Endeca® Content Assembler API

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

Version 2.1.2 • December 2011

Contents

Preface.....7
About this guide.....7
Who should use this guide.....7
Conventions used in this guide.....8
Contacting Endeca Customer Support.....8

Chapter 1: Introduction to the Content Assembler API.....9
Overview of the Content Assembler API 9
Content Assembler API components.....10
Overview of the Content Assembler reference application.....11
About handling dynamic content.....11
The reference application model for dynamic content.....12
List of reference application cartridges.....13
Connecting to a different MDEX Engine.....15
About skinning the reference application.....15

Chapter 2: Working with the Content Assembler API.....17
Writing applications with the Content Assembler API 17
About using the Content Assembler with the RAD Toolkit for ASP.NET 17
Creating a ContentNavigationDataSource control 17
About implementing custom trigger conditions.....18
About content XML validation.....21
Building cartridges to render template-based content.....21
About working with content items.....21
Using the Content Assembler reference application controls.....22
Writing user controls to render dynamic content 22
About rendering customized navigation refinements.....24
About rendering customized results lists.....24
About customized results.....25
About rendering record lists.....26
Generating see-all links.....27
About the DynamicContentPlaceHolder.....28
Using the DynamicContentPlaceHolder to render pages.....28
Using the DynamicContentPlaceHolder to render cartridge content.....29
About using the RAD Toolkit for ASP.NET server controls with the Content Assembler.....30
Using the Content Assembler API for programmatic querying 32

Chapter 3: Extending the Content Assembler with Tag Handlers.....35
About tag handlers in the Content Assembler.....35
Scenarios for extending Page Builder and the Content Assembler.....36
Life cycle of a Content Assembler query.....37
Class overview.....38
Implementing the tag handler interface.....39
Resources managed by the ContentContext object.....39
About invoking other tag handlers.....40
Integrating a tag handler into the Content Assembler.....41
Registering a tag handler.....42
Standard tag handlers in the Content Assembler.....42
About the sample tag handler.....43
Installing the sample tag handler.....44
About extending the Content Assembler to validate custom XML.....45

Copyright and disclaimer

Preface

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

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

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

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

About this guide

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

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

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

Who should use this guide

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

If you are a new user of the Endeca Information Access Platform or the Endeca Commerce Suite and you are not familiar with developing Endeca applications, Endeca recommends reading the following guides prior to this one:

1. *Endeca Commerce Suite Getting Started Guide*
2. *Endeca Basic Development Guide*
3. *Endeca Advanced Development Guide*

4. *Endeca RAD Toolkit for ASP.NET Developer's Guide*
5. *Page Builder Developer's Guide*

If you are an existing user of the Endeca Information Access Platform or the Endeca Commerce Suite and you are familiar with developing Endeca applications, Endeca recommends reading the following guides prior to this one:

1. *Endeca Commerce Suite Getting Started Guide*
2. *Endeca RAD Toolkit for ASP.NET Developer's Guide*
3. *Page Builder Developer's Guide*

Remember: All documentation is available on the Endeca Developer Network (EDeN) at *http://eden.endeca.com*.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Endeca Customer Support

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

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

Chapter 1

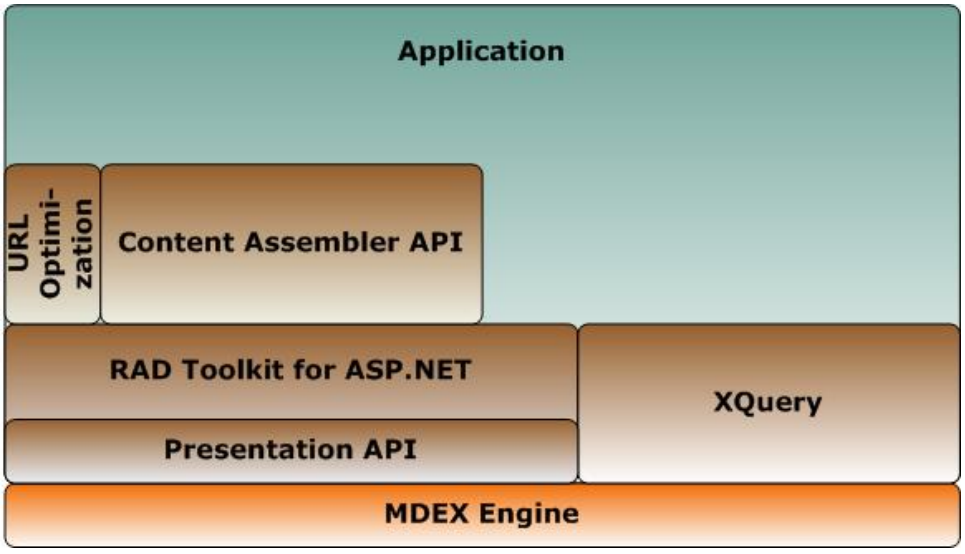
Introduction to the Content Assembler API

This section provides an overview of the Content Assembler API for the RAD Toolkit for ASP.NET and the associated reference application.

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 and is used in conjunction with other Endeca APIs to build configurable Web applications.

The Content Assembler API is designed primarily for search and navigation queries and returns dynamic content if any dynamic pages are triggered by those queries. The Content Assembler API uses the RAD Toolkit for ASP.NET to query the MDEX Engine and provides convenient methods for accessing the content tree that is returned as part of the query results. This content tree reflects the page configuration created by a content administrator in the Page Builder. The tree may contain results from additional queries executed by the Content Assembler that are used to populate page sections based on the configuration returned for the initial query.



