Endeca Content Assembler

Experience Manager Developer's Guide

Version 2.1.x • March 2012

ORACLE®

ENDECA

Contents

Copyright and disclaimer

Preface

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

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

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

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

About this guide

This guide describes the major tasks involved in supporting content administrators using Endeca Experience Manager and creating templates to drive dynamic pages. It also covers extending Experience Manager functionality using the Experience Manager Editor SDK.

This guide assumes that you have read the *Oracle Endeca Experience Manager Getting Started Guide* and that you are familiar with Endeca’s terminology and basic concepts.

This guide covers only the features of the Content Assembler API, and is not a replacement for the available material documenting other Endeca products and features. For a list of recommended reading, please refer to the section "Who should use this guide."

Who should use this guide

This guide is intended for developers who are building Endeca applications using the Endeca Workbench and the Content Assembler API.

If you are a new user of Oracle Endeca Experience Manager and are are not familiar with developing Endeca applications, Oracle recommends reading the following guides prior to this one:

1. *Oracle Endeca Experience Manager Getting Started Guide*
2. *Endeca Basic Development Guide*
3. *Endeca Advanced Development Guide*

Once you have familiarized yourself with the concepts in this *Oracle Endeca Experience Manager Developer's Guide*, Oracle recommends reading one of the following:

- *Endeca Content Assembler API and Reference Application Guide for Java* (Content Assembler API for Java users)
- *Endeca Content Assembler API and Reference Application Guide for the RAD Toolkit for ASP.NET* (Content Assembler API for the RAD Toolkit for ASP.NET users)

If you are an existing user of Oracle Endeca Guided Search or Oracle Endeca Experience Manager and you are familiar with developing Endeca applications, Oracle recommends reading the *Oracle Endeca Experience Manager Getting Started Guide* before this guide.

Remember: All documentation is available on the Oracle Technology Network.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Endeca Customer Support

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

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

Chapter 1

Developing Applications with Template-Based Pages

This section provides an overview of working with Experience Manager and the Content Assembler API to create an Endeca application with template-driven dynamic pages.

Overview of roles

This introduction discusses several roles involved in creating a Web application with template-based pages.

- *Application developers*, who create a set of custom templates as well as a front-end application that can render template-based content.
- *Content administrators*, who are typically site editors responsible for managing the information presented in the Web application. In the case of an eCommerce site, this role may be filled by merchandisers.
- *Managers* who are responsible for approving pages for publication. These managers may have titles such as Editorial Director or Director of Merchandising.
- A *creative team*, which may consist of an art director, Web designer, and graphic artist, or any combination of those roles.

About cartridges

A cartridge is a functional component that a content administrator can place on a dynamic page using Experience Manager. Examples of cartridges may include a Guided Navigation cartridge, results list cartridge, or rotating banner cartridge.

A cartridge is comprised of several parts:

- A configuration file (XML **template**) that defines the content structure and the editing interface in Experience Manager. Experience Manager ships with a collection of reference templates, but application developers should customize these for their specific needs.
- Zero or more Experience Manager **editors** that allow the content administrator to configure cartridge content. Experience Manager ships with several standard editors, but application developers can develop editors that are tailored to particular business needs if so desired.
- One or more Content Assembler **tag handlers** to do additional query processing before returning results to the application. The Content Assembler ships with several standard tag handlers.

Developing tag handlers is usually not necessary, but can be useful for interfacing with third-party systems.

- **Rendering code** to display the content in the Web application. Each template is associated with a particular piece of rendering code, enabling developers to build applications (and content administrators to configure their content) in a modular fashion.

Templates serve as a basis for the dynamic pages that content administrators create in Experience Manager. The templates are XML documents that define the content structure of a dynamic page, or part of a page. A template can be thought of as a content object definition that declares what properties the content contains, and how those properties can be configured in Experience Manager.

Because a template defines the properties in the content objects, you can write code that is tailored to render the content driven by a specific template. For example, one cartridge template may contain properties for a banner image and a link, while another template may be designed to contain several records. The corresponding cartridge code for each of these templates then uses the configured values of these properties to create a banner promotion or a Content Spotlighting section of a page.

By building cartridges and associating specific rendering logic with every template in the application, developers ensure that the application can render any page configuration created in Experience Manager. This enables content administrators to flexibly combine content in a modular manner while maintaining a consistent look-and-feel throughout an entire site.

For more information about writing editors for Experience Manager, see "Extending Experience Manager with Community Editors" in this guide. For more information about writing tag handlers for the Content Assembler, see the *Content Assembler API Developer's Guide*.

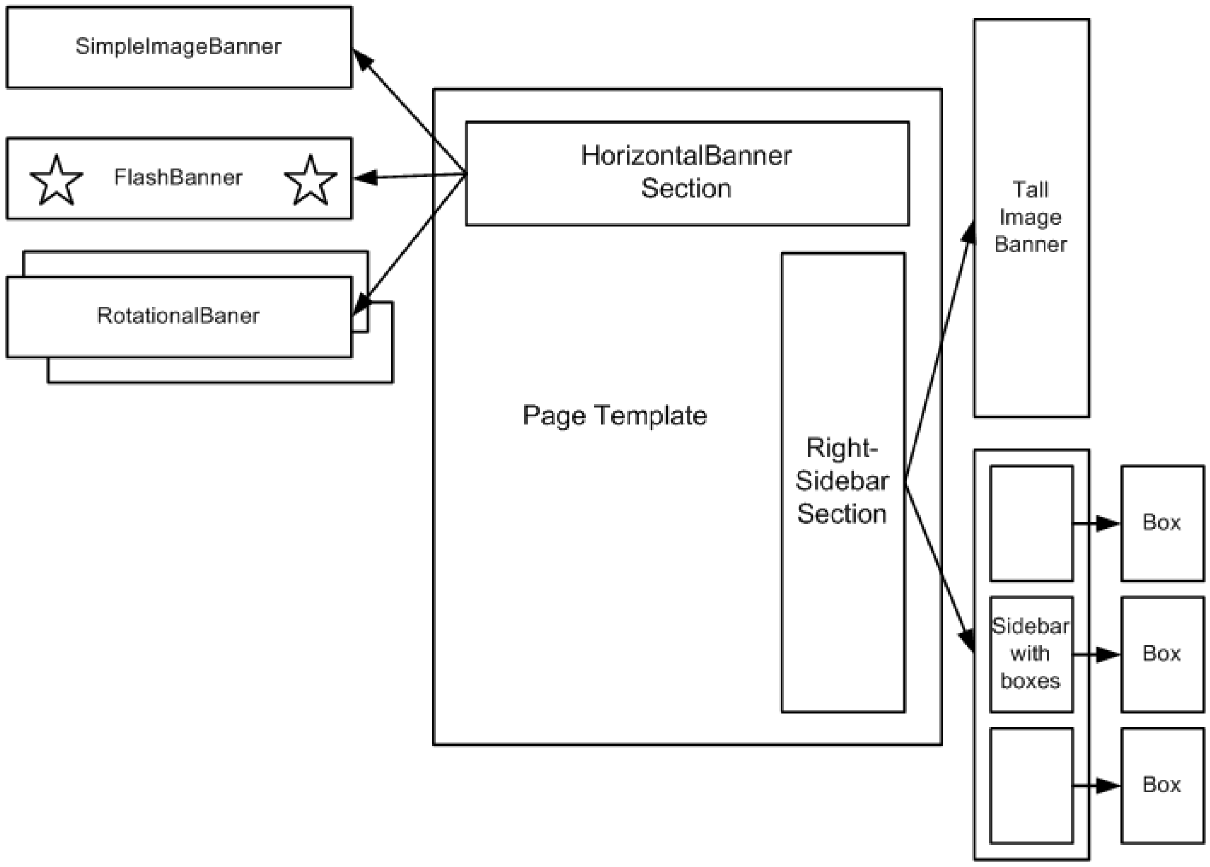
About templates and Experience Manager

Templates can either define the content structure of an entire page or a section of a page.

Page templates drive the content of an entire Web page. They define parts of the page called *sections*.

Sections can be thought of as slots that can be filled with content such as a banner image, Content Spotlighting, or search results). Typically, they represent a physical area on the page, but can also represent content that is not visible (for example, meta keywords used for search engine optimization). They can also represent content that may be rendered in a number of ways (for example, a page element that may display in the left or right column of a page depending on context).

The content in sections is also driven by templates known as *cartridge templates*. The following diagram shows a page template that includes two sections, and the cartridge templates that can be applied to each section.



A template defines what kinds of content can be placed in each of its sections. In this example, the SimpleImageBanner, FlashBanner, and RotationalBanner cartridges can be inserted in any section of type HorizontalBanner. A page template may include multiple sections of the same type. The same type of section can also be used in multiple templates. Cartridge templates can in turn define sections within them.

Content administrators configure dynamic pages in Experience Manager by selecting cartridges to insert into sections and then populating the cartridges with content. The interface for populating cartridges in Experience Manager is driven by the template definition. Some cartridges may be prepopulated by the application developer with information that the application can use to render predefined content, without the need for additional configuration by the content administrator. Such prepopulated cartridges can still present contextually relevant content through the use of Endeca query refinements.

About content items and the Content Assembler API

When a content administrator creates pages in Experience Manager, the resulting configurations are saved as XML documents in the MDEX Engine.

If a template is seen as a content object definition, then these page configurations represent instances of the content objects that have specific values for the properties defined in the template.

The Content Assembler retrieves these page configurations, evaluates the XML and returns content item objects based on the configuration and any additional queries that need to be made. For example, if the template defines a record list property and a content administrator specifies a navigation query to populate that property, the assembler executes that query to return specific records in the content item object.

Each template corresponds to a single content item object, which may contain other content items within it. In other words, the entire page is returned by the Content Assembler API as a single root content item, and each cartridge within the page is returned as a nested content item property within the parent content item.

A typical workflow for creating a template-based application

Applications built with template-based pages present dynamic content with a consistent look and feel, while Experience Manager enables updates to that content with relatively little maintenance.

The process of developing the application generally begins with the creative team. This team develops mockups of the page layouts that are used throughout the application, as well as any custom images or rich media that the design requires. Although there may be several variations, the set of layouts for a site typically follow a common high-level structure that results in a unified appearance across an entire site. For example, certain elements (such as a logo or banner) may always be present in the same location, or the proportions or relative position of various areas of the page may remain constant even if the content within those areas changes.

The application developer then creates cartridges based on the layouts from the creative team. This involves writing templates that describe the overall content structure of a page, including page sections that can be filled by certain cartridges. Typically, an application has only a few top-level page templates that define the overall page layouts that are used in the site, and many cartridge templates that drive the behavior of specific parts of a page. For each template, the developer writes code for the front-end application that can render the content items based on that template. Because templates define the kinds of content that are allowed in a page, an application that is aware of the templates being used within the site can include very specific rendering logic for pages based on those templates.

The developer uploads the templates to Experience Manager, and specifies an application that uses the cartridge code as the preview application in Endeca Workbench.

A content administrator can then use Experience Manager to create and configure dynamic pages. The content administrator can control the conditions under which a page should display by applying triggers based on navigation state, search terms, date ranges, or user profiles. Configuring a page in Experience Manager consists of associating it with a particular trigger or set of triggers, and designating the content to display by inserting and configuring cartridges in each section. Cartridges can contain content that is either static or dynamically populated based on queries to the MDEX Engine.

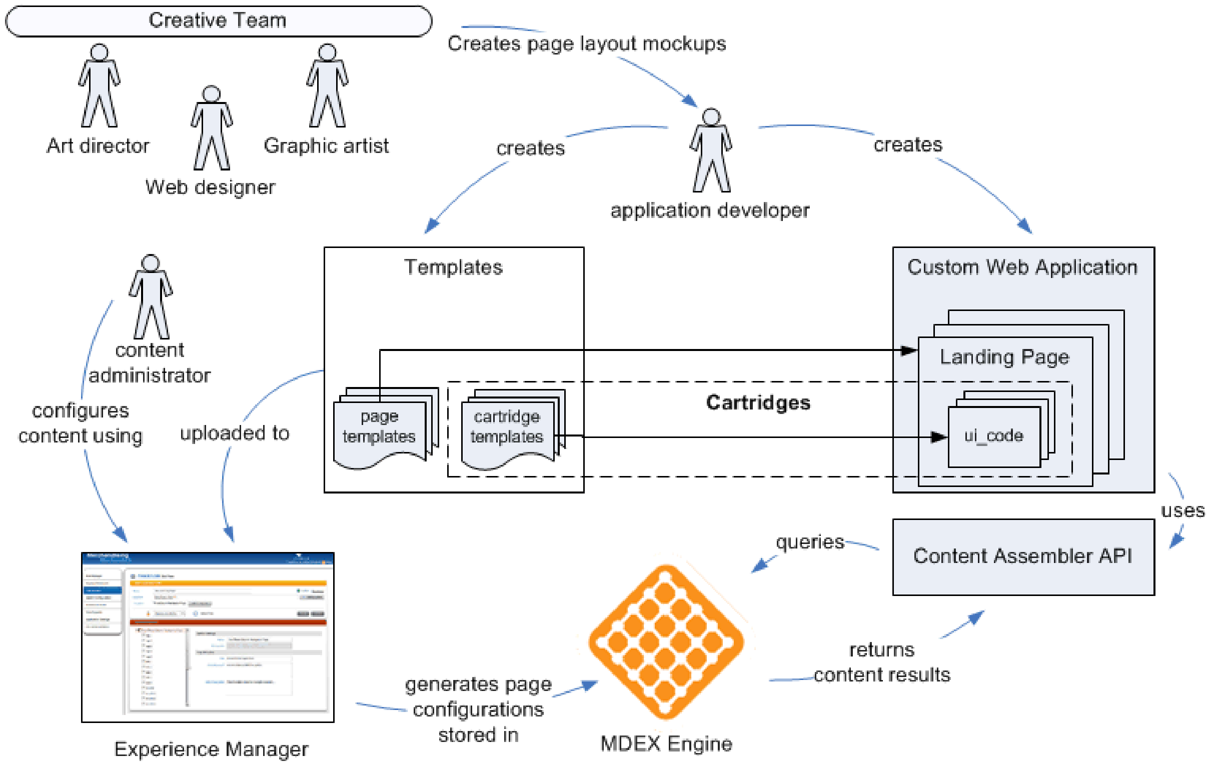
Experience Manager allows the content administrator to save progress incrementally and also to preview dynamic pages to make sure that the application renders the content as desired.

Once a page configuration is complete, the content administrator can request that a manager activate the page. Once the manager activates the page in Experience Manager, it is published and accessible to the end users of the Web site. The workflow model within Experience Manager also allows content administrators to make dramatic changes to pages in the tool and request re-activation from their manager without the need for any changes in the application code.

Experience Manager and Content Assembler API architecture

Experience Manager and the Content Assembler API combine to enable the creation and display of dynamic pages.

This diagram shows the life cycle of a dynamic page that is created using Experience Manager and rendered by an application built with the Content Assembler API:



The application developer creates cartridges based on designs from the creative team and incorporates the cartridge code into a Web application.

Based on the templates that have been uploaded to Experience Manager, the content administrator configures specific pages and sets them to display based on a set of criteria such as navigation state, user profile, and date range. Experience Manager outputs these page configurations as XML documents that are stored in the MDEX Engine.

As users search or navigate within the site, the application queries the MDEX Engine using the Content Assembler API, retrieves the dynamic page content that applies at the appropriate navigation state, and renders the content.

Chapter 2

Working with Templates for Dynamic Pages

This section describes the process of creating templates that are used to drive dynamic pages.

Template prerequisites

Experience Manager leverages functionality from dynamic business rules such as triggers, priority, and workflow to manage dynamic pages. Because dynamic pages are stored as dynamic business rules in the MDEX Engine, some of the same supporting configuration is required for pages as for rules.

Before you create templates for use in Experience Manager, you must have the following in place for the dynamic pages in your application:

- One or more rule groups (the "default" rule group is automatically created)
- One or more rule zones
- A rule style

You specify the zone and style for a page in the top-level page templates that you create.

About dynamic pages and rule groups

Dynamic pages use the same group mechanism as dynamic business rules.

You can use one rule group for all your dynamic pages, or you can use multiple groups to organize the pages in your application, for example, into an Electronics group and a Jewelry group. Multiple groups also allow you to manage permissions independently for each rule group.

If you are using both traditional dynamic business rules and dynamic pages in your application, create rule groups for use with dynamic pages that are distinct from those used for dynamic business rules. For example, if you group your rules and pages by category, you can have separate rule groups for "Sports rules" and "Sports pages." For details about creating rule groups, see the *Oracle Endeca Developer Studio Help*.

The Group List pages of both the Rule Manager and Experience Manager display all groups that a user has permission to view, regardless of whether they are used for dynamic business rules or dynamic pages. Within a rule group, only rules that represent dynamic pages display in Experience Manager. Both dynamic business rules and dynamic pages display in the Rule Manager, but dynamic pages are read-only in the Rule Manager for all users regardless of rule group permissions. Users who have the pages role should have permissions to access groups that you have set up for use with dynamic pages, and users with the rules role should have permissions for groups that are used only for dynamic business rules.

Workflow, resource locks, and priority for groups with dynamic pages function exactly as they do for dynamic business rules. For more information about rule groups, see the *Endeca Advanced Development Guide*.

About using zones with dynamic pages

Zones enable the display of dynamic pages in the application. While a single zone is usually sufficient, multiple zones can enable finer-grained control over the display of dynamic pages.

Unlike dynamic business rules, which generally control only one aspect of a page that is divided up into zones, dynamic pages drive the presentation of the page as a whole. In the context of dynamic pages, *sections* represent the parts of a page and *zones* enable you to provide different perspectives on the same page.

For example, if you want to present a tabbed pane that displays either product details or user reviews of the same product (or summaries of different information for products based on the same navigation state), you can use separate zones for each view. In this example, you would create a ProductDetails zone and a UserReviews zone, with associated templates for each zone.

Related Links

Creating a zone for dynamic pages on page 16
Dynamic pages must be assigned a rule zone in order to display. You should create at least one zone in your application for use exclusively with dynamic pages.

Creating a zone for dynamic pages

Dynamic pages must be assigned a rule zone in order to display. You should create at least one zone in your application for use exclusively with dynamic pages.

Although the procedure to create a zone for dynamic pages is the same as for dynamic business rules, there are certain properties you should set for dynamic page zones.

To create a zone for use with dynamic pages:

1. In the **Project Explorer** of Developer Studio, expand **Dynamic Business Rules**.
2. Double-click **Zones** to open the **Zones** view.
3. Click **New** to open the **New Zone** editor.
4. In the **Name** field, provide a unique name for the zone.
5. In the **Rule Limit** field, enter 1.
6. Select **Valid for search**.

   Leave **Shuffle rules** and **Unique by this dimension/property** unselected.

Note: If you have both dynamic pages and business rules in your application, be aware that conflicts may occur if a zone being used for dynamic pages is assigned to a business rule. If a rule and a page within the same zone have overlapping triggers and the rule has higher priority, the page may never display. In order to avoid this situation, assign names to the zones being used for pages that clearly indicate that such zones are to be used for dynamic pages only so that users do not assign them to dynamic business rules in the Rule Manager.

Creating a style for dynamic pages

Before you create dynamic pages, you must have at least one style defined in your application.

Oracle recommends that you create one style exclusively for use with dynamic pages. The style that you assign to a dynamic page does not affect how it displays; it is only required to make the rule that contains the page configuration valid.

Although the procedure to create a style for dynamic pages is the same as for dynamic business rules, there are certain properties you should set for dynamic page styles.

