

# **Endeca® Content Assembler API**

## **Developer's Guide for the RAD Toolkit for ASP.NET**





# Contents

<b>Preface.....</b>	<b>7</b>
About this guide.....	7
Who should use this guide.....	7
Conventions used in this book.....	8
Contacting Endeca Customer Support.....	8
<b>Chapter 1: Developing Applications with Template-Based Pages.....</b>	<b>9</b>
Overview of roles.....	9
About the Endeca Page Builder.....	9
About the Content Assembler API.....	10
About cartridges.....	10
About templates and the Page Builder.....	10
About content items and the Content Assembler API.....	11
A typical workflow for creating a template-based application.....	12
Page Builder and Content Assembler API architecture.....	12
<b>Chapter 2: Working with Templates for Dynamic Pages.....</b>	<b>15</b>
Template prerequisites.....	15
About dynamic pages and rule groups.....	15
About using zones with dynamic pages.....	16
Creating a zone for dynamic pages.....	16
Dynamic pages and styles.....	17
About creating templates.....	17
About template validation.....	18
About the type and ID for a template.....	18
Specifying the zone and style for a template.....	19
About using thumbnail images in the Page Builder.....	20
Specifying the description and thumbnail image for a template.....	21
About saving templates.....	21
About defining content properties.....	22
Specifying the default name for a ContentItem.....	22
About content properties.....	23
Types of property elements.....	24
Adding a string property.....	25
Adding a record list property.....	26
Adding a content item property.....	26
About defining the editing interface for properties.....	27
Adding a string editor.....	28
Adding a record selector.....	30
About cartridge selectors.....	31
Adding a group label.....	31
About using XML pass-through properties.....	32
About using pass-through content properties.....	32
About passing arbitrary XML to the front-end application.....	34
<b>Chapter 3: Supporting the Page Builder.....</b>	<b>35</b>
Making templates available in the Page Builder.....	35
Uploading templates to the Page Builder.....	35
About modifying templates that are used by existing pages.....	36
About updating templates.....	37
Updating templates in the Page Builder.....	38
Retrieving the current templates from the Page Builder.....	38
About removing templates.....	39
Removing templates from the Page Builder.....	40
Troubleshooting invalid templates.....	40
Troubleshooting invalid pages.....	41

About the preview application.....	42
<b>Chapter 4: Working with the Content Assembler API.....</b>	<b>45</b>
Overview of the Content Assembler API .....	45
Content Assembler API components.....	45
Installing the Content Assembler API and reference application.....	46
Prerequisites for installing the Content Assembler API.....	46
Installing the Content Assembler API .....	46
Installation package contents .....	47
Adding the Content Assembler API to Visual Studio .....	48
About the Content Assembler reference application.....	48
Deploying the Content Assembler reference application.....	49
Using the Content Assembler reference application.....	50
Uninstalling the Content Assembler reference application.....	50
Uninstalling the Content Assembler API.....	51
Writing applications with the Content Assembler API .....	51
About using the Content Assembler with the RAD Toolkit for ASP.NET .....	51
Creating a ContentNavigationDataSource control .....	51
About setting the content rule zone.....	52
About content XML validation.....	53
Building cartridges to render template-based content.....	53
About working with content items.....	53
Using the Content Assembler reference application controls.....	54
Writing user controls to render dynamic content .....	54
About the DynamicContentControl.....	56
Using the DynamicContentControl to render pages.....	56
Using the DynamicContentControl to render cartridge content.....	57
Using the Content Assembler API for programmatic querying .....	59



---

## Copyright and disclaimer

Product specifications are subject to change without notice and do not represent a commitment on the part of Endeca Technologies, Inc. The software described in this document is furnished under a license agreement. The software may not be reverse engineered, decompiled, or otherwise manipulated for purposes of obtaining the source code. The software may be used or copied only in accordance with the terms of the license agreement. It is against the law to copy the software on any medium except as specifically allowed in the license agreement.

No part of this document may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying and recording, for any purpose without the express written permission of Endeca Technologies, Inc.

Copyright © 2003-2008 Endeca Technologies, Inc. All rights reserved. Printed in USA.

Portions of this document and the software are subject to third-party rights, including:

Corda PopChart® and Corda Builder™ Copyright © 1996-2005 Corda Technologies, Inc.

Outside In® Search Export Copyright © 1991, 2007 Oracle.

Rosette® Globalization Platform Copyright © 2003-2005 Basis Technology Corp. All rights reserved.

Teragram Language Identification Software Copyright © 1997-2005 Teragram Corporation. All rights reserved.

### **Trademarks**

Endeca, the Endeca logo, Guided Navigation, Endeca – The Next Generation of Search and Information Access, Find/Analyze/Understand, MDEX Engine, Endeca Latitude, Endeca Profind, Endeca Navigation Engine, and other Endeca product names referenced herein are registered trademarks or trademarks of Endeca Technologies, Inc. in the United States and other jurisdictions.

All other product names, company names, marks, logos, and symbols are trademarks of their respective owners.

Endeca Content Assembler API Developer's Guide for the RAD Toolkit for ASP.NET • June 2008



# Preface

The Endeca® Information Access Platform is the foundation for building applications that help people understand complex information, fostering discovery and improving daily decision-making. These applications instantly summarize data and content for users -- even for unanticipated requests. The Guided Summarization experience leads to unexpected insights in millions of everyday decisions, increasing revenue, decreasing costs, and accelerating operations.

The Endeca Information Access Platform is powered by MDEX Engine™ technology, a new class of database designed for exploring information, not managing transactions. The MDEX Engine is supported by:

- The Information Transformation Layer that unites and enriches disparate sources of information while maintaining, augmenting, and even creating structures across the data and content.
- An adaptive application component library that enables the rapid development of information access applications that automatically adapt to changes in the data and content.
- A Web-based management suite that empowers managers to highlight the right information at the right time to end users through adaptive presentation rules and dynamic pages.

These essential capabilities are delivered as an enterprise-class platform, with the scalability, reliability, and security that leading organizations demand.

## About this guide

This guide describes the major tasks involved in developing an Endeca application using the Content Assembler API for the RAD Toolkit for ASP.NET.

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

This guide covers the features of the Content Assembler API for the RAD Toolkit for ASP.NET. This guide is not a replacement for *Endeca Developer's Guide for .NET* or the *RAD Toolkit for ASP.NET Developer's Guide*.

## Who should use this guide

This guide is intended for developers who are building Endeca applications using the Content Assembler API for the RAD Toolkit for ASP.NET.

If you are a new user of the Endeca Information Access Platform or you are not familiar with developing Endeca applications with the RAD Toolkit for ASP.NET, read this guide in conjunction with:

- The *Endeca Developer's Guide for .NET* located in %ENDECA\_ROOT%\doc (in a default installation, that is C:\Endeca\MDEXEngine\5.1.3\doc), which provides detailed feature descriptions that are not covered in this guide.
- The *RAD Toolkit for ASP.NET Developer's Guide*, which covers using the Endeca RAD Toolkit for ASP.NET to build Web applications.

If you are an existing user of the Endeca Information Access Platform and you are familiar with developing Endeca applications with the RAD Toolkit for ASP.NET, this guide should provide enough

information to help you build a new application using the Content Assembler API for the RAD Toolkit for ASP.NET.

Also, see the *Endeca API Reference for the Content Assembler API for the RAD Toolkit for ASP.NET*. In a default installation, this is located in the `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\1.0.0\doc` directory. This file contains reference information for the Content Assembler API classes, and is also available on the Endeca Developer Network (EDeN) at <http://eden.endeca.com>.

## Conventions used in this book

This book uses the following typographical conventions:

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

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

## Contacting Endeca Customer Support

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

You can contact Endeca Standard Customer Support through the online Endeca Support Center <https://support.endeca.com>.





## Chapter 1

---

# Developing Applications with Template-Based Pages

This section provides an overview of working with the Page Builder 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 the Endeca Page Builder

The Endeca Page Builder is a powerful template-based tool that enables the rapid creation of rich, data-driven pages. The Page Builder is part of the Endeca Workbench suite of tools.

With the Page Builder, content administrators can create dynamic pages based on a set of custom templates, such as landing pages for a particular search term or navigation state. The ability to combine content in a modular fashion within a template framework allows a wide degree of flexibility in crafting pages while maintaining a consistent look-and-feel across an entire site.

For the content administrator, the tool offers a holistic interface to manage the placement and display of content (including rich media, refinements, promotions, or Content Spotighting) all within the overall context of a page, rather than as isolated content sections. Each dynamic page can be used in any number of locations across a site while presenting content that is contextually relevant to an end user's navigation state. This functionality greatly decreases the time and effort normally associated with the creation and maintenance of custom pages.

## About the Content Assembler API

The Content Assembler API provides a simple interface to access template-driven content for rendering in Web-based applications.

The Content Assembler API enables a Web application to query the MDEX Engine and retrieve the appropriate dynamic content based on a user's navigation state or other triggers. The Content Assembler returns both Endeca query results familiar from the Presentation API or RAD API as well as a content item object that encapsulates the page configuration specified by the content administrator. All the content for a page, including the results of any additional queries needed for spotlighting or merchandising, are wrapped in the content item object, simplifying the logic in the front-end application by reducing the need to manage sub-queries in the application layer.

## About cartridges

A cartridge consists of a template and its associated rendering code, separating the structure of dynamic page content from its presentation.