Because the Content Assembler uses classes from the RAD Toolkit for ASP.NET such as `NavigationDataSource` and `NavigationCommand`, all queries to the MDEX Engine can be sent through

the Content Assembler API. You can use regular RAD Toolkit methods to access and process query results. Note that only search or navigation queries that trigger a dynamic page return a content tree.

In addition, an Endeca application built with the Content Assembler API can also use the URL Optimization API, available as part of the optional Search Engine Optimization Module. The URL Optimization API also works with the RAD Toolkit for ASP.NET to enable developers to create application links using directory-style URLs with embedded keyword metadata.

Applications built on top of the MDEX Engine version 6.1 or later can also leverage the MDEX API through XQuery, available as part of the Advanced Query Module. There is no explicit support for XQuery within the current version of the Content Assembler; that is, the Content Assembler does not use the MDEX API through XQuery to process queries to the MDEX Engine. However, XQuery for Endeca enables developers to extend MDEX Engine functionality through custom XQuery modules.

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.

The Content Assembler API includes the following additions to the RAD Toolkit for ASP.NET:

Endeca.Data.Content namespace

Class or Interface	Description
`IContentItem`	Defines a content item that represents the dynamic page content configured in the Page Builder. Content items contain a collection of `IProperty` objects that may include child content items.
`IContentItemList`	A list of content items.
`IProperty`	Represents a property defined in the template and configured by the content administrator.

Endeca.Data.Content.Navigation namespace

Class or Interface	Description
`NavigationContentItemCreator`	Generates content item objects from a RAD Toolkit for ASP.NET `NavigationCommand` and `NavigationResult`.
`INavigationRecords`	Used for rendering results lists from a `NavigationRecords` property that has been configured in the Page Builder, including custom sort, relevance ranking, and records-per-page behavior.
`IRecordListProperty`	Used for rendering results lists from a `RecordList` property that has been configured in the Page Builder, including information to create "see-all" links.

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 utility controls are included (along with full source code) in the Content Assembler reference application:

- The `DynamicContentPlaceHolder` 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.

Endeca.Data.Content.Assembler namespace

In addition to the components listed above, the classes in this namespace provide access to core Content Assembler functionality that you can use to extend the Content Assembler. For more information, see "Extending the Content Assembler with Tag Handlers" in this guide.

Overview of the Content Assembler reference application

The Content Assembler reference front-end application demonstrates best practices for using the Content Assembler API to develop configurable applications.

The Content Assembler reference application and sample project is designed to show a typical approach to building cartridges -- that is, templates and their associated rendering code -- and demonstrate how the configuration specified by the content administrator in the Page Builder can affect the display of content in the front-end application. The templates and application code are based on UI best practices developed by Endeca specifically for Guided Navigation applications.

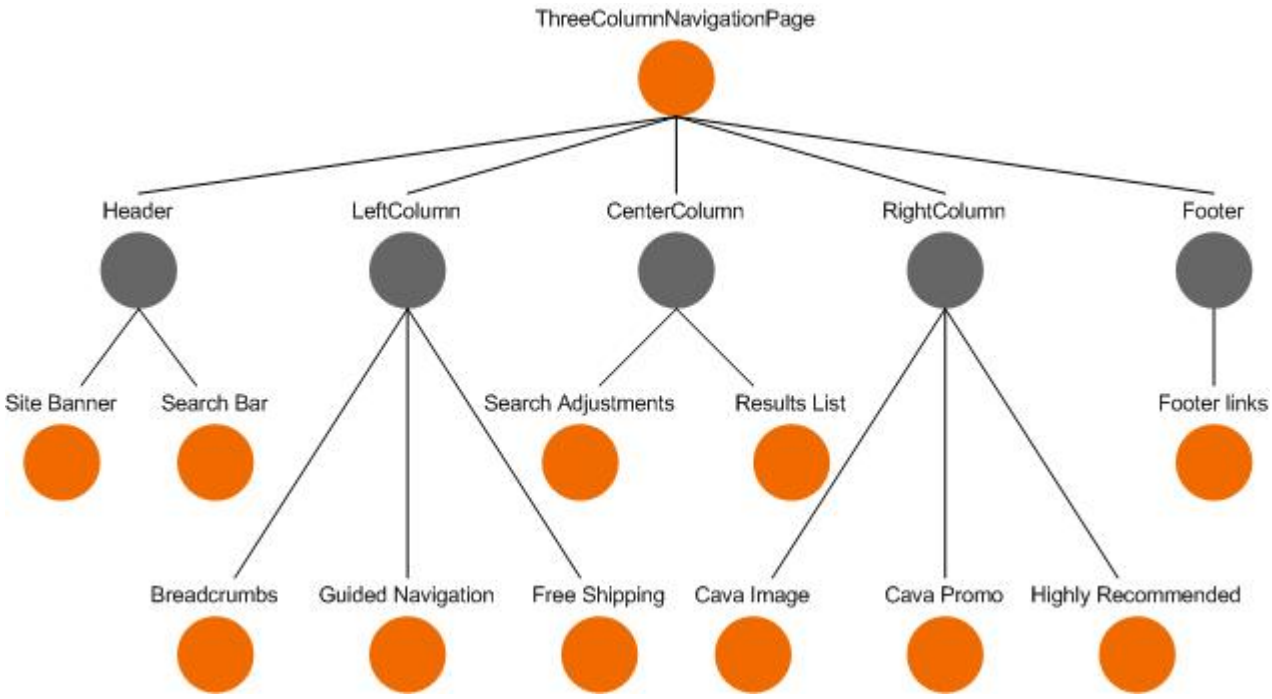
Unlike other Endeca reference applications, the Content Assembler reference application is not intended as a general-purpose data navigator. In order to show realistic examples of cartridge development, the reference application is closely tied to the sample wine data project that is provided with the Content Assembler. For this reason, it is not intended as a generic preview application for the Page Builder in Endeca Workbench.