To create a style for use with dynamic pages:

1. In the **Project Explorer** of Developer Studio, expand **Dynamic Business Rules**.
2. Double-click **Styles** to open the **Styles** view.
3. Click **New** to open the **New Style** editor.
4. In the **Name** field, provide a unique name for the style.

   The style **Title** is optional, and does not display in Experience Manager.
5. In the **Records** area:
   a) For the **Minimum** value, enter `0`.
   b) For the **Maximum** value, enter `1`.

About maintaining consistent dimension value IDs

Because the trigger locations for landing pages and the dynamic queries for record list cartridges are both stored based on the ID of the selected dimension value, it is important that these IDs remain stable.

If dimension value IDs change after landing pages have already been created based on the old IDs, then the following may occur:

- IDs referenced in triggers and dynamic record lists for existing pages may be reassigned to a different dimension value. In this case, the affected pages behave as if they were configured to trigger or return results based on the new dimension value rather than the one designated by the content administrator.
- If a landing page has a trigger with an invalid dimension value ID (that is, one that no longer corresponds to a dimension value), then content administrators cannot save any changes that they make to the landing page by clicking **OK** in the **Edit View**. This situation may occur if the selected dimension value is deleted or the original dimension value is assigned a new ID and no other dimension value is assigned the old ID.
- If any landing page in a landing page group has a trigger with an invalid dimension value ID, content administrators cannot save changes they make to any page in that group by clicking **Save All Changes** in the **List View**.
- A page with an invalid ID in the trigger never fires because the trigger criteria cannot be satisfied. A page with an invalid ID specified for a dynamic record list cartridge returns zero records for that cartridge.

The Endeca ITL process generally ensures that dimension value IDs remain consistent across updates. However, if your application uses externally managed taxonomies, dimension value IDs may change if you make changes to the dimension hierarchy. In order to avoid the problems caused by invalid and inconsistent dimension value IDs, follow the recommendations in "Node ID requirements and identifier management in Forge" in the *Endeca Platform Services Forge Guide*.

About creating templates

Templates are XML documents that define the content structure of a dynamic page or part of a page and enable content administrators to specify page content in Experience Manager.

Top-level templates, which define an entire page, and cartridge templates, which drive the content of sections, share the same structure and are defined by the same schema.

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. For top-level page templates, this part of the template also allows you to specify a zone and style, which the tool assigns to any pages that are created from that template.
- **Property definitions.** In this part of the template, you explicitly declare all the properties of the `ContentItem` object that a template represents. Some properties also allow you to specify default values.
- **Property editors.** These allow you to specify whether a property can be configured and some attributes of the editing interface in Experience Manager.

Properties may include simple string properties, record lists, or template sections. Most properties are configurable and enable content administrators to define the behavior of pages that they build within the tool. However, properties can also be used to pass information directly to the front-end application, for example details about how to render the content within a template. By defining the properties in the template along with how they can be configured in the tool, you ensure that the content items returned by the Content Assembler API can be properly handled by the presentation logic in your application.

In general, when creating page templates, you have a page layout provided by your creative team. Working from a sample design or mockup, identify the high-level structure of the page -- this structure informs the sections you define in your page template. Recall that the structure of each section is in turn driven by a cartridge template, so if one portion of your page can contain either a large banner image or a three-column content area, you can implement this as one page template with a section that allows two different cartridge templates, rather than two different top-level templates.

Then, for each section, identify the information that your front-end application uses to render the content in that section. This information is then modeled in the cartridge template as properties that the content administrator can configure.

While most template properties and sections affect 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 include embedded code that does not render in the page but enables functionality such as Web analytics reporting. Sections can also represent content that may be rendered in a number of ways (for example, a page element that may display in the left or right column of a page depending on context).

About template validation

Templates are validated against the Experience Manager template schemas when you upload them to Experience Manager.

Before you upload your templates to Experience Manager, ensure that the templates validate against the template schema. All templates must include the following schema declaration:

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

Although the `<RuleInfo>` element is not required by the schema, it is required by Experience Manager for all top-level page templates. If the `<RuleInfo>` element is missing in a page template, or if a zone or style specified in the element attributes does not exist in the application's instance configuration, the template is not available in Experience Manager and an error is written to `webstudio.log`.

A copy of the template schema (`content-template.xsd`) and associated content schemas (`content.xsd` and `content-tags.xsd`) is located for your reference in `\doc\schemas` (on Windows) or `/doc/schemas` (on UNIX) in your Content Assembler API installation. In general, the template schema describes the format of page and cartridge templates, while the content schemas describe the format of the page configurations created in Experience Manager, although the content-tags schema is also used for some types of template configuration.

Related Links

Troubleshooting problems with uploading templates on page 59

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

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. Top-level, or *page templates*, describe the structure of an entire Web page. These templates can include sections, which are placeholders for content driven by templates known as *cartridge templates*. Cartridge templates can in turn include sections within them to allow for further nested content.

Page templates are identified by a special `type` string. Any template designed to be a top-level template must be of type `PageTemplate`.

Cartridge templates can be of any type you specify. This allows you to constrain the cartridges that can be inserted in a particular section. For example, if you have a page or cartridge 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 do not display in Experience Manager. The value should be as descriptive as possible to help the user select the appropriate template, for instance, "ThreeColumnWithLargeBanner" or "HolidaySalePromotion."

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

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  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 a hyphen (`-`)

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

Specifying the zone and style for a template

Page templates are required to specify a rule zone and a style. When a page is created in Experience Manager, the zone and style are applied to any pages based on that template.

Zones and styles are only used for page templates, not cartridge templates.

To specify the zone and style for a page template:

Insert a `<RuleInfo>` element immediately after the opening `<ContentTemplate>` tag as in the following example:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  type="PageTemplate" id="ThreeColumnNavigationPage">
  <RuleInfo zone="NavigationPageZone" style="PageStyle"/>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

The value of the `zone` attribute must be the exact name of the zone that is defined in your application's instance configuration and that you want to apply to all pages created with this template.

The value of the `style` attribute is the exact name of any style that is defined in your application's instance configuration. Oracle recommends that you create one style exclusively for use with all dynamic pages. Styles are required to make pages valid, but do not affect their display.

Specifying custom rule properties for a template

You can optionally specify custom rule properties in any page template. When a page is created in Experience Manager, the rule properties are applied to any pages based on that template.

Before specifying custom properties, you must already have inserted a `<RuleInfo>` element in your template.

Rule properties are key-value pairs that are passed back to the application along with query results. They allow you to associate supplementary information with a landing page or to exclude certain landing pages from consideration by the MDEX Engine.

Custom rule properties are only used for page templates, not cartridge templates.

To specify custom rule properties for a page template:

1. Insert a `<Property>` element within `<RuleInfo>` as in the following example:

   ```
   <ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
     type="PageTemplate" id="ThreeColumnNavigationPage">
     <RuleInfo zone="NavigationPageZone" style="PageStyle">
       <Property name="my.property.name" value="My property value"/>
     </RuleInfo>
     <!-- additional elements deleted from this example -->
   </ContentTemplate>
   ```

   The value of the `name` attribute can be any string you choose to use for a property key. The `name` cannot begin with the string `endeca.internal.` (this prefix is reserved for use by Endeca components only).

   The value of the `value` attribute can be any string.

2. Insert additional `<Property>` elements to specify additional rule properties for pages based on this template.

For more information about using rule properties with landing pages, see the *Content Assembler API and Reference Application Guide*.

About using thumbnail images in Experience Manager

You can specify thumbnail images for page templates and section templates that display along with the template description in the template selector and cartridge selector dialog boxes in Experience Manager. These images can help the content administrator identify the appropriate template to use for the pages they create.

The Content Assembler sample application includes a Web application for hosting the thumbnail images for the sample templates in the Endeca Tools Service. You can add your own thumbnail images to this application, or the images may be hosted on a separate Web server from your Experience Manager instance. If the thumbnail image for a template is either not specified or not accessible, a default image displays in the dialog box.

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.

Related Links

File hosting and security considerations on page 62
Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

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 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").
`<ThumbnailUrl>`	The absolute URL to a thumbnail image that shows a sample page or section that is based on the template. The images are 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"
  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 saving templates

Templates are saved as XML files that are then uploaded to the 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, whether it is a page template or a cartridge template, should be in a separate file.
- Name each template file using the following format: `TemplateType-TemplateID.xml`. For example, `PageTemplate-ThreeColumnNavigationPage.xml` or `HorizontalBanner-ImageMap.xml`

Note: Template file names cannot have spaces in them.

Endeca also recommends that you treat page and cartridge 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 defining content properties

When you create a template, you specify all the properties that are necessary to render a page or section. These properties are returned as part of the content item object in the Content Assembler API.

You define properties within the `<ContentItem>` element in the template. Each `<ContentItem>` must have a `<Name>` property. In addition, you can define any number of properties for use by your front-end application. For each property, you specify a name and a property type. In some cases, you can optionally specify a default value for the property.

Properties can be associated with editing interfaces that enable configuration within Experience Manager. Content properties may include text, image URLs, or records that the content administrator can specify. One type of property is a section, which allows content administrators to insert a cartridge to drive the content of a specific part of a page.

You can choose not to expose a particular property in Experience Manager and simply pass its value to your front-end application. Examples of this usage can include a reference to the cartridge code that should be used to render the template content, or queries to the MDEX Engine that are hidden from the content administrator in the tool.

Related Links

About defining the editing interface for properties on page 34
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 the default name for a ContentItem

`Name` is a required property on a `ContentItem`. Generally the content administrator provides a value for it in Experience Manager, but you can specify a default name as a placeholder.

To specify a default name for a `ContentItem`:

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

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  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 page</Name>
    <!-- additional elements deleted from this example -->
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

`<Name>` is a required element, but you do not need to specify a value for the name. If you insert an empty `<Name/>` element, an empty text field displays in Experience Manager and the content administrator supplies a value.

About content properties

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

Each property must have a name that is unique within the template. This is the key by which your application can access that property through the Content Assembler API. The 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.

The child elements of `<Property>` allow you to specify the type of property. The template schema provides several basic property types, such as `<String>`, `<Boolean>`, and `<RecordList>`. Additional property types including `<NavigationRefinements>` and `<NavigationRecords>` are defined in the content-tags schema.

The `<ContentItem>` element within `<Property>` allows you to define a section property. As the template structure suggests, a section is in essence a placeholder for a nested content item defined by a separate cartridge template. (Recall that each template, whether it is a page template or cartridge template, defines a corresponding content item.)

In addition, the `<Property>` element can also contain content pass-through elements that cannot be configured in Experience Manager or arbitrary XML that is passed directly to your front-end application.

The following example shows the properties of a page template that defines several cartridge sections.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <Property name="autogen_meta">
      <Boolean>true</Boolean>
    </Property>
    <Property name="title">
      <String>Endeca Content Assembler Reference Application</String>
    </Property>
    <Property name="meta_keywords">
      <String>content assembler reference application</String>
    </Property>
    <Property name="meta_description">
      <String>Endeca Content Assembler reference application.</String>
    </Property>
    <Property name="Header">
      <ContentItemList type="FullWidthContent"/>
    </Property>
    <Property name="LeftColumn">
      <ContentItemList type="SidebarItem" />
    </Property>
    <Property name="CenterColumn">
      <ContentItemList type="MainColumnContent" />
    </Property>
    <Property name="RightColumn">
      <ContentItemList type="SidebarItem" />
    </Property>
    <Property name="Footer">
      <ContentItemList type="FullWidthContent"/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

Each of the `<ContentItemList>` sections accept cartridges. For example, the properties of the left column section may look similar to the following:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="SidebarItem" id="ThreeRecordBox">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three Record Spotlight Box</Name>
    <Property name="title">
      <String>Featured Items</String>
    </Property>
    <Property name="record_list">
      <RecordList/>
    </Property>
    <Property name="link_text">
      <String/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

Types of property elements

Each property type corresponds with a particular object type that is returned by the Content Assembler API.

Configurable content properties

These property types can be associated with property editors to enable configuration by the content administrator in Experience Manager.

Template element	Object type returned by API (Java)	Object type returned by API (RAD Toolkit for ASP.NET)
`<Boolean>`	`java.lang.Boolean`	`bool`
`<ContentItem>`	`com.endeca.content.Con¬tentItem`	`Endeca.Data.Content.ICon¬tentItem`
`<ContentItemList>`	`com.endeca.content.Con¬tentItemList`	`Endeca.Data.Content.ICon¬tentItemList`
`<NavigationRecords>`	`com.endeca.con¬tent.ene.Navigation¬Records`	`Endeca.Data.Content.Naviga¬tion.INavigationRecords`
`<NavigationRefine¬ments>`	`com.endeca.naviga¬tion.DimensionList`	`Endeca.Data.DimensionStatesRe¬sult`
`<RecordList>`	`com.endeca.naviga¬tion.ERecList`	`System.Collections.ObjectMod¬el.ReadOnlyCollection<Record>`
`<String>`	`java.lang.String`	`string`

Pass-through content properties

These properties cannot be exposed for configuration in the Experience Manager tool. They allow you to embed MDEX Engine query results in the content item object that your application accesses through the Content Assembler API.

Template element	Object type returned by API (Java)	Object type returned by API (RAD Toolkit for ASP.NET)
`<NavigationResult>`	`com.endeca.naviga¬tion.ENEQueryResults`	`Endeca.Data.NavigationResult`
`<NavQuery>`	`com.endeca.naviga¬tion.ERecList`	`System.Collections.ObjectMod¬el.ReadOnlyCollection<Record>`
`<RecordQuery>`	`com.endeca.naviga¬tion.ERecList`	`System.Collections.ObjectMod¬el.ReadOnlyCollection<Record>`
`<Supplement>`	`com.endeca.naviga¬tion.Supplement`	`Endeca.Data.BusinessRule`
`<UrlEneQuery>`	`com.endeca.naviga¬tion.ERecList`	`System.Collections.ObjectMod¬el.ReadOnlyCollection<Record>`

Custom property elements

You can insert your own arbitrary XML within a `<Property>` element. In this case, the API returns the contained element directly as an `org.w3c.dom.Element` (in Java) or a `string` (for the RAD Toolkit for ASP.NET).

Related Links

About passing arbitrary XML to the front-end application on page 52
You can nest arbitrary XML in templates within a `<Property>` element.

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, you can 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"
type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <Property name="title">
      <String>Endeca Content Assembler Reference Application</String>
    </Property>
    <Property name="meta_keywords">
      </String>
    </Property>
    <Property name="meta_description">
      </String>
    </Property>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

Related Links

Adding a string editor on page 35
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.

Adding an image preview on page 40
An image preview displays an image in the Experience Manager interface based on a URL.

Adding a combo box editor on page 37
Predefining values through a combo box editor allows you more control over page content and provides a more efficient mechanism for populating templates and cartridges in the editing interface.

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 examples show different uses of Boolean properties:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- additional elements deleted from this example -->
    <Name>New Three-Column Navigation Page</Name>
    <Property name="autogen_meta">
      <Boolean>true</Boolean>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="MainColumnContent" id="DimensionSearchResults">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Dimension Search Results</Name>
    <Property name="display_compound_dimensions">
      <Boolean>true</Boolean>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

Related Links

Adding a checkbox editor on page 42
Checkbox editors provide Experience Manager users with a quick and efficient way to specify the value of a Boolean property.

Adding a navigation refinements property

When you add a navigation refinements property, the content administrator can use the default dimension order returned by the Endeca MDEX Engine or specify a custom default dimension order.

To add a navigation refinements property to a template:

1. Insert a `<NavigationRefinements>` element inside a `<Property>` element.
2. Specify the content-tags schema `xmlns` attribute for `<NavigationRefinements>`.
   For example: `<NavigationRefinements xmlns="http://endeca.com/schema/content-tags/2008" />`

3. Specify a default ordering:
   - To accept the default dimension order returned by the Endeca MDEX Engine, leave the `<Nav¬igationRefinements>` element empty.
   - To create a custom dimension list, insert a `<DimensionList>` element.
4. Optionally, insert zero or more `<Dimension>` elements inside the `<DimensionList>` element with the dimension ID set as the `id` attribute.
   For example: `<Dimension id="123"/>`.

   Note: You can leave the `<DimensionList>` element empty to create an empty dimension list.

The following example shows the definition of a navigation refinement property using the MDEX Engine default dimension order:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="Cartridge25PercentWidth" id="GuidedNavigation">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Guided Navigation</Name>
    <Property name="refinements">
      <NavigationRefinements xmlns="http://endeca.com/schema/content-
tags/2008" />
    </Property>
  </ContentItem>
<!-- additional elements deleted from this example -->
</ContentTemplate>
```

The following example shows the definition of a navigation refinement property with an empty dimension list. In this case, the Dimension Selector dialog box in Experience Manager displays with an unpopulated Selected Dimensions list.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="Cartridge25PercentWidth" id="GuidedNavigation">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Guided Navigation</Name>
    <Property name="refinements">
      <NavigationRefinements xmlns="http://endeca.com/schema/content-
tags/2008" >
        <DimensionList/>
      </NavigationRefinements>
    </Property>
  </ContentItem>
<!-- additional elements deleted from this example -->
</ContentTemplate>
```

The following example shows the definition of a navigation refinement property with a custom default dimension order:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="Cartridge25PercentWidth" id="GuidedNavigation">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Guided Navigation</Name>
    <Property name="refinements">
      <NavigationRefinements xmlns="http://endeca.com/schema/content-
tags/2008" >
        <DimensionList>
          <Dimension id="146"/>
          <Dimension id="212"/>
          <Dimension id="123"/>
        </DimensionList>
      </NavigationRefinements>
    </Property>
  </ContentItem>
<!-- additional elements deleted from this example -->
</ContentTemplate>
```

Related Links

Adding a navigation refinements selector on page 43

You add a navigation refinements selector to enable an interface in Experience Manager for content administrators to choose and order specific dimensions to display for a landing page.

About record properties

There are two types of content properties that enable a content administrator to customize the display of records on a page: navigation records and record lists.

Navigation records represent the main results of a search or navigation query based on an end user's navigation state. Content administrators can specify configuration options for navigation records such as sort, relevance ranking, and number of records to display per page.

Record lists represent supplemental queries, for example, to populate promotions or Content Spotlighting cartridges. Content administrators can designate either specific records or a navigation query that returns records for spotlighting.

Adding a navigation records property

When you add a navigation records property, a content administrator can configure the display of the main results of an end user's query.

To add a navigation records property to a template:

1. Insert a `<NavigationRecords>` element inside a `<Property>` element.
2. Specify the content-tags schema `xmlns` attribute for `<NavigationRecords>`.
   For example: `<NavigationRecords xmlns="http://endeca.com/schema/content-tags/2008" />`
3. Optionally, you can specify default sort, relevance ranking, and records-per-page behavior.
4. To specify a default sort, insert a `<Sort>` element inside `<NavigationRecords>`. For example:

   ```
   <Sort>P_Price</Sort>
   ```

   The content of the element is the name of the property by which to sort. The `<Sort>` element takes the following optional attributes:

Attribute	Description
`ascending`	A Boolean value specifying whether the records should be sorted in ascending order. Default is `true`.

Attribute	Description
`latitude`	The latitude value of a geocode sort key for geospatial sorting.
`longitude`	The longitude value of a geocode sort key for geospatial sorting.

If you specify more than one default `<Sort>`, the sorts are applied in order. If no sort order is explicitly specified for this property, the default sorting behavior from the MDEX Engine is used.

5. To specify a default relevance ranking strategy, insert a `<RelevanceRanking>` element inside `<NavigationRecords>`. For example:

   ```
   <RelevanceRanking>freq</RelevanceRanking>
   ```

   The content of the element is an MDEX Engine URL relevance ranking strategy string. For details about the format of these strings, see "Controlling relevance ranking at the query level" in the *Endeca MDEX Engine Advanced Development Guide*. If no relevance ranking strategy is explicitly specified for this property, the default relevance ranking behavior from the MDEX Engine is used.

6. To specify a default number of records to return per page, specify a value for the optional `recordsPerPage` attribute. For example:

   ```
   <NavigationRecords recordsPerPage="12"/>
   ```

   The value of `recordsPerPage` should be an integer greater than or equal to `1`. If the value is less than `1` or if it is outside the valid range specified by the navigation records editor, the value is discarded and the field is left empty in Experience Manager by default. If no value for records per page is explicitly specified for this property, the default is `10`.

The following example shows the definition of a navigation records property with no defaults defined:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="MainColumnContent" id="ResultsList">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Results List</Name>
    <Property name="navigation_records">
     <NavigationRecords xmlns="http://endeca.com/schema/content-tags/2008"
 />
    </Property>
  </ContentItem>
<!-- additional elements deleted from this example -->
</ContentTemplate>
```

The following example shows the definition of a navigation records property with defaults for sort, relevance ranking, and records per page:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="MainColumnContent" id="ResultsGrid">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Results Grid</Name>
    <Property name="navigation_records_with_defaults">
      <NavigationRecords recordsPerPage="12"
        xmlns="http://endeca.com/schema/content-tags/2008">
        <Sort ascending="false">P_Score</Sort>
       <RelevanceRanking>static(P_SalesRank,descend),exact</RelevanceRank¬
ing>
      <NavigationRecords>
```

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

Note: If you specify a default sort or relevance ranking strategy, ensure that the option is also available in the associated navigation records editor. If the sort or relevance ranking strategy is not specified in the editor, Experience Manager discards the default property value and defaults to an empty `<Sort>` or `<RelevanceRanking>` element, representing the default MDEX Engine behavior.

Related Links

About record properties on page 29
There are two types of content properties that enable a content administrator to customize the display of records on a page: navigation records and record lists.

Adding a navigation records editor on page 44
You add a navigation records editor to enable an interface in Experience Manager to configure the display of record results on a landing page.

Adding a record list property

A record list property can contain one or more Endeca records for merchandising or Content Spotlighting.

To add a record list property to a template:

Insert a `<RecordList>` element inside a `<Property>` element.

Note: Although you cannot specify a default value for the `<RecordList>` element, you can specify default records or queries using pass-through content elements.

The following example shows the definition of a record list property:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="SidebarItem" id="ThreeRecordBox">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three Record Spotlight Box</Name>
    <!-- additional properties deleted from this example -->
    <Property name="record_list">
      <RecordList/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentTemplate>
```

You can configure behavior such as the maximum number of records that can be returned or whether record filters are applied to this property when you add the record selector to the template.