Templates serve as a basis for the dynamic pages that content administrators create in the Page Builder. 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 the Page Builder.

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 the Page Builder. This enables content administrators to flexibly combine content in a modular manner while maintaining a consistent look-and-feel throughout an entire site.

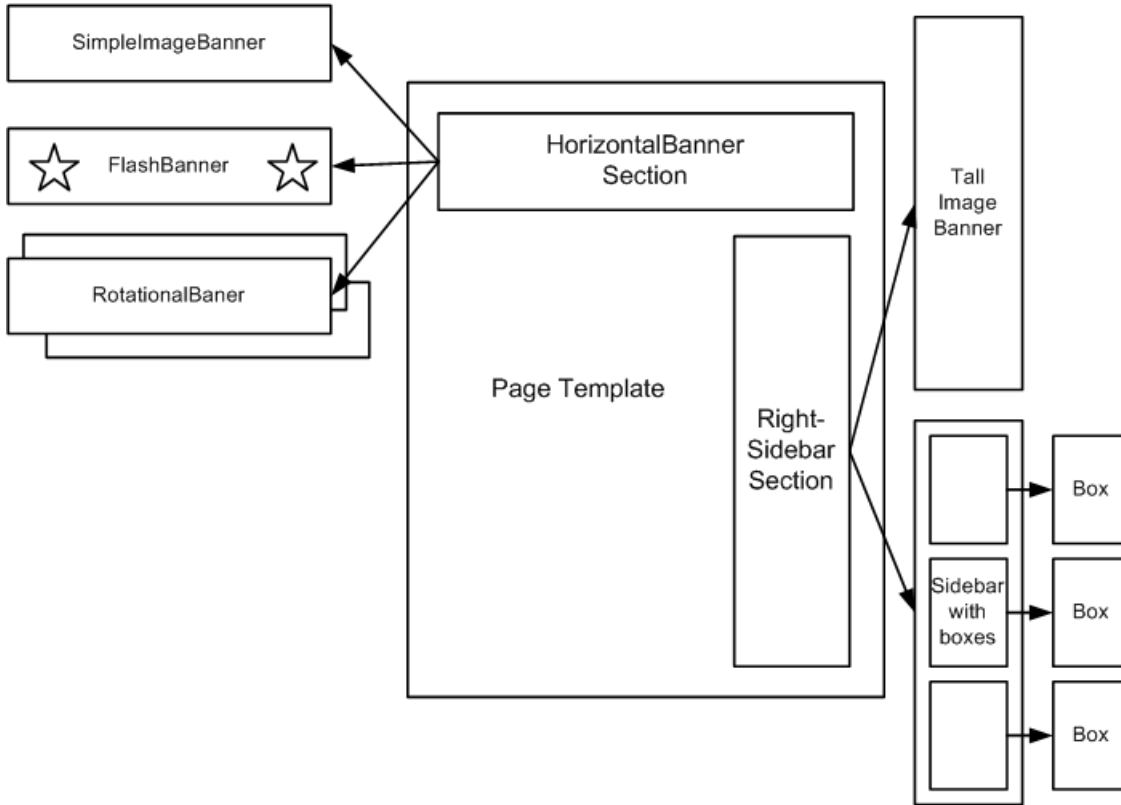
## About templates and the Page Builder

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 cartridges can be used 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 the Page Builder by selecting cartridges to insert into sections and then populating the cartridges with content. The interface for populating cartridges in the Page Builder 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 the Page Builder, 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. 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 the Page Builder 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 the Page Builder, and specifies an application that uses the cartridge code as the preview application in Endeca Workbench.

A content administrator can then use the Page Builder 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 the Page Builder 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.

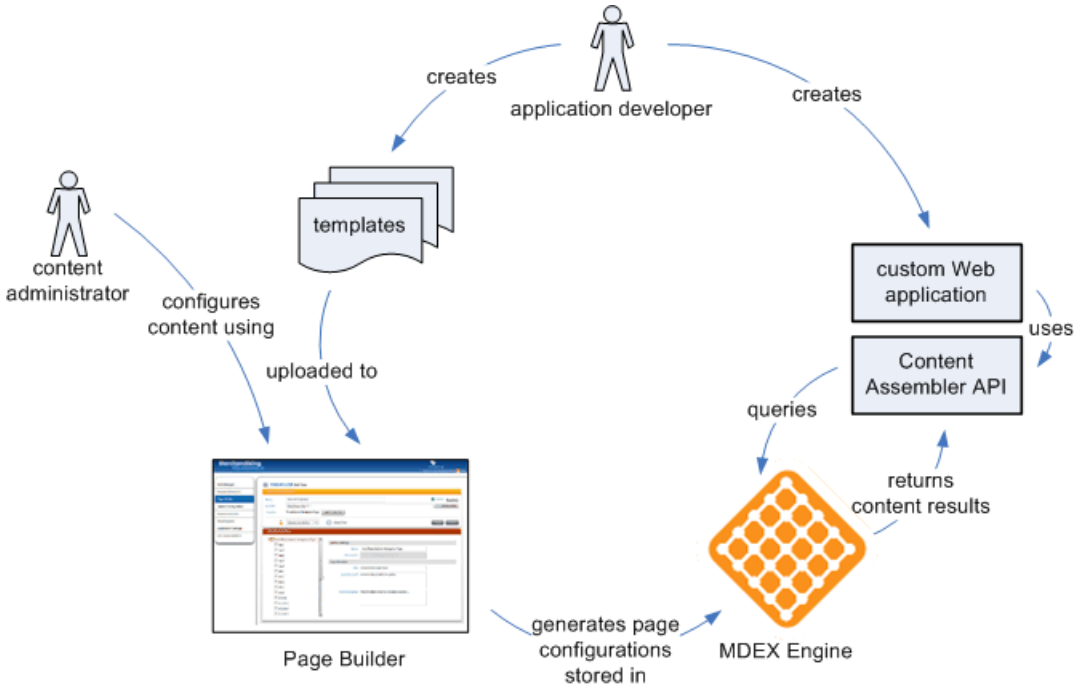
The Page Builder allows the content administrator to save progress incrementally and also to preview dynamic pages to make sure that the application renders the content as desired.

Once a page configuration is complete, the content administrator can request that a manager activate the page. Once the manager activates the page in the Page Builder, it is published and accessible to the end users of the Web site. The workflow model within the Page Builder 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.

## Page Builder and Content Assembler API architecture

The Page Builder 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 the Page Builder 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 the Page Builder, 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. The Page Builder 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

The Page Builder 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 the Page Builder, 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 *Endeca Developer Studio Help*.

The Group List pages of both the Rule Manager and the Page Builder 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 the Page Builder. 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 "Grouping rules" in the *Endeca Developer's Guide*.

## About using zones with dynamic pages

Zones enable the display of dynamic pages in the application. While a single zone can be sufficient, multiple zones allow 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* allow you to provide different perspectives on the same page. Two common use cases for using multiple zones are for search results and for switching between views for the same data, such as product listings and reviews.

### Search results

Dynamic pages are generally configured to display based on a navigation trigger. This means however that the page for a particular location displays even if a user has entered a search term on your Web site from that location. For example, you may have set up a highly branded dynamic page to display as your site's home page (at location N=0) that does not include any search results. This page displays even if a user has performed a search from the home page location, unless a page has been configured specifically to trigger on that search term. In an application with a single zone, generic search results pages may never display.

You can enable more robust handling of search results pages by creating a separate zone for searches. You then create a template that uses the search zone and the content administrator creates a search results page based on this template. In the simplest case, you can have one search results page in this zone that applies everywhere. When your application receives a search query, it displays the page for the search zone rather than the navigation zone.

### Switching between views

Another use of multiple zones is to enable different views on the same data. For example, if you want to present a tabbed pane that displays either product details or user reviews of the same product, 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.

- [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.
- [Specifying the zone and style for a template](#) on page 19  
Page templates are required to specify a rule zone and a style. When a page is created in the Page Builder, the zone and style are applied to any pages based on that template.

## 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 **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.

## Dynamic pages and styles

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

Endeca 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.

## 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 the Page Builder.

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 the Page Builder 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 the Page Builder.

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 when they are uploaded to the Page Builder.

Before you upload your templates to the Page Builder, 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">
```

A copy of the schema is located for reference in `\doc\schema\content-template.xsd` (on Windows) or `/doc/schema/content-template.xsd` (on UNIX) in your Content Assembler API installation.

Although the `<RuleInfo>` element is not required by the schema, it is required by the Page Builder 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 the Page Builder and an error is written to `webstudio.log`.

- [Specifying the zone and style for a template](#) on page 19  
Page templates are required to specify a rule zone and a style. When a page is created in the Page Builder, the zone and style are applied to any pages based on that template.
- [Troubleshooting invalid templates](#) on page 40  
Some template errors are returned to the `emgr_update` command line call, but all errors are detailed in the `emgr.log` or `webstudio.log` files.

## 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 a 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 cartridgesw 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 the Page Builder.

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 the Page Builder. 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 `<ContentItemTemplate>` 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.

- [About templates and the Page Builder](#) on page 10  
Templates can either define the content structure of an entire page or a section of a page.

## 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 the Page Builder, 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 `<ContentItemTemplate>` 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 -->
</ContentItemTemplate>
```

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. Endeca 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.

- [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 can be sufficient, multiple zones allow finer-grained control over the display of dynamic pages.