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

About handling dynamic content

Your application should contain logic to iterate through the content tree returned by the Content Assembler and pass the embedded content items to the appropriate code for rendering.

Recall that the structure of the templates you provide in the Page Builder determines the structure of the content in the page configuration. Templates enable you to specify `<ContentItem>` or `<ContentItemList>` elements that serve as place holders for the content configured by the content administrator. The diagram below shows an example of a fully configured dynamic page.



In this example, each orange dot represents a content item while the gray dots (such as Header and LeftColumn) represent content item lists. You can use both content items and content item lists in your templates, but generally only content items are actually rendered.

Because the template dictates the number and type of properties in a content item, you can write rendering code that is closely tailored to handle the content items based on a particular template. There are several ways that you can then match the content items in the content tree to the appropriate rendering code, for example:

- inspecting the `TemplateId` of the content item
- using a naming convention based on the template id
- using a string property in the template that specifies the name of the class to use for rendering content items based on the template

Content Assembler reference application for the RAD Toolkit for ASP.NET uses a mapping between the template id and the rendering code, specified in the site's `Web.config` file.

The reference application model for dynamic content

In the Content Assembler reference application for the RAD Toolkit for ASP.NET, the `DynamicCon¬tentPlaceHolder` manages the logic of finding the appropriate control to handle each content item.

The `DynamicContentPlaceHolder` is a data-bound control that automatically loads a user control to handle nested cartridge content based on the template id and the mapping specified in the site's `Web.config` file.

The following example from `Web.config` shows the format of the mapping that is used by the `Dy¬namicContentPlaceHolder` between the template id and the path to the class designed to render cartridges based on that template.

The class specified here is the class that is initially loaded to handle the content item. In some cases, the content is then passed to another class for the actual rendering. See `GuidedNavigation.ascx` and `ResultsList.ascx` for examples.

```
<configuration>
  <configSections>
<!- additional elements not shown in this example -->
    <sectionGroup name="content.config"
      type="ContentRef.Config.ContentConfigSectionGroup">
      <section name="templateHandlers"
        type="ContentRef.Config.TemplateHandlersSection"/>
    </sectionGroup>
  </configSections>
<!- additional elements not shown in this example -->
  <content.config>
    <templateHandlers>
      <add templateId="Breadcrumbs" handlerPath="~/Resources/ContentCon¬
trols/Navigation/Breadcrumbs.ascx" />
     <add templateId="GuidedNavigation" handlerPath="~/Resources/Content¬
Controls/Navigation/GuidedNavigation.ascx" />
     <add templateId="ImageBox" handlerPath="~/Resources/ContentControls/Ba¬
sics/Image.ascx" />
     <add templateId="ThreeRecordBox" handlerPath="~/Resources/ContentCon¬
trols/Spotlights/ThreeRecordBox.ascx" />
     <add templateId="TextBox" handlerPath="~/Resources/ContentControls/Ba¬
sics/Text.ascx" />
     <add templateId="SearchAdjustments" handlerPath="~/Resources/Content¬
Controls/Search/SearchAdjustments.ascx" />
     <add templateId="DimensionSearchResults" handlerPath="~/Resources/Con¬
tentControls/Search/DimensionSearchResults.ascx" />
      <add templateId="ImageBanner" handlerPath="~/Resources/ContentCon¬
trols/Basics/Image.ascx" />
      <add templateId="OneRecordBanner" handlerPath="~/Resources/Content¬
Controls/Spotlights/OneRecordBanner.ascx" />
      <add templateId="ResultsList" handlerPath="~/Resources/ContentCon¬
trols/Records/ResultsList.ascx" />
      <add templateId="TextBanner" handlerPath="~/Resources/ContentCon¬
trols/Basics/Text.ascx" />
     <add templateId="ThreeRecordBanner" handlerPath="~/Resources/Content¬
Controls/Spotlights/ThreeRecordBanner.ascx" />
      <add templateId="ImageSiteBanner" handlerPath="~/Resources/Content¬
Controls/Basics/Image.ascx" />
      <add templateId="SearchBar" handlerPath="~/Resources/ContentCon¬
trols/Search/SearchBar.ascx" />
      <add templateId="ThreeColumnNavigationPage" handlerPath="~/Re¬
sources/ContentControls/Pages/ThreeColumnNavigationPage.ascx" />
    </templateHandlers>
  </content.config>
<!- additional elements not shown in this example -->
</configuration>
```

Note that the same code may be used to handle more than one template, if the properties defined in the templates are sufficiently similar.

List of reference application cartridges

The reference application includes sample cartridges that enable configuration of a variety of front-end features.

For implementation details, refer to the templates (located in your reference application deployment at *[appDir]*\config\page_builder_templates) and the rendering code (located in C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET*version*\reference\ContentAssemblerRefApp\Resources\ContentControls).