Related Links

About record properties on page 29

There are two types of content properties that enable a content administrator to customize the display of records on a page: navigation records and record lists.

Adding a record selector on page 47

You add a record selector to enable an interface in Experience Manager to specify featured records or queries for a record list property.

Adding a content item property

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

Recall that each template corresponds to a content item object, so a cartridge template is returned by the Content Assembler API as a nested content item. 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 goes to choose the cartridge to insert 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 to template types 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 front-end application expects this kind of content.

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

Related Links

About cartridge selectors on page 49

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

Adding a content item list property on page 33

A content item list allows Experience Manager users 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.

Adding a content item list property

A content item list allows Experience Manager users 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 must contain four sections:

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

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"
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 administrator goes to choose a cartridge to insert in a `banners` section, only templates of type `BuyingGuideSection` display in the **Select Cartridge** dialog box.

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

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

   By default `maxContentItems="0"`, which means that there is no limit to the number of content items in a content item list.

Related Links

Adding a content item property on page 32
A content item property defines a template section by creating a placeholder for a nested content item.

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.

The template schema provides elements that define editors for string and record list properties. For example, this excerpt from a sample template defines a configurable string property named `title`:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <Property name="title">
      <String/>
    </Property>
    <!-- additional properties deleted from this example -->
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <StringEditor propertyName="title" label="Title"/>
    </BasicContentItemEditor>
    <!-- additional editors deleted from this example -->
  </EditorPanel>
</ContentTemplate>
```

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, a `<StringEditor>` cannot be associated with a `<RecordList>` property. If you define a content editor for a property that does not exist, or that is of the wrong type, an 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 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 for primitive property types (such as String and Boolean). 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. For complex Endeca property types such as navigation refinements, navigation records, and record list, you can associate only one editor with each complex property in the template.

Related Links

About defining content properties on page 22

When you create a template, you specify all the properties that are necessary to render a page or section. These properties are returned as part of the content item object in the Content Assembler API.

Adding a group label on page 49

In the Experience Manager interface, group labels can serve as a visual cue that several properties are related.

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 combo box.

To add a string editor to a template:

1. Insert a `<StringEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the string 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.
`label`	This attribute allows you to specify a more descriptive label for this field in Experience Manager. If no label is specified, the property name is used by default.
`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 string property. Properties are enabled by default.
`wordWrap`	If this attribute is set to `true`, word wrap is enabled for long strings in the Experience Manager text field. Word wrap is enabled by default.
`width`	The width in pixels of the text field presented in the Experience Manager interface. The default width is 300 pixels.
`height`	The height in pixels of the text field presented in the the Experience Manager interface. The default height is 24 pixels. Set this higher to enable a multiline text area.

Note: If the expected value of a string editor is an image URL, you may want to add an image preview to display that image within Experience Manager.

The following example shows a variety of editing options for string properties:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="FullWidthContent" id="ImageSiteBanner">
  <!-- additional elements deleted from this example -->
  <!-- First define all the content properties -->
  <ContentItem>
    <Name>New Site Banner</Name>
    <Property name="image_src">
      <String>/images/WineDemoImages/site-banner-barrels.jpg</String>
    </Property>
    <Property name="image_href">
      <String/>
    </Property>
    <Property name="image_alt">
      <String/>
    </Property>
  </ContentItem>
  <!--  Define editors for properties that should be configurable -->
  <EditorPanel>
    <BasicContentItemEditor>
      <!--
      # This example allows the content administrator to
      # specify an image URL.
      # A default value was provided above as a placeholder,
      # and it is editable in Experience Manager.
      -->
      <StringEditor propertyName="image_src" label="Image name" en¬
abled="true"/>

      <!--
      # The image_href property allows a content administrator
      # to specify an image link URL.
      # This property is editable, but there is no default
      # value.
      -->
      <StringEditor propertyName="image_href" label="Image link URL" en¬
abled="true"/>

      <!--
      # This example allows the content administrator to
      # specify an alternative image text.
      -->
      <StringEditor propertyName="image_alt" label="Image alt text" en¬
abled="true"/>

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

Note: Neither Experience Manager nor the Content Assembler API 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.

Related Links

Adding a string property on page 26

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.

Adding an image preview on page 40
An image preview displays an image in the Experience Manager interface based on a URL.

Adding a combo box editor on page 37
Predefining values through a combo box editor allows you more control over page content and provides a more efficient mechanism for populating templates and cartridges in the editing interface.

Adding a combo box editor

Predefining values through a combo box editor allows you more control over page content and provides a more efficient mechanism for populating templates and cartridges in the editing interface.

Combo box editors affect the value of a string property. For example, you might use a combo box to provide image options:

```
     <ComboBox propertyName="image_src" label="Image name" usePrompt="false"
 editable="true" >
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-empty-
glass.jpg" label="Empty glass with strawberries"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-green-
grapes.jpg" label="Green grapes"/>
        <ComboBoxItem data="/images/WineDemoImages/one-record-banner-wine-
being-poured.jpg" label="Rose wine being poured"/>
      </ComboBox>
```

To add a combo box editor:

1. Insert a `<ComboBox>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the combo box 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 combo box editor.
`editable`	If set to `true`, this attribute allows Experience Manager users to specify custom string values. By default, combo boxes are not editable.
`enabled`	If set to `false`, the combo box displays in Experience Manager but the value cannot be changed by the user. By default, combo boxes are enabled.
`label`	This attribute allows you to specify a more descriptive label for this field in Experience Manager. If no label is specified, the property name is used by default.
`usePrompt`	If `usePrompt` is set to `false`, the first `ComboBoxItem` in the list is used as the displayed value. If the first `ComboBoxItem` is not set as the default property value, it is not saved if the content administrator exits the **Edit View** without selecting a value.

Attribute	Description
	By default, this attribute is set to `true`, and a prompt displays at the top of the combo box editor.
`prompt`	Specifies a custom prompt. The default prompt is "Select one..."
`rows`	Specifies the maximum number of rows appearing in a drop down list before a scroll bar displays. The default value is `5`.
`maxWidth`	Controls with width in pixels of the editor. The default value is `300`.

3. Specify one or more menu options for the combo box by adding `<ComboBoxItem>` elements.
   `<ComboBoxItem>` takes the following attributes:

Attribute	Description
`data`	Required. The string value to assign to the associated property if this `<ComboBox¬Item>` 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 for `data` is used by default. You must either specify a `label` for all of the combo box items or none of them. You cannot have labels for some items and not others.

Note: If you choose to make a combo box editable, you should not use the `label` attribute for combo box items. Instead, the combo box 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 image of wine being poured (`/images/WineDemoImages/one-record-banner-wine-being-poured.jpg`) as the default choice for a combo box with `propertyName="image_src"`:

```
<Property name="image_src">
 <String>/images/WineDemoImages/one-record-banner-wine-being-
poured.jpg</String>
</Property>
```

Note: Ensure that the default value for the property is one of the options defined for the combo box in a `<ComboBoxItem>` element.

The following example illustrates a combo box with a custom prompt ("Please select an image"):

```
<EditorPanel>
    <BasicContentItemEditor>
<!-- additional elements deleted from this example -->
     <ComboBox propertyName="image_src" label="Image name" prompt="Please
 select an image">
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-empty-
```

```
glass.jpg" label="Empty glass with strawberries"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-green-
grapes.jpg" label="Green grapes"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-wine-
being-poured.jpg" label="Rose wine being poured"/>
      </ComboBox>
    </BasicContentItemEditor>
  </EditorPanel>
```

The following example depicts a combo box configured as "editable." The combo box displays the image options specified in the `<ComboBoxItem>` elements, but also allows a Experience Manager user to enter an image URL.

```
<EditorPanel>
    <BasicContentItemEditor>
<!-- additional elements deleted from this example -->
      <ComboBox propertyName="image_src" label="Image name" use¬
Prompt="false" editable="true">
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-empty-
glass.jpg" label="Empty glass with strawberries"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-green-
grapes.jpg" label="Green grapes"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-wine-
being-poured.jpg" label="Rose wine being poured"/>
      </ComboBox>
    </BasicContentItemEditor>
  </EditorPanel>
```

The following example shows a combo box configured with a default value. The selected value when the editor is first instantiated is `/images/WineDemoImages/one-record-banner-green-grapes.jpg`, which displays with the label "Green grapes" in the drop-down menu. Experience Manager users can choose to select a different image.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="MainColumnContent" id="OneRecordBanner">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New One Record Banner</Name>
    <!-- additional properties deleted from this example -->
    <Property name="image_src">
      <String>/images/WineDemoImages/one-record-banner-green-
grapes.jpg<String/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
  <EditorPanel>
    <BasicContentItemEditor>
<!-- additional elements deleted from this example -->
      <ComboBox propertyName="image_src" label="Image name" >
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-empty-
glass.jpg" label="Empty glass with strawberries"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-green-
grapes.jpg" label="Green grapes"/>
       <ComboBoxItem data="/images/WineDemoImages/one-record-banner-wine-
being-poured.jpg" label="Rose wine being poured"/>
      </ComboBox>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Related Links

Adding a string property on page 26
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.

Adding a string editor on page 35
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.

Adding an image preview on page 40
An image preview displays an image in the Experience Manager interface based on a URL.

Adding an image preview

An image preview displays an image in the Experience Manager interface based on a URL.

You can construct an image preview URL from a hard-coded value, or from any number of string properties.

Image preview supports JPEG, GIF, and PNG image formats. Note that if the images are hosted on a different server from Workbench, you may have to enable the Flash player to access those resources.

To add an image preview to a template:

1. Insert an `<ImagePreview>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the image preview:

Attribute	Description
`urlExpression`	Required. The source of the image URL. You can construct `urlExpres¬sion` from any number of string properties, or you can enter a static value.
`label`	A description for the image preview. There is no default value.
`maxHeight`	The height in pixels of the image preview presented in the Experience Manager interface. The default value is `100`.
`maxWidth`	The width in pixels of the image preview presented in the Experience Manager interface. The default value is `300`.
`displayUrl`	A boolean indicating whether to display the processed URL. The default value is `true`.

If you are using more than one string property to compose the URL, you may want to use a group label to indicate to Experience Manager users that these properties are related.

The following examples show options for constructing an image preview.

In the first example, an `image_src` property is created:

```
    <Property name="image_src">
      <String/>
    </Property>
```

Then, in the `<EditorPanel>`, `urlExpression` takes the value of the `image_src` property, specified in a `<StringEditor>` above the `<ImagePreview>` element:

```
    <StringEditor propertyName="image_src" label="Image name" en¬
abled="true"/>
```

```
    <ImagePreview
      urlExpression="http://localhost:8006/ContentAssemblerRefApp/{im¬
age_src}"
      label="Site banner image"
      maxWidth="200"
      maxHeight="100" />
```

In the second example, `urlExpression` parses some of the image URL from string properties determined by values selected in combo boxes above the `<ImagePreview>` element:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="ContentCartridge" id="ThreeRecordBanner-Advanced">
<!-- additional elements deleted from this example -->
<!-- First define all the content properties -->
  <ContentItem>
    <Name>Three Record Banner</Name>
    <Property name="title">
      <String>Recommended</String>
    </Property>
    <Property name="image_collection">
      <String/>
    </Property>
    <Property name="image_src">
      <String/>
    </Property>
    <Property name="record_list">
      <RecordList/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <GroupLabel label="Image configuration"/>
      <StringEditor propertyName="title" label="Banner Title"/>
      <ComboBox propertyName="image_collection" label="Image Collection">
        <ComboBoxItem data="images/stock" label="Stock Photos"/>
        <ComboBoxItem data="images/promo" label="Promotional Images"/>
       <ComboBoxItem data="images/promo/product" label="Product Images"/>

        <ComboBoxItem data="web/images" label="Web Graphics"/>
        <ComboBoxItem data="web/logos" label="Logos"/>
      </ComboBox>
      <StringEditor propertyName="image_src" label="Image Name"/>
      <ImagePreview urlExpression="http://www.example.com/{image_collec¬
tion}/{image_src}" label="Image Preview" />
      <RecordSelector propertyName="record_list" label="Featured Records"
 maxRecords="3"/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Related Links

File hosting and security considerations on page 62

Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Adding a string property on page 26

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.

Adding a string editor on page 35
: 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.

Adding a combo box editor on page 37
: Predefining values through a combo box editor allows you more control over page content and provides a more efficient mechanism for populating templates and cartridges in the editing interface.

About the urlExpression for image preview

This section contains guidelines for constructing the `urlExpression` for an image preview.

- Any string within {braces} is treated as the name of a string property defined in the same template.
- All string properties that the `urlExpression` depends on must be specified in Experience Manager before the preview renders. There is no way to specify an "optional" property.
- If the URL that you specify is a relative rather than absolute path, this is interpreted for preview purposes as relative to Experience Manager (and not, for example, relative to the front-end application server).

Adding a checkbox editor

Checkbox editors provide Experience Manager users with a quick and efficient way to specify the value of a Boolean property.

For example, you might use a checkbox editor to give the content administrator the choice to auto-generate metadata information.

```
<CheckBox propertyName="autogen_meta" label="Auto-generate meta information"
 />
```

To add a checkbox editor:

1. Insert a `<CheckBox>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the combo box 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 checkbox 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.
`label`	This attribute allows you to specify a more descriptive label for this field in Experience Manager. If no label is specified, the property name is used by default.

The following example illustrates an "autogen_meta" checkbox with a default value of "true":

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
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="autogen_meta">
      <Boolean>true</Boolean>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
  <EditorPanel>
    <BasicContentItemEditor>
<!-- additional elements deleted from this example -->
      <CheckBox propertyName="autogen_meta" label="Auto-generate meta in¬
formation" />
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Related Links

Adding a Boolean property on page 27
Boolean properties represent a true or false value and can be used to enable or disable features in your application.

Adding a navigation refinements selector

You add a navigation refinements selector to enable an interface in Experience Manager for content administrators to choose and order specific dimensions to display for a landing page.

To add a navigation refinements selector to a template:

1. Insert a `<NavigationRefinementsSelector>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the dimension selector:

Attribute	Description
`propertyName`	Required. The `name` of the navigation refinements property that this editor is associated with. This property must be declared in the same template as the navigation refinements selector.
`label`	This attribute allows you to specify a more descriptive label for this editor in Experience Manager. If no label is specified, the property name is used by default.

The following example shows a navigation refinements selector associated with a "refinements" property.

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="Cartridge25PercentWidth" id="GuidedNavigation">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Guided Navigation</Name>
    <Property name="refinements">
      <NavigationRefinements xmlns="http://endeca.com/schema/content-
tags/2008" />
        <DimensionList>
          <Dimension id="146"/>
          <Dimension id="212"/>
          <Dimension id="123"/>
        <DimensionList/>
      <NavigationRefinements/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <NavigationRefinementsSelector propertyName="refinements" label="Re¬
finements" />
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Related Links

Adding a navigation refinements property on page 27

When you add a navigation refinements property, the content administrator can use the default dimension order returned by the Endeca MDEX Engine or specify a custom default dimension order.

Adding a navigation records editor

You add a navigation records editor to enable an interface in Experience Manager to configure the display of record results on a landing page.

The navigation records editor is a collection of controls that enable a content administrator to specify sort order, relevance ranking, and the number of records to display per page.

To add a navigation records editor to a template:

1. Insert a `<NavigationRecordsEditor>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the navigation records editor:

Attribute	Description
`propertyName`	Required. The `name` of the navigation records property that this editor is associated with. This property must be declared in the same template as the navigation records editor.
`label`	This attribute enables you to specify a more descriptive label for this editor in Experience Manager. If no label is specified, the property name is used by default.

3. Specify sort options by inserting a `<SortChoices>` element within `<NavigationRecordsEdi¬tor>`.

   `<SortChoices>` takes the following optional attributes:

Attribute	Description
`enabled`	If set to `false`, the default sort order displays in the Content Details Panel in Experience Manager, but the content administrator cannot configure the sort order for this cartridge. Sort order configuration is enabled by default.
`label`	This attribute enables you to specify a more descriptive label for this drop down menu in Experience Manager. If no label is specified, the default is **Sort by**.

   If you omit the `<SortChoices>` element, the drop down menu to specify sort order does not display in Experience Manager.

4. Specify the available sort options by inserting one or more `<SortChoice>` elements within `<SortChoices>`.

   `<SortChoice>` takes a required `label` attribute to specify a more descriptive label, such as `Name (A-Z)`, for this option in the drop down list.

   `<SortChoice>` can contain one or more `<Sort>` elements. The content of the element is the name of the property by which to sort. The `<Sort>` element takes the following optional attributes:

Attribute	Description
`ascending`	A Boolean value specifying whether the records should be sorted in ascending order. Default is `true`.
`latitude`	The latitude value of a geocode sort key for geospatial sorting.
`longitude`	The longitude value of a geocode sort key for geospatial sorting.

   If you specify more than one `<Sort>` within a `<SortChoice>`, selecting that option applies the sorts in order. For example, the following sort choice applies a record sort in descending order of price followed by sales rank:

```
<SortChoice label="Price (Descending) then Sales Rank (Descending)">
  <Sort ascending="false">P_Price</Sort>
  <Sort ascending="false">P_SalesRank</Sort>
</SortChoice>
```

5. Specify relevance ranking options by inserting a `<RelevanceRankingChoices>` element within `<NavigationRecordsEditor>`.

   `<RelevanceRankingChoices>` takes the following optional attributes:

Attribute	Description
`enabled`	If set to `false`, the default relevance ranking strategy displays in the Content Details Panel in Experience Manager, but the content administrator cannot configure relevance ranking for this cartridge. Relevance ranking configuration is enabled by default.
`label`	This attribute enables you to specify a more descriptive label for this drop down list in Experience Manager. If no label is specified, the default is **Relevance ranking**.

   If you omit the `<RelevanceRankingChoices>` element, the drop down menu to specify relevance ranking strategy does not display in Experience Manager.

6. Specify the available relevance ranking options by inserting one or more `<RelevanceRanking¬Choice>` elements within `<RelevanceRankingChoices>`.

   `<RelevanceRankingChoice>` takes the following attributes:

Attribute	Description
`label`	Required. This attribute enables you to specify a more descriptive label for this option in the drop down list.
`relevanceRanking`	Required. The value of this attribute is an MDEX Engine URL relevance ranking strategy string. For details about the format of these strings, see "Controlling relevance ranking at the query level" in the *Endeca Advanced Development Guide*.

Note: If you use certain relevance ranking strategies across several templates, you can create a search interface in Developer Studio to define commonly used combinations of relevance ranking modules. For details, see the *Oracle Endeca Developer Studio Help*.

7. Enable a field for specifying the number of records to return per page by inserting a `<RecordsPerPage>` element within `<NavigationRecordsEditor>`.

   `<RecordsPerPage>` takes the following optional attributes:

Attribute	Description
`enabled`	If set to `false`, the default records per page value (if you specified one for the property in the template) displays in the Content Details Panel in Experience Manager. However, the content administrator cannot configure records per page for this cartridge. Configuration of records per page is enabled by default.
`label`	This attribute enables you to specify a more descriptive label for this field in Experience Manager. If no label is specified, the default is **Records per page**.
`minimumRecords`	The minimum value for records per page that can be specified in Experience Manager. Must be an integer greater than or equal to `1`.
`maximumRecords`	The maximum value for records per page that can be specified in Experience Manager. Must be an integer greater than the value of `minimumRecords`.

   If you omit the `<RecordsPerPage>` element, the field to specify the number of records per page does not display in Experience Manager.

The following example shows a navigation results editor that enables configuration of sort, relevance ranking, and records per page:

```
<?xml version="1.0" encoding="UTF-8"?>
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="MainColumnContent" id="ResultsList">
<!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Results List</Name>
    <Property name="navigation_records">
     <NavigationRecords xmlns="http://endeca.com/schema/content-tags/2008"
 />
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <NavigationRecordsEditor propertyName="navigation_records"
        label="Record List">
        <SortChoices label="Sorts">
          <SortChoice label="Score (Ascending)">
            <Sort ascending="true">P_Score</Sort>
          </SortChoice>
          <SortChoice label="Score (Descending)">
            <Sort ascending="false">P_Score</Sort>
          </SortChoice>
          <SortChoice label="Price (Ascending)">
            <Sort ascending="true">P_Price</Sort>
          </SortChoice>
          <SortChoice label="Price (Descending)">
```

```
            <Sort ascending="false">P_Price</Sort>
          </SortChoice>
          <SortChoice label="Sales Rank (Ascending)">
            <Sort ascending="true">P_SalesRank</Sort>
          </SortChoice>
          <SortChoice label="Sales Rank (Descending)">
            <Sort ascending="false">P_SalesRank</Sort>
          </SortChoice>
        </SortChoices>
        <RelevanceRankingChoices label="Relevance ranking">
          <RelevanceRankingChoice label="First"
            relevanceRanking="first" />
          <RelevanceRankingChoice label="By Sales Rank (Static)"
            relevanceRanking="static(P_SalesRank)" />
          <RelevanceRankingChoice label="By Price (Static)"
            relevanceRanking="static(P_Price)" />
          <RelevanceRankingChoice label="Frequency"
            relevanceRanking="freq" />
        </RelevanceRankingChoices>
        <RecordsPerPage label="Records per page" minimumRecords="1"
          maximumRecords="100" />
      </NavigationRecordsEditor>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Related Links

About record properties on page 29

There are two types of content properties that enable a content administrator to customize the display of records on a page: navigation records and record lists.

Adding a navigation records property on page 29

When you add a navigation records property, a content administrator can configure the display of the main results of an end user's query.

Adding a record selector

You add a record selector to enable an interface in Experience Manager to specify featured records or queries for a record list property.

The record selector dialog box allows a content administrator to designate specific records to spotlight in a section, or to specify a query to return a dynamic list of records. If the content administrator designates a record ID that does not exist, an error message displays.

To add a record selector to a template:

1. Insert a `<RecordSelector>` element within `<BasicContentItemEditor>`.
2. Specify additional attributes for the record selector:

Attribute	Description
`propertyName`	Required. The name of the record list property that this editor is associated with. This property must be declared in the same template as the record selector.
`label`	This attribute allows you to specify a more descriptive label for this editor in Experience Manager. If no label is specified, the property name is used by default.

Attribute	Description
`maxRecords`	Sets the maximum number of records that this property can contain. If the content administrator designates specific records in Experience Manager, the number of records cannot exceed the value of `maxRecords`. If the content administrator specifies a query, Experience Manager returns no more than this number of records. When configuring this property, the content administrator may choose to designate fewer static records or to further limit the number of records returned by a query. The default value for `maxRecords` is `10`.
`augmentRecord¬ Filter`	If set to `true`, record filters are applied to the query that is used to evaluate the associated record list property. The default value is `true`.
`previewRecordId¬ Label`	This attribute allows you to specify a more descriptive label for the record ID field in the Record Selector dialog box. For example, `previewRecordIdLa¬ bel="SKU"` prompts a content administrator to enter a SKU number. The default label is **Record ID**.

3. Optionally, specify one or more `PreviewProperty` elements.

   If a preview property is specified, an additional column appears in the Record Selector dialog box to showcase the preview properties. Preview properties display relevant information about records, allowing content administrators to confirm that they are entering the correct record IDs. `Preview¬ Property` takes the following attributes:

Attribute	Description
`name`	Required. Specify the property to display with record IDs in the Record Selector dialog box. **Note:** You must specify a property; dimensions cannot be used.
`label`	This attribute allows you to specify a more descriptive label for this property in the Record Selector dialog box. If no label is specified, the value for `name` is used by default.

The following example shows a Record Selector associated with a "record_list" property. This allows a content administrator to specify up to three specific records (SKU numbers) or a query that returns up to three records. If a content administrator chooses to enter SKU numbers, the name and year of each chosen record displays in the Record Selector dialog box.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="SidebarItem" id="ThreeRecordBox">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Three Record Spotlight Box</Name>
    <!-- additional properties deleted from this example -->
    <Property name="record_list">
      <RecordList/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <!-- additional editors deleted from this example -->
      <RecordSelector propertyName="record_list" label="Featured records"
```

```
 maxRecords="3">
         <PreviewProperty name="P_Name" label="Name"/>
      </RecordSelector>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

Related Links

About record properties on page 29
: There are two types of content properties that enable a content administrator to customize the display of records on a page: navigation records and record lists.

Adding a record list property on page 31
: A record list property can contain one or more Endeca records for merchandising or Content Spotlighting.

About sort options for the record selector

The record selector enables the content administrator to designate either specific featured records or a query that returns a dynamic list of records for spotlighting. When selecting dynamic records, the content administrator can specify a sort order for the records.

Unlike the navigation records editor, the sort options for the record selector are not specified in the template. The sort keys that are available in the record selector are all the properties or dimensions that are configured for precomputed sort (that is, those that have the **Prepare Sort Offline** option selected in Developer Studio). For more information about configuring properties and dimensions for precomputed sort, see the *Oracle Endeca Developer Studio Help*.

Only one sort key can be applied to the query for each dynamic spotlighting cartridge. It is not possible to specify a default sort order for the record list property.

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.

Related Links

Adding a content item property on page 32
: A content item property defines a template section by creating a placeholder for a nested content item.

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"
 type="BuyingGuideSection" id="BuyingGuideSection">
  <!-- additional elements deleted from this example -->
  <EditorPanel>
    <BasicContentItemEditor>
      <GroupLabel label="Page metadata"/>
      <CheckBox propertyName="autogen_meta" label="Auto-generate meta
information" />
      <StringEditor propertyName="title" label="Title" enabled="true"/>
      <StringEditor propertyName="meta_keywords" label="Meta keywords"
enabled="true" height="72"/>
      <StringEditor propertyName="meta_description" label="Meta descrip¬
tion" enabled="true" height="72"/>
      <GroupLabel label="Configuration"/>
      <StringEditor propertyName="title" label="Banner title" en¬
abled="true"/>
     <RecordSelector propertyName="record_list" label="Featured records"
 maxRecords="3">
         <PreviewProperty name="P_Name" label="Name"/>
      </RecordSelector>
      <StringEditor propertyName="link_text" label="See-all link text"
enabled="true"/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

`<GroupLabel>` is an empty tag that allows you to specify the label text with the `label` attribute.

About using XML pass-through properties

In addition to configurable content properties, the template schema also allows you to define non-configurable properties that are passed directly to the front-end application.

While you can use hidden string properties to pass simple pieces of information to the application, pass-through properties can be useful if the following conditions apply:

- The property never needs to be configured or exposed in Experience Manager.
- The property contains structured data that can be represented in XML.

Pass-through properties may take the form of *pass-through content properties* or any arbitrary XML that does not conform to the template schema, as long as you specify a different namespace from the Endeca template and content schemas.

About using pass-through content properties

Pass-through content properties allow you to embed MDEX Engine query results in the content item object that you can access through the Content Assembler API.

Pass-through content properties follow the same schema as the page configurations generated by Experience Manager. When the Content Assembler processes these properties with the values you specified in the template, it evaluates them and executes any necessary queries exactly as if the property had been configured with specific values by a content administrator in Experience Manager.

The schema to use for pass-through content properties is located in `doc\schemas\content-tags.xsd` (on Windows) or `doc/schemas/content-tags.xsd` (on UNIX) in your Content Assembler API installation. You must specify the namespace for the content-tags schema in order for the Content Assembler to evaluate the properties as if they were content properties.

Note: The `<NavigationRefinements>` and `<NavigationRecords>` properties in the content-tags schema are configurable in Experience Manager (via the `<NavigationRefinementsSelector>` and `<NavigationRecordsEditor>`, respectively). Other properties defined in the content-tags schema do not have editors available in Experience Manager. `<NavigationRefinements>` and `<NavigationRecords>` can be used as pass-through properties by specifying defaults in the property definition without defining an associated editor in the template.

The following example shows several types of pass-through content properties:

```
<Property name="sample_navigation_query">
  <NavQuery xmlns="http://endeca.com/schema/content-tags/2008"
    augment="true" maxRecords="7">
    <DimensionValue id="60" dimensionId="2"/>
    <DimensionValue id="40" dimensionId="1"/>
  </NavQuery>
</Property>

<Property name="sample_urlenequery">
  <UrlEneQuery xmlns="http://endeca.com/schema/content-tags/2008"
    maxRecords="3">N=8021&amp;Ns=P_Price</UrlEneQuery>
</Property>

<Property name="sample_supplement_property">
  <Supplement xmlns="http://endeca.com/schema/content-tags/2008">
    <SupplementId>547</SupplementId>
  </Supplement>
</Property>

<Property name="another_sample_supplement_property">
  <Supplement xmlns="http://endeca.com/schema/content-tags/2008">
    <Zone>ZoneOne</Zone>
    <Style>StyleOne</Style>
  </Supplement>
</Property>

<!-- In the unusual case that you need specific records
     hard-coded into the template -->
<Property name="sample_record_query">
  <RecordQuery xmlns="http://endeca.com/schema/content-tags/2008">
    <RecordId>123</RecordId>
    <RecordId>456</RecordId>
    <RecordId>789</RecordId>
    <RecordId>abc</RecordId>
  </RecordQuery>
</Property>

<!-- <NavigationResult> is an empty tag that enables access to
     query results from nested content items -->
<Property name="navigation_results">
 <NavigationResult xmlns="http://endeca.com/schema/content-tags/2008"/>

</Property>
```

About passing arbitrary XML to the front-end application

You can nest arbitrary XML in templates within a `<Property>` element.

Embedding arbitrary XML in template properties allows you to pass structured data to your application such as configuration for third-party packages used by your front-end application. If the Content Assembler does not recognize a tag, it returns the XML as an `org.w3c.dom.Element` (in Java) or a `string` (for the RAD Toolkit for ASP.NET).

The only requirement is that the namespace must be different from any of the Endeca template or content schemas located in `doc\schemas\` (on Windows) or `doc/schemas/` (on UNIX) in your Content Assembler API installation. Additionally, the Content Assembler API must be able to access the namespace that you specify.

Note: Experience Manager does not perform any validation on XML within a different namespace from the content-template or content-tags schemas. If you are using custom XML pass-through properties, be sure to validate your templates before uploading them.

The following example shows XML inserted within a property:

```
  <Property name="sample_XML_pass-through">
    <widget xmlns="http://mycompany.com/schema/widgets">
      <name>Example widget</name>
      <description>Sample for embedded XML in a template</description>
      <icon src="icons/example.png" />
      <content src="index.html"/>
      <access network="true"/>
    </widget>
  </Property>
```

Chapter 3

Supporting Experience Manager

This section describes the tasks needed to enable content administrators to create pages in Experience Manager.

Making templates available in Experience Manager

This section describes how to manage Experience Manager templates using the emgr_update utility.

The emgr_update utility assists you in updating the instance configuration of a production system based on the changes made with the Endeca tools in a staging environment. You can also use emgr_update to add, retrieve, and remove templates from Experience Manager.

For a complete list of accepted emgr_update syntax, refer to the *Oracle Endeca Guided Search Administrator's Guide*.

Uploading templates to Experience Manager

Before Experience Manager users can access new templates, you must upload them using the emgr_update utility.

Note: Template file names cannot have spaces in them.

To upload a new template:

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

Parameter	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 to which you want the templates to apply.
`--dir`	The path to the local directory where your templates are stored.

The following is a Windows example:

```
emgr_update.bat --action set_templates --host localhost:8006
--app_name My_application --dir c:\endeca-app\templates\
```

The following is a UNIX example:

```
emgr_update --action set_templates --host localhost:8006
--app_name My_application --dir /apps/endeca/templates/
```

If templates do not display in Experience Manager after uploading them using emgr_update, check the log in `%ENDECA_TOOLS_CONF%\logs\webstudio.log` (on Windows) or `$ENDECA_TOOLS_CONF/logs/webstudio.log` (on UNIX) for possible causes.

Related Links

Updating templates in Experience Manager on page 56
Updating templates using emgr_update is a multi-step process.

About updating templates on page 55
When updating templates in Experience Manager, you should be aware of how conflicts are handled.

Troubleshooting invalid templates on page 61
Some templates may be successfully uploaded to Workbench, but still contain errors that lead to unexpected behavior in Experience Manager.

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 page or cartridge templates and update them in Experience Manager.

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

For example, a record list editor can be altered in a landing page template to have its `maxRecords` value lowered from five to two. In this case, if the content administrator has previously selected more than two static records, Experience Manager removes the excess record selections the next time it loads the record list editor. Any such changes are saved when the content administrator manually saves the page.

Note: Existing page configurations are not upgraded to the new template until a content administrator edits and saves the affected page or cartridge 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 page or cartridge 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 page or cartridge 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 page or cartridge 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 record list) within a template, the corresponding property editor (if one is specified) becomes available in Experience Manager when a content administrator edits a page or cartridge based on the updated template. The existing value for the property does not display in Experience Manager and is replaced when the content administrator saves the content.
- If a content item property has changed to specify a different cartridge 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, whether it is one of the built-in content properties or a custom XML property, it is only applied to new pages or cartridges 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.

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.

Editor-specific behavior

Some editors impose constraints on the content that can be allowed for a particular page or cardtridge:

- If the `maxRecords` value for a record selector is lower than the previous value and the content administrator had specified static records to display, any records beyond the new maximum value are deleted.

Managing template changes

Because existing content is not automatically updated to the new templates, and both XML pass-through 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 relevant 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.

Related Links

About updating templates on page 55
When updating templates in Experience Manager, you should be aware of how conflicts are handled.

Updating templates in Experience Manager on page 56
Updating templates using emgr_update is a multi-step process.

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.

Related Links

About modifying templates that are used by existing pages on page 54
: During the development and testing phase of your application deployment, you may need to make adjustments to your page or cartridge templates and update them in Experience Manager.

Updating templates in Experience Manager on page 56
: Updating templates using emgr_update is a multi-step process.

Updating templates in Experience Manager

Updating templates using emgr_update is a multi-step process.

Note: Before updating templates in 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.

To update existing templates in Experience Manager:

1. Retrieve the current set of templates from Experience Manager.
2. Make any necessary edits to the templates on your local machine.
3. Remove all templates from Experience Manager.
4. Upload the revised templates from your local directory to the Experience Manager.

Related Links

About modifying templates that are used by existing pages on page 54
: During the development and testing phase of your application deployment, you may need to make adjustments to your page or cartridge templates and update them in Experience Manager.

About updating templates on page 55
: When updating templates in Experience Manager, you should be aware of how conflicts are handled.

Uploading templates to Experience Manager on page 53
: Before Experience Manager users can access new templates, you must upload them using the emgr_update utility.

Retrieving the current templates from Experience Manager on page 57
: If you need to view or edit an existing template on a local machine, use emgr_update to copy the templates from Experience Manager into a local directory.

Removing templates from Experience Manager on page 58
: You can remove all the templates from Experience Manager using the emgr_update utility.

Retrieving the current templates from Experience Manager

If you need to view or edit an existing template on a local machine, use emgr_update to copy the templates from Experience Manager into a local directory.

If you need to retrieve the current set of templates:

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

Parameter	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 retrieve the templates.
`--dir`	The path to the local directory to which you want the templates copied.

The following is a Windows example:

```
emgr_update.bat --action get_templates --host localhost:8006
--app_name My_application --dir c:\endeca-app\templates\
```

The following is a UNIX example:

```
emgr_update --action get_templates --host localhost:8006
--app_name My_application --dir /apps/endeca/templates/
```

About removing templates

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

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

- If the missing template is the page template, then the top-level page properties cannot be edited in the content details panel, but the content tree is still active. The content administrator can still change or edit the cartridges in that page as long as their corresponding templates are available.
- If the missing template is a cartridge template, the properties of that cartridge cannot be edited in the content details panel. All other cartridges, including cartridges that are nested within the missing cartridge, can still be edited via the content tree.

In both cases, all the configured values of the missing template's properties are preserved unless the content administrator removes or changes the template.

The content administrator has the following options:

- Leave the existing content as is. The Content Assembler continues to evaluate and process page configurations regardless of whether the template exists in Experience Manager, and existing pages continue to display in the front-end 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.