## About using thumbnail images in the Page Builder

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

The images may be hosted on a separate Web server from your Page Builder 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.

### Hosting and security considerations

The images must be hosted on a Web server accessible from the Page Builder server. The Page Builder makes an anonymous request to the image server to fetch the images. That is, even though content administrators are authenticated when they log in to the Page Builder, the tool does not use their credentials when requesting thumbnail images.

The Page Builder also respects the cross-domain policy file of the server hosting the images. To ensure that the Page Builder can load the images, place a `crossdomain.xml` file in the root directory of the image 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 Page Builder server, a security error similar to the following displays in the Page Builder when the template selector or cartridge selector dialog box attempts to load the images:

```
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 the Page Builder 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`:

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

```
<allow-access-from domain="105.216.0.40" />
</cross-domain-policy>
```

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

## 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 the Page Builder. 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 `<ContentItemTemplate>`:

Element	Description
<code>&lt;Description&gt;</code>	One or two sentences to help the content administrator identify the template in the Page Builder. This can include information about the visual layout of the template ("Three-column layout with large top banner") or its intended purpose ("Back to school promotion").
<code>&lt;ThumbnailUrl&gt;</code>	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 Page Builder server.

### 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 -->
</ContentItemTemplate>
```

## About saving templates

Templates are saved as XML files that are then uploaded to the Page Builder.

It is possible to have multiple templates in a single file, however, for ease of maintenance Endeca 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 the Page Builder. 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 the Page Builder 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.

- [About defining the editing interface for properties](#) on page 27  
After you have defined the content properties in your template, you can define how those properties can be configured by the content administrator in the Page Builder.

## 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 the Page Builder, 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 -->
</ContentItemTemplate>
```

`<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 the Page Builder 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. A `<String>` element specifies a string property, and a `<RecordList>` specifies a property that can contain one or more Endeca records.

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 special content pass-through elements or arbitrary XML that is passed directly to your front-end application.

The following example shows the properties of a cartridge template that defines a buying guide. This buying guide can contain three subsections that spotlight specific products.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  type="HorizontalBanner" id="BuyingGuide">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Buying Guide</Name>
    <Property name="ui_code">
      <String>/HorizontalBanner/BuyingGuide.jsp</String>
    </Property>
    <Property name="title">
      <String/>
    </Property>
    <Property name="header">
      <String/>
    </Property>
    <Property name="footer">
      <String/>
    </Property>
    <Property name="section1">
      <ContentItem type="BuyingGuideSection"/>
    </Property>
    <Property name="section2">
      <ContentItem type="BuyingGuideSection"/>
    </Property>
    <Property name="section3">
      <ContentItem type="BuyingGuideSection"/>
    </Property>
  </ContentItem>
```

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

The properties of the buying guide sections may look similar to the following:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  type="BuyingGuideSection" id="BuyingGuideSection">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Buying Guide Section</Name>
    <Property name="title">
      <String/>
    </Property>
    <Property name="header">
      <String/>
    </Property>
    <Property name="img_src">
      <String/>
    </Property>
    <Property name="footer_text">
      <String/>
    </Property>
    <Property name="footer_href">
      <String/>
    </Property>
    <Property name="products">
      <RecordList/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentItemTemplate>
```

## 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 the Page Builder.

Template element	Object type returned by API (Java)	Object type returned by API (RAD Toolkit for ASP.NET)
<String>	java.lang.String	string
<RecordList>	com.endeca.navigation.ERecList	System.Collections.ObjectModel.ReadOnlyCollection<Record>
<ContentItem>	com.endeca.content.ContentItem	Endeca.Data.Content.IContentItem



## Pass-through content properties

These properties cannot be exposed for configuration in the Page Builder 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)
<UrlEneQuery>	com.endeca.navigation.ERecList	System.Collections.ObjectModel.ReadOnlyCollection<Record>
<Supplement>	com.endeca.navigation.Supplement	Endeca.Data.BusinessRule
<RecordQuery>	com.endeca.navigation.ERecList	System.Collections.ObjectModel.ReadOnlyCollection<Record>
<NavQuery>	com.endeca.navigation.ERecList	System.Collections.ObjectModel.ReadOnlyCollection<Record>
<NavigationResult>	com.endeca.navigation.ENEQueryResults	Endeca.Data.NavigationResult

## 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).

- [About working with content items](#) on page 53  
You can access the `IContentItem` that contains dynamic page content from a `ContentNavigationDataSource`.

## 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="ui_code">
      <String>/ThreeColumnNavigationPage.jsp</String>
    </Property>
    <Property name="title">
      <String>Endeca Wine Superstore</String>
```

```

</Property>
<Property name="meta_keywords">
  <String/>
</Property>
<Property name="meta_description">
  <String/>
</Property>
</ContentItem>
<!-- additional elements deleted from this example -->
</ContentItemTemplate>

```

- [Adding a string editor](#) on page 28  
You add a string editor to enable configuration of string properties in the Page Builder.

## Adding a record list property

A record list property can contain one or more Endeca records.

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="BuyingGuideSection" id="BuyingGuideSection">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Buying Guide Section</Name>
    <!-- additional properties deleted from this example -->
    <Property name="products">
      <RecordList/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentItemTemplate>

```

- [Adding a record selector](#) on page 30  
You add a record selector to enable an interface in the Page Builder to specify featured records or queries.

## 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 the Page Builder 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 will be presented as options for the content administrator to choose from in the Page Builder. For example, when a content administrator goes to choose the cartridge to insert in a `RecommendedContent` section, only templates of type `RecommendedContent` will display in the **Select Cartridge** dialog box. (Recall that the cartridge template is the part of a cartridge that is exposed in the Page Builder.) 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 three 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 page</Name>
    <!-- additional properties deleted from this example -->
    <Property name="top">
      <ContentItem type="HorizontalBanner"/>
    </Property>
    <Property name="left">
      <ContentItem type="VerticalBanner"/>
    </Property>
    <Property name="right">
      <ContentItem type="VerticalBanner"/>
    </Property>
  </ContentItem>
  <!-- additional elements deleted from this example -->
</ContentItemTemplate>
```

- [About cartridge selectors](#) on page 31  
Unlike string and record list properties, section properties are always editable; you do not need to explicitly specify an editor in the template.
- [About content items and the Content Assembler API](#) on page 11  
When a content administrator creates pages in the Page Builder, the resulting configurations are saved as XML documents in the MDEX Engine.

## 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 the Page Builder.

You add content editors inside the `<EditorPanel>` element in the template. The `<BasicContentItemEditor>` allows you to specify individual property editors that display in the Page Builder 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 Sample Section</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 the Page Builder 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 the Page Builder, in the order in which you define them in the template. If you do not want a property to be exposed in the Page Builder interface, do not define an editor associated with it.

It is possible to create more than one editor associated with the same property, however, be aware that all editors that you define in the template display in the Page Builder, 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.

- [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 31  
In the Page Builder 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 in the Page Builder.

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
<b>propertyName</b>	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.
<b>label</b>	This attribute allows you to specify a more descriptive label for this field in the Page Builder. If no label is specified, the property name is used by default.
<b>editable</b>	If set to <i>false</i> , this attribute makes the property read-only in the Page Builder so that the value of the property is visible, but it cannot be edited. Use this option only if you specify a default value in the definition of the string property. Properties are editable by default.
<b>width</b>	The width in pixels of the text field presented in the Page Builder interface. The default width is 300 pixels.
<b>height</b>	The height in pixels of the text field presented in the Page Builder interface. The default height is 24 pixels.

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

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- additional elements deleted from this example -->
  <!-- First define all the content properties -->
  <ContentItem>
    <Name>New Sample Page</Name>
    <Property name="title">
      <String>Page Title</String>
    </Property>
    <Property name="meta_keywords">
      <String/>
    </Property>
    <Property name="img_src">
      <String>/imgs/logo.jpg</String>
    </Property>
    <Property name="ui_code">
      <String>~/Resources/UserControls/HorizontalBanner/SamplePage.as-
cx</String>
    </Property>
  </ContentItem>
  <!-- Define editors for properties that should be configurable -->
  <EditorPanel>
    <BasicContentItemEditor>
      <!--
      # This example allows the content administrator to
      # specify the title of the page.
      # A default value was provided above as a placeholder,
      # and it is editable in the Page Builder. The label
      # indicates that this field must be filled in by the
      # content administrator.
      -->
      <StringEditor propertyName="title" label="Title (required)"/>

      <!--
      # The meta.keywords property allows a content administrator
      # to specify keywords for search engine optimization that the
      # application can insert into the meta tags when it renders
      # a page. The default width and height values have been
```

```

# overridden to provide a larger text box to enter content.
# This property is editable, but there is no default
# value.
-->
<StringEditor propertyName="meta_keywords"
  label="Meta keywords" width="300" height="120"/>

<!--
# In this example the template includes a particular
# image that displays on every page that is created
# using this template. This information can be
# displayed in the Page Builder for information only
# but it cannot be edited or deleted.
-->
<StringEditor propertyName="img_src" label="Logo URL"
  editable="false"/>