Template name	Rendering code	Description
FullWidthContent-ImageSiteBanner	Basics\Image.ascx	Displays the site banner image with an optional link.
FullWidthContent-SearchBar	Search\SearchBar.ascx	Displays the search bar.
MainColumnContent-DimensionSearchResults	Search\Dimension¬SearchResults.ascx	Displays dimension search results. Content administrators can configure whether or not to display compound dimension search results.
MainColumnContent-ImageBanner	Basics\Image.ascx	Displays an image banner with an optional link.
MainColumnContent-OneRecordBanner	Spotlights\OneRecord¬Banner.ascx	Displays one record spotlight with an image.
MainColumnContent-ResultsList	Records\Result¬sList.ascx	Displays search and navigation results in a list view.
MainColumnContent-SearchAdjustments	Search\SearchAdjust¬ments.ascx	Displays search adjustment messaging such as Did You Mean or spelling correction.
MainColumnContent-TextBanner	Basics\Text.ascx	Displays promotional text with a title and an optional link.
MainColumnContent-ThreeRecordBanner	Spotlights\ThreeRe¬cordBanner.ascx	Displays a three record spotlight banner.
SidebarItem-Breadcrumbs	Navigation\Bread¬crumbs.ascx	Displays breadcrumbs appropriate to the current refinement state.
SidebarItem-GuidedNavigation	Nsvigation\GuidedNav¬igation.ascx	Displays Endeca Guided Navigation with configurable display of dimensions.
SidebarItem-ImageBox	Basics\Image.ascx	Displays an image with an optional link.
SidebarItem-TextBox	Basics\Text.ascx	Displays promotional text with a title and an optional link.
SidebarItem-ThreeRecordBox	Spotlights\ThreeRe¬cordBox.ascx	Displays a three record spotlight box.

Note: Text.ascx in the reference application applies HTML escaping to the strings specified by the content administrator in the Page Builder. If you want to allow content administrators to enter HTML-formatted text in the Page Builder, create a separate cartridge with rendering code that does not escape HTML strings.

The reference application also includes a page template named PageTemplate-ThreeColumnNavigationPage, which controls the overall page content and ThreeColumnNavigationPage.ascx (located in C:\Endeca\ContentAssemblerAPIs\RAD

Toolkit for ASP.NET\2.0.0\reference\ContentAssemblerRefApp\Resources\ContentControls\Pages), which controls the overall rendering of the page.

Connecting to a different MDEX Engine

By default the Content Assembler reference application attempts to connect to an MDEX Engine running on localhost port 15000 (the default port in the sample wine deployment). If you are running the MDEX Engine on a different host or port, you can update the configuration in the site's `Web.config` file.

To specify a different MDEX Engine host or port:

1. Navigate to the location of the Content Assembler reference application. In a typical installation, this is: `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\2.0.0\reference\ContentAssemblerRefApp`.
2. Open the `Web.config` file and locate the following section:

```
 <endeca>
<!- additional elements not shown in this example -->
  <servers>
   <clear/>
   <add name="Local" hostName="localhost" port="15000" certificatePath=""/>

      </servers>
 </endeca>
```

3. To change the host name of the MDEX Engine server, update the value of the `hostName` parameter.
4. To change the port of the MDEX Engine server, update the value of the `port` parameter.
5. Save and close the file.
6. Restart IIS.

About skinning the reference application

The styling of the reference application is implemented through external CSS style sheets, which can be easily customized.

The style sheets are located in the `reference/ContentAssemblerRefApp/css` directory of your Content Assembler API installation. In a typical installation, this is `C:\Endeca\ContentAssemblerAPIs\RAD Toolkit for ASP.NET\`*`version`*`\reference\ContentAssemblerRefApp\css`.

Each cartridge component (or type of component) in the reference application has a corresponding style sheet that controls the appearance of that component.

Chapter 2

Working with the Content Assembler API

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

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.

      In most cases you only need one zone for all your landing pages. Using multiple zones can enable you to provide different perspectives on the same navigation state within your application.

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

```
<cc1:ContentNavigationDataSource
     ID="ContentNavigationDS1"
     runat="server"
     MdexPort="7900"
     MdexHostName="smith-690"
     ContentRuleZone="NavigationPageZone"
     ContentValidation="false">
   <PermanentRefinementConfigs>
      <end:RefinementConfig DimensionValueId="3">
      </end:RefinementConfig>
   </PermanentRefinementConfigs>
</cc1:ContentNavigationDataSource>
```

About implementing custom trigger conditions

Because the Content Assembler API retrieves page content based on Endeca's dynamic business rules functionality, pages can only be triggered on record-filtering dimension values, specific search terms, a date range, or a single user profile identifier.

These limitations can make it difficult to handle certain scenarios such as the following:

- **Search results pages.** 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 page to display as your site's home page (at location N=0) that does not include any record 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.
- **Record offset pages.** There is no simple way to explicitly trigger different content for the first page of record results (at offset=0) and for subsequent pages, with different page configurations specified by the content administrator in the Page Builder.
- **Alternate views on the same navigation state.** Use cases include A/B testing or toggling between a product details view and a customer reviews view. By default, the Content Assembler API returns a single content tree representing a dynamic page for any given navigation state or trigger condition.

There are various approaches that can be used to handle these use cases:

- Filtering landing pages based on rule properties
- Using hidden dimensions
- Using multiple rule zones
- Using multiple user profiles

Any of these strategies can be applied to the scenarios listed above. They can also be used to implement other custom trigger conditions that you may require. Which approach you use depends on the scenario you are trying to address and the specifics of your application. For guidance on selecting the appropriate option (or combination of options) and assistance with implementation, contact your Endeca representative.