Related Links

Removing templates from Experience Manager on page 58
You can remove all the templates from Experience Manager using the emgr_update utility.

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
```

Related Links

About removing templates on page 57
If you remove a page or cartridge template that is being used for an existing page, the properties of the page or section are no longer editable in Experience Manager.

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 platforms
- `$ENDECA_TOOLS_CONF/logs` on UNIX platforms

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\page_builder_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.

Missing <RuleInfo> element

A template of type `PageTemplate` that does not have a `<RuleInfo>` element passes schema validation but is still an invalid template. When such a template is uploaded to Experience Manager, it is unavailable to content administrators and an error message is returned to the emgr_update command line call similar to the following:

```
The template "ThreeColumnNavigationPage" is missing a required <RuleInfo>
element.
```

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

Invalid zone or style

If a template is uploaded and refers to an invalid zone or style, the template is unavailable in Experience Manager and an error message is returned to the emgr_update command line call similar to the following:

```
[ERROR 1] The template "NavigationPage" has an invalid style ("PageStyle3").
[ERROR 2] The template "NavigationPage" has an invalid zone ("Navigation¬
PageZone3").
```

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

The `zone` and `style` attributes of the `<RuleInfo>` element must correspond to one of the zones and styles defined in the instance configuration of the application that you specified in the emgr_update call. Note that you may also see this message when the name of application you specified does not exist in the EAC.

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\page_builder_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.

Related Links

About template validation on page 18

Templates are validated against the Experience Manager template schemas when you upload them to Experience Manager.

Troubleshooting invalid templates

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

The main scenario is when a property is associated with an editor with constraints, such as a combo box or a navigation records editor, and the default value of the property does not meet the editor's constraints. For example:

- A default value for a string property that is not specified as one of the options in a combo box editor (unless the combo box is editable)
- A default sort order for navigation records that is not specified as one of the options in the navigation records editor
- A default relevance ranking strategy that is not specified as one of the options in the navigation records editor
- A default number of records per page that is outside the specified range for the navigation records editor

In the case of the navigation records editor, Experience Manager discards the default value and the following messsage displays 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 Merchandising Workbench Help.
```

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

Related Links

Adding a combo box editor on page 37
Predefining values through a combo box editor allows you more control over page content and provides a more efficient mechanism for populating templates and cartridges in the editing interface.

Adding a navigation records property on page 29
When you add a navigation records property, a content administrator can configure the display of the main results of an end user's query.

Adding a navigation records editor on page 44
You add a navigation records editor to enable an interface in Experience Manager to configure the display of record results on a landing page.

Troubleshooting invalid pages

If a page is displaying in Experience Manager as invalid, it is using a template with invalid zones or styles.

To determine whether the template is referring an invalid style or an invalid zone:

1. Retrieve the currently loaded instance configuration using one of the following methods:
   - Using the "Get Instance Configuration" feature of Developer Studio, copy the configuration files into the project folder.
   - Using the emgr_update utility, specify a destination directory.

2. In the destination directory, locate and open the *appname*.merch_rule_group_*groupname*.xml file that corresponds to the invalid template's rule group.
   For example, if the application name is "wineapp" and the rule group is "dynamicpages", the file would be wineapp.merch_rule_group_dynamicpages.xml.
3. Look for the invalid.zone or invalid.style properties:

```
//Invalid zone property:
 <PROP NAME ="endeca.internal.landingpage.invalid.zone">
 <PVAL>true</PVAL>
 </PROP>

//Invalid style property:
 <PROP NAME ="endeca.internal.landingpage.invalid.style">
 <PVAL>true</PVAL>
 </PROP>
```

   These properties are for debugging purposes only, and can be safely removed.

4. Locate the zone and style configuration for the rule:
   For example:

```
<MERCH_RULE ID="17" TITLE="Champagne Landing Page"
  ZONE_NAME="NavigationPageZone" STYLE_NAME="PageStylee"
  SHUFFLE_RECS="FALSE" SELF_PIVOT="TRUE">
```

Once you have identified the invalid zone or style, you can either restore the zone or style or edit the rule to use a valid zone or style. You must run a baseline update for your changes to appear in the preview application.

Related Links

Creating a zone for dynamic pages on page 16
Dynamic pages must be assigned a rule zone in order to display. You should create at least one zone in your application for use exclusively with dynamic pages.

About using zones with dynamic pages on page 16
Zones enable the display of dynamic pages in the application. While a single zone is usually sufficient, multiple zones can enable finer-grained control over the display of dynamic pages.

File hosting and security considerations

Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Experience Manager makes an anonymous request to the file server to fetch resources. That is, even though content administrators are authenticated when they log in to Experience Manager, the tool does not use their credentials when requesting images, editors, or other files.

Experience Manager also respects the cross-domain policy file of the server hosting the external files. To ensure that Experience Manager can load these files, place a crossdomain.xml file on the file server. This file allows you to enable access to media on this server from a specific IP address, a specific domain, or any domain. If this policy file does not allow access from the Experience Manager server, a security error similar to the following displays when Experience Manager attempts to load the resource:

```
Error #2044: Unhandled securityError:. text=Error #2048: Security sandbox
violation: http://pagebuilder.mycompany.com/tmgr/tmgr.swf cannot load data
 from http://www.example.com/images/3column.gif.
```

The following example of a `crossdomain.xml` file enables access from any domain to files hosted on www.example.com:

```
<?xml version="1.0"?>
<!-- http://www.example.com/crossdomain.xml -->
<cross-domain-policy>
  <allow-access-from domain="*" />
</cross-domain-policy>
```

You can also restrict access to specific domains or IP addresses, for instance, for the server on which Experience Manager is running. Wildcards are allowed in domain names but not IP addresses. The following example shows a policy file for www.example.com that allows access from anywhere in the example.com domain, www.customer.com, and 105.216.0.40. It includes a `by-content-type` meta-policy that allows policy files with a `Content-Type` of exactly `text/x-cross-domain-policy`:

```
<?xml version="1.0"?>
<!-- http://www.example.com/crossdomain.xml -->
<cross-domain-policy>
  <site-control permitted-cross-domain-policies="by-content-type"/>
  <allow-access-from domain="*.example.com" />
  <allow-access-from domain="www.customer.com" />
  <allow-access-from domain="105.216.0.40" />
</cross-domain-policy>
```

Important: In addition to cross-domain policy files, you must set up a meta-policy for each server. These are configuration settings that manage what cross-domain policies are allowed on a server. Using the default configuration without an explicit policy file is allowed by Flash version 9, but a meta-policy is required with the use of version 10.

For more information about meta-policies and cross-domain policy files, see the Adobe Flash documentation.

Chapter 4

Extending Experience Manager with Community Editors

The Experience Manager Editor SDK enables application developers to introduce new functionality into Experience Manager via community editors. This enables merchandisers to manage content in a way that makes sense to them, and takes into account their existing business processes.

About Experience Manager editors

Experience Manager ships with a set of standard editors that address most common use cases. You can extend the functionality of the Experience Manager interface by writing editors to support a specific use case or business process.

Experience Manager editors are Flex components that provide a graphical interface for the content administrator to configure properties in Experience Manager.

Editors are registered with Experience Manager through a configuration file that describes available editor modules. These modules exist as compiled SWFs. A single module may be able to instantiate several kinds of editors.

The editors themselves consist of Flex MXML and ActionScript that together specify editor behavior, and drive the user interface for creating and editing content. Editors can use the Flex `ExternalInterface` class to invoke JavaScript, which can in turn call methods implemented in other languages.

Editors use the content model provided by the API to manipulate content. When a landing page is loaded in Experience Manager, its content is deserialized into ActionScript objects that can be programmatically manipulated using the interfaces provided by the Experience Manager Editor API.

Standard editors are included as part of the Experience Manager product, and cannot be removed or modified. Additional editors developed by the Endeca community (including Endeca Professional Services, partners, or customers) using the Experience Manager Editor API are called **community editors**.

Note: For the purposes of this section, all references to Experience Manager editors refer to community editors, unless specifically stated otherwise.

What can editors do?

The Experience Manager Editor API enables editors to perform the following actions:

- They can edit one or more properties on a content item, including both standard and custom XML property types.
- They can locate templates based on a template type and instantiate content items.
- They can access children of a content item, or anything lower in the content tree.

Editors have the following limitations:

- They cannot access trigger or other rule information, and therefore cannot modify rule properties.
- They cannot access the parent of the content item, or anything higher in the content tree.
- They cannot add or remove `<Property>` elements from a template.

Scenarios for extending Experience Manager and the Content Assembler

You can use either community editors on their own, community tag handlers on their own, or both of them in combination to extend the functionality of Experience Manager.

A tag handler enables you to extend the processing logic in the Content Assembler for custom XML property types.

Following are some common scenarios and their implications for community editors or tag handlers:

Scenario	Use community editor?	Use community tag handler?
Include application-specific information in the template as a pass-through XML property. *Example:* Information that the application uses to render the cartridge, but is of no interest to the content administrator.	**No** If content administrators do not need to modify the configuration of a property on a per-page basis, you do not need to write a specialized editor.	**No** The Content Assembler returns the XML to the rendering code for your application.
Include external configuration in the template as a pass-through XML property. *Example:* Hard-coded configuration for a third-party system that applies to any page that uses this template.	**No** If content administrators do not need to modify the content of a property on a per-page basis, you do not need to write a specialized editor.	**Yes** The Content Assembler uses the information contained in the XML to query a third-party system, and returns the results to the rendering code.
Provide a new interface for content administrators to configure existing Experience Manager properties. *Example:* A variation of the record selector dialog box that enables content administrators to browse for featured records, instead of entering a record ID.	**Yes** This editor is bound to a standard property. (In the example, the editor modifies a `<RecordList>` property.)	**No** The community editor outputs standard Endeca content XML, which is processed by the standard tag handler for record lists. No additional work is necessary.

Scenario	Use community editor?	Use community tag handler?
Provide an interface to configure functionality that is not supported by Experience Manager out-of-the-box. *Example:* An editor that enables content administrators to specify reviews to display for a particular navigation state, including number of reviews, sort order, and additional filtering options.	**Yes** The editor provides a specialized interface for selecting data to populate a cartridge. The configuration is saved as a custom XML property.	There are two options: **No** The Content Assembler returns the XML to the application's rendering code, which can then fetch the reviews from the CMS where they are stored. **Yes** (preferred) The Content Assembler fetches the reviews from the CMS before returning the content results to the rendering code for your application. Similarly, you can use a tag handler and community editor to send customized queries to an MDEX Engine and return results to the rendering code.

Related Links

Working with custom XML properties on page 86
You can use the `IXMLProperty` interface to retrieve and manipulate custom XML properties.

What is the Experience Manager Editor SDK?

The Experience Manager Editor SDK consists of the following components:

- **Experience Manager Editor API** — A programmatic interface for managing and instantiating Experience Manager editors and accessing and manipulating the underlying content model.
- **Sample Editor** — A Flex Builder project and source code for a sample rich text editor module.
- ***Experience Manager Editor API Reference*** — The generated ActionScript documentation for the Experience Manager Editor API.

The `pagebuilderEditorSDK.zip` archive is included in the `reference` directory of the Workbench installation. When extracted, it contains the following:

Directory	Contents
`asdoc`	This directory contains the *Experience Manager Editor API Reference* documentation.
`sample-editor-config`	The `sample_editors.xml` editor configuration file is included here for reference.

Directory	Contents
`sample-editor-project`	This directory includes a sample Flex Builder project that you can import into your workspace in order to view, modify, and build the sample rich text editor module.
`sample-page-templates`	The `SidebarItem-RichText.xml` file is provided as an example of a cartridge that uses the rich text editor.
`schema`	The included `editor-config.xsd` file defines the schema for editor configuration files. You can use this file to validate your own editor configuration files.

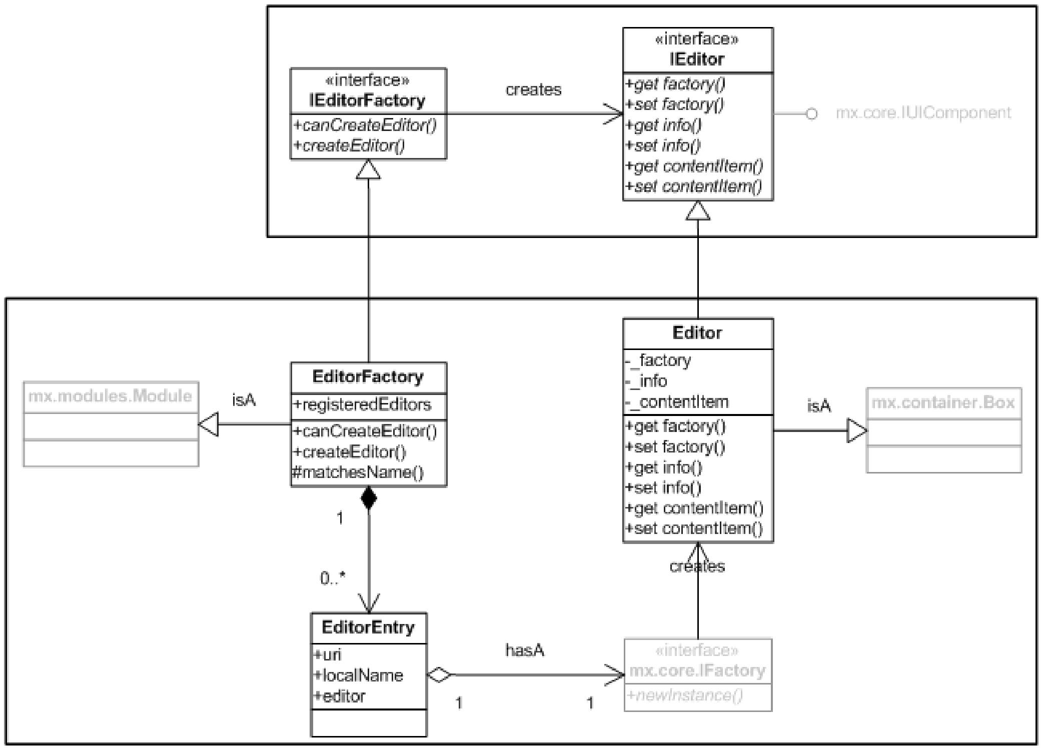
About the Experience Manager Editor API

There are three main parts to the Experience Manager Editor API: the first includes interfaces for editors and modules, and allows you to manage and instantiate editors. The second is a set of interfaces to the content model. The third is the Experience Manager eventing model, which allows you to dispatch different types of events on the content model.

About Experience Manager editor representation in the API

The images below show the Experience Manager editor interface and its basic implementation.

The following image shows how the Experience Manager Editor API represents editors. The interface is shown in the top box, with the basic implementation provided in `Experience ManagerEditorAPI.swc` shown below:



This basic implementation is included with Experience Manager, and is designed to support most common uses cases.

If you write your own implementation of these interfaces, note that `EditorModule` must be a `Module` or a subclass of `Module`. If you encounter a runtime error where an editor in the **Content Details Panel** is stuck in the "loading" state, check that the `EditorModule` has been implemented as a `Module` rather than an `Application`.

Related Links

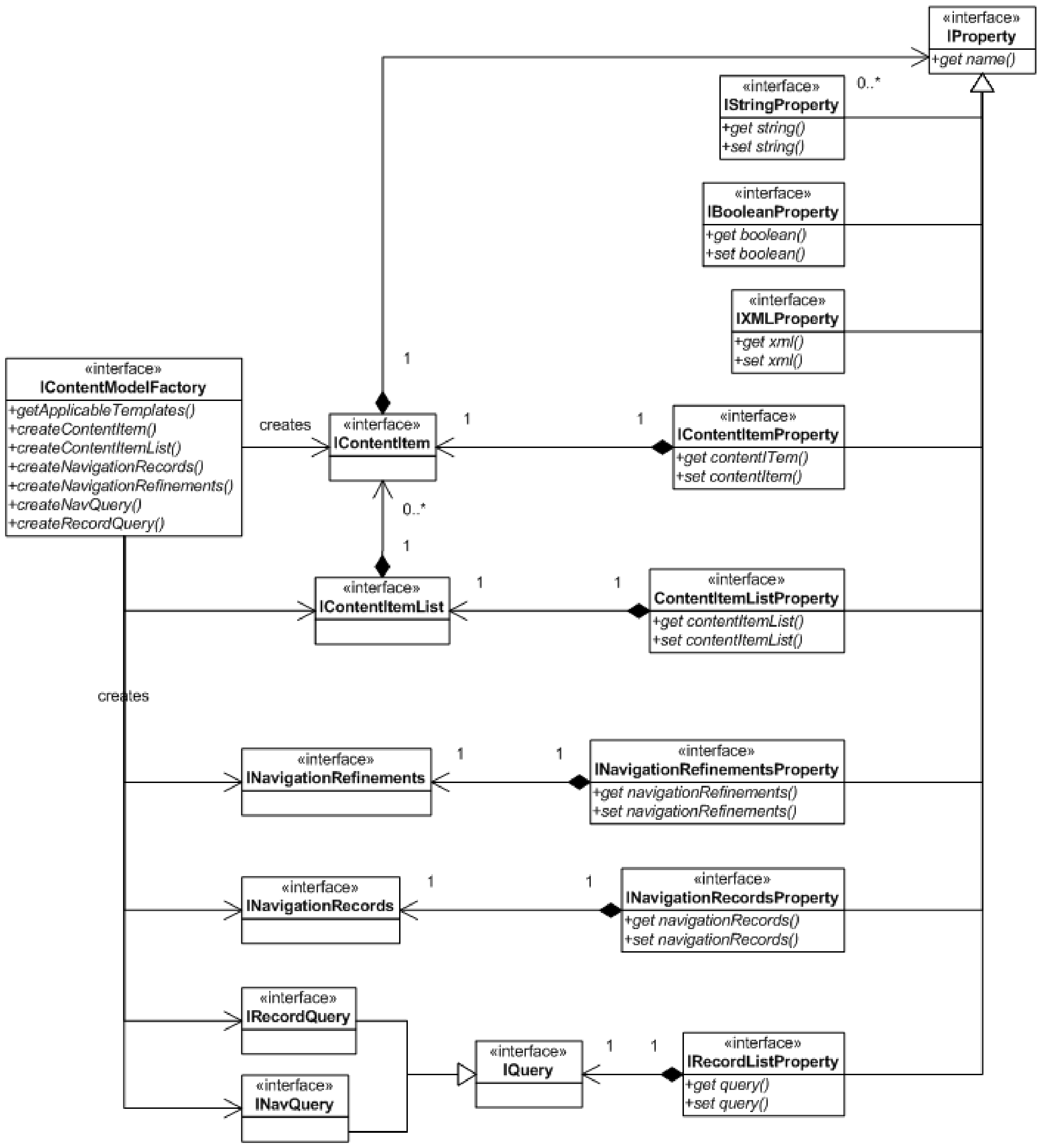
About developing editors on page 76

This section includes information on writing editors in Flex and MXML, as well as editors that use Flex to communicate with JavaScript.

The Experience Manager Editor API content model

The diagram below shows the Experience Manager Editor API content model.

Experience Manager reads and outputs XML representing a landing page. This XML content document gets deserialized into ActionScript objects, which you can access via a set of interfaces that are provided by the Experience Manager Editor API. These interfaces provide structured programmatic access to an object-based representation of the content document that is the landing page. The `IContentModelFactory` interface allows editors to create any required objects, such as properties and content items. The individual object interfaces provide methods for manipulating them.