<!--
# Because the app.code property is used only by the
# front-end application and is not of interest to the
# content administrator, there is no editor to expose
# it in the Page Builder.
-->
</BasicContentItemEditor>
</EditorPanel>
</ContentItemTemplate>

```

- [Adding a string property](#) on page 25  
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 record selector

You add a record selector to enable an interface in the Page Builder to specify featured records or queries.

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.

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
<code>propertyName</code>	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.
<code>label</code>	This attribute allows you to specify a more descriptive label for this editor in the Page Builder. If no label is specified, the property name is used by default.
<code>maxRecords</code>	Sets the maximum number of records that this property can contain. If the content administrator designates specific records in the Page Builder, the number of records cannot exceed the value of <code>maxRecords</code> . If the content administrator specifies a query, it will return no more than this number of records. When configuring this property, the content administrator may choose to designate

Attribute	Description
	fewer static records or to further limit the number of records returned by a query. The default value for <code>maxRecords</code> is 10.

The following example shows a record selector associated with a "recommended" property. This allows a content administrator to specify up to three specific records or a query that returns up to three records.

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="BuyingGuideSection" id="BuyingGuideSection">
  <!-- additional elements deleted from this example -->
  <ContentItem>
    <Name>New Buying Guide Section</Name>
    <!-- additional properties deleted from this example -->
    </Property>
    <Property name="products">
      <RecordList/>
    </Property>
  </ContentItem>
  <EditorPanel>
    <BasicContentItemEditor>
      <!-- additional editors deleted from this example -->
      <RecordSelector propertyName="products" label="Products"
maxRecords="3"/>
    </BasicContentItemEditor>
  </EditorPanel>
</ContentTemplate>
```

- [Adding a record list property](#) on page 26  
A record list property can contain one or more Endeca records.

## About cartridge selectors

Unlike string and record list properties, section properties are always editable; you do not need to explicitly specify an editor in the template.

In the Page Builder, 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.

- [Adding a content item property](#) on page 26  
A content item property defines a template section by creating a placeholder for a nested content item.

## Adding a group label

In the Page Builder 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 the Page Builder 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 the Page Builder.

The editor panel in the Page Builder 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="Image info"/>
      <StringEditor propertyName="title" label="Title"/>
      <StringEditor propertyName="header" label="Header text"/>
      <StringEditor propertyName="img_src" label="Image file"/>
      <GroupLabel label="Mouse-over info"/>
      <StringEditor propertyName="footer_text" label="Footer text"/>
      <StringEditor propertyName="footer_href" label="Footer link"/>
      <RecordSelector propertyName="products" label="Products"
maxRecords="3"/>
    </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 the Page Builder.
- 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 template schema.



**Note:** The Page Builder does not perform any validation on XML within a different namespace from the template schema. If you are using pass-through properties, be sure to validate your templates before uploading them.

## 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 the Page Builder. When the Content Assembler processes these properties, it evaluates them and executes any necessary queries exactly as if the property had been configured with specific values by a content administrator in the Page Builder.

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

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 template or content schemas located in `doc\schema\` (on Windows) or `doc/schema/` (on UNIX) in your Content Assembler API installation. The Page Builder does not perform any validation on XML within a different namespace from the template schema. However, the Content Assembler API must be able to access the namespace that you specify.

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 the Page Builder

---

This section describes the tasks needed to enable content administrators to create pages in the Page Builder.

## Making templates available in the Page Builder

This section describes how to manage Page Builder 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 the Page Builder.

For a complete list of accepted `emgr_update` syntax, refer to "emgr\_update syntax" in the *Endeca Administrator's Guide*.

## Uploading templates to the Page Builder

Before Page Builder 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
<code>--host</code>	The machine name and port for the staging Endeca Workbench environment, in the format <i>host.port</i> .
<code>--app_name</code>	The name of the application to which you want the templates to apply.
<code>--dir</code>	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:8888
--app_name My_application --dir c:\endeca-app\templates\
```

The following is a UNIX example:

```
emgr_update --action set_templates --host localhost:8888
--app_name My_application --dir /apps/endeca/templates/
```

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

- [Updating templates in the Page Builder](#) on page 38  
Updating templates using `emgr_update` is a multi-step process.
- [About updating templates](#) on page 37  
When updating templates in the Page Builder, you should be aware of how conflicts are handled.
- [Troubleshooting invalid templates](#) on page 40  
Some template errors are returned to the `emgr_update` command line call, but all errors are detailed in the `emgr.log` or `webstudio.log` files.

## 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 the Page Builder.

When updating templates, you should be aware of the following effects on existing pages that use an updated template:



**Note:** Existing page configurations are not updated to the new template until a content administrator edits and saves the affected page or cartridge in the Page Builder.

- If a template has new properties that did not previously exist, any corresponding property editors become available in the Page Builder when a content administrator edits a page or cartridge based on the updated template. If you specify any default values for new properties, they are applied when a content administrator edits and saves the page or cartridge using the updated template.
- If a template no longer contains properties that previously existed, the corresponding property editors no longer display in the Page Builder 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 the Page Builder when a content administrator edits a page or cartridge based on the updated template. The existing value for the property does not display in the Page Builder and is deleted or 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 a property has not changed its name or type, the existing values are migrated to the new template. 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.
- If XML pass-through properties have been changed, they are updated when a content administrator edits and saves the page or cartridge using the updated template.
- If the default value of an existing property has changed, 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.

- 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 the Page Builder, use the `label` attribute instead.



**Note:** Because existing content is not automatically updated to the new templates, and default values can never be updated in existing pages, any changes that you make to your rendering code to reflect changes to a template must be backward-compatible. 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.

If you do need to update templates that are being used in production, you can include a version number in a property that your code can check so that it can render the content appropriately. In order for this number to be updated when existing content is updated to a new version of a template, it must be stored as the `template name` or as an XML pass-through property.

- [About updating templates](#) on page 37  
When updating templates in the Page Builder, you should be aware of how conflicts are handled.
- [Updating templates in the Page Builder](#) on page 38  
Updating templates using `emgr_update` is a multi-step process.

## About updating templates

When updating templates in the Page Builder, you should be aware of how conflicts are handled.

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

Templates with non-unique IDs do not display in the template selector or cartridge selector dialog box. If you upload two template files with the same ID but different file names, then two separate templates are stored in the Page Builder but neither one displays to content administrators. For this reason, you should avoid renaming template files after they have been uploaded to the Page Builder unless you make sure to remove the old template first.

If a template is created with a duplicate ID, no error message is returned to the `emgr_update` command line call when uploading templates. Instead, an error message is written to the `webstudio.log` file similar to the following:

```
Jan 1, 2008 12:34:56 AM com.endeca.etools.io.ConfigStoreAPI loadContentTemplates
SEVERE: The template "HorizontalBanner-ImageMap.xml" has a non-unique ID ("ImageMap").
Jan 1, 2008 12:34:56 AM com.endeca.etools.io.ConfigStoreAPI loadContentTemplates
SEVERE: The template "VerticalBanner-ImageMap.xml" has a non-unique ID ("ImageMap").
```

To re-enable the templates, edit the `id` attribute of the `<ContentTemplate>` element so that each template ID is unique, remove the templates from the Page Builder, and re-upload the templates. In

general, it is a best practice to remove templates from the Page Builder and upload a complete set of templates whenever you need to update templates.

- [About modifying templates that are used by existing pages](#) on page 36  
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 the Page Builder.
- [Updating templates in the Page Builder](#) on page 38  
Updating templates using `emgr_update` is a multi-step process.

## Updating templates in the Page Builder

Updating templates using `emgr_update` is a multi-step process.



**Note:** Before updating templates in the Page Builder, be sure you have a backup of the current set of templates. Endeca 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 the Page Builder.

To update existing templates in the Page Builder:

1. Retrieve the current set of templates from the Page Builder.
  2. Make any necessary edits to the templates on your local machine.
  3. Remove all templates from the Page Builder.
  4. Upload the revised templates from your local directory to the Page Builder.
- [About modifying templates that are used by existing pages](#) on page 36  
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 the Page Builder.
  - [About updating templates](#) on page 37  
When updating templates in the Page Builder, you should be aware of how conflicts are handled.
  - [Uploading templates to the Page Builder](#) on page 35  
Before Page Builder users can access new templates, you must upload them using the `emgr_update` utility.
  - [Retrieving the current templates from the Page Builder](#) on page 38  
If you need to view or edit an existing template on a local machine, use `emgr_update` to copy the templates from the Page Builder into a local directory.
  - [Removing templates from the Page Builder](#) on page 40  
You can remove all the templates from the Page Builder using the `emgr_update` utility.

## Retrieving the current templates from the Page Builder

If you need to view or edit an existing template on a local machine, use `emgr_update` to copy the templates from the Page Builder 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
<code>--host</code>	The machine name and port for the staging Endeca Workbench environment, in the format <i>host.port</i> .
<code>--app_name</code>	The name of the application from which you want to retrieve the templates.
<code>--dir</code>	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:8888
--app_name My_application --dir c:\endeca-app\templates\
```

The following is a UNIX example:

```
emgr_update --action get_templates --host localhost:8888
--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 the Page Builder.

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 the Page Builder, 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.

- [Removing templates from the Page Builder](#) on page 40  
You can remove all the templates from the Page Builder using the `emgr_update` utility.

## Removing templates from the Page Builder

You can remove all the templates from the Page Builder using the `emgr_update` utility.



**Note:** Before removing templates from the Page Builder, be sure you have a backup of the current set of templates. Endeca 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 the Page Builder.

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

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

To remove templates from the Page Builder:

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

Parameters	Value
<code>--host</code>	The machine name and port for the staging Endeca Workbench environment, in the format <code>host.port</code> .
<code>--app_name</code>	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:8888
--app_name My_application
```

The following is a UNIX example:

```
emgr_update --action remove_templates --host localhost:8888
--app_name My_application
```

- [About removing templates](#) on page 39  
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 the Page Builder.

