You must also specify a relRankKey.
relRankMatchMode	Nrm	(Optional) The match mode that determines the subset of results to apply Relevance Ranking to. You must set a relRankStrategy, relRankKey, and relRankTerms if you specify a value for this property.
whyMatchEnabled	whymatch	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.
whyRankEnabled	whyrank	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 snippeting 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.

Snippeting is configured as part of a search interface. You can enable snippeting on one or more properties in a search interface, typically properties that contain multiple lines of text.

To enable snippeting:

1. Enable snippeting on one or more properties in the relevant search interface.

   For more information about configuring snippeting, refer to the *Basic Developer's 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 snippeting, you would specify the property product.short_desc.Snippet, as in the following example:

```
  <bean id="CartridgeHandler_ResultsList" class="com.endeca.infront.car¬
tridge.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 features

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
`fieldNames`	A list of record fields to pass through from the record to the `Record` output model of the `RecordDetailsHandler`.
`subRecordField¬Names`	For aggregate records, a list of sub-record fields to pass through from each sub-record to the `Record` output model of the `RecordDetailsHandler`.

MDEX Engine configuration for the Record Details page

No special configuration is required 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 features

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
`maxNumRecords`	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 `maxNumRecords`. If the content administrator specifies a query, the Assembler returns no more than this number of records.
`recordSelection`	A `RecordSpotlightSelection` object that represents the records selected for spotlighting. This includes the specified filter state, sort options, and result limit.
`showSeeAllLink`	A Boolean that determines whether to display the "See All" link. The link requires a value for `seeAllLinkText` in order to display.
`seeAllLinkText`	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.
`fieldNames`	A list of record fields to pass through from the record to the `Record` output model of the `RecordSpotlightHandler`.

Property name	Description
`subRecordField¬ Names`	For aggregate records, a list of sub-record fields to pass through from each sub-record to the `Record` output model of the `RecordSpotlightHandler`.

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 101
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`

In addition, the cartridge template includes the following pass-through property:

Property name	Description
`title`	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 `Ac¬tionLabel` 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 `MediaObject` representing the image or video asset to display in the application.
`link`	The `LinkBuilder` 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.

Related Links

Link Builder property reference The Link Builder modifies three properties in the content item template: path, linkType, and queryString.

Dynamic triggering features

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.

Related Links

About content collections on page 16

Before a content administrator can configure dynamic content items within an application, you must create content collections to contain those items. Content items within the same collection 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.

Chapter 8

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\dis¬cover-electronics-authoring` (`$ENDECA_TOOLS_ROOT/reference/discover-electron¬ics-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:

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)

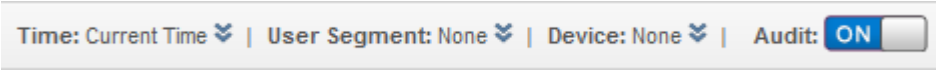
Adding a device for preview

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.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
previewEnabled	`true`
previewModuleUrl	`http://localhost:8006/preview`

   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.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.SLF4JAssemblerEvent¬
Logger" />
        </list>
    </constructor-arg>
</bean>
```

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

   Preview is only supported in an authoring environment. In the Discover Electronics reference implementation, `isAuthoring` takes the value of the `preview.enabled` property, since any content source with preview enabled must be an authoring content source:

```
<!--
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~

    ~ Content Sources
-->
<bean id="contentSource" class="com.endeca.infront.content.source.Work¬
benchContentSource"
      scope="singleton" init-method="init" destroy-method="destroy">
    <property name="isAuthoring" value="${preview.enabled}"/>
    <property name="appName" value="${workbench.app.name}" />
    <property name="defaultSiteRootPath" value="/pages" />
    <property name="host" value="${workbench.host}" />
    <property name="workbenchPort" value="${workbench.port}" />
    <property name="clientPort" value="${workbench.publishing.client¬
Port}"/>
</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>
        <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.assem¬bler.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.SpringPre¬
viewLinkServlet
    </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 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.

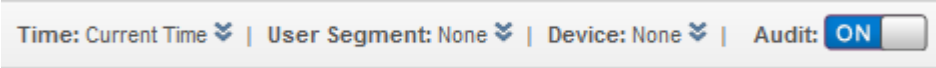
Testing your preview application

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.

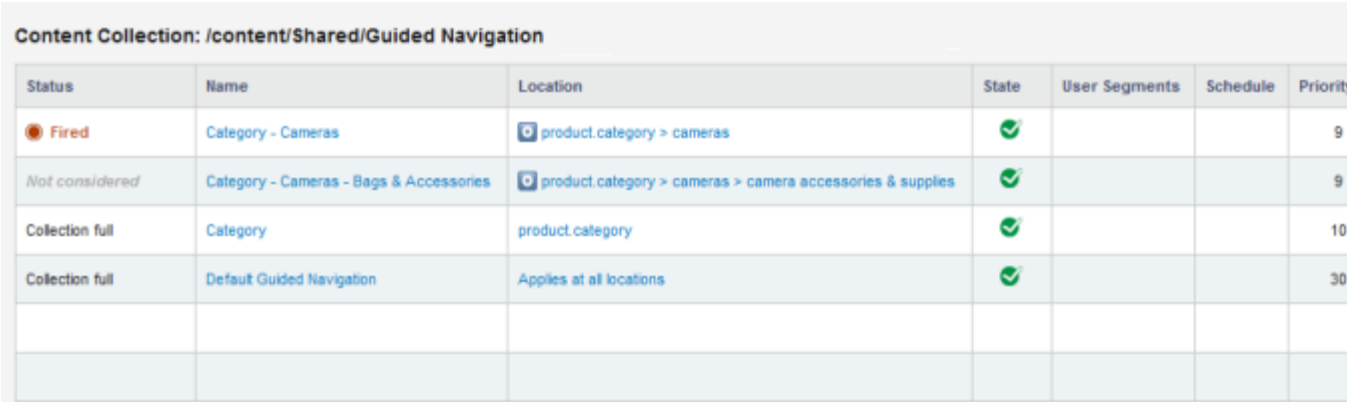
To test your custom preview application:

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:



   The Preview Toolbar expands to show configuration options.
   b) Select a device from the **Device** dropdown and click **Submit**.

   Specifying a preview device lets you see how the application renders on that device.
6. To test audit functionality:
   a) Mouseover the cartridge you wish to audit.
   The **Audit** button appears.
   b) Click the **Audit** button.
   The **Audit Panel** appears with a list of all content items for the specified content collection:



   c) Click any of the listed **Locations** to navigate to that location in the preview application.

d) Click the name of any of the listed content items and confirm that you return to that item in Experience Manager.

Chapter 9

Enabling Endeca Query Language in your Assembler application

Endeca Query Language allows you to apply additional filters to an end user's search and navigation state in your application.

About Endeca Query Language

Endeca Query Language (EQL) enables you to filter the records returned from the MDEX Engine based on your own criteria. By combining an EQL filter with an end user's search and navigation state, you can present a more relevant set of results.

Using EQL in your application enables you to develop for the following scenarios:

- *Your data set requires you to enable search against more than one search interface at a time* — For example, if you wish to enable both thesaurus expansion and wildcard search on an end user's search terms, you must configure two separate search interfaces (since wildcard search is not compatible thesaurus expansion). In this case, you can configure the wildcard search as an "or" in the EQL expression.
- *You are dealing with a data set that requires record joins* — This can be a product catalog where each "product" record contains different properties or properties with slightly differing values, depending on the location that entered the data. In this case, you may wish to configure a default EQL filter that performs a record join and executes a search against the combined record information.

For a detailed overview of the syntax and capabilities of EQL, see "Using the Endeca Query Language" in the *MDEX Engine Advanced Development Guide*.

Applying an EQL filter in a Java application

If you are developing a Java Assembler application, you can apply an EQL filter to an end user's filter state by creating the expression as an instance of the `EqlFilter` class, and calling the `setEqlFil¬ter()` method to apply it to your filter state.

Note: The approach outlined below is one suggested implementation of the EQL feature. If you are relying on the packaged services to interface with the Assembler, you must instead pass

EQL filters to your application through the `Nrs` URL parameter. You can also configure a single, default EQL filter that applies across your entire application.

To apply an EQL filter programatically:

1. Decide where to insert your EQL logic.

   Oracle recommends extending the `UrlNavigationStateBuilder` class.

2. Duplicate the current filter state in the application as an EQL expression:

```
//Retrieve the search key, terms, and match mode from the search filter
 in the parsed FilterState
if (filterState.getSearchFilters() != null && filterState.getSearchFil¬
ters().size() > 0){
   String searchTerms = filterState.getSearchFilters().get(0).getTerms();

    String searchKey = filterState.getSearchFilters().get(0).getKey();
    String matchMode = filterState.getSearchFilters().get(0).getMatch¬
Mode().toString();

    //Build the EQL expression using the first search filter
    String eqlExpression;
    if (matchMode == null){
        eqlExpression = "collection()/record[endeca:matches(.,\"" +
searchKey + "\",\"" + searchTerms + "\");
    } else {
        eqlExpression = "collection()/record[endeca:matches(.,\"" +
searchKey + "\",\"" + searchTerms
            + "\",\"" + matchMode + "\");
    };
```

3. Make any desired customizations to the EQL expression.

   For example, to execute a normal search but wildcard search terms when matching against the "product.id" dimension, you could use the following expression:

```
    if (matchMode == null){
        eqlExpression = "collection()/record[endeca:matches(.,\"" +
searchKey + "\",\"" + searchTerms
            + "\") or endeca:matches(.,\"product.id\",\"" + searchTerms
 + "*\")]";
    } else {
        eqlExpression = "collection()/record[endeca:matches(.,\"" +
searchKey + "\",\"" + searchTerms
            + "\",\"" + matchMode + "\") or endeca:matches(.,\"prod¬
uct.id\",\"" + searchTerms + "*\")]";
    };
```

4. Create your EQL filter as an instance of the `EqlFilter` class and set the EQL expression:

```
    //Create a EqlFilter and set it on the filterState
    EqlFilter eqlFilter = new EqlFilter();
    eqlFilter.setExpression(eqlExpression);
```

5. Return the completed filter state to your application and handle it as necessary.

About configuring an EQL filter using the Nrs URL parameter

You can pass an EQL expression to your application through the `Nrs` URL parameter.

Once you have created an EQL expression, you can use the `Nrs` URL parameter pass it into the `UrlNavigationStateBuilder`, for example:

```
Nrs=collection()/record[endeca:matches(.,"product.description",searchTerms)]
```

For additional information about the `UrlNavigationStateBuilder` and a full list of associated URL parameters, refer to the class documentation in the *Assembler API Reference (Javadoc)*.

About configuring a default EQL filter

You can apply an EQL filter to your application's default filter state.

You may wish to apply a specific, consistent EQL expression to all queries in your application.

You can configure a default EQL filter for your application by specifying it as a `eqlFilter` property on the default `FilterState`.

In the Discover Electronics reference application, this is configured in the Assembler context file:

```
<bean id="navigationStateBuilder" scope="request"
    class="com.endeca.infront.navigation.url.UrlNavigationStateBuilder">
    <property name="defaultFilterState">
        <bean scope="singleton" class="com.endeca.infront.navigation.mod¬
el.FilterState">
            <property name="eqlFilter">
                <bean scope="request" class="com.endeca.infront.naviga¬
tion.model.EqlFilter">
                    <property name="expression" value="collec¬
tion()/record[...]" />
                </bean>
            </property>
            <!-- additional elements removed from this example -->
        </bean>
    </property>
    <!-- additional elements removed from this example -->
</bean>
```

For additional information about search interfaces, see the *MDEX Engine Basic Development Guide*.

About applying Relevance Ranking to EQL results

You can apply Relevance Ranking to search keys, terms, and match modes other than those specified in the search filter.

You can specify your relevance ranking settings programatically through the `ResultsListConfig` object, or through the relevance ranking URL parameters. To apply these to your EQL query rather than the end user's search filter, pass in the EQL-specific values when setting the `relRankStrategy`, `relRankKey`, and `relRankTerms` properties (or the associated URL parameters). You must use one of these two methods to set the properties if you wish to apply relevance ranking to the EQL search results rather than the results from the end user's search filter.

For additional information, see the *Assembler API Reference (Javadoc)* and the documentation for the Results List cartridge.

Related Links

Configuration model for the Results List cartridge on page 100

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.

About troubleshooting EQL performance

If you wish to determine the impact of EQL expressions on your application, you should review the performance of cartridge handlers that are EQL-enabled.

EQL-enabled queries are not categorized separately in application logs; instead, you should review the performance metrics and behavior of cartridges that use EQL expressions. If you are modifying a cartridge handler to use EQL or creating a new cartridge handler that includes the feature, Oracle recommends supplementing the logging information for that cartridge to reflect the presence of EQL.

Chapter 10

Using an MDEX Engine to Manage 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 enables Guided Navigation in the Experience Manager Media Browser, allowing content administrators to easily navigate across resources when selecting media assets for a content item.

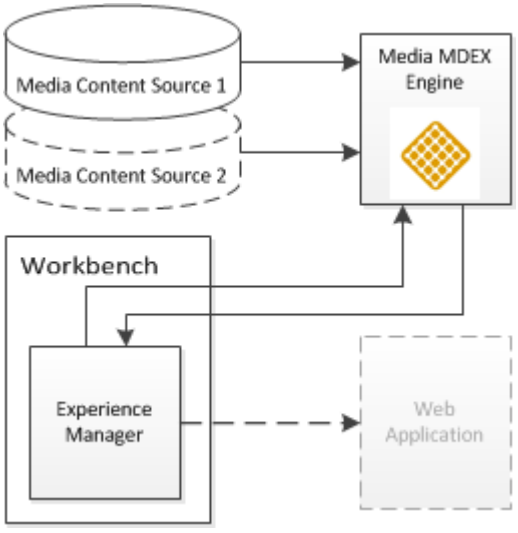
Interaction between an Endeca application and the media MDEX Engine components

The interactions between a media MDEX Engine, Experience Manager, and Endeca Web application are summarized below.

Note that for simplicity, the MDEX Engine backing the Endeca Web application is not shown in the diagrams below.

Interaction between the media MDEX Engine and Experience Manager

Experience Manager retrieves media asset information as follows:

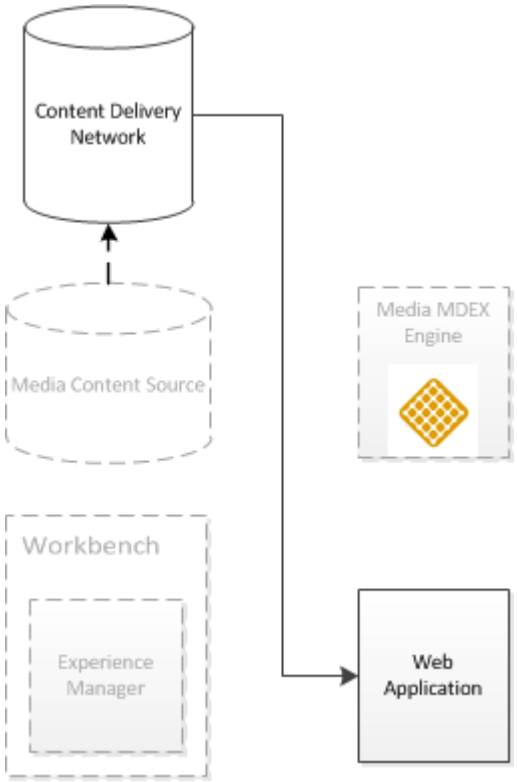


1. A CAS crawl and Forge pipeline populate the media MDEX Engine with media asset metadata and URIs from one or more content sources.

2. 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.
3. 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 Endeca application

In a production environment, the Endeca Web 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 Web application retrieves media from this content delivery network.

Note: The server hosting your 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 data application, the Endeca Configuration Repository is used as the authoring content source. For more information on configuring content sources, see "About Media editor configuration" in the "Template Property and Editor Reference" Appendix.

Related Links

About Media editor configuration on page 171
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.

About resolving media paths in content items on page 175

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.

Uploading media content for use in Experience Manager

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. You run the script after you locally add content to `<app dir>\config\media`. After uploading the 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. Start a command prompt (on Windows) or a shell (on UNIX).
3. Navigate to the `<app dir>\control` directory of your deployed application.

   This is located under your application directory. For example: `C:\Endeca\apps\<app name>\control`.
4. Run the `set_media` script.

Overview of the reference data application

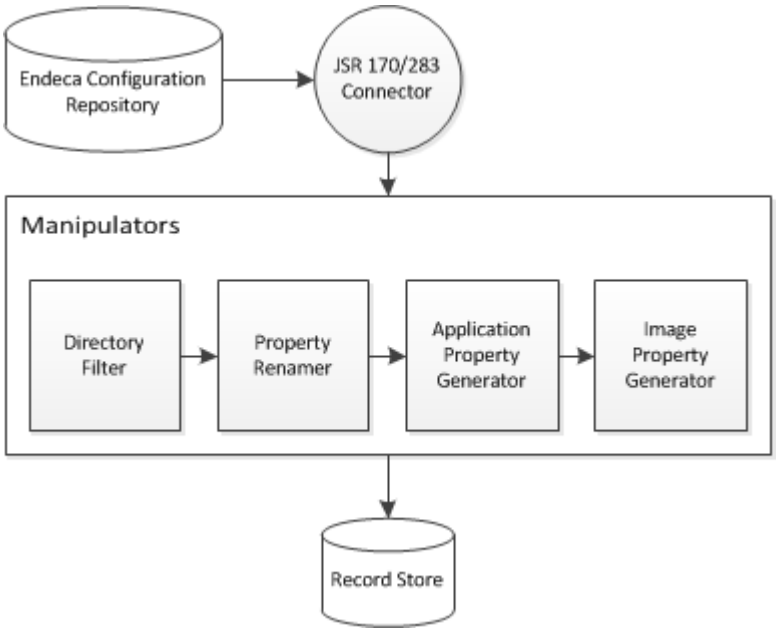
The Tools and Frameworks package includes a reference implementation of a media MDEX Engine that includes a CAS crawl and Forge pipeline for crawling resources in the Endeca Configuration Repository and indexing the corresponding metadata and URIs. The Experience Manager can then query the MDEX Engine for record information.

Endeca software requirements

In addition to the hardware and software required for Oracle Endeca Tools and Frameworks, the data ingest process for the media MDEX Engine requires the Oracle Endeca Content Acquisition System. You must have CAS 3.1.1 installed on the machine on which you are hosting your media MDEX Engine.

Media MDEX CAS crawl

Below is a representation of the CAS crawl for the media MDEX Engine:



Important: A connector for the JSR-170 Compliant data source is provided for the reference application, but requires a separate license for use. To obtain licensing information for this connector, contact your Oracle sales representative.

The crawl uses the following manipulators:

1. **Directory Filter**: This manipulator filters out directory records, so that only media files are output to the MDEX Engine.
2. **Property Renamer**: This manipulator maps crawled property value names to the schema used by the Media Browser in Experience Manager. For more information, see "Media MDEX Engine schema definition."
3. **Application Property Generator**: This manipulator analyzes record paths to determine which application a record is associated with (if any) and assigns it to a `media.site` property. 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. **Image Property Generator**: This manipulator analyzes image binaries to determine their width and height. It then adds corresponding `height` and `width` properties to each record.

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 a baseline update to index records to the media MDEX Engine.

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 a custom repository adapter and custom 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. For details, see "Media MDEX Engine schema definition."

Related Links

Media MDEX Engine schema definition on page 130
In order for the Media Browser in Experience Manager to have sufficient information for forming content XML, any media MDEX Engine that you configure must define specific properties and dimensions.

Deploying the media MDEX Engine data application

The 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. Additionally, your Discover Electronics reference application must be deployed and running.

To deploy the media MDEX Engine 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-3.1.1.jar`
      - `google-collections-1.0.jar`
2. Set up CAS for crawling JSR-170/ 283 compliant Jackrabbit repositories:
   a) Navigate to the `%ENDECA_TOOLS_ROOT%\reference\media-mdex-cas\cas\dependencies` directory.
   b) Copy the CAS dependencies to the `%CAS_ROOT%\lib\server` directory:
      - jackrabbit-jcr-rmi-2.2.9.jar
      - jcr-2.0.jar
3. Restart the CAS Service.
4. 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 *<Endeca Directory>*`/ToolsAndFrameworks/<version>/server/bin/setenv.bat` to set the environment variables for the current command window.

b) Navigate to the `C:\Endeca\ToolsAndFrameworks\<version>\deployment_template\bin` directory on Windows, or `/usr/local/endeca/ToolsAndFrameworks/<version>/deployment_template/bin` on UNIX.
c) Run `deploy.bat` or `deploy.sh` with the following options:

```
deploy --app <Endeca Directory>/ToolsAndFrameworks/<version>/refer¬
ence/media-mdex-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 is running on.
   By default, this is port 8006.
j) Specify the Dgraph and LogServer ports for your reference data application.

   Note: These must be different from those used for the Discover Electronics reference application.

   By default, this is port 17000 for the Dgraph and port 17010 for the LogServer. If you change these values, you must also update the configuration for the `MediaEditor` in the `<app dir>\config\editors_config\editors.xml` file.

k) Specify the CAS installation directory.
l) Enter the port that CAS runs on.
m) Enter the hostname that Workbench runs on.
   This information enables the data application to locate and query the Endeca Configuration Repository during the data ingest process.
n) Enter the name of the Discover Electronics application as specified during deployment.
   By default, this should be `Discover`.

5. Initialize and run a baseline update on the media MDEX Engine:
   a) Navigate to the `control` directory of your deployed data application.
   b) Run the `initialize_services` batch or shell script.
   c) Run the `baseline_update` batch or shell script.

Media MDEX Engine schema definition

In order for the Media Browser in Experience Manager to have sufficient information for forming content XML, any media MDEX Engine that you configure 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.site	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 media MDEX Engine includes the following such fields:

Field	Type	Description
media.mime_type	Dimension	The MIME type of the media asset. This enables filtering by media type and file extension in the Media Browser.
media.author	Dimension	The user who uploaded the specified asset to the Endeca Configuration Repository. This enables filtering by author in the Media Browser.
media.creation_date	Property	The date and time that the file was created on the Endeca Configuration Repository.
media.last_modifica¬tion_date	Property	The date and time that the file was last modified prior to being crawled by the Content Acquisition System.

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 in the 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.

Related Links

About the Media Browser on page 170

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.

Chapter 11

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.

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 Endeca 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.nav¬
igation.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 `Dgraph.WhyMatch` property on the record.
Why Rank	Returns information about why each record was ranked the way it was in a `Dgraph.WhyRank` property on the record.
Why Precedence Rule Fired	Returns information about precedence rules that fired on a query in a `DGraph.WhyPrecedenceRuleFired` property on each root dimension value.

Feature	Results
Word Interpretation	Returns information about word or phrase substitutions as a map that can be accessed via `getInterpretedTerms()` on the `SearchAd¬justments` model.

For details about the format of the debugging results, refer to the *MDEX Engine Advanced 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="sin¬
gleton">
    <!-- <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.ResultsListCon¬fig` 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.

Chapter 12

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(Re¬
```

```
questEventFactory.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 NavigationEventWrap¬
per(assemblerRequestEvent);
        // 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: "+navigation¬
```

```
Event.getSearchTerms());
        } else {
            System.out.println("There were no search terms in the current
request");
        }
    }
}
```

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="${preview.enabled}" />
            <property name="previewModuleUrl" value="http://${workbench.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.Con¬
tentItemAugmentAdapter">
                <constructor-arg ref="springUtility"/>
            </bean>
            <bean class="com.endeca.infront.navigation.event.LogServer¬
Adapter">
                <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 reference application includes two request event adapters, `ContentItemAugmentAdapter` and `LogServerAdapater`.

Adapter	Description
`com.endeca.infront.assembler.re¬ quest.ContentItemAugmentAdapter`	Appends request event information to the Content Item returned by the `assemble()` method. Information is included as a nested Content Item of type `AssemblerRe¬ questEvent`, with the key `endeca:assembler¬ RequestInformation`. For a list of attributes that are available out-of-the-box, see *Request Event Attributes* on page 197.
`com.endeca.infront.naviga¬ tion.event.LogServerAdapter`	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 `WEB-INF\assembler.properties` 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
IN_DIM_SEARCH	`Boolean`	Did the user select a dimension search result.
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*.

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.
`NumericStepperEdi¬tor`	`<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.

Editor	Property Type	Functionality
`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.
`RecordListEditor`	`<xavia:List>`	**Important:** This editor is deprecated. Use the `SpotlightSelectionEditor` instead. Displays as a button that launches the microbrowser and allows the content administrator to select the list of records that populates a `<xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection"/>` 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 `<xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelection"/>` 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 `<xavia:List>` properties, one for boosted dimension refinements and one for buried dimension refinements.

Editor	Property Type	Functionality
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 <xavia:List> properties, one for boosted records and one for buried records.
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	<ContentItem_List>	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.infront.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

Editor	Property Type	Functionality
		can browse through media in the configured source repository, and generate a link to a selected asset.
`RecordStratifica¬tionEditor`	`<xavia:List>`	**Important:** This editor is deprecated. Use the `BoostBuryRecordEditor` 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. The editor populates two `<xavia:List>` properties, one for boosted records and one for buried records.
`RichTextEditor`	`<String>`	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`	`<xavia:Item>`	Displays as a dropdown. The content administrator selects a sort order from those configured in the editor. The editor includes multiple `<xavia:Item class="com.endeca.infront.navigation.mod¬el.SortOption"/>` item properties that each specify an available sort option. For more information, see "Adding a Sort editor."

Related Links

Basic content properties on page 147
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 161
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 `propertyName` 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"
    id="ResultsList">

  <!-- 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.mod¬
el.SortSpec">
                         <xavia:Property name="key">product.analytics.to¬
tal_sales</xavia:Property>
                      <xavia:Property name="descending">false</xavia:Prop¬
erty>
                     </xavia:Item>
```

```
                </xavia:List>
             </xavia:Property>
          </xavia:Item>
      </Property>
      <Property name="relRank">
          <!-- Margin Bias -->
          <String>nterms,maxfield,exact,static(product.analytics.conver¬
sion_rate,descending)</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" id="DimensionNavigation">
   <!-- 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
`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.
`enabled`	If set to `false`, 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. Set this to `false` only if you specify a default value in the definition of the string property. Editors are enabled by default.
`width`	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.
`height`	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"
                id="ThreeColumnNavigationPage">
  <!-- 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>
```