«interface» IProperty
+get name()
0..*
«interface» IStringProperty
+get string()
+set string()
«interface» IBooleanProperty
+get boolean()
+set boolean()
«interface» IXMLProperty
+get xml()
+set xml()
«interface» IContentModelFactory
+getApplicableTemplates()
+createContentItem()
+createContentItemList()
+createNavigationRecords()
+createNavigationRefinements()
+createNavQuery()
+createRecordQuery()
creates
«interface» IContentItem
1
0..*
«interface» IContentItemProperty
+get contentITem()
+set contentItem()
«interface» IContentItemList
«interface» ContentItemListProperty
+get contentItemList()
+set contentItemList()
creates
«interface» INavigationRefinements
«interface» INavigationRefinementsProperty
+get navigationRefinements()
+set navigationRefinements()
«interface» INavigationRecords
«interface» INavigationRecordsProperty
+get navigationRecords()
+set navigationRecords()
«interface» IRecordQuery
«interface» INavQuery
«interface» IQuery
«interface» IRecordListProperty
+get query()
+set query()

The above image is meant to illustrate the structure and relationships of objects in the content model; it is not a comprehensive reference for the Experience Manager Editor API. Note the recursive nature of the content document, as defined by the relationship between content items and properties.

The content model includes interfaces for manipulating all standard Experience Manager property types. In addition, the `IXMLProperty` enables editors to output custom XML properties.

For complete information regarding the Experience Manager Editor API, please refer to the *Experience Manager Editor API Reference*.

Related Links

Working with the content model on page 83
To modify any content model object, you must retrieve it from the content item after your editor has been instantiated.

Working with custom XML properties on page 86

You can use the `IXMLProperty` interface to retrieve and manipulate custom XML properties.

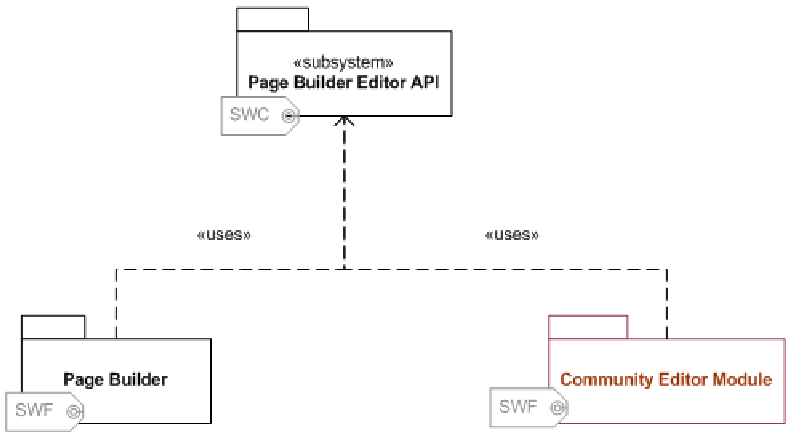
About eventing in Experience Manager

Experience Manager uses a standard Flex eventing model that includes the events below:

const	String	Usage
`ContentModelEvent.CHANGE`	`contentModelChange`	Experience Manager dispatches `contentModelChange` events in response to changes to the content model. You can use event listeners or bind your editor properties to content model objects if you want them to update automatically in response to changes to the content model.
`ContentModelEvent.UPGRADE`	`contentUpgrade`	If a template has been changed such that it is no longer compatible with the content that uses that template, an editor can dispatch this event to display a warning that the content may have changed.
`EditorEvent.READY`	`editorReady`	Experience Manager dispatches this event after it has instantiated an editor and set the `info`, `factory`, and `contentItem` properties on it. Editors should be set to listen for this event in order to retrieve and validate whatever properties they manipulate on the content item.
`EditorEvent.ERROR`	`editorError`	An editor can dispatch a `EditorEvent.ERROR` event to display an error message in place of the editor interface.

Compile-time and runtime dependencies

The main Experience Manager SWF and Experience Manager editor modules rely on the same Editor API, but have different compile time and runtime dependencies.



The following dependencies exist among the Experience Manager components that are used to implement editors:

- Experience Manager relies on the Experience Manager Editor API library, `Experience ManagerEditorAPI.swc`, as a compile-time resource. It is compiled into the Experience Manager SWF in its entirety.
- During development, Experience Manager editor modules require `Experience ManagerEditorAPI.swc` as a compile-time resource.
- When you build editor modules for use in Experience Manager, you should configure `Experience ManagerEditorAPI.swc` as a dynamically linked library, rather than compiling it into the SWF. You can do this by setting the link type to **External** in Flex Builder. Your editors will use the SWC compiled into Experience Manager at runtime.

About setting up the sample editor

The Workbench installation includes a sample editor project and associated template and configuration files. This section assumes that you have one or more applications deployed in Workbench that you can use to test the sample editor.

The sample rich text editor is a simple, all-Flex editor that should be bound to a property that contains a string. Before setting up the sample editor, you must set up your development environment so that you can generate the sample editor module from the provided source code.

Note: The rich text editor uses the Flex `RichTextEditor` control and outputs a string property that contains escaped Flex HTML text. This is a subset of HTML that is supported by Flash player and AIR, with the addition of a `<textformat>` tag, which is not standard HTML. The Content Assembler reference application does not include rendering code that is intended to handle strings in Flex HTML text format. To see the markup generated by the rich text editor, click the **XML View** tab and scroll to the `rich_text` property within the content tree. For more information about Flex HTML text, refer to the Adobe documentation.

After you have written and compiled your editor module, making the sample editor available to Experience Manager requires the following steps:

- Make the editor available via a URL.
- Add an entry for the editor to the editor configuration file.
- Configure Workbench to specify the location of the editor configuration file.

Using the editor requires the following steps:

- Create or modify a template to use the editor.
- Upload the template to Experience Manager using the emgr_update utility.
- In Experience Manager, create a cartridge using the above-mentioned template.

About Flex

Experience Manager editor modules should be written and compiled in Adobe Flex 3.2. Other versions are unsupported and may cause editors to fail at runtime.

Flex is an open source development framework created and maintained by Adobe. It supports common design patterns and is based on MXML and ActionScript 3.

This section assumes that you are comfortable working with Flex. If not, you may find the following resources helpful:

- *http://www.adobe.com/products/flex/overview* — The Adobe Flex website provides an overview of the Flex development framework and includes download links to the Flex SDK.
- *http://www.adobe.com/support/documentation/en/flex* — The Adobe Flex resources page contains links to Flex documentation, including the *Adobe Flex Language Reference* and *Flex 3 Developer Guide*.
- *http://opensource.adobe.com/wiki/display/flexsdk/Coding+Conventions* — The Adobe Open Source wiki page on Flex SDK best practices and coding conventions provides a set of standards for coding in Flex.

Setting up your development environment using the sample project

The Workbench installation includes a sample Flex project that you can use as a starting point for development. Once you have installed Adobe Flex Builder 3, you can import the project from the `reference` directory of your Workbench installation.

The sample project demonstrates the use of the `EditorModule` class, and is configured to use the `Experience ManagerEditorAPI.swc` as an external library.

Before starting development on your external editors, you should have an application provisioned in Workbench that you can use to test the sample editor. You can use the Content Assembler reference application for this purpose; see "Working with the Sample Application" in the *Oracle Endeca Experience Manager Getting Started Guide* for details.

To set up your development environment using the sample project:

1. Install Adobe Flex Builder 3 either as a standalone IDE or an Eclipse Plug-in.

   Ensure that you are developing against the latest stable build of Adobe Flex SDK 3.2. As of this writing, the recommended build is available from the following link: *http://opensource.adobe.com/wiki/display/flexsdk/download?build=3.2.0.3794&pkgtype=1*
2. Navigate to the `reference` subfolder of your Workbench installation and select the `pagebuilderEditorSDK.zip` archive.

   If you installed Workbench to the default location, this will be under `%ENDECA_TOOLS_ROOT%\reference` on Windows, or `$ENDECA_TOOLS_ROOT\reference` on UNIX.
3. Extract the archive to a location of your choice.
4. Open Flex Builder and select **File** > **Import** > **Flex Project...**.
5. Make sure the **Project folder** radio button is selected, and click **Browse**.

6. Navigate to the location where you extracted the archive in Step 3 and select the `sample-editor-project` directory.
7. Click **Finish**.
8. Confirm that the `Experience ManagerEditorAPI.swc` is configured as a dynamically linked library.
   a) In Flex Builder, right-click the **sample-editors** project and select **Properties** from the drop-down menu.
   b) Select **Flex Build Path** from the tree control in the left panel.
   c) Select the **Library path** tab from the **Flex Build Path** panel on the right.
   d) Expand the **libs** folder in the **Build path libraries** view.
   e) Ensure that the **Link Type** is set to **External**.
9. Click **Finish**.
   Flex Builder creates the new project.

Hosting the sample editor module

After you have built the editor module, you must host the module on a server that can be accessed by Experience Manager.

For simplicity, this procedure describes hosting the sample editor module in the Endeca Tools Service, which allows you to bypass some security configuration. This is sufficient for working with the sample project in a development environment.

Note: In a production environment, hosting editors in the Workbench container is unsupported, and causes them to compete with Workbench for resources.

To add your sample editor application to the Endeca Tools Service:

1. Navigate to `%ENDECA_TOOLS_ROOT%\server\webapps` on Windows, or `$ENDECA_TOOLS_ROOT/server/webapps` on UNIX.
2. Create a new directory, `sample_editor_app`.
3. Open the sample project in Flex Builder 3.
   a) Select the `sample_editor_app` directory you created in Step 2 as the output location.
   b) Build the sample editor SWF.
4. Navigate to `%ENDECA_TOOLS_CONF%\conf\Standalone\localhost` on Windows, or `$ENDECA_TOOLS_CONF/conf/Standalone/localhost` on UNIX.
5. Create a new context file inside the directory, `sample_editor_app.xml`, and include the following (Note that the `docBase` path given is for a default Windows installation):
```
<!-- Context file for the sample editor application -->
<Context
 path="/sample_editor_app"
 docBase="C:\Endeca\Workbench\2.1.0\server/webapps/sample_editor_app"
    debug="0"
    privileged="false">
</Context>
```
6. Save and close the file.

After you create the context configuration file and add the editor module to your `sample_editor_app` directory, you need to modify the sample editor configuration file.

Modifying the sample editor configuration file

You must update the editor configuration file to specify the location of the sample editor module.

To specify the current location of the editor module:

1. Navigate to `%ENDECA_TOOLS_ROOT%\reference` on Windows, or `$ENDECA_TOOLS_ROOT/reference` on UNIX.
2. Extract the sample editor configuration file, `sample_editors.xml`, from the `pagebuilderEditorSDK.zip` archive.

   The sample editor configuration file is located in the `pagebuilder-editor-sdk\sample-editor-config` directory.
3. Open the file.
4. Locate the line that specifies the url of the editor module, as in the example below:

   ```
   <EditorModule url="http://example.com:8080/test-editors.swf"
     crossDomainUrl="http://example.com:8080/crossdomain.xml">
   ```
5. Modify the `url` attribute to point to the new location of the editor module, for example:

   ```
   <EditorModule url="http://localhost:8006/sample_editor_app/sample_edi¬
   tors.swf">
   ```
6. Save the modified editor configuration file inside the `%ENDECA_TOOLS_ROOT%\server\webapps\sample_editor_app` directory on Windows, or `$ENDECA_TOOLS_ROOT/server/webapps/sample_editor_app` directory on UNIX.

After updating the editor configuration file, you must specify its location in `webstudio.properties`.

Modifying webstudio.properties to enable the sample editor

In order for Experience Manager to use the sample editor configuration file, you must specify the location of the file in `webstudio.properties`.

To specify the location of the editor configuration file:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf/` (on UNIX).
3. Open the `webstudio.properties` file.
4. Find the line that specifies the location of the editor configuration file, for example:

   ```
   # The URL from which to load editor configuration for Experience Manager
   #com.endeca.webstudio.pagebuilder.editors.config=http://my.compa¬
   ny.com/2009/myeditors.xml
   ```
5. Uncomment and change it to point to the absolute URL of your editor configuration file, `sample_editors.xml`:

   ```
   # The URL from which to load editor configuration for Experience Manager
   com.endeca.webstudio.pagebuilder.editors.config=http://localhost:8006/sam¬
   ple_editor_app/sample_editors.xml
   ```
6. Save and close the file.
7. Start the Endeca Tools Service.

Incorporating the sample cartridge

The Workbench installation includes a sample cartridge that uses the sample editor. You can add it to your application's template directory and use the emgr_update utility to upload it to Experience Manager.

To add the sample cartridge to your Web application:

1. Navigate to `%ENDECA_TOOLS_ROOT%\reference` on Windows, or `$ENDECA_TOOLS_ROOT/reference` on UNIX.
2. Navigate to the location where you extracted the `pagebuilderEditorSDK.zip` archive.
3. Copy the sample cartridge, `SidebarItem-RichText.xml`, from the `pagebuilder-editor-sdk\sample-page-templates` directory to the template directory of your application.
4. Run the emgr_update utility to upload the templates to Experience Manager.

   The example below is for the Content Assembler Reference Application installed with defaults in Windows:

   ```
   emgr_update.bat --action set_templates --host localhost:8006
   --app_name ContentAssemblerRefApp --dir C:\Endeca\apps\ContentAssembler¬
   RefApp\config\page_builder_templates
   ```
5. Log in to Workbench.
6. Create or modify a landing page to use the RichTextEditor Sidebar Item.

Note: If you encounter a security sandbox violation error when Experience Manager tries to load your editor, confirm that your security certificates are set up correctly and that your browser accepts the validity of your certificates.

About developing editors

This section includes information on writing editors in Flex and MXML, as well as editors that use Flex to communicate with JavaScript.

The editor creation workflow

This section provides an overview of the steps required to create and use an editor.

The steps involved in creating and installing an editor are as follows:

1. Create or open a Flex Builder project.

   The Workbench installation includes a sample project that you can use for editor development against the Experience Manager SDK.
2. Create a new Flex component or modify an existing editor.

   This component should implement the `IEditor` interface.
3. In the MXML that represents your `EditorModule`, add an entry for your editor similar to the following:

   ```
     <editor:EditorEntry
       uri="http://endeca.com/sample/2010"
   ```

```
    localName="MyEditor"
    editor="com.endeca.tools.pagebuilder.samples.editors.MyEditor" />
```

4. Configure `Experience ManagerEditorAPI.swc` as a dynamically linked library by setting the link type to **External** in Flex Builder.
5. Build the Flex project to generate the editor module.
6. Upload your editor module to a chosen host.

   Note: If you are hosting your editor module on a different security domain from the editor configuration file, you must also create a cross-domain policy file for the editor module.

7. Specify the location of the editor module (and its cross-domain policy file, if you created one) in your editor configuration file.
8. Edit the `webstudio.properties` file, located in `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX) to specify the following:
   - The location of the editor configuration file
   - The cross-domain policy file for the editor configuration file

   Unless you change the location of the editor configuration file or its associated `crossdomain.xml` file, this step only needs to be taken once.
9. Add the editor to a Experience Manager template, either by creating a new template, or modifying an existing one.

   Note that the element name and namespace together form the QName that you specified for the editor in the editor configuration file.
10. Use the emgr_update utility to upload your new or updated templates to Experience Manager.
11. Test the editor by logging into Experience Manager.

    If the editor is valid and configured properly in Experience Manager, it displays in the **Content Detail Panel** when the appropriate cartridge is selected.

Note the following:

- If you make changes to `webstudio.properties`, you must restart the Endeca Tools Service.
- If you update the editor configuration file, you need to reload the main Experience Manager SWF.
- If you update the editor module, you need to reload the main Experience Manager SWF.
- Changes to the editor SWF may not be loaded if your browser is using a cached version of the SWF. Check your browser settings for options related to checking for new versions of cached content, or clear your cache manually to resolve the issue.

Related Links

File hosting and security considerations on page 62
: Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Uploading templates to Experience Manager on page 53
: Before Experience Manager users can access new templates, you must upload them using the emgr_update utility.

About modifying templates that are used by existing pages on page 54
: During the development and testing phase of your application deployment, you may need to make adjustments to your page or cartridge templates and update them in Experience Manager.

Specifying editors for Experience Manager on page 92

The editor configuration file lists the editors available to Experience Manager.

Enabling the editor configuration file on page 94
: In order for Experience Manager to use the editor configuration file, you must set up a `crossdomain.xml` file and specify the locations of both the cross-domain policy file and the editor configuration file in `webstudio.properties`.

Using editors in templates on page 95
: Adding editors to a template is similar to adding standard editors, except that you must specify a non-Endeca namespace for your editor name.

About the Experience Manager editor life cycle

This section provides an overview of the processes within Experience Manager that govern the instantiation and life cycle of editors.

1. At startup, Experience Manager populates an internal editor registry based on the editor configuration file specified in `webstudio.properties`:

   ```
   # The URL from which to load editor configuration for Experience Manager
   com.endeca.webstudio.pagebuilder.editors.config=http://my.compa¬
   ny.com/2009/myeditors.xml
   ```

2. When the content administrator navigates to the **Edit View** and selects a content item in the **Content Tree**, Experience Manager loads editors according to the information in the `<EditorPanel>` element of the template XML:

   ```
   <EditorPanel>
     <BasicContentItemEditor>
       <GroupLabel label="Page metadata"/>
       <CheckBox propertyName="autogen_meta" label="Auto-generate meta in¬
   formation" />
       ...
       <MyEditor xmlns="http://my.company.com/2009" propertyName="meta_de¬
   scription" label="Meta description" enabled="true" height="72"/>
     </BasicContentItemEditor>
   </EditorPanel>
   ```

3. When Experience Manager encounters an editor in the `<EditorPanel>` element, it concatenates the namespace and name to form a qualified name:

   ```
   <MyEditor xmlns="http://my.company.com/2009"... />
   ```

4. Experience Manager then refers to the editor registry in memory, and searches for the first match to the qualified name in order to identify the correct editor module:

   ```
   <EditorModule url="http://localhost/foo/bar/myeditors.swf">
     <editor name="http://my.company.com/2009:MyEditor"/>
   </EditorModule>
   ```

5. Experience Manager reads the `url` attribute of the `<EditorModule>` element to locate the community editor module:

   ```
   <EditorModule url="http://localhost/foo/bar/myeditors.swf">
     <editor name="http://my.company.com/2009:MyEditor"/>
   </EditorModule>
   ```

6. Experience Manager loads the editor module.

   Note: Once an editor module has been loaded, Experience Manager does not unload or reload it unless you reload the Experience Manager SWF. Browser caching settings may interfere with this behavior. For more information, please see the "Troubleshooting an editor" section.

7. Experience Manager uses the newly created editor module to instantiate the appropriate `editor` object.
8. After instantiating the editor, Experience Manager sets the `info`, `factory`, and `contentItem` properties on it.

   Note: Experience Manager always passes an editor the entire `ContentItem`, so that the editor can modify multiple properties if desired.