## Troubleshooting invalid templates

Some template errors are returned to the `emgr_update` command line call, but all errors are detailed in the `emgr.log` or `webstudio.log` files.

The `emgr.log` and `webstudio.log` files are located in:

- `%ENDECA_CONF%\logs` on Windows platforms
- `$/ENDECA_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\ContentAssemblerAPIJava\1.0.0\reference\templates>emgr_update --
host localhost:8888 --app_name wine --action set_templates --dir
\apps\myapp\template
ERROR: Failed to set the following components: type='CONTENT' loca-
tor='PageTemplate-ThreeColumnPage.xml' size='2236' , type='CONTENT' loca-
tor='Banner-ImageMap.xml' size'2034' ,
ERROR: Failed to set app config. Make sure you can connect to http://local-
host:8888.
```

Each template that fails validation will appear as a separate component. If you receive a schema validation message, check the `emgr.log` file for a more detailed validation error.

### Invalid zone or style

If a template is uploaded and refers to an invalid zone or style, no error message is returned to the `emgr_update` command line call, but the template is not available in the Page Builder. An error message is written to the `webstudio.log` file similar to the following:

```
Jan 1, 2008 12:34:56 AM com.endeca.ertools.io.ConfigStoreAPI loadContentTem-
plates
SEVERE: The template "NavigationPage" has an invalid style("PageStyle3").
Jan 1, 2008 12:34:56 AM com.endeca.ertools.io.ConfigStoreAPI loadContentTem-
plates
SEVERE: The template "NavigationPage" has an invalid zone("NavigationPage-
Zone3").
```

The `zone` and `style` attributes of the `<RuleInfo>` element must correspond to one of the zones and styles defined in your application's instance configuration in Endeca Workbench.

### Empty directory

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