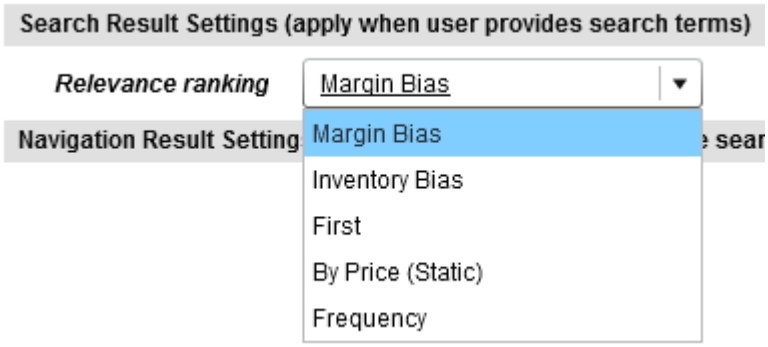
```
  <EditorPanel>
    <BasicContentItemEditor>
      <GroupLabel label="Page metadata"/>
      <editors:StringEditor propertyName="title" label="Title" en¬
abled="true"/>
     <editors:StringEditor propertyName="metaKeywords" label="Meta keywords"

        enabled="true" height="72"/>
     <editors:StringEditor propertyName="metaDescription" label="Meta de¬
scription"
        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
`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 choice editor.
`editable`	If set to `true`, this attribute allows Experience Manager users to specify custom string values. By default, choice editors are not editable.

Attribute	Description
`enabled`	If set to `false`, the choice editor displays in Experience Manager but the value cannot be changed by the user. By default, choice editors are enabled.
`prompt`	Specifies a custom prompt. The default prompt is an empty string.
`tooltip`	If present, specifies optional help text to display in a tool tip window. The default behavior is no tool tip.
`width`	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
`value`	Required. The string value to assign to the associated property if this `<choice>` is selected.
`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 `label` 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.conver¬
sion_rate,descending)</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.con¬`

version_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"
    id="ResultsList">
    <!-- 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.conver¬
sion_rate,descending)</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,stat¬
ic(product.analytics.conversion_rate,descending)" />
         <choice label="Inventory Bias" value="nterms,maxfield,exact,stat¬
ic(product.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
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 choice editor.
enabled	If set to false, 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
value	Required. The string value to assign to the associated property if this <choice> is selected.
label	This attribute allows you to specify a more descriptive label for the radio button associated with this option. If no label is specified, the value 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="showDis¬abledRefinements":

```
        <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" id="ExampleRadioGroupCartridge">
   <!-- 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
`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 60.
`height`	The height, in pixels, of the editor. The default height is 24.
`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.
`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" id="DimensionNavigation">
```

```
    <!-- 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" en¬
abled="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.

Attribute	Description
`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" id="ExampleSliderCar¬
tridge">
  <!-- 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"
                 id="SearchBox">
    <!-- 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
`propertyName`	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.
`enabled`	If set to `false`, 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"
                 id="SearchBox">
    <!-- 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" id="ResultsList">
  <!-- 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.to¬
tal_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 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 corresponding 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" id="ResultsList">
  <!-- 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.to¬
tal_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>
```

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"
                 id="RecordSpotlight">
    <!-- additional elements omitted from this example -->
    <ContentItem>
        <!-- additional elements omitted from this example -->
        <Property name="recordSelection">
          <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlight¬
Selection" />
        </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"
                 id="RecordSpotlight">
    <!-- 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" la¬
```

```
bel="Spotlight Records">
                <PreviewProperty name="product.name"/>
            </editors:RecordListEditor>
            <editors:StringEditor propertyName="seeAllLink" label="See
All 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 Discover Electronics reference application makes MDEX Engine configuration information available through a data service. The data service configuration file is used by editors that need to query the authoring MDEX Engine used in the reference application, such as those that use the microbrowser.

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 `initialize_services` 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.
`wildcard¬ SearchEnabled`	Optional. Wildcard search is enabled by default. If your configuration does not index dimensions by wildcard index, you must explicitly set this property to `false`.
`recordName¬ Field`	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: • `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 and updates all editors that rely on the data service.

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

To make MDEX Engine configuration information available to editors, you must configure a data service. In the Discover Electronics reference application, the data service configuration file is used by editors that need to query the authoring MDEX Engine used in the reference application, such as those that use the **Select Records** dialog.

The configuration file, <app dir>\config\editors_config\services\endecaBrowserService.json, is shown below:

```
{
    "host": "myhost.mydomain.com",
    "port": "15002",
    "recSpecProp": "common.id",
    "recAggregationKey": "product.code",
    "recFilter": "",
    "recImgUrlProp" : "product.img_url_thumbnail",
   "recDisplayProps": [ "product.name", "product.price", "product.short_de¬
sc" ],
    "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 initialize_services 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.
recSpecProp	The dimension used as the record specifier. This must be a unique identifier.
recAggregation¬Key	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.
recDis¬playProps	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.
textSearch¬MatchMode	Optional. Specifies the match mode to apply to text searches in the **Select Records** dialog.

You can modify these values as necessary. Running <app dir>\control\set_editors_config pushes changes to the Discover Electronics reference application and updates all editors that rely on the data service.

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 a content collection and number of content items with which to populate that section.