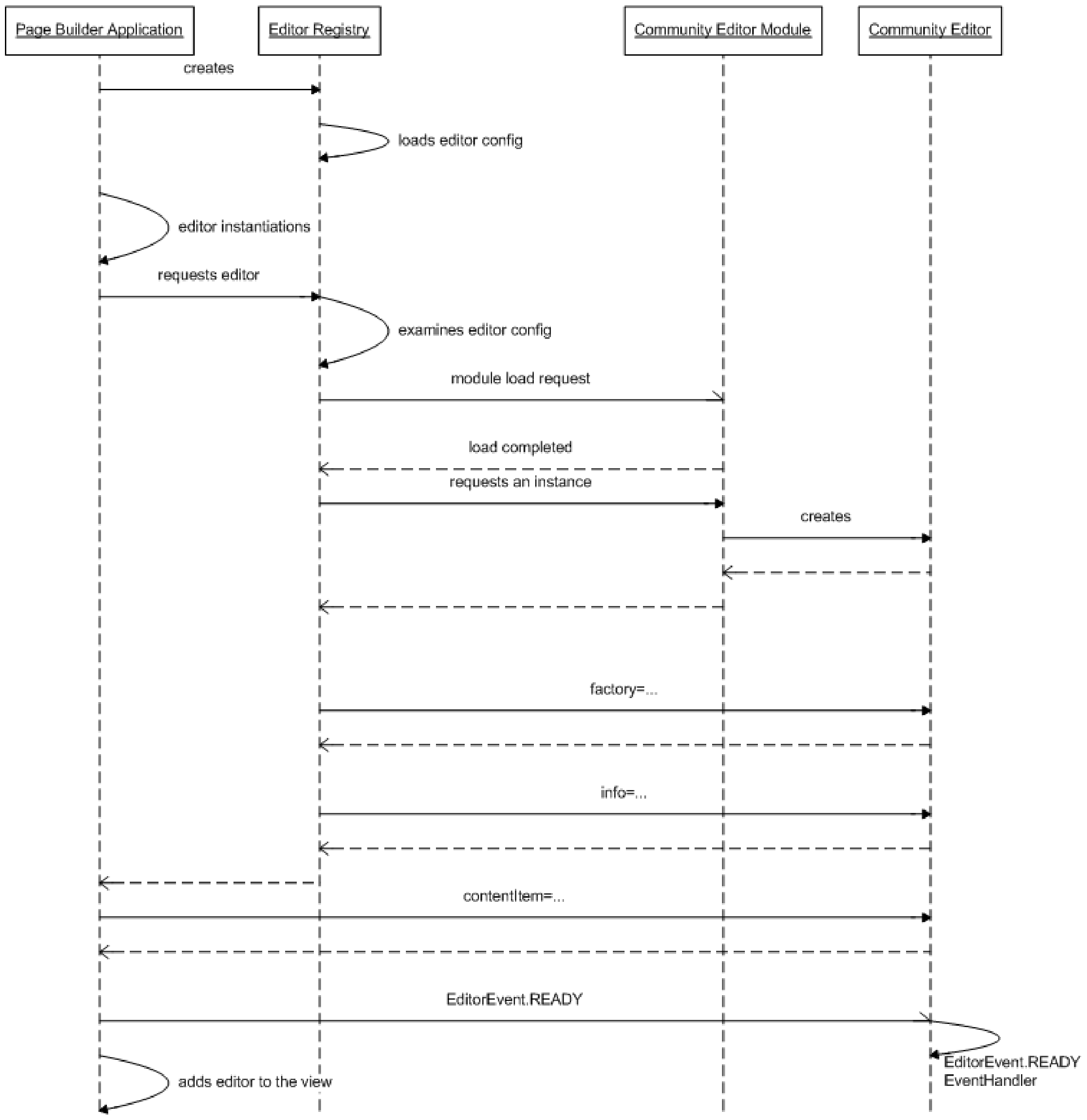
9. Once all three properties have been set, Experience Manager dispatches an `editorReady` event. Typically, editors listen for this event to retrieve and validate specific properties from the content item.
10. Experience Manager adds the rendered editor to the **Content Detail Panel** on the right side of the **Content Editor Pane**.
11. The content administrator uses the editor interface to make any desired changes. Saved changes are serialized into the content XML that represents the landing page.
12. All editor objects are destroyed when the content administrator selects another content item in the content tree or navigates away from the **Edit View**.

Related Links

Ensuring Experience Manager loads the current editor modules on page 98
After you update a hosted editor module, Web browsers may continue to load cached versions of the old editor module.

Experience Manager editor life cycle diagram

Below is a sequence diagram that depicts the Experience Manager editor life cycle.



About editor instantiation

Generally, you should set your editor to listen for the `EditorEvent.READY` event that Experience Manager dispatches when an editor is instantiated.

The `EditorEvent.READY` event signals that an editor can access its configuration to retrieve and validate whatever properties it manipulates on the content item. This event is only dispatched after Experience Manager has set the following properties on a newly instantiated editor:

`factory`	The factory object of type `IContentModelFactory`, which the editor can use to create content model objects.

`info`	An object of type `IEnvironmentInfo` that includes environmental and configuration information, which Experience Manager uses to determine how to render the editor.
`contentItem`	The `IContentItem` object that contains the property or properties that the editor modifies. The `getPropertyByName` method on the `ContentItem` object can be used to retrieve a property by name.

The sample rich text editor uses the `editorReadyHandler()` method below, which retrieves the property name that the editor is linked to, and sets up a listener for `ContentModelEvent.CHANGE` events to monitor the property for changes:

```
private function editorReadyHandler(event:EditorEvent):void
{
  if (contentItem != null && info.templateConfig != null)
  {
    var prop:IProperty = contentItem.getPropertyByName(info.templateCon¬
fig.@propertyName);
    property.addEventListener("contentModelChange", contentModelChangeHan¬
dler());
  }
}
```

About configuring editors

You can specify editor configuration information in one of three places.

When configuring your editors, you should consider the impact of specifying configuration information in one of the three available locations:

- **Hardcoded** — specifying configuration in the editor source ensures consistent behavior across all instances of that editor.
- **Editor Configuration File** — specifying configuration in the editor configuration file results in consistent behavior for a given Workbench installation. This is useful for information that may vary across installations, such as URL and user credentials for a third-party service.
- **Landing Page Template** — adding configuration information to the template ensures consistency for that template only.

Experience Manager sets global, editor-specific, and template-specific configuration information as properties of an `IEnvironmentInfo` object when an editor is instantiated.

Related Links

Adding editor configuration information to the editor configuration file on page 93
  You can specify arbitrary configuration information in the editor configuration file. This can be done on a global or per-editor basis.

Using editors in templates on page 95
  Adding editors to a template is similar to adding standard editors, except that you must specify a non-Endeca namespace for your editor name.

About retrieving editor configuration information

The `info` property provides information regarding the environment in which an editor is created. This includes global, template-specific, and editor-specific configuration.

Each of the three configuration types is returned as an XML object:

Object	Usage

`globalEditorCon¬fig`	This includes configuration specified in the editor configuration file on a global level, either as attributes or nested elements of the `<GlobalEditorConfig>` tag:
	`<EditorConfig xmlns="...">`
	`  **<GlobalEditorConfig username="u" password="p">**`
	`    **<ExampleCustomGlobalConfigXML />**`
	`  **</GlobalEditorConfig>**`
	`  ...`
	`  <Editor ...>`
	`  </Editor>`
	`</EditorConfig>`

`editorConfig`	This includes configuration specified in the editor configuration file on a per-editor basis:
	`<Editor name="http://my.company.com/2009/editors:MyEdi¬tor">`
	`  **<EditorConfig>**`
	`    **<Arbitrary foo="bar" size="10" resizeable="false"/>**`
	`  **</EditorConfig>**`
	`</Editor>`

`templateConfig`	This includes configuration that is specified in a landing page template as attributes of the editor tag: `<MyEditor xmlns="..." **propertyName="meta_description" label="Meta description" enabled="true" height="72"**/>`

For example:

```
/*
* The property name is stored in the propertyName attribute of the
* templateConfig (in this case, the RichTextEditor element in the
* page template). In the sample template the RichTextEditor
* templateConfig looks like this:
*
* <RichTextEditor propertyName="rich_text"
*    xmlns="http://endeca.com/sample/2010" />
*/
  var prop:IProperty = contentItem.getPropertyByName(info.templateCon¬
fig.@propertyName);
```

For more details about the `IEnvironmentInfo` interface, refer to the *Experience Manager Editor API Reference*.

Related Links

About configuring editors on page 81
You can specify editor configuration information in one of three places.

About retrieving the Experience Manager MDEX host name and port

The `info` property includes the Experience Manager MDEX host name and port as part of the environment information it provides to an editor.

Experience Manager refers to the MDEX properties specified in the Endeca Application Controller for MDEX hostname and port information. You can access these properties on the `IEnvironmentInfo` object, as with the example below:

```
var hostname:String = info.webstudioMDEXHostname;
var port:uint = info.webstudioMDEXPort;

queryMDEX(hostname, port, query);
```

For more information on specifying which MDEX Engine to use with Workbench, see the *Oracle Endeca Workbench Administrator's Guide*.

About binding an editor to multiple properties

You can bind an editor to multiple properties by specifying multiple property attributes in the `<editor>` element of a landing page or cartridge template.

For example, a landing page template can include an editor element with multiple properties specified as attributes, as in the example below:

```
<MyEditor xmlns="..." propOne="foo" propTwo="bar"/>
```

After your editor is instantiated, you can retrieve the names of these properties from the `template¬Config` property of the `IEnvironmentInfo` object. With these names, you can access the property objects by calling the `ContentItem.getpropertyByName()` method:

```
var propOne:IProperty = contentItem.getPropertyByName(info.templateCon¬
fig.@propOne);
var propTwo:IProperty = contentItem.getPropertyByName(info.templateCon¬
fig.@propTwo);
```

Note: If you need to be notified of any changes to either property, you can listen for content model events on the content item that has these two properties.

Working with the content model

To modify any content model object, you must retrieve it from the content item after your editor has been instantiated.

Experience Manager instantiates an `IContentItem` object based on the `<ContentItem>` definition in the template and sets it as the `contentItem` property on the editor. Editors cannot add or remove properties from the content item.

The code below assumes that the name of the property containing the object is specified in the `propertyName` attribute of the editor configuration in the template. The value of the `propertyName` attribute from the template configuration is retrieved from the `info` property, assigned to a variable, and passed into the content item `getPropertyByName()` method.

```
// Retrieve the name of the property to be used by this
// editor from the template XML.
var propertyName:String = info.templateConfig.@propertyName;

// Now, retrieve the property itself from our content item.
```

```
var prop:IProperty = contentItem.getPropertyByName(propertyName);

// If the property is null or the wrong type, then the template has
// been misconfigured and there is nothing we can do here except
// dispatch an error event.
if (prop == null || !(prop is INavigationRecordsProperty))
{
      dispatchEvent(EditorEvent.createErrorEvent("Invalid property"));
      return;
}

// Otherwise, we can retrieve and use the value of the property.
var property:INavigationRecordsProperty = INavigationRecordsProperty(prop);

displayToUser(property.navigationRecords);

// We can also replace it with a new value which we create using
// the content model factory...
if (property.navigationRecords == null)
{
      property.navigationRecords = factory.createNavigationRecords();
}

// ...and modify it.
property.navigationRecords.recordsPerPage = 15;
```

For more information, see "About the content model interfaces" in the *Experience Manager Editor API Reference*.

Related Links

About retrieving editor configuration information on page 81
The `info` property provides information regarding the environment in which an editor is created. This includes global, template-specific, and editor-specific configuration.

About handling error conditions on page 88
An editor dispatches an `EditorEvent.ERROR` event ito display an error message in place of the editor.

Working with IRecordListProperties

Unlike other objects in the content model, the `IRecordListProperty` object does not simply contain a `RecordList` property. Instead, it has a `query` property that can be of type `INavQuery` or `IRecordQuery`.

A community editor can create both record queries and navigation queries for a record list property:

- A **navigation query** populates the record list property by returning all records that match a selected navigation state (dynamic records).
- A **record query** populates the record list property by taking a specific list of record IDs (featured records).

Each query type contains query parameters that the Content Assembler can use to populate the record list before returning the results to your application.

In order to specify queries, you must first retrieve the `IRecordListProperty` from the content item after your editor has been instantiated. The code below assumes that the name of the property is specified in the `propertyName` attribute of the editor configuration in the template:

```
// Get the property associated with this editor in the template XML
var prop:IProperty = contentItem.getPropertyByName(info.templateConfig.@prop¬
ertyName);
var property:IRecordListProperty = IRecordListProperty(prop);
```

Below is an example of how you can add a navigation query to a record list property:

```
//Create a new navigation query for this property
var navQuery:INavQuery = factory.createNavQuery();
navQuery.dimensionValues.addItem(factory.createDimensionValue("5564","12"));
navQuery.maxRecords = 12;
navQuery.sort = factory.createRecordSort("Price", true);
property.query = navQuery;
```

The following example shows how you can add a record query to a record list property:

```
//Create a new record query for this property
var recordQuery:IRecordQuery = factory.createRecordQuery();
recordQuery.recordIds.addItem("16342");
recordQuery.recordIds.addItem("28314");
property2.query = recordQuery;
```

For additional information about record list properties and the query objects, see the *Experience Manager Editor API Reference*.

About handling changes to the content model

Experience Manager dispatches a `ContentModelEvent.CHANGE` when an object in the content model is changed in any way.

Listening for content model change events

Note that a `ContentModelEvent.CHANGE` event dispatched within the content model hierarchy bubbles up, triggering any listeners on objects higher up in the hierarchy.

This event includes the following information:

- The `source` of the change within the content model
- The `property` on an object that was changed, including the `oldValue` and `newValue`
- The `kind` of change; possible values are `UPDATE`, `ADD`, `MOVE`, `REMOVE`, `REPLACE`, `REFRESH`, `RESET`, and `null`

For more details about the `ContentModelEvent` class, refer to the *Experience Manager Editor API Reference*.

You can rely on standard Flex bindings to update property values with any changes. The example below shows a standard checkbox component whose `selected` property is bound to the `boolean` property of a `BooleanProperty` object:

```
<mx:CheckBox id="propertyValueCheckBox"
change="booleanProperty.boolean = propertyValueCheckBox.selected;"
selected="{booleanProperty.boolean}" />
```

The `selected` property binding ensures that whenever a `ContentModelEvent.CHANGE` event is dispatched by `mBooleanProperty`, the check box `selected` value is updated to reflect the new value.

Note: All content model objects that represent collections of objects (such as `Record¬Query.recordIds`) implement the `mx.collections.IList` interface. You can bind these objects in the same fashion as any other built-in Flex construct that implements an `IList` interface.

If you want to call a method in response to content model changes, you can use an event listener to monitor a property for changes. For example, if you have an editor that displays an image preview, you can add a listener on the property or properties that represent the image URL so that you can update the preview to reflect any changes.

Dispatching content model change events

Experience Manager dispatches events for changes to all objects in the content model. This includes objects of type `IXMLProperty` when the `xml` property is set directly. However, changes to the underlying XML object itself do not trigger an event. If you modify the object in this manner, you must dispatch a `ContentModelEvent.CHANGE` event as part of your code:

```
/*
* Setting the "xml" property of an IXMLProperty object automatically
* dispatches an event with type ContentModelEvent.CHANGE.
*/
 xmlProperty.xml = foo;

/*
* Making changes to the object which is referred to by the "xml" property
* does not dispatch a ContentModelEvent.CHANGE event. You must dispatch
* the event manually.
*/
xmlProperty.xml.bar[0] = <baz/>;
xmlProperty.dispatchEvent(new ContentModelEvent(
   ContentModelEvent.CHANGE, null, null, null, null, null, xmlProperty.xml));
```

Working with custom XML properties

You can use the `IXMLProperty` interface to retrieve and manipulate custom XML properties.

Adding custom XML to your templates enables you to work outside of the standard content object model. For example, you can:

- Create properties similar to standard Endeca properties, such as a `MyRecordList` property with a roll-up key for aggregate records.
- Pass in any other information your application requires, such as configuration for a third-party application.

To add custom XML to an editor:

1. Create a `<Property>` tag inside the `<ContentItem>` in your template.
2. Add custom XML to the `<Property>` element as nested attributes:

   ```
   <ContentItem>
     <Property name="custom">
       <Milkshake xmlns="http://mycompany.com/schema/2010/data"
         flavor="vanilla"/>
     </Property>
   </ContentItem>
   ```

   Explicitly define a namespace for your custom XML that is not one of the following Endeca namespaces:

   - http://endeca.com/schema/content/2008
   - http://endeca.com/schema/content-template/2008
   - http://endeca.com/schema/content-tags/2008

Note: Because Experience Manager does not perform validation on XML that is not in one of the Endeca namespaces, Oracle recommends validating any templates that contain custom or pass-through XML properties prior to uploading them.

3. Add your editor to the `<EditorPanel>` and bind it to your custom XML property:

```
<EditorPanel>
  <BasicContentItemEditor>
    <CustomXMLEditor propertyName="custom" xmlns="http://mycompa¬
ny.com/schema/2010/editors"/>
  </BasicContentItemEditor>
</EditorPanel>
```

4. After your editor is instantiated and its properties are set, Experience Manager dispatches the `ed¬itorReady` event. To modify the XML in an `IXMLProperty`, use the `getPropertyByName` method and specify the `propertyName` from the `<EditorPanel>` that ties your editor to the custom XML property. You can set the XML directly, or modify the underlying object:

```
var customProperty:IXMLProperty = IXMLProperty(contentItem.getPropertyBy¬
Name(info.templateConfig.@propertyName));

/*
* Set the XML directly. Experience Manager automatically dispatches a
* contentModelChange event
*/
customProperty.xml = <Milkshake xmlns="http://mycompany.com/schema/2010/da¬
ta" flavor="vanilla">
                       A tall, frosty beverage
                     </Milkshake>

/*
* Modify the underlying object and manually dispatch a
* contentModelChange event
*/
customProperty.xml.@flavor = "banana";
customProperty.dispatchEvent(new ContentModelEvent(ContentMod¬
elEvent.CHANGE, null, null, null, null, null, customProperty.xml));
```

Note: Keep in mind when modifying the XML property that only the setter method automatically dispatches a `ContentModelEvent.CHANGE` event. The content model cannot detect internal changes to an XML object. If you modify the XML data by reference, you must manually dispatch a `ContentModelEvent.CHANGE` event.

The custom XML is passed to the Content Assembler as part of the overall page configuration. You may also consider writing a tag handler for your custom XML. A tag handler enables you to extend the processing logic in the Content Assembler before it returns the content results to your application. For more information, see the *Content Assembler API Developer's Guide*.

Related Links

Scenarios for extending Experience Manager and the Content Assembler on page 66
You can use either community editors on their own, community tag handlers on their own, or both of them in combination to extend the functionality of Experience Manager.

About read-only mode

If a user does not have sufficient privileges to make modifications to a landing page, or a selected landing page group is locked by another Workbench user, Experience Manager enters read-only mode. Your editor needs to respect read-only mode by placing any UI controls it contains into a read-only mode as well.

When a landing page is in read-only mode, all its child content items and properties are also in read-only mode. Attempting to modify them results in a runtime exception.

You can place your UI controls in read-only mode by binding any relevant attributes to the `readOnly` property of the content item or any of its child objects in the Flex code. Which attributes you bind in this manner will depend on the UI control you are using. See the Adobe Flex documentation for details.

The example below shows a `TextArea` UI control. The `enabled` and `editable` attributes are set to the negated value of the string property's `readOnly` property (which is inherited from the content item):

```
<mx:TextArea id="propertyValueTextInput"
  text="{stringProperty.string}"
  enabled="{!stringProperty.readOnly}"
  editable="{!stringProperty.readOnly}"
  change="stringProperty.string = propertyValueTextInput.text;" />
```

The `TextArea` is only editable if the content item associated with the editor is not in read-only mode.

Note: If your UI controls allow the modification of read-only objects, and the content administrator tries to modify those objects, Experience Manager throws an error that is presented to the content administrator.

About handling error conditions

An editor dispatches an `EditorEvent.ERROR` event ito display an error message in place of the editor.