```
C:\Endeca\ContentAssemblerAPIJava\1.0.0\reference\templates>emgr_update --
host localhost:8888 --app_name wine --action set_templates --dir
\apps\myapp\template
There are no templates in the specified directory.
```

If you receive this message, check to make sure that you specified the correct directory.

- [About template validation](#) on page 18  
Templates are validated when they are uploaded to the Page Builder.
- [Specifying the zone and style for a template](#) on page 19  
Page templates are required to specify a rule zone and a style. When a page is created in the Page Builder, the zone and style are applied to any pages based on that template.

## Troubleshooting invalid pages

If a page is displaying in the Page Builder 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 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.

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.

- [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.
- [Specifying the zone and style for a template](#) on page 19  
Page templates are required to specify a rule zone and a style. When a page is created in the Page Builder, the zone and style are applied to any pages based on that template.
- [About using zones with dynamic pages](#) on page 16  
Zones enable the display of dynamic pages in the application. While a single zone can be sufficient, multiple zones allow finer-grained control over the display of dynamic pages.

## About the preview application

The preview application in the Page Builder allows content administrators to verify the behavior of the pages they create. The Page Builder shares the same preview application as the Rule Manager.

Before content administrators create pages in the Page Builder, be sure to replace the default preview application with one that can render pages appropriately based on the templates you have created. This ensures that the preview application can provide an accurate representation of the way pages display in the final front-end application. In addition, in order to allow content administrators to save their progress and preview pages incrementally, the application should be able to gracefully handle empty sections or cartridges that have not been fully configured.

Note that if your preview application is not instrumented, the status messages for dynamic pages do not update when previewing by navigating in the preview application, making it difficult to determine

why a page may not be firing. (The status messages do update when previewing through the Location links in the List View.) For more information about setting up a preview application, see the *Endeca Administrator's Guide*.





## Chapter 4

# Working with the Content Assembler API

---

This section provides information on working with the Endeca Content Assembler API classes and server controls.

## Overview of the Content Assembler API

The Content Assembler API for the RAD Toolkit for ASP.NET extends the Endeca RAD Toolkit for ASP.NET to enable access to dynamic page content.

The Content Assembler retrieves the appropriate page configuration (created by a content administrator in the Page Builder) based on search, navigation state (also known as refinement state), date, and user profile triggers. The Assembler then executes any additional queries specified by that configuration, such as navigation or record queries for spotlighted content, and returns the assembled page content as a `IContentItem` object. Content items can contain various properties including strings, records, and nested content items as specified in the page or cartridge templates.

## Content Assembler API components

The Content Assembler API for the RAD Toolkit for ASP.NET provides some additional classes and controls for accessing and rendering dynamic page content.

### Content Assembler RAD API

The Content Assembler API includes the following additions to the RAD API:

- The `IContentItem` interface defines a content item that represents the dynamic page content configured in the Page Builder.
- Content items contain a collection of `IProperty` objects that represent the properties defined in the template and configured by the content administrator.
- The `NavigationContentItemCreator` generates content item objects from a RAD Toolkit for ASP.NET `NavigationCommand` and `NavigationResult`.

### The `ContentNavigationDataSource` control

The Content Assembler API includes a new server control. The `ContentNavigationDataSource` extends the `NavigationDataSource` control to enable access to dynamic page content.

### Reference application controls

In addition, the following controls are included with the Content Assembler reference application:

- The `DynamicContentControl` dynamically loads other user controls in order to render template-based content.
- The `IContentControl` interface defines a content item-aware user control for rendering section content.

## Installing the Content Assembler API and reference application

This section describes the prerequisites and installation tasks for the Content Assembler API and reference application for the RAD Toolkit for ASP.NET.

### Prerequisites for installing the Content Assembler API

This topic provides system requirements for installing the Content Assembler API and reference application for the RAD Toolkit for ASP.NET.

The Content Assembler API for the RAD Toolkit for ASP.NET is compatible with Endeca Information Access Platform version 5.1.3 and the RAD Toolkit for ASP.NET version 1.1.0.

The following software must be installed before you install the Content Assembler API for the RAD Toolkit for ASP.NET:

- Microsoft .NET Framework. The following versions are supported: 2.0 with SP1, 3.0, and 3.5 (including the 32-bit and 64-bit versions).
- Microsoft Visual Studio 2005 with Service Pack 1 (any edition) or Visual Studio 2008.
- On machines running the Content Assembler reference application, you need Internet Information Services (IIS) 5.1 or later.

#### Additional requirements

The Endeca Page Builder, a component of the Endeca Workbench suite of tools, is required for content administrators to configure template-based pages. For details about installing Endeca Workbench, see the *Endeca Merchandising Workbench Installation and Migration Guide*.

## Installing the Content Assembler API

This topic describes how to install the Content Assembler API for the RAD Toolkit for ASP.NET.



**Note:** The RAD Toolkit for ASP.NET version 1.1.0 and the Endeca Presentation and Logging APIs for .NET version 5.1.3 are part of the Content Assembler API installation. When building applications using the Content Assembler API, you must use the assemblies that are included in the Content Assembler API installation.

To install the Content Assembler API for the RAD Toolkit for ASP.NET:

1. In your local environment, locate the Content Assembler API package that you downloaded from the Endeca Support Center (<https://support.endeca.com>).

The Content Assembler API package is named `ContentAssemblerRADNET-version.msi`, where *version* is the version number of the Content Assembler API.

2. Double click the installation file, and on the **Welcome** screen, click **Next**.
3. On the **Select Installation Folder** screen, either accept the default installation location, or click **Browse...** and browse to the directory where you want to install. Endeca recommends that you accept the default installation path of `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\1.0.0`.
4. On the **Ready to Install** screen, click **Install**.
5. Click **Finish**.

To complete setup of your development environment for the Content Assembler API, add the Endeca server controls and assemblies to Visual Studio.

- [Adding the Content Assembler API to Visual Studio](#) on page 48  
This procedure adds the Endeca data source controls and Endeca user interface controls to the Toolbox window of Visual Studio and also adds references from your Web site project to Endeca assemblies.
- [Deploying the Content Assembler reference application](#) on page 49  
These instructions assume a typical system configuration that includes IIS 5.1, IIS 6 Manager, and the .NET Framework 2.0. There may be minor configuration differences if you are using other versions of IIS, IIS Manager, or the .NET Framework.

## Installation package contents

This topic describes in detail the directories that are created in the Content Assembler API installation.

The `\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\1.0.0` directory contains the release notes for the RAD Toolkit for ASP.NET (`README_RAD.txt`) and the Content Assembler API (`README_CARN.txt`) as well as the following directories:

Directory	Contents
<code>\bin</code>	The assemblies for the Content Assembler API for the RAD Toolkit for ASP.NET.
<code>\doc</code>	The documentation for the RAD Toolkit for ASP.NET and the Content Assembler API.
<code>\doc\schemas</code>	The XML schema documents for templates ( <code>content-template.xsd</code> ) and for the page configurations generated by the Page Builder ( <code>content.xsd</code> and <code>content-tags.xsd</code> ).
<code>\reference\ContentAspNetRef</code>	The Content Assembler reference application for the RAD Toolkit for ASP.NET.
<code>\reference\templates</code>	A collection of sample templates for use with the Page Builder and the Content Assembler API.

## Adding the Content Assembler API to Visual Studio

This procedure adds the Endeca data source controls and Endeca user interface controls to the Toolbox window of Visual Studio and also adds references from your Web site project to Endeca assemblies.

This procedure describes how to add the RAD Toolkit and Content Assembler API to Visual Studio 2005. The procedure may be slightly different in Visual Studio 2008.

To add the RAD Toolkit for ASP.NET and the Content Assembler API to Visual Studio:

1. Start Visual Studio and open your Web site.
2. In the **Toolbox** window, right-click and select **Add Tab**.
3. Type `Endeca Content Assembler API`.
4. Right-click in the grey box under the entry for the Content Assembler API.  
The grey box contains the text "There are no usable controls in this group.."
5. Select **Choose Items...**  
After a short delay, the **Choose Toolbox Items** dialog box displays.
6. On the **.NET Framework Components** tab of the **Choose Toolbox Items** dialog box, click **Browse...**
7. Browse to the `bin` directory of your Content Assembler API installation. In a typical installation, this is `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\1.0.0\bin\`
8. Select `Endeca.Web.Content.dll` and click **Open**.
9. Repeat the previous step for `Endeca.Web.dll` and `Endeca.Web.UI.WebControls.dll`.
10. Click **OK**.  
The Endeca data source controls, user interface controls, and Content Assembler controls display in the **Toolbox** under Endeca Content Assembler.
11. In the **Solution Explorer** window, right-click your Web site.
12. Select **Add Reference...**
13. In the **Add Reference** dialog, select the **Browse** tab.
14. Browse to the `bin` directory of your Content Assembler API installation.
15. Select `Endeca.Data.Content.dll`, `Endeca.Data.dll`, `Endeca.Logging.dll`, `Endeca.Navigation.AccessControl.dll`, and `Endeca.Navigation.dll`. You do not need to select `Endeca.Web.Content.dll`, `Endeca.Web.dll` or `Endeca.Web.UI.WebControls.dll`.  
Visual Studio copies the selected files into the Web site project.

## About the Content Assembler reference application

The Content Assembler API distribution includes a modified version of the RAD Toolkit for ASP.NET reference application for use with the Page Builder and the Content Assembler API.

The Content Assembler reference application is intended primarily as a validation and diagnostic tool for developers. Although it demonstrates how to query the MDEX Engine using the Content Assembler API, the reference application does not represent the best practices for building an ASP.NET application using cartridges.

The reference application enables you to browse your data in the same way as the reference application for the RAD Toolkit for ASP.NET. In addition, the Content Assembler reference application features a new section called **ContentControl.ascx** that allows you to browse the content items that are returned by the Content Assembler API based on your navigation state.



The `ContentRadAspNetRef\App_Code\ContentRadAspNetRef` directory also includes the `DynamicContentControl` and the `IContentControl`. These controls are provided as a starting point for building applications that can dynamically load user controls to render page content, although the reference application does not demonstrate how to use these controls.

- [Using the Content Assembler reference application controls](#) on page 54  
The `DynamicContentControl` and the `IContentControl` work together to allow you to dynamically load user controls to render page content.

## Deploying the Content Assembler reference application

These instructions assume a typical system configuration that includes IIS 5.1, IIS 6 Manager, and the .NET Framework 2.0. There may be minor configuration differences if you are using other versions of IIS, IIS Manager, or the .NET Framework.

To deploy the Content Assembler reference application for the RAD Toolkit for ASP.NET:

1. From the Windows Control Panel, select **Administrative Tools** > **Internet Information Services (IIS6) Manager**.
2. In the IIS tree pane, expand the machine icon for the local machine, then expand the **Web Sites** directory.
3. Right click the **Default Web Site** and select **New** > **Virtual Directory....**
4. Complete the **Virtual Directory Creation Wizard** as follows:
  - a) Click **Next**.
  - b) Type an alias name such as `ContentAssembler`.
  - c) Click **Next**.
  - d) In the **Web Site Content Directory** screen, click **Browse** and locate the reference application that is packaged with the Content Assembler API for the RAD Toolkit for ASP.NET. In a typical installation, this is in `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\1.0.0\reference\ContentRadAspNetRef`.
  - e) Click **Next**.
  - f) In the **Access Permissions** window, leave the default settings in place.
  - g) Click **Next**, and then click **Finish**.
5. In the IIS tree pane, expand the machine icon and locate the virtual directory named `ContentAssembler` that you created in the step above.
6. Right click `ContentAssembler` and select **Properties**.
7. Select the **Virtual Directory** tab and perform the following tasks:
  - a) Under the **Application Settings** section, click **Create**.
  - b) From the **Execute permissions** list, select **Scripts only**.
  - c) Click **Apply**.
8. Select the **Documents** tab and perform the following tasks:
  - a) Check **Enable default content page**.
  - b) Click **Add...**
  - c) In the **Default content page** field, type `GuidedNavigation.aspx`.
  - d) Click **OK**.
  - e) Select `GuidedNavigation.aspx` and click **Move Up** until the file is at the top position.
  - f) Click **OK**.
9. Select the **ASP.NET** tab and from the **ASP.NET version** list, select 2.0.x or later.

10. Restart IIS.

## Using the Content Assembler reference application

The Content Assembler reference application allows you to browse the content items that are returned by the Content Assembler API based on your navigation state.

In order to use the reference application to view dynamic page results, you must first have created dynamic page configurations in the Page Builder.

After connecting the reference application to an MDEX Engine, you can choose to run the application in Postback mode, the URL mode, the RAD Toolkit Server Controls Postback mode, or the RAD Toolkit Server Controls URL mode by clicking the corresponding link in header of the reference application.

To view dynamic page results in the reference application:

1. On your Windows machine, start Internet Explorer and navigate to `http://host/ContentAssembler/GuidedNavigation.aspx`.  
Replace *host* with the host name or IP address of the server running the reference application and *ContentAssembler* with the name of the virtual directory in IIS.
2. If you have an MDEX Engine running and want to view its records in the reference application, provide the host and port information of the machine running an MDEX Engine and click **Go**.
3. Search or navigate to a location that triggers a dynamic page.
4. Select a zone from the **Content Rule Zone** drop-down list in the **ContentControl.ascx** area.



**Note:** Choose the zone that corresponds to the zone for the page you want to view. The zone is set based on the zone specified in the page template. Only the zones that have rules triggered by the current navigation state display in the list, although not all the rules contain dynamic page content.

If there is a dynamic page in the specified zone for the current navigation state, an XML representation of the content item returned at that navigation state displays. Otherwise, the following message displays: `There are no content results for zone 'ZoneName' at the current navigation state.`

5. Search or navigate to another location trigger (resetting the zone as necessary) to view the page content that is returned for each trigger.



**Note:** The XML that displays in the reference application is not the same as the XML that represents the page configuration in the MDEX Engine. Rather, it is an XML representation of the content items that are returned by the API after the Content Assembler has evaluated the page configurations and includes the results of any additional queries the Assembler makes to the MDEX Engine.

## Uninstalling the Content Assembler reference application

If you wish to uninstall the Content Assembler reference application, remove the virtual directories in IIS.

To uninstall the Content Assembler reference application for the RAD Toolkit for ASP.NET:

1. From the Windows Control Panel, select **Administrative Tools** > **Internet Information Services Manager**.
2. In the IIS tree pane, expand the machine icon for the local machine, then expand the **Web Sites** directory.
3. Select **Default Web Site**.
4. In the right pane, right-click **ContentAssembler**, or the name of your application's site, click **Delete**, and click **Yes** to confirm the deletion.
5. Close the Internet Information Services window.

## Uninstalling the Content Assembler API

This topic describes how to uninstall the Endeca RAD Toolkit from a system.

To uninstall the Content Assembler API for the RAD Toolkit for ASP.NET:

1. From the Windows Control Panel, select **Add or Remove Programs**.
2. Select **Endeca Content Assembler for RAD Toolkit for ASP.NET** from the list of installed software.
3. Click **Remove** and click **Yes** to confirm the removal.
4. Start Visual Studio and open your Web site.
5. In the Toolbox window, right-click the **Endeca Content Assembler API** tab and select **Delete Tab**.

## Writing applications with the Content Assembler API

This section describes how to use the Content Assembler API for the RAD Toolkit for ASP.NET to query the MDEX Engine and access dynamic page content.

### About using the Content Assembler with the RAD Toolkit for ASP.NET

The Content Assembler API is used in conjunction with the RAD Toolkit for ASP.NET.

Use a `ContentNavigationDataSource` in place of a `NavigationDataSource` to access content results from the Content Assembler in addition to MDEX Engine record data.

If you are using the RAD API for programmatic querying, you can access content items from the results of a `NavigationCommand`. The Content Assembler is not intended for use with records detail, dimension search, or metadata queries.

### Creating a ContentNavigationDataSource control

You create and configure a `ContentNavigationDataSource` control in order to provide dynamic page content and MDEX Engine records to other controls in your Web site.

The `ContentNavigationDataSource` provides the same design time functionality to populate the other controls (e.g. user interface controls) with Endeca record properties as a `NavigationDataSource`, with the addition of the content items returned by the Content Assembler.



**Note:** For more information about configuring `NavigationDataSource` controls, see the *Endeca RAD Toolkit for ASP.NET Developer's Guide*.

To create and configure an Endeca **ContentNavigationDataSource** control:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **ContentNavigationDataSource** on to the **Design** tab of your Web page.
4. From the smart tag, check **Preview Endeca data** to populate other controls you add later with representative data from the MDEX Engine.
5. From the smart tag, select **Configure Data Source....**
6. On the **Choose Endeca server** screen, specify the host and port on which the MDEX Engine is running.
7. At this point, you can either click **Finish** to finish configuring the data source, or you can click **Next** to continue through the wizard and configure optional data source parameters and specify optional Analytics query information. Adding data source parameters makes them available to other controls on the page.
8. In the **Properties** window of Visual Studio, modify the properties for the data source control if necessary. Many of the properties are set when you run the **Configure Data Source...** wizard.
  - a) Specify a value for the **ContentRuleZone** property. This property is required and corresponds to the zone that is specified on the template for the dynamic pages that you want to access with this data source.

The code generated on the **Source** tab is similar to the following:

```
<ccl:ContentNavigationDataSource
  ID="ContentNavigationDS1"
  runat="server"
  MdexPort="7900"
  MdexHostName="smith-690"
  ContentRuleZone="NavigationPageZone"
  ContentValidation="false">
  <PermanentRefinementConfigs>
    <end:RefinementConfig DimensionValueId="3">
    </end:RefinementConfig>
  </PermanentRefinementConfigs>
</ccl:ContentNavigationDataSource>
```

- [About setting the content rule zone](#) on page 52  
When querying the MDEX Engine using the Content Assembler API, you must specify the zone that corresponds to that of the dynamic page content that you want to retrieve.
- [About content XML validation](#) on page 53  
You can enable XML validation of page configurations by setting the **ContentValidation** property of the **ContentNavigationDataSource** to true.

## About setting the content rule zone

When querying the MDEX Engine using the Content Assembler API, you must specify the zone that corresponds to that of the dynamic page content that you want to retrieve.

Using multiple zones can enable your application to implement more fine-grained trigger functionality than is provided by the dynamic business rules feature of the MDEX Engine. For example, zones can allow the front-end application to retrieve dynamic content based on:

- Searches (`Next` and `Nextk` parameters) for terms that are not designated as specific search term triggers, including search within results.
- Record offset (`No` parameter) to present different content when a user is browsing subsequent pages of search results.
- Different views of the same content, for example to present different content depending on whether a user is viewing product details or a product reviews page.

Your front-end application can set the zone for the content query based on conditions like the ones above. However, because the zone for a dynamic page is set based on the `zone` attribute of the `<RuleInfo>` element in the page template, the content administrator must have set up a page intended for a particular condition based on a template that uses the appropriate zone. You can provide information in the template `id` (for example, `ThreeColumnPage-Search`) or `description` to help the content administrator select the appropriate template.

- [Specifying the zone and style for a template](#) on page 19  
Page templates are required to specify a rule zone and a style. When a page is created in the Page Builder, the zone and style are applied to any pages based on that template.

## About content XML validation

You can enable XML validation of page configurations by setting the `ContentValidation` property of the `ContentNavigationDataSource` to `true`.

Validation can be useful in a testing environment for debugging purposes, particularly if templates are changing often. Because of the performance impact of validating content XML, this option should never be used in production. XML validation is disabled in the `ContentNavigationDataSource` by default.

## Building cartridges to render template-based content

Cartridges consist of cartridge templates and their associated rendering code, allowing you to separate the structure of dynamic page content from its presentation.

Building an ASP.NET application based on cartridges involves the following tasks:

- Writing user controls to render content items based on each template.
- For controls that render content items that contain nested content items, adding logic to load the appropriate user control to render the nested content.

The examples in this section use the `DynamicContentControl` and the `IContentControl` provided with the reference application. You can use these controls as a starting point for writing applications to render dynamic page content and extend them with further functionality as needed.

## About working with content items

You can access the `IContentItem` that contains dynamic page content from a `ContentNavigationDataSource`.

An `IContentItem` contains a `KeyedCollection` of `IProperty` objects. An `IProperty` can contain any type of object returned by the MDEX Engine. The type of object depends on the property elements specified in the template. Common object types include:

- `string`

- `ReadOnlyCollection<Record>`
- `IContentItem`
- `NavigationResult`

Because the properties are defined by the template on which a content item is based, you can access the content properties directly based on the property name defined in the template. Typically, you access a specific property value using `ContentItem.Properties["name"].Value` and cast it to the appropriate object type.

- [Types of property elements](#) on page 24  
Each property type corresponds with a particular object type that is returned by the Content Assembler API.

## Using the Content Assembler reference application controls

The `DynamicContentControl` and the `IContentControl` work together to allow you to dynamically load user controls to render page content.

These controls are provided with the reference application code, although the reference application does not demonstrate how to use these controls.

To use the reference application controls in your application:

1. Open Windows explorer and navigate to the reference application directory. In a typical installation this is `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\1.0.0\reference\ContentRadAspNetRef\`
2. Navigate to the following subdirectory: `App_Code\ContentRadAspNetRef`
3. Copy the `DynamicContentControl.cs` and `IContentControl.cs` files to a directory of your choice within your Web site directory structure.

When using a `DynamicContentControl`, you must add the following line to your code:

```
<%@ Register TagPrefix="end" Namespace="ContentRadAspNetRef" %>
```

## Writing user controls to render dynamic content

User controls designed to render template-based content must implement the `IContentControl` interface. This allows a `DynamicContentControl` to load this control and pass a reference to the content item it should render.

To create user control to render dynamic content:

1. Add the following includes at the top of your code:

```
using System.Web.UI;
using Endeca.Data.Content;
using Endeca.Web.Content.UI;
```

2. Implement the `IContentControl` interface.

This example shows the code-behind for a basic implementation:

```
public partial class ContentUserControl : UserControl, IContentControl
{
    public string BaseContentNavigationDataSourceID
    {
        get
```

```

        {
            return baseDataSourceID;
        }

        set
        {
            baseDataSourceID = value;
        }
    }

    public IContentItem ContentItem
    {
        get
        {
            return contentItem;
        }

        set
        {
            contentItem = value;
        }
    }

    private string baseDataSourceID;
    private IContentItem contentItem;
}

```

### 3. In the in-line code, access the properties of the content item for rendering.

For example, if you have the following properties defined in a cartridge template:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
    type="BuyingGuideSection" id="BuyingGuideSection">
    <!-- additional elements not shown in this example -->
    <ContentItem>
        <Name>New Buying Guide Section</Name>
        <Property name="title">
            <String/>
        </Property>
        <Property name="header">
            <String/>
        </Property>
        <Property name="img_src">
            <String/>
        </Property>
        <Property name="products">
            <RecordList/>
        </Property>
    </ContentItem>
    <!-- additional elements not shown in this example -->
</ContentTemplate>

```

The code to render content items based on this template could look like the following:

```

<div class="header">
    <h1><asp:Literal runat="server"
        Text='<%= ContentItem.Properties["title"].Value %>' /></h1>
    <p><asp:Literal runat="server"
        Text='<%= ContentItem.Properties["header"].Value %>' /></p>
    ' />" />

```

```

</div>
<div class="bg_products">
  <ul>
    <asp:Repeater ID="recordsRepeater" runat="server"
      DataSource='<%=# ContentItem.Properties["products"].Value %>'>
      <ItemTemplate>
        <li class="record">
          <h3><%=# ((Record)Container.DataItem).Properties["P_Name"].Value
%></h3>
          <div><%=# ((Record)Container.DataItem).Properties["P_Price"].Value
%></div>
        </li>
      </ItemTemplate>
    </asp:Repeater>
  </ul>
</div>

```

## About the DynamicContentControl

You use a `DynamicContentControl` when rendering dynamic pages or any cartridges that have nested content items within them.

The `DynamicContentControl` is a data-bound control that automatically loads a user control to handle nested cartridge content. Using the `DynamicContentControl` class assumes that you have included a hidden property in each template specifying the name of a user control that should be used to render any content based on that template. For example:

```

<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  type="PageTemplate" id="ThreeColumnPage">
  <!-- additional elements removed from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <Property name="ui_code">
      <String>~/Resources/UserControls/ThreeColumnPage.ascx</String>
    </Property>
    <!-- additional properties removed from this example -->
  </ContentItem>
  <!-- additional elements removed from this example -->
</ContentTemplate>

```

The `~` indicates that the path is relative to the Web site root.

## Using the DynamicContentControl to render pages

Because a content administrator specifies the template that drives a page, you need to be able to dynamically load the appropriate code to render them.

Typically, you will have a generic page that includes a `ContentNavigationDataSource` and a single `DynamicContentControl` to load the code that handles the actual rendering of the page.

To render pages based on different templates:

1. Add and configure a `ContentNavigationDataSource` control.
2. Add a `DynamicContentControl` and set the following properties:



Property	Value
<b>DataSourceID</b>	The ID of the <code>ContentNavigationDataSource</code> to use.
<b>DataMember</b>	The string "ContentItem".
<b>ControlPathPropertyName</b>	The name of the property on the content item that contains the name of the .ascx file to use to render the page. This file should contain a user control that implements <code>IContentControl</code> .
<b>BaseContentNavigationDataSourceID</b>	Set this property to the same value as the <code>DataSourceID</code> .