The dynamic slot editor takes a single property, zoneType. When the content administrator edits this cartridge in Experience Manager, the editor queries the Endeca Configuration Repository for content collections of the appropriate type. The evaluation limit of the selected collection determines the maximum possible evaluation limit for the dynamic slot, though the slot may be configured to return fewer content items than this maximum.

Creating a cartridge template with a dynamic slot

You should configure a separate cartridge template for each template type into which you wish to include a dynamic slot.

To create a cartridge template with a dynamic slot:

1. Insert a `ContentItem` that includes a `contentCollection` and `ruleLimit` property, as in the following example:

```
<ContentItem>
    <Name>Search Box Slot</Name>
    <Property name="contentCollection"/>
    <Property name="ruleLimit"></Property>
</ContentItem>
```

2. Within the `ruleLimit` property, insert a `String` element that specifies the maximum number of matching content items to return for a query:

```
<ContentItem>
    <Name>Search Box Slot</Name>
    <Property name="contentCollection"/>
    <Property name="ruleLimit"><String>1</String></Property>
</ContentItem>
```

3. Insert a corresponding `<editors:DynamicSlotEditor>` element within `<BasicContentItemEd¬itor>`.
4. Specify the `zoneType` attribute for the editor. This value must match the `id` attribute for the corresponding template type:

```
<EditorPanel>
    <BasicContentItemEditor>
        <editors:DynamicSlotEditor zoneType="SearchBox"/>
    </BasicContentItemEditor>
</EditorPanel>
```

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 dynamic slot Search Box cartridge. Note that although the content type for this cartridge is `HeaderContent`, the zone type for items in the Search Box Content collection is `SearchBox`, which is a more specific constraint on what content items can populated the dynamic slot:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  xmlns:editors="editors"
  type="HeaderContent"
  id="SearchBoxSlot">
    <!-- additional elements omitted from this example -->
    <ContentItem>
```

```
        <Name>Search Box Slot</Name>
        <Property name="contentCollection"/>
        <Property name="ruleLimit"><String>1</String></Property>
    </ContentItem>

    <EditorPanel>
        <BasicContentItemEditor>
            <editors:DynamicSlotEditor zoneType="SearchBox"/>
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>
```

If the content administrator wishes to change the specified collection, they are restricted to collections of the appropriate `zoneType`, in this case, `SearchBox`.

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 configured separately.

Once you have created a cartridge template that uses a dynamic slot, you must specify a cartridge handler for that template. Your cartridge handler should inherit the `CartridgeHandler_ContentSlot` handler.

The response model for a dynamic slot cartridge is a `ContentItem` that includes a `contents` property containing a list of content items. The `ContentItem` also returns its `contentCollection` and `ruleLimit`.

The examples below use the cartridge handler configuration in the Assembler context file of the Discover Electronics application, located in `%ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring\WEB-INF\assembler-context.xml`.

To add a cartridge handler for your dynamic slot template:

1. Stop the Endeca Tools Service.
2. Open your Spring context configuration file.

   In the Discover Electronics reference application, this is located in `%ENDECA_TOOLS_ROOT%\reference\discover-electronics-authoring\WEB-INF\assembler-context.xml`.
3. Locate the `CartridgeHandler_ContentSlot` bean:

   ```
   <!--
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   ~ BEAN: CartridgeHandler_ContentSlot
   -->
   <bean id="CartridgeHandler_ContentSlot" class="com.endeca.infront.car¬
   tridge.ContentSlotHandler"
       scope="prototype">
     <property name="contentManager" ref="cartridgeManager" />
   </bean>
   ```
4. Add a new `CartridgeHandler` bean for your dynamic slot cartridge.

   Set the `id` attribute to correspond to the ID of your dynamic slot cartridge template, and set the `parent` attribute to the `CartridgeHandler_ContentSlot` handler:

   ```
   <bean id="CartridgeHandler_MyPageSlot" parent="CartridgeHandler_Con¬
   tentSlot"
         scope="prototype"/>
   ```

5. Save and close the file.
6. Start the Endeca Tools Service.

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"
  id="MediaBanner">
    <!-- 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 Desti¬
nation" 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/en¬
decaBrowserService.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.

Property	Description
`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.
`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 163
To make MDEX Engine configuration information available to editors, you must configure a data service. In the Discover Electronics reference application, the data service configuration file is used by editors that need to query the authoring MDEX Engine used in the reference application, such as those that use the **Select Records** dialog.

About the Select Records dialog on page 163
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" en¬
abled="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"
                 id="MediaBanner">
  <!-- 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" en¬
abled="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 174
The `set_media` script uploads any media assets in your application's local `config\media` directory to the Endeca Configuration Repository.

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/Dis¬
cover/media/</default>
                    <IFCRSource>http://myhost.mydo¬
main.com:8006/ifcr/sites/Discover/media/</IFCRSource>
                </mediaRoots>
                <mdexPort>17000</mdexPort>
                <mdexHost>myhost.mydomain.com</mdexHost>
                <videoFormats>flv|f4v|3pg|mov|mp4</videoFormats>
                <imageFormats>jpg|jpeg|png|gif</imageFormats>
                <mediaURI>/ifcr/sites/Discover/media/</mediaURI>
            </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 `false`.
`mediaRoot`	This property specifies the absolute URLs to available media repositories. It includes a nested `default` 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.
`IFCRSource`	The absolute URL to the media repository used by the Media Browser. The property name is the same as that of the associated CAS crawl name used by the media MDEX Engine.
`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-delineated 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-delineated 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.

Related Links

Media MDEX Engine schema definition on page 130
In order for the Media Browser in Experience Manager to have sufficient information for forming content XML, any media MDEX Engine that you configure must define specific properties and dimensions.

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/Discov¬
er/media/</default>
            <IFCRSource>http://myhost.mydomain.com:8006/ifcr/sites/Dis¬
cover/media/</IFCRSource>
        </mediaRoots>
        <mdexPort>17000</mdexPort>
        <mdexHost>mymediahost.mydomain.com</mdexHost>
        <videoFormats>flv|f4v|3pg|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 your media host and corresponds to the value of the `media.repository_id` property assigned to each record. 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 `IFCRSource`:

```
<mediaRoots>
    <default>http://myhost.mydomain.com:8006/ifcr/sites/Discover/media/</de¬
fault>
    <IFCRSource>http://myhost.mydomain.com:8006/ifcr/sites/Discover/me¬
```

```
dia/</IFCRSource>
</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 Manage Media Assets on page 125
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 enables Guided Navigation in the Experience Manager Media Browser, allowing content administrators to easily navigate across resources when selecting media assets for a content item.