About filtering landing pages based on rule properties

If you specify custom rule properties in a page template, you can use those properties to exclude certain landing pages from consideration by the MDEX Engine on a per-query basis.

Filtering based on rule properties can enable your application to implement more fine-grained trigger functionality than is available in the Page Builder.

Because the rule properties for a dynamic page are set based on the properties specified within the `<RuleInfo>` element in the page template, the content administrator must have set up a page intended for a particular trigger condition based on a template with the appropriate property. You can provide information in the template `id` (for example, `ThreeColumnPage-Search`) or `description` to help the content administrator select the appropriate template.

For the purposes of priority, pages based on templates with custom rule properties should be treated as if they have more specific trigger conditions than the same page with no such properties. (In general, pages with more specific triggering conditions should have higher priority than more generic pages.)

Because the Page Builder preview functionality cannot replicate your custom logic for filtering pages, the preview status messages may be misleading when you exclude certain pages from consideration. However, if your preview application includes the appropriate logic, the correct page displays in the preview pane even if the status messages indicate that a different page fired.

Use case: Search results

You can enable more robust handling of search results pages by creating a template that specifies a custom rule property with a key such as `search_results` and a value of `true`. The content administrator can then create search results pages based on this template. You can add logic to your application to consider these pages only for search queries (that is, queries that include `Ntt` and `Ntk` parameters). If there are no search parameters present, you can augment the query with a filter such as `Nmrf=not(search_results:true)` before you pass it to the MDEX Engine via the Content Assembler API.

For more information about working with rule properties, see "Promoting records with dynamic business rules" in the *Advanced Development Guide*.

About using hidden dimensions to trigger landing pages

You can create specialized dimensions in your application to expose additional trigger conditions.

This approach involves some additional work in your data pipeline to apply the dimension values to the records. Once this is done, the content administrator can select the trigger condition in the Page Builder using the same process as any navigation state.

Use case: Record offset

You can enable different landing pages based on record offset by creating a dimension such as Offset with dimension values such as First Page and Next Pages. During the ITL process, apply both the Offset > First Page and Offset > Next Pages dimension values to all records. The content administrator can then set up pages for each trigger condition.

You can add logic to your application to augment the navigation filter (`N` parameter) based on the record offset value (the `No` parameter).

For more information about working with dimensions, see the *Forge Guide*, *Basic Development Guide*, and the *Endeca Developer Studio Help*.

About using multiple rule zones for landing pages

Using multiple zones can enable you to provide different perspectives on the same navigation state within your application.

Because the zone for a 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 display condition based on a template that uses the appropriate zone. You can provide information in the template `id` or `description` to help the content administrator select the appropriate template for each case.

Because the Page Builder preview functionality does not limit the query to a single zone, the preview status messages may be misleading when you use multiple zones. However, if your preview application includes the appropriate logic, the correct page should display in the preview pane even if the status messages indicate that more than one page fired.

Also note that although the Content Assembler API only retrieves the content tree from a specific zone, the results from all zones with triggered content are returned as part of the query response, so excessive use of multiple zones may lead to a noticeable increase in the size of the query response.

Use case: A/B testing

You can enable A/B testing scenarios by setting up different zones such as Control, VariableA, VariableB, and so on. You then create different templates for each zone, and the content administrator can create pages based on the different templates.

Your front-end application can set the zone for the content query based on various conditions for which you want to expose different views on the data.

For more information about setting up rule zones for landing pages, see the *Page Builder Developer's Guide*.

About using multiple user profiles for custom trigger conditions

You can use the user profile functionality to provide different views on the same navigation states.

You can set up specialized user profiles to enable content administrators to set up different pages in the Page Builder for different scenarios. However, if you are already using user profiles for other purposes, this usage may interfere with other user profile triggers.

Use case: Different front-end sites backed by the same data

You can present different views on the same data by creating different user profiles in Developer Studio such as SiteAUser and SiteBUser. In the Page Builder, the content administrator can set the user profile to use for each page.

You can add logic to your application to add the appropriate user profile to the query by setting `NavigationCommand.UserProfiles` in the query.

For more information about setting up user profiles, see the *Endeca Developer Studio Help*. For more information about working with user profiles, see "Implementing User Profiles" in the *Advanced Development Guide*.

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 `DynamicContentPlaceHolder` and the `IContentControl` that are included as part of the reference application. You can adapt the entire reference application, or simply 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:

- `bool`
- `string`
- `IContentItem`
- `IContentItemList`
- `INavigationRecords`
- `ReadOnlyCollection<Record>`

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.

Using the Content Assembler reference application controls

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

These controls are included as part of the reference application. You can use these controls as components in your own custom applications and extend them with further functionality as needed.

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\`*`version`*`\reference\ContentAssemblerRefApp\`
2. Navigate to the `App_Code` subdirectory.
3. Copy the following files to a directory of your choice within your Web site directory structure:
   - `ContentPathManager.cs`
   - `DynamicContentPlaceHolder.cs`
   - `IContentControl.cs`
   - `Config\ContentConfigSectionGroup.cs`
   - `Config\TemplateHandler.cs`
   - `Config\TemplateHandlersSection.cs`

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

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

If you modify the code and change the namespace of your custom `DynamicContentPlaceHolder`, update this line accordingly.