You can do this in the case of an irrecoverable error, for example, when the requisite property on a content item does not exist, or the property exists, but is of the wrong type.

Below is an example that dispatches an `editorError` event:

```
dispatchEvent(EditorEvent.createErrorEvent(errorMessage));
```

About handling conflicts between content and editor definitions

You can dispatch a `ContentModelEvent.UPGRADE` event to display a message box that alerts the content administrator of changes to the content.

When Experience Manager populates the **Content Detail Panel** for a landing page or cartridge, it checks the content XML of the loaded page against the template XML. If the template has been changed such that it is no longer compatible with the content, Experience Manager displays a warning and updates existing content that is bound to standard editors to fit the current template definition.

You can add similar logic to your editor, and display a warning by using the eventing model.

Note that you should dispatch `ContentModelEvent.UPGRADE` events from the `IProperty` that has been modified. The events bubble up the hierarchy to the appropriate content model object.

Below is an example:

```
property.dispatchEvent(ContentModelEvent.createContentModelUpgradeEvent(prop¬
erty));
```

Related Links

About modifying templates that are used by existing pages on page 54
During the development and testing phase of your application deployment, you may need to make adjustments to your page or cartridge templates and update them in Experience Manager.

About writing editor components outside of Flex

An editor can use the Flex `ExternalInterface` API to enable a Web application to interact with the Experience Manager SWF. This is one method of implementing external editor functionality with minimal use of ActionScript.

Due to JavaScript domain issues, any Web application that uses JavaScript to interact with the Experience Manager SWF must be hosted on the same domain as Workbench. These are the same restrictions that exist for the preview application in Workbench.

Calling JavaScript from ActionScript

`ExternalInterface` includes a `call()` method that allows ActionScript to call a function in the container application. This method requires at least one parameter:

- The first, required parameter is the name of the function to call.
- Any additional parameters are passed, in order, as arguments to the function being called.

The example below calls the JavaScript `window.open()` method to open a window that displays a Web application:

```
ExternalInterface.call("window.open", <external application URL>,
  <window title>,<JavaScript window.open arguments>);
```

If you use this approach, you should specify the Flex portion of the editor in the editor configuration file.

Calling ActionScript from JavaScript

The Flex `ExternalInterface` class provides an `addCallback()` method that registers an ActionScript function so that it can be called by JavaScript. This method takes two parameters:

- A string, which is the function name you call from your JavaScript
- An ActionScript function, which executes when your Web application makes a JavaScript call matching the specified string

The following example shows how you can combine the use of `addCallback()` and `call()` to communicate with your Web application via JavaScript.

The editor includes the ActionScript below:

```
private function openExternalWindow(event:Event):void
{
  // When addRecord() is called on the editor, the
  // list of records in Experience Manager editor is
  // updated to include the given record if it doesn't already.

  // Calling ExternalInterface.addCallback() adds a JavaScript
  // callback to the Experience Manager SWF.
  ExternalInterface.addCallback("addRecord",
    function(recordSpec:String):void
    {
      if( !records.contains(recordSpec)) {
      records.addItem(recordSpec);
    }
    property.dispatchEvent(new Event("featuredRecordsChange"));
    });
    // Open the window. Calling the JavaScript window.open()
    // method opens a window displaying the Web application
    // you want to use to call addRecord() on the Experience Manager SWF.
    ExternalInterface.call("window.open", http://mycompany.com/we¬
bapp/record_browser, "featuredRecordsEditor",
      "menubar=1,resizeable=1,width=1000,height=800,loca¬
tion=yes,menubar=yes,scrollbars=1,toolbar=1");
}
```

In the Web application, the call to `addRecord()` is treated as though a local function is being called. The following JavaScript is used to execute this call on the Experience Manager SWF:

```
function addRecord(recordSpec) {
  var movie;
  try {
    // the Experience Manager movie is called “pagebuilder”. Firefox and
    // IE provide different ways to access it – check for both
    // Firefox and IE

    // Check IE
    if( window.opener.pagebuilder != undefined)
    {
      movie = window.opener.pagebuilder;
    }
    // Check Firefox
    else
    {
      movie = window.opener.document.pagebuilder;
    }
    // Call the JavaScript method addRecord() on the Experience Manager SWF

    movie.addRecord(recordSpec);
  }
  catch(e) {
    alert("error opening window or movie: " + e.description);
  }
}
```

Note: For additional information about the `ExternalInterface` class, consult the Adobe documentation.

Related Links

Adding editor configuration information to the editor configuration file on page 93

You can specify arbitrary configuration information in the editor configuration file. This can be done on a global or per-editor basis.

About JavaScript domain considerations

Web applications that use JavaScript to interact with the Experience Manager SWF rely on the same mechanism for managing the JavaScript domain as the preview application. You must host any such applications on the same domain as Workbench.

In order to enable JavaScript communication, you must ensure that the `document.domain` in both the Workbench preview application and your editor application are set to the same value. This value must be the first common domain shared by Workbench, the preview application, and your editor.

For example, if the JSP editor is on `foo.company.com`, the preview application is on `bar.company.com`, and Workbench is on `bar.company.com`, both the editor domain setting and the preview application domain setting must be set to `company.com` :

```
document.domain = "company.com";
```

Additionally, if either the Workbench or editor container is running in SSL mode, the other must be set to SSL mode as well for the two to communicate. Editors written in Flex do not have this limitation.

The **Preview App Settings** page in Endeca Workbench provides a field where you must enter the JavaScript domain information for the preview application. This is analogous to declaring the domain in your Javascript headers. You must enter this information correctly regardless of whether you are actually using the preview application. This setting also applies to your editor application.

For more information about configuring JavaScript domain settings in the preview application, consult the *Oracle Endeca Workbench Administrator's Guide*.

Related Links

About using SSL and editors on page 96

If your Workbench server is running in SSL mode, you must enable communication with any remote editor modules by configuring the module server's cross-domain policy file.

About client-server communication in editors

After Experience Manager loads editor modules, the Flash player treats module code as if it originates from the host where Experience Manager is served, regardless of the location where the editor module is hosted. If your editor includes client-server communication, you need to set up a cross-domain policy file that allows the instance of the Experience Manager client to talk to your server-side code.

For security purposes, when the modules are first loaded, Experience Manager checks the cross-domain policy file for the location where the editor module is hosted.

Related Links

File hosting and security considerations on page 62

Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

About debugging an editor

Because the Experience Manager SWF is not compiled and shipped in debug mode, compiling your editor modules in debug mode does not expose any additional information. You can write your own application SWF for step-through debugging purposes. Additionally, Oracle recommends that you use a debugging version of Flash Player.

Setting up a debug-mode application SWF

When a module in debug mode is loaded by an application SWF that was not compiled in debug mode, debugging information for that module is not exposed. You can debug your editors by compiling your own simple application SWF in debug mode, loading in a debugging version of your editor module, and supplying the editor with a mock content item.

Using a debugging version of Flash Player

Regardless of whether you are using a debug-mode application SWF, a debugging version of Flash Player displays a dialog box when it encounters an ActionScript error in your code, giving you some useful traceback information. A non-debugging Flash player fails silently and can potentially leave your application in an incomplete state.

You may find the following resources helpful for setting up a debugging version of Flash Player:

- *http://www.playerversion.com* — This site is a good place to look for various versions of Flash Player.
- *http://www.pluginswitcher.de* — The Kewbee Plugin Switcher tracks the versions of all your installed players, and allows you to revert to earlier installed versions. It can differentiate between debugging and non-debugging players.

Installing an editor

In order to use an editor module in Experience Manager, it must be described in the configuration file and hosted in a location where Experience Manager can access it.

You can specify your editor modules within the editor configuration file. To enable them, you must configure the `webstudio.properties` file to point to the configuration file. In order for content administrators to use a new editor to configure content in Experience Manager, you must create a template that uses that editor.

Specifying editors for Experience Manager

The editor configuration file lists the editors available to Experience Manager.

You can specify multiple editor modules in your editor configuration file; each module is associated with one or more editors and contains the factory methods for instantiating those editors.

To specify editor modules:

1. Insert an `<EditorModule>` element within `<EditorConfig>`.
2. Set the `url` attribute of the `<EditorModule>` element to the fully qualified URL of the compiled editor module you wish to use. For example:

   ```
   <EditorModule url="http://localhost:8080/foo/bar/myeditors.swf">
       ...
   </EditorModule>
   ```

3. Insert an `<editor>` element inside the `<EditorModule>` element for each editor within the module that you want to enable.
4. Set the `name` attribute of each `<editor>` element to a qualified name that includes the namespace and the local name for the editor. For example:

   ```
   <editor name="http://my.company.com/2009/editors:MyEditor"/>
   ```

The specified local name must exactly match the one to be used in Experience Manager templates. For the above example, the `<EditorPanel>` element in a template that used `MyEditor` would have to include a line similar to the one below:

```
<MyEditor xmlns="http://my.company.com/2009/editors" propertyName="foo"/>
```

Note: In the case of multiple editors with the same qualified name, Experience Manager defaults to the first instance of a given editor. Oracle recommends unique qualified names for each editor.

5. Repeat steps 2-5 for any additional editor modules.
6. If your editor module is hosted on a different security domain from the editor configuration file, you must specify the location of its cross-domain policy file. Insert a `crossDomainUrl` attribute inside the `<EditorModule>` element and specify the absolute URL to the `crossdomain.xml` file, as in the example below:

```
<EditorModule url="http://<host>:<port>/foo/bar/myeditors.swf"
  crossDomainUrl="http://<host>:<port>/foo/bar/crossdomain.xml">
  ...
</EditorModule>
```

Note: The cross-domain policy file may be located on the server root, or within another directory on the server, depending on your server configuration. Consult the Adobe Flash documentation for additional information.

7. Save and close the file.
8. Upload the editor configuration file to a server where Experience Manager can access it.

The following example shows a sample editor list with a cross-domain policy file specified:

```
<?xml version="1.0" encoding="utf-8"?>
<EditorConfig>
  <EditorModule url="http://localhost:8080/foo/bar/myeditors.swf"
    crossDomainUrl="http://localhost:8080/foo/bar/crossdomain.xml">
    <Editor name="http://my.company.com/2009/editors:MyEditor"/>
    <Editor name="http://my.company.com/2009/editors:MyOtherEditor"/>
  </EditorModule>
</EditorConfig>
```

You should validate your editor configuration file against the schema file, `editor-config.xsd`, to ensure that it conforms to the schema definition. This file is available in the `pagebuilder-editor-sdk\schema` directory of the `pageBuilderEditorSDK.zip` archive, located in the `reference` directory of your Workbench installation.

Related Links

File hosting and security considerations on page 62
Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Adding editor configuration information to the editor configuration file

You can specify arbitrary configuration information in the editor configuration file. This can be done on a global or per-editor basis.

Adding editor configuration information to the editor configuration file results in the same behavior for all instances of a specific editor in a given Workbench installation. Global configuration applies to all editors for a given Workbench installation.

To add configuration information to the editor configuration file:

1. Stop the Endeca Tools Service.
2. Navigate to your editor configuration file.
3. To add configuration information to a single editor:
   a) Insert an `<EditorConfig>` element directly inside the `<Editor>` tag of the editor you wish to modify.
   b) 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="http://my.company.com/2009/editors:MyEditor">
        <EditorConfig>
          <Arbitrary foo="bar" size="10" resizeable="false"/>
        </EditorConfig>
      </Editor>
      ```

4. To add global configuration information:
   a) Insert a `<GlobalEditorConfig>` tag directly inside the main `<EditorEditorConfig>` element.
   b) Add your arbitrary configuration information.

      The example below includes configuration both as attributes of `GlobalEditorConfig`, and as nested elements. You may use either approach, or both:

      ```
      <EditorConfig xmlns="...">
        <GlobalEditorConfig username="u" password="p">
          <ExampleCustomGlobalConfigXML />
        </GlobalEditorConfig>
        ...
        <Editor ...>
        </Editor>
      </EditorConfig>
      ```

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

The editor can access this information through the `editorConfig` and `globalEditorConfig` properties of the `IEnvironmentInfo` object.

Related Links

About retrieving editor configuration information on page 81
The `info` property provides information regarding the environment in which an editor is created. This includes global, template-specific, and editor-specific configuration.

Enabling the editor configuration file

In order for Experience Manager to use the editor configuration file, you must set up a `crossdomain.xml` file and specify the locations of both the cross-domain policy file and the editor configuration file in `webstudio.properties`.

Before updating `webstudio.properties` in your server configuration, make sure you have uploaded both the editor configuration file and its cross-domain policy file to the desired location.

To specify the location of the editor configuration:

1. Stop the Endeca Tools Service.
2. Navigate to `%ENDECA_TOOLS_CONF%\conf` (on Windows) or `$ENDECA_TOOLS_CONF/conf` (on UNIX).
3. Open the `webstudio.properties` file.
4. Find the line that specifies the location of the editor configuration file, for example:
   ```
   # The URL from which to load editor configuration for Experience Manager
   #com.endeca.webstudio.pagebuilder.editors.config=http://my.company.com/ed¬
   itors/myeditors.xml
   ```
5. Uncomment and change it to point to the absolute URL of your editor configuration file, for example:
   ```
   # The URL from which to load editor configuration for Experience Manager
   com.endeca.webstudio.pagebuilder.editors.config=http://<host>:<port>/re¬
   sources/config/myeditors.xml
   ```
6. Find the line that specifies the location of the cross-domain policy file for your editor configuration file, for example:
   ```
   # The URL from which to load the crossdomain.xml policy file (necessary
    when the above file is on a server that is distinct from the Experience
    Manager server)
   #com.endeca.webstudio.pagebuilder.editors.crossdomain=http://my.compa¬
   ny.com/editors/crossdomain.xml
   ```
7. Uncomment and change it to point to the absolute URL of your cross-domain policy file, for example:
   ```
   # The URL from which to load the crossdomain.xml policy file (necessary
    when the above file is on a server that is distinct from the Experience
    Manager server)
   com.endeca.webstudio.pagebuilder.editors.config=http://<host>:<port>/cross¬
   domain.xml
   ```
8. Save and close the file.
9. Start the Endeca Tools Service.

Once you have specified the location of the editor configuration file and its `crossdomain.xml` file, you can alter the editor configuration file without restarting the Endeca Tools Service, but you must reload the main Experience Manager SWF for the configuration changes to take effect. If you make any changes to `webstudio.properties`, you must restart the Endeca Tools Service for those changes to take effect.

Related Links

File hosting and security considerations on page 62

Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Using editors in templates

Adding editors to a template is similar to adding standard editors, except that you must specify a non-Endeca namespace for your editor name.

To add an editor to a landing page template:

1. Include any necessary property definitions and default values for your template.
2. Add your editor to the template's `<EditorPanel>` element and specify a non-Endeca namespace.

   Experience Manager uses the following three Endeca namespaces:

   - http://endeca.com/schema/content/2008
   - http://endeca.com/schema/content-template/2008
   - http://endeca.com/schema/content-tags/2008
3. Set any additional configuration information for your editor.

   You can include this information either as attributes of the editor element, or as nested elements. When an editor is instantiated, this information is included in the `info` property.

   As part of your configuration information, you can choose to map your editors to properties using attributes. In the example below, the `propertyName` attribute maps the editor to the `myProperty` property:

   ```
   <EditorPanel>
     <BasicContentItemEditor>
       <MyEditor xmlns="http://my.company.com/editors" propertyName="myProp¬
   erty" label="editor_label"/>
     </BasicContentItemEditor>
   </EditorPanel>
   ```

   Note: Oracle recommends the use of a formal schema to define your editor configuration and validate your templates, but you must perform this validation before uploading your templates to Experience Manager. Experience Manager does not validate any XML outside of standard Endeca schemas.

Related Links

About defining the editing interface for properties on page 34
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.

About using SSL and editors

If your Workbench server is running in SSL mode, you must enable communication with any remote editor modules by configuring the module server's cross-domain policy file.

The following possible configurations exist when serving editor modules:

Workbench	Editor Modules	Notes
SSL	no SSL	You do not need to make any changes to your cross-domain policy file with this setup. If you use this configuration, a browser warning may display informing users that the site contains both secure and non-secure items.
SSL	SSL	Oracle recommends this approach if you are using SSL. This configuration requires the `secure` attribute of `<allow-access-from domain>` to be `"true"`. Because `"true"` is the default value,

Workbench	Editor Modules	Notes
		you do not need to make any changes to your cross-domain policy file unless you have previously modified the `secure` attribute.
no SSL	SSL	This configuration requires you to set the `secure` attribute of `<al¬low-access-from domain>` to `"false"`. **Note:** This configuration can compromise your server's security. See the Adobe documentation for details.

If your Web application includes non-Flex editors that use JavaScript to communicate with Experience Manager, then all of the application containers must use the same SSL mode (enabled or disabled) for the communication to take place.

For more information about configuring Workbench to use SSL, see the *Oracle Endeca Workbench Administrator's Guide*.

Related Links

File hosting and security considerations on page 62
Experience Manager occasionally needs to access files hosted on a different server. Certain security issues may apply.

Troubleshooting an editor

Common problems and their solutions are outlined in this section. For editor-specific issues, editors must rely on their own error handling code.

Related Links

About handling error conditions on page 88
An editor dispatches an `EditorEvent.ERROR` event ito display an error message in place of the editor.

About using the Experience Manager XML View

You can use the Experience Manager XML View to evaluate the XML representation of the landing page.

You can access the Experience Manager XML View by clicking the **XML View** tab alongside the **Content Editor Pane** in the **Edit View**.

Content XML and Rule XML

You can select the **Content XML** view to see only the content XML for the landing page.

The **Rule XML** view shows the XML representation for the rule that contains this page configuration. In this representation, the content XML tags are encoded.

Saved Copy and Working Copy

You can switch between your current page XML and the version that was most recently saved in Experience Manager by using the **Saved Copy** and **Working Copy** toggle buttons. The saved copy is updated only when you click **Save All Changes** in Experience Manager **List View**.

In the case of a landing page that has had its template updated, any automatic changes made to the content are reflected in the **Working Copy**.

Ensuring Experience Manager loads the current editor modules

After you update a hosted editor module, Web browsers may continue to load cached versions of the old editor module.

When you roll out updated versions of community editor modules, you should instruct your users to clear their browser cache.

In a development environment where you may be iterating rapidly on editor modules, you can avoid clearing your cache manually by setting Internet Explorer to automatically check for updates to cached pages.

To set Internet Explorer to check for updates to cached assets:

1. From the menu, select **Tools** > **Internet Options**.
   The **Internet Options** window opens to the **General** tab.
2. Access the caching settings:

Browser	UI Control
IE6	In the **Temporary Internet Files** section, click the **Settings** button.
IE7 or IE8	In the **Browsing History** section, click the **Settings** button.

3. Set the browser to check for new versions of the page on each visit:

Browser	UI Control
IE6	Under **Check for newer versions of stored pages**, select the **Every visit to page** radio button.
IE7 or IE8	Under **Check for newer versions of stored pages**, select the **Every time I visit the web page** radio button.

Changes to the appearance of Experience Manager when using community editors

The last styles loaded into Experience Manager take precedence. Because community editor modules are loaded after the main Experience Manager SWF, this can lead to styles defined in community editors overriding Experience Manager styles.

Styling a common component, such as `mx.controls.Button`, can lead to unexpected behavior in Experience Manager. In order to avoid conflicting with Experience Manager styles, you should use a unique `styleName` for any Flex component you wish to apply styles to. Oracle recommends prefixing styles with a unique identifier, such as your company name.

Index