Media MDEX Engine schema definition on page 130
In order for the Media Browser in Experience Manager to have sufficient information for forming content XML, any media MDEX Engine that you configure must define specific properties and dimensions.

Uploading media to the Endeca Configuration Repository

The `set_media` script uploads any media assets in your application's local `config\media` directory to the Endeca Configuration Repository.

These steps are based on local copies of your files being up to date and located in the `<app dir>\config\media` directory.

To upload images to the Endeca Configuration Repository:

1. Confirm that local copies of the images you wish to upload have been copied to the `<app dir>\config\media` directory, for example, `C:\Endeca\apps\Discover\config\media.`
2. From a command prompt, navigate to the `<app dir>\control` directory, for example, `C:\Endeca\apps\Discover\control`.
3. Run the `set_media` batch or shell script.
   This script uploads media assets to the `/ifcr/sites/<App_Name>/media` node in the Endeca Configuration Repository, where they become available to the Media editor.
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.
      In the Discover Electronics reference application, navigate to **Web** > **Web Browse Pages** > **Category - Cameras**and select the **Media banner** template.
   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/</my¬
MediaSource>
```

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.MediaSourceCon¬
fig">
                <property name="sourceName" value="MyMediaSource" />
                <property name="sourceValue" value="http://myhost.mydo¬
main.com:8006/myCMS/Discover/media/" />
            </bean>
           <bean class="com.endeca.infront.cartridge.model.MediaSourceCon¬
fig">
                <property name="sourceName" value="default" />
                <property name="sourceValue" value="http://myhost.mydo¬
main.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.MediaSourceCon¬
fig">
                <property name="sourceName" value="MyMediaSource" />
                <property name="sourceValue" value="http://myhost.mydo¬
main.com:8006/myBiggerFasterCMS/Discover/media/assets/" />
            </bean>
           <bean class="com.endeca.infront.cartridge.model.MediaSourceCon¬
fig">
                <property name="sourceName" value="default" />
                <property name="sourceValue" value="http://myhost.mydo¬
main.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/im¬
ages/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.

Related Links

Interaction between an Endeca application and the media MDEX Engine components on page 125
The interactions between a media MDEX Engine, Experience Manager, and Endeca Web application are summarized below.

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
`propertyName`	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.

Attribute	Description
buryProperty	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://endeca.com/schema/content-template/2008"
    xmlns:editors="editors"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    type="MainContent" id="ResultsList">
  <!-- 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 163
To make MDEX Engine configuration information available to editors, you must configure a data service. In the Discover Electronics reference application, the data service configuration file is used by editors that need to query the authoring MDEX Engine used in the reference application, such as those that use the **Select Records** dialog.

About the Select Records dialog on page 163
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:identi¬
   fier" />
   </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" id="GuidedNavigation">
   <Description>Creates a container for navigation cartridges.</Description>

   <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/Secondary_Guid¬
edNav.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" dimensionProper¬
ty="endeca:name" />
                    <endeca:Property name="dimensionId" dimensionProper¬
ty="endeca: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 `<BasicContentItemEdi¬tor>`.
2. Specify additional attributes for the editor:

Attribute	Description
`propertyName`	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.
`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 `false`, 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" id="RefinementMenu">
   <Description>Displays Endeca Facet Navigation in a Text Link Rendering.
 For Flat Dimensions only.</Description>
   <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/dimension_navi¬
gation.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"
 idProperty="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 163
To make MDEX Engine configuration information available to editors, you must configure a data service. In the Discover Electronics reference application, the data service configuration file is used by editors that need to query the authoring MDEX Engine used in the reference application, such as those that use the **Select Records** dialog.

About the Select Records dialog on page 163
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
propertyName	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" id="DimensionSearchResults">

    <Description>Displays dimension search results.</Description>
   <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/Main_Dimension¬
SearchResults.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 `dimen¬sionId` 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 false, 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 boostList and buryList 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" id="RefinementMenu">
   <Description>Displays Endeca Facet Navigation in a Text Link Rendering.
 For Flat Dimensions only.</Description>
   <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/dimension_navi¬
gation.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"
buryProperty="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
`propertyName`	Required. The name of the List property that represents the selected dimension values. The property must be declared in the same template.
`dimensionId`	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" id="RefinementMenu">
   <Description>Displays Endeca Facet Navigation in a Text Link Rendering.
 For Flat Dimensions only.</Description>
   <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/dimension_navi¬
gation.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>
```

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.RecordSpotlightSe¬lection` 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
`propertyName`	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
`name`	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"
                 id="RecordSpotlight">
    <!-- 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.RecordSpotlight¬
Selection" />
        </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
`label`	A descriptive label that displays in a drop-down menu in Experience Manager.
`sorts`	A list of one or more items of class `com.endeca.infront.naviga¬tion.model.SortSpec`. Each `SortSpec` has two properties: • `key` — A string representing the name of an Endeca property or dimension on which to sort • `descending` — 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" id="ResultsList">
  <!-- 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.to¬
tal_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>
```

Related Links

Cartridge handler configuration for the Results List cartridge on page 102

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.

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 <BasicContentItemEd¬itor>.
2. Specify additional attributes for the editor:

Attribute	Description
propertyName	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.
buryProperty	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://endeca.com/schema/content-template/2008"
    xmlns:editors="editors"
    xmlns:xavia="http://endeca.com/schema/xavia/2010"
    type="MainContent" id="ResultsList">
  <!-- 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 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" id="ResultsList">
  <!-- 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.to¬
tal_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>

  <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.Sort¬
Spec">
               <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>
       <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.Sort¬
Spec">
                <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.Sort¬
Spec">
                <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.Sort¬
Spec">
                <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.RecordSpotlightSe¬lection`.
   In the following example, this is the `recordSelection` property:

```
<Property name="recordSelection">
    <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlightSelec¬
tion" />
</Property>
```

2. Insert an `<editors:RecordSpotlightSelectionEditor>` element within `<BasicCon¬tentItemEditor>`.
3. Specify label attributes and the additional attributes for the editor:

Attribute	Description
`propertyName`	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.

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"
                 id="RecordSpotlight">
    <Description>Displays selected records in seconday content area.</De¬
scription>
    <ThumbnailUrl>/ifcr/tools/xmgr/img/template_thumbnails/Sec¬
ondary_RecordSpotlight.png</ThumbnailUrl>
    <ContentItem>
        <Name>Spotlight Records</Name>
        <!-- additional elements omitted from this example -->
        <Property name="recordSelection">
          <xavia:Item class="com.endeca.infront.cartridge.RecordSpotlight¬
Selection" />
        </Property>
        <!-- additional elements omitted from this example -->
    </ContentItem>

    <EditorPanel>
        <BasicContentItemEditor>
            <GroupLabel label="Define Spotlight"/>
            <!-- additional elements omitted from this example -->
          <editors:SpotlightSelectionEditor propertyName="recordSelection"
 label="Spotlight Records" />
            <!-- additional elements omitted from this example -->
        </BasicContentItemEditor>
```

```
    </EditorPanel>
</ContentTemplate>
```

Related Links

Select Records data service configuration reference on page 163
To make MDEX Engine configuration information available to editors, you must configure a data service. In the Discover Electronics reference application, the data service configuration file is used by editors that need to query the authoring MDEX Engine used in the reference application, such as those that use the **Select Records** dialog.