Writing user controls to render dynamic content

User controls designed to render template-based content must implement the `IContentControl` interface. This allows a `DynamicContentPlaceHolder` 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;
```

2. Implement the `IContentControl` interface.
   This example shows the code-behind for a basic implementation:

```
  public partial class ContentUserControl : UserControl, IContentControl

  {

     public IContentItem ContentItem
     {
        get
        {
           return contentItem;
        }

        set
```

```
        {
           contentItem = value;
        }
     }

     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="SidebarItem" id="TextBox">
  <!-- additional elements not shown in this example -->
  <ContentItem>
    <Name>New Text Box</Name>
    <Property name="title">
      <String/>
    </Property>
    <Property name="body">
      <String/>
    </Property>
    <Property name="link_text">
      <String/>
    </Property>
    <Property name="link_href">
      <String/>
    </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="TextBanner">
  <div class="Title"><%# (string)ContentItem.Properties["title"].Value
%></div>
  <div class="Body"><%# (string)ContentItem.Properties["body"].Value
%></div>
  <div class="Link">
    <a href="<%# (string)ContentItem.Properties["link_href"].Value %>">
      <%# (string)ContentItem.Properties["link_text"].Value %>
    </a>
  </div>
</div>
```

The following example shows a content item list property in a page template and how the corresponding rendering code can display the results.

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

```
<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="left_column">
      <ContentItemList type="SidebarItem" />
    </Property>
<!-- additional elements deleted from this example -->
```

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

The associated rendering code (`ThreeColumnNavigationPage.ascx`) may look similar to the following:

```
<div id="LeftColumn">
  <asp:Repeater runat="server" DataSource='<%#ContentItem.Proper¬
ties["left_column"].Value %>'>
    <ItemTemplate>
      <!-- Dynamically include rendering code for each content item in the
 list. -->
      <end:DynamicContentPlaceHolder runat="server" ContentItem='<%# Con¬
tainer.DataItem %>' />
    </ItemTemplate>
  </asp:Repeater>
</div>
```

About rendering customized navigation refinements

User controls designed to render customized navigation refinements must implement the `IContent¬Control` interface to access the configured `DimensionList` values.

For example:

```
DimensionStatesResult refinements = (DimensionStatesResult)ContentItem.Prop¬
erties["refinements"].Value;
```

This code is equivalent to using the `DimensionStatesResult` from a `ContentNavigationData¬Source` or a `NavigationCommand`, except that the dimensions returned reflect the content administrator's configuration specified in the Page Builder.

Note: If you have precedence rules defined in your application, they still apply to the customized `DimensionList`. This means that if the landing page definition specifies certain dimensions for display that should not display for that navigation state (whether it is due to precedence rules or because it is not a valid refinement), those invalid dimensions are not included in the `Dimen¬sionList` object.

The Content Assembler reference application provides a sample Endeca Guided Navigation cartridge (including rendering code) that uses a navigation refinements property.

About rendering customized results lists

If you enable content administrators to customize the display of record results, the results object returned by the Content Assembler API is different from the object returned by the RAD Toolkit for ASP.NET.

Recall that you can specify a `<NavigationRecords>` property in a template with a `<Navigation¬RecordsEditor>` that enables a content administrator to specify sort order, relevance ranking, and the number of records to display per page.

To render the customized navigation results, retrieve the list of records from the navigation records property, which is of type `INavigationRecords`. For example:

```
INavigationRecords navRecs =
   (INavigationRecords) ContentItem.Properties["navigation_records"].Value;
ReadOnlyCollection<Record> recs = navRecs.RecordsResult.Records;
```

This code is equivalent to using the `RecordsResult` from a `ContentNavigationDataSource` or a `NavigationCommand`, except that the records returned reflect the content administrator's configuration specified in the Page Builder.

You can also use members of the `INavigationRecords` object as a data source. The Content Assembler reference application provides a sample cartridge for rendering results lists, including sample code for using members of `INavigationRecords` as a data source. Refer to the `MainColumnContent-ResultsList.xml` template and the `ResultsSet.ascx` and `RecordsControl.ascx` classes for more details.

The `INavigationRecords` object also exposes the following members with values based on the modified query that is used to retrieve the customized results:

- `ActiveSorts`
- `AggregateRecordsResult`
- `AggregationKey`
- `AggregationKeys`
- `RecordsResult`
- `SortKeys`

When working with customized results lists, you must use the `INavigationRecords` members, rather than the `NavigationResults` from the main `ContentNavigationDataSource`.

For example, when rendering a pager component for a customized record list, you should use `navRecs.RecordsResult.RecordsPerPage` because the content administrator may have specified a different number of records per page from the main query (which is reflected in `NavigationRe¬sults.RecordsResult.RecordsPerPage`).

For further details, refer to the *Endeca API Reference for the Content Assembler API for the RAD Toolkit for ASP.NET*.

About customized results

The Content Assembler handles sort order, relevance ranking, and records-per-page customization slightly differently than the RAD Toolkit for ASP.NET. See the sections below for details about how the Content Assembler handles each configuration option.

The Content Assembler performs an additional query in order to retrieve the customized record results from the MDEX Engine. If no custom behavior was specified in the Page Builder, no additional query is made.

Sort order

The sort order specified by the content administrator in the Page Builder is used as a default. End users of the Web application can override this setting if you enable a control for users to specify sort order.

Relevance ranking

If the content administrator specifies both a sort order and a relevance ranking strategy for a single landing page and the query that triggers the page contains a search, the Content Assembler passes only the relevance ranking strategy on to the query to retrieve the customized navigation records. If no search is present, both the sort order and the relevance ranking strategy are passed on to the second query. In this case, the sort order overrides the relevance ranking.

The relevance ranking strategy specified by the content administrator for a landing page always overrides any other relevance ranking setting (whether it is coded as default behavior in the application or -- less typically -- specified by an end user).

Records per page

The `NavigationRecordsEditor` provides an optional interface for the content administrator to specify the number of records to return per page for a given navigation state.

The case where a content administrator has configured a value for records per page and an end user also specifies a value can lead to undefined and unexpected behaviors. For this reason, if you enable configuration of records-per-page display in the Page Builder, it is not recommended that you enable a control for end users to specify records per page in the application.

About rendering record lists

Record list properties represent the results of supplemental queries, for example, to populate promotions or Content Spotlighting cartridges.

Properties containing record list values are returned as instances of `IRecordListProperty`, which is a sub-interface of `IProperty`. Content administrators can designate either specific records or a navigation query that returns records for spotlighting. An `IRecordListProperty` that is configured to display specific featured records always returns a `ReadOnlyCollection<Record>`.

Note: When a cartridge is configured to display specific featured records and any of the specified record IDs are invalid, the Content Assembler API for the RAD Toolkit for ASP.NET returns only the records that have valid IDs.

An `IRecordListProperty` that is populated with a `NavigationCommand` returns either a `ReadOnlyCollection<Record>` or a `ReadOnlyCollection<AggregateRecord>`, depending on whether the `NavigationCommand` that triggered a landing page has the `AggregationKey` property set.

If you use rollup keys for aggregated records in your application, then you must check the type of list being returned for any `IRecordListProperty` in one of two ways:

- Check the type of the record list `IProperty.Value` to determine whether it is a `ReadOnlyCollection<AggregateRecord>` or a `ReadOnlyCollection<Record>`.
- Cast the `IProperty` to an `IRecordListProperty`, and check the `bool` value of the `ContainsAggregateRecords` property.

Based on the type of list returned, your application must handle records or aggregated records as appropriate.

If you prefer to render records rather than aggregated records for a Content Spotlighting cartridge on a page with a rollup key, you can render a representative record from the list of constituent records. For example, for each aggregated record, the application can retrieve the first constituent record in the list as follows:

```
Record record = aggregateRecord.Records[0];
```

where `aggregateRecord` is the `AggregateRecord` object.

The Content Assembler reference application provides several sample spotlight cartridges that demonstrate how to render a record list property.

Generating see-all links

You can provide front-end users with a "see-all" link to display the full results set of a navigation query that was used to populate a spotlight cartridge.

The `IRecordListProperty` interface has additional public properties for the corresponding `NavigationCommand` object that aids in creating see-all links.

Note: See-all links cannot be generated for record lists that are returned from record queries. The `NavigationCommand` object is null when a record query is used to specify the record list.

In URL-based RAD.NET applications, the `NavigationCommand` object can be serialized using the RAD.NET `CommandSerialization` class that uses the application `CommandSerializationProvider`. The serialized command can then be used when forming the see-all link URL.

To create a see-all link:

1. Retrieve the record list `IProperty` object off an `IContentItem`.

```
IProperty property = contentItem.Properties["products"];
NavigationCommand navigationCommand = null;
if (property is IRecordListProperty) {

   //Cast the Property to IRecordListProperty
   IRecordListProperty recListProperty = (IRecordListProperty)property;

   navigationCommand = recListProperty.NavCommand;

}
```

2. Check that the `NavigationCommand` is not null, then use the `CommandSerialization` class to serialize the command.

```
//Check that the navigationCommand is not null:

if (navigationCommand != null) {

   String serializedCommand = CommandSerialization.Serialize(navigationCommand);

}
```

   The serialized command can then be used to generate the link URL.

If you plan to construct URLs using the `UrlManager` object from the RAD Toolkit for ASP.NET, please refer to the *RAD Toolkit for ASP.NET Developer's Guide* for more information. To see an example cartridge that uses a `UrlManager`, refer to the `ThreeRecordBannerSpotlightControl.ascx.cs` file located in the `Resources\UserControls` directory of your Content Assembler reference application installation directory.

About the DynamicContentPlaceHolder

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

The `DynamicContentPlaceHolder` is a data-bound control that automatically loads a user control to handle nested cartridge content. Using the `DynamicContentPlaceHolder` class to render a dynamic page, you would have a generic page that includes a `ContentNavigationDataSource` and a single `DynamicContentPlaceHolder` to load the code that handles the actual rendering of the page.

Related Links

Using the DynamicContentPlaceHolder to render pages on page 28
Because a content administrator chooses which template drives a page, you need to be able to dynamically load the appropriate code to render them.

Using the DynamicContentPlaceHolder to render cartridge content on page 29
When working with content items that contain nested content items, you can use a `Dynamic¬ContentPlaceHolder` to load the appropriate user control to render the nested content.

Using the DynamicContentPlaceHolder to render pages

Because a content administrator chooses which template 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 `DynamicContentPlaceHolder` 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 `DynamicContentPlaceHolder` and set the following properties:

Property	Value
`DataSourceID`	The `ID` of the `ContentNavigationDataSource` to use.
`DataMember`	The string `"ContentItem"`.

The following example shows a simple `ContentNavigationDataSource` and a `DynamicContent¬PlaceHolder` that loads the appropriate code to render dynamic page content.

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

   <end:DynamicContentPlaceHolder
     ID="content"
     runat="server"
     DataSourceID="dsNav"
     DataMember="ContentItem" />


   <end:ContentNavigationDataSource
      ID="dsNav"
      runat="server"
     MdexHostName='<%$ EndecaConfig:Servers.Servers["Local"].HostName %>'
```

```
      MdexPort='<%$ EndecaConfig:Servers.Servers["Local"].Port %>'
      ContentRuleZone="NavigationPageZone"
      EnableExposeAllRefinements="true" />
```

Related Links

Using the DynamicContentPlaceHolder to render cartridge content on page 29
When working with content items that contain nested content items, you can use a `DynamicContentPlaceHolder` to load the appropriate user control to render the nested content.

About the DynamicContentPlaceHolder on page 28
You use a `DynamicContentPlaceHolder` when rendering dynamic pages or any cartridges that have nested content items within them.

Using the DynamicContentPlaceHolder to render cartridge content

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

The `DynamicContentPlaceHolder` 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 `DynamicContentPlaceHolder` to render cartridge content:

1. Add a `DynamicContentPlaceHolder` and set the following property:

Property	Value
`ContentItem`	The `ContentItem` object that the loaded control should render.

2. Repeat the previous step for each nested content item that the current content item can contain. For example, if your template defines sections called `LeftColumn`, `CenterColumn`, and `RightColumn`, you would add three `DynamicContentPlaceHolder` controls, one to handle each nested content item.

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

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

```
<ContentTemplate xmlns="http://endeca.com/schema/content-template/2008"
type="PageTemplate" id="ThreeColumnNavigationPage">
  <!-- additional elements removed from this example -->
  <ContentItem>
    <Name>New Three-Column Navigation Page</Name>
    <!-- additional properties removed from this example -->
    <Property name="left_column">
      <ContentItemList type="SidebarItem" />
    </Property>
    <Property name="center_column">
      <ContentItemList type="MainColumnContent" />
    </Property>
    <Property name="right_column">
      <ContentItemList type="SidebarItem" />
```

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

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

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

<div class="LeftColumn">
  <asp:Repeater runat="server" DataSource='<%#ContentItem.Proper¬
ties["left_column"].Value %>'>
    <ItemTemplate>
      <end:DynamicContentPlaceHolder runat="server" ContentItem='<%# Con¬
tainer.DataItem %>' />
    </ItemTemplate>
  </asp:Repeater>
</div>

<div id="CenterColumn">
    <div id="CenterContent">
   <asp:Repeater runat="server" DataSource='<%#ContentItem.Properties["cen¬
ter_column"].Value %>'>
      <ItemTemplate>
        <end:DynamicContentPlaceHolder runat="server" ContentItem='<%#
Container.DataItem %>' />
      </ItemTemplate>
  </asp:Repeater>
</div>

<div id="RightColumn">
    <div id="CenterContent">
    <asp:Repeater runat="server" DataSource='<%#ContentItem.Proper¬
ties["right_column"].Value %>'>
      <ItemTemplate>
        <end:DynamicContentPlaceHolder runat="server" ContentItem='<%#
Container.DataItem %>' />
      </ItemTemplate>
  </asp:Repeater>
</div>
```

Related Links

Using the DynamicContentPlaceHolder to render pages on page 28
Because a content administrator chooses which template drives a page, you need to be able to dynamically load the appropriate code to render them.

About the DynamicContentPlaceHolder on page 28
You use a `DynamicContentPlaceHolder` when rendering dynamic pages or any cartridges that have nested content items within them.

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

If you prefer not to use user controls similar to those in the Content Assembler reference application to render certain Endeca features, you can use the RAD Toolkit for ASP.NET server controls. In this

case, the way in which the data sources are passed to the server controls is slightly different from the way it is normally done in the RAD Toolkit for ASP.NET.

To use server controls to render features on landing pages that may have customized navigation refinements or results lists, you can use a content item member as a data source. The `DataSource` is used to render the feature, while the `NavigationDataSourceID` represents the overall navigation state for the page to generate URLs for any follow-on queries.

Guided Navigation

The following is a typical example in the RAD Toolkit for ASP.NET:

```
<cc1:GuidedNavigation
  ID="GuidedNavigation1"
  runat="server"
  DataSourceID="dsNav">
</cc1:GuidedNavigation>
```

The same example with the Content Assembler:

```
<cc1:GuidedNavigation
   ID="guidedNavigationServerControl"
   runat="server"
  DataSource='<%# new object[] {ContentItem.Properties["refinements"].Value}
 %>'
   NavigationDataSourceID="dsNav">
   <DimensionStateGroupTemplate></DimensionStateGroupTemplate>
</cc1:GuidedNavigation>
```

Note: The default behavior of the server control renders dimensions arranged by dimension groups. By specifying an empty `DimensionStateGroupTemplate` you suppress the display of dimension group names in cases where a content administrator may have customized the display of dimensions.

Breadcrumbs

Breadcrumbs server controls work with the Content Assembler without any changes, as in this example:

```
<cc1:Breadcrumbs
   ID="breadcrumbsServerControl" runat="server"
   DataSourceID="dsNav"
   RemoveImageUrl="<%~/images/x.gif%>" />
```

Pager

The following is a typical example in the RAD Toolkit for ASP.NET:

```
<cc1:Pager
   ID="topPaging1"
   runat="server"
   DataSourceID="dsNav" />
```

The same example with the Content Assembler:

```
<cc1:Pager
   ID="topPaging1"
   runat="server"
   ItemsType='<%# getPagerItemsType() %>'
```

```
   DataSource='<%# getRecordsResult() %>'
   NavigationDataSourceID="dsNav" />
```

In addition, the following two methods need to be defined in the code behind:

```