The following example shows a simple `ContentNavigationDataSource` and a `DynamicContentControl` that will load the appropriate code to render dynamic page content.

```
<%@ Register TagPrefix="end" Namespace="ContentRadAspNetRef" %>

<end:DynamicContentControl
  ID="dynamicContent"
  runat="server"
  DataSourceID="dsNav"
  DataMember="ContentItem"
  ControlPathPropertyName="ui_code"
  BaseContentNavigationDataSourceID="dsNav" />

<end:ContentNavigationDataSource
  ID="dsNav"
  runat="server"
  MdexPort="7900"
  MdexHostName="smith-690"
  ContentRuleZone="NavigationPageZone"
  ContentValidation="false" />
```

## Using the `DynamicContentControl` to render cartridge content

When working with content items that contain nested content items, you can use a `DynamicContentControl` to load the appropriate user control to render the nested content.



**Note:** This procedure assumes that you have a `ContentNavigationDataSource` defined further up in the control hierarchy.

The `DynamicContentControl` loads the control to render cartridges in the same way as it loads the appropriate control to render dynamic pages. The only difference is that you do not need to specify a `DataSourceID` or `DataMember`, as you are setting the data source via the `ContentItem` property.

To use the `DynamicContentControl` to render cartridge content:

1. Add a `DynamicContentControl` and set the following properties:

Property	Value
<b>ControlPathPropertyName</b>	The name of the property on the content item that contains the name of the .ascx file to use to render the page. This file should contain a user control that implements <code>IContentControl</code> .

Property	Value
<b>ContentItem</b>	The ContentItem object that the loaded control should render.
<b>BaseContentNavigationDataSourceID</b>	The BaseContentNavigationDataSourceID of the current user control, which is passed to the new user control that is loaded.

- Repeat the previous step for each nested content item that the current content item can contain. For example, if your template defines sections called `top`, `left`, `middle`, and `right`, you would add four `DynamicContentControl` controls, one to handle each nested content item.

The following example shows a page template and the corresponding user control that uses a `DynamicContentControl` to render cartridge content:

If the template (`PageTemplate-ThreeColumnPage.xml`) includes the following:

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
  type="PageTemplate" id="ThreeColumnPage">
  <!-- additional elements removed from this example -->
  <ContentItem>
    <Name>New Three-Column Page</Name>
    <Property name="ui_code">
      <String>~/Resources/UserControls/ThreeColumnPage.ascx</String>
    </Property>
    <!-- additional properties removed from this example -->
    <Property name="title">
      <String/>
    </Property>
    <Property name="left">
      <ContentItem type="VerticalBanner"/>
    </Property>
    <Property name="middle">
      <ContentItem type="HorizontalBanner"/>
    </Property>
    <Property name="right">
      <ContentItem type="VerticalBanner"/>
    </Property>
  </ContentItem>
  <!-- additional elements removed from this example -->
</ContentTemplate>
```

The code for the associated user control (`ThreeColumnPage.ascx`) may look similar to the following:

```
<%@ Register TagPrefix="end" Namespace="ContentRadAspNetRef" %>

<div class="header"><h1><asp:Literal runat="server"
  Text='<%= ContentItem.Properties["title"].Value %>' /></h1></div>

<div class="left">
  <end:DynamicContentControl
    ID="DynamicContentControl_left"
    runat="server"
    ContentItem='<%= ContentItem.Properties["left"].Value %>'
    ControlPathPropertyName="ui_code"
    BaseContentNavigationDataSourceID=
      '<%= BaseContentNavigationDataSourceID %>' />
</div>

<div class="middle">
```

```
<end:DynamicContentControl
  ID="DynamicContentControl_middle"
  runat="server"
  ContentItem='<## ContentItem.Properties["middle"].Value %>'
  ControlPathPropertyName="ui_code"
  BaseContentNavigationDataSourceID=
    '<## BaseContentNavigationDataSourceID %>' />
</div>

<div class="right">
  <end:DynamicContentControl
    ID="DynamicContentControl_right"
    runat="server"
    ContentItem='<## ContentItem.Properties["right"].Value %>'
    ControlPathPropertyName="ui_code"
    BaseContentNavigationDataSourceID=
      '<## BaseContentNavigationDataSourceID %>' />
</div>
```

## Using the Content Assembler API for programmatic querying

This example code connects to an MDEX Engine, creates and executes a `NavigationCommand`, and retrieves the root content item for a query.

To retrieve content results from a `NavigationCommand`:

1. Add the following includes at the top of your code:

```
using System.Collections.ObjectModel;
using System.Collections.Generic;
using Endeca.Data;
using Endeca.Data.Provider;
using Endeca.Data.Provider.PresentationApi;
using Endeca.Data.Content;
using Endeca.Data.Content.Navigation;
```

2. Create and execute a `NavigationCommand`.

```
// A PresentationApiConnection is an EndecaConnection that uses
// the Presentation API as a transport.
// Future EndecaConnections can use XQuery or other transport mechanisms
PresentationApiConnection conn =
  new PresentationApiConnection("localhost", 8000);

// A NavigationCommand represents a query to the engine that requests
// everything except record/aggregate record details and single/compound
// dimension search
NavigationCommand cmd = new NavigationCommand(conn);

// ... additional code not shown to set Navigation Command values ...

// NavigationResult contains Records, AggregateRecords, Dimensions,
// Breadcrumbs, Analytics, BusinessRules, MetaData, and Supplemental
// Objects.
NavigationResult res = cmd.Execute();
```

### 3. Get the root content item.

```
IContentItem contentItem = NavigationContentItemCreator.Create(cmd,  
    res, "NavigationPageZone", false)
```

In the `Create()` method, the third parameter is the content rule zone and the final parameter controls content XML validation. Validation should never be enabled in a production environment.

For more details on using the RAD API for programmatic querying, see the *Endeca RAD Toolkit for ASP.NET Developer's Guide*.

- [About setting the content rule zone](#) on page 52  
When querying the MDEX Engine using the Content Assembler API, you must specify the zone that corresponds to that of the dynamic page content that you want to retrieve.
- [About content XML validation](#) on page 53  
You can enable XML validation of page configurations by setting the **ContentValidation** property of the `ContentNavigationDataSource` to `true`.

# Index

## A

- application development
  - architecture 12
  - roles 9
  - workflow 12

## C

- cartridges 10, 18, 53, 54
  - and user controls 54
  - building 53
  - definition 10
  - rendering code 54
  - type 18
- content administrator
  - introduced 9
- Content Assembler API
  - and property types defined in template 24
  - and RAD API 59
  - architecture 12
  - introduced 10
- Content Assembler API for the RAD Toolkit for ASP.NET and the RAD Toolkit for ASP.NET 51
  - components 45
  - directory structure 47
  - installing 46, 48
  - overview 45
  - system prerequisites 46
  - uninstalling 51
- Content Assembler reference application
  - deploying 49
  - introduced 48
  - uninstalling 50
  - using 50
- Content Assembler reference application controls
  - using 54
- content items
  - introduced 11
  - and Content Assembler API 53
- content properties
  - accessing 53
- content query
  - and content rule zone 52
  - and content XML validation 53
  - executing 59
  - results 59
- content XML 11
- ContentNavigationDataSource
  - and content rule zone 52
  - and content XML validation 53
- ContentNavigationDataSourceControl 51

## D

- dynamic pages 9, 42
  - and preview application 42
  - troubleshooting 42
- DynamicContentControl 54, 56, 57
  - introduced 56
  - using 56, 57

## E

- emgr\_update
  - get\_templates 38
  - introduced 35
  - remove\_templates 40
  - set\_templates 35
  - updating templates 38
- Endeca Support
  - contacting 8

## G

- group labels 31

## I

- IContentControl 54
  - installing 46, 48

## P

- Page Builder 9, 10, 12
  - architecture 12
  - introduced 9
- preview application 42
- properties 22, 23, 24, 25, 26, 27, 32, 33, 34
  - and configuration in Page Builder 27
  - content item 26
  - defining 22
  - overview 23
  - record list 26
  - string 25
  - template section 26
  - types 24
  - using arbitrary XML 34
  - using pass-through content properties 33
  - XML pass-through 32
- property editors 27, 28, 30, 31
  - cartridge selector 31
  - default value 28
  - grouping 31
  - introduced 27

property editors (*continued*)  
record selector 30  
section 31  
string 28

## R

rule groups 15  
rule styles 15, 17, 19, 40, 42  
and invalid pages 42  
and invalid templates 40  
specifying for page templates 19

## S

styles, See rule styles  
system prerequisites 46

## T

template description  
specifying 21  
template properties  
and types returned by API 24  
templates 10, 15, 17, 18, 19, 21, 22, 23, 24, 26, 27, 35,  
36, 37, 38, 39, 40  
cartridge templates 10  
defining editors 27  
See also property editors  
defining properties 22  
defining sections 26  
managing in Page Builder 35  
modifying 36, 37, 38  
Name property 22  
page templates 10, 19  
pages based on missing templates 39  
prerequisites 15

templates (*continued*)  
properties 23  
See also properties  
property types 24  
removing from Page Builder 39, 40  
retrieving from Page Builder 38  
saving 21  
troubleshooting 40  
updating 36, 37, 38  
uploading 35  
validation 18, 40  
creating 17  
id 18  
page templates 10, 19  
schema 18  
type 18  
validation 18, 40  
thumbnail images  
specifying 21  
using 20

## U

uninstalling  
Content Assembler API for the RAD Toolkit for  
ASP.NET 51  
Content Assembler reference application 50

## Z

zones 15, 16, 19, 40, 42, 52  
and content query 52  
and invalid pages 42  
and invalid templates 40  
creating 16  
specifying for page templates 19  
zones, and dynamic pages overview 16