About the Select Records dialog on page 163
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 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.

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 `<BasicContentItemEd¬itor>`.
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','Paste¬
   FromWord','-','Undo','Redo' ] },
     { name: 'insert',      items : [ 'Image','Table','HorizontalRule','Spe¬
   cialChar'] },
     { name: 'paragraph',   items : [ 'NumberedList','BulletedList','-
   ','Outdent','Indent','-','JustifyLeft',
                                   'JustifyCenter','JustifyRight','Justi¬
   fyBlock' ] },
     { name: 'links',       items : [ 'Link','Unlink','Anchor' ] },
     '/',
     { name: 'basicstyles', items : [ 'Bold','Italic','Under¬
   line','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"
  id="MediaBanner">
    <!-- 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="De¬
scription" enabled="true" height="200"
                toolbar="[
                { name: 'document',    items : [ 'Source'] },
                { name: 'clipboard',   items : [
'Cut','Copy','Paste','PasteText','PasteFromWord','-','Undo','Redo' ] },
               { name: 'insert',      items : [ 'Image','Table','Horizon¬
talRule','SpecialChar'] },
                { name: 'paragraph',   items : [ 'NumberedList','Bullet¬
edList','-','Outdent','Indent','-',
                                               'JustifyLeft','JustifyCen¬
ter','JustifyRight','JustifyBlock' ] },
                { name: 'links',       items : [ 'Link','Unlink','Anchor'
 ] },
                '/',
                { name: 'basicstyles', items : [ 'Bold','Italic','Under¬
line','Strike','Subscript','Superscript' ] },
                { name: 'styles',      items : [ 'Styles','For¬
mat','Font','FontSize' ] },
                { name: 'colors',      items : [ 'TextColor'] }
              ]"/>
        </BasicContentItemEditor>
    </EditorPanel>
</ContentTemplate>
```

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
"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.subRecordsPerAggregatRecord	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

Feature	URL Parameter	Global Configuration <class>.<property>	Cartridge Handler(s)
Geocode range filter	Nfg	FilterState.rangeFilters	UrlNavigationState¬ Builder
Set preview time	Ende¬ ca_Time	UserState.date	- -
Relevance ranking Match Mode	Nrm	FilterState.SearchFil¬ ters.MatchMode	ResultsList
Relevance ranking strategy	- -	ResultsListConfig.rel¬ RankStrategy, Dimension¬ SearchResultsConfig.rel¬ RankStrategy	ResultsList
Relevance ranking search terms	Nrt	- -	- -
Relevance ranking search key	Nrk	- -	ResultsList
EQL filter	Nrs	FilterState.eqlFilter	UrlNavigationState¬ Builder
Sort key Sort order	Ns	ResultsListConfig.sortOp¬ tion, RefinementMenu.sort	ResultsList, Refinement¬ Menu
Compute phrasings Rewrite query with alternate phrasing	Ntp	SearchAdjustmentsCon¬ fig.phraseSuggestionEn¬ abled	UrlNavigationState¬ Builder
Search terms	Ntt	FilterState.SearchFil¬ ters.terms	UrlNavigationState¬ Builder
Search mode	Ntx	FilterState.SearchFil¬ ters.matchMode	UrlNavigationState¬ Builder
"Did You Mean"	Nty	SearchAdjustmentsCon¬ fig.spellSuggestionEn¬ abled	UrlNavigationState¬ Builder
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	- -	DimensionSearchResultCon¬ fig.relRank	DimensionSearchRe¬ sultHandler
Dimension search scope	N with Dy=1	See N	See N

Feature	URL Parameter	Global Configuration <class>.<property>	Cartridge Handler(s)
Dimension search result offset	- -	- -	- -
Dimension search dimVal count	- -	DimensionSearchResultCon¬ fig.maxResultsPerDimen¬ sion	DimensionSearchRe¬ sultHandler
Dimension search record filter	Nr with Dy=1	See Nr	See Nr
Dimension search refinement configuration	- -	DimensionSearchResultCon¬ fig.showCountsEnabled	DimensionSearchRe¬ sultHandler
Dimension search EQL filter	Nrs with Dy=1	See Nrs	See Nrs
Dimension search options	- -	- -	- -

Appendix B

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.

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
`endeca:sessionId`	`String`	The unique identifier for a browser session. To retrieve this information, you must register an implementation of `SessionIdProvider` in the request event adapter constructor.
`endeca:assemblyStartTimestamp`	`long`	The time (in milliseconds from POSIX Epoch) that the `assemble()` method started
`endeca:assemblyFinishTimestamp`	`long`	The time (in milliseconds from POSIX Epoch) that the `assemble()` method finished

Navigation request event attributes

The following describes the schema for an Assembler navigation 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
`endeca:autocorrectTo`	`String`	The suggested auto-correct term, if it triggers for the request.

Attribute	Type	Description
endeca:contentPath	String	The content path of the page corresponding to the request.
endeca:didYouMeanTo	List <String>	The suggested "Did You Mean" term, if it triggers for the request.
endeca:dimensions	List <String>	The dimension names selected for navigation.
endeca:dimensionVal¬ues	List <String>	The dimension value names selected for navigation.
endeca:eneTime	Long	The time, in milliseconds, that it takes the MDEX Engine to run the query.
endeca:numRecords	Long	The number of records returned for the request.
endeca:numRefine¬ments	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: • 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 fieldName|<*Descending*|*Ascending*> .

Attribute	Type	Description
`endeca:spotlights`	`List <String>`	The list of spotlights triggered for the request.

Index

E

F

G

H

J

K

L

M

N

P

Q

R

S

T

U

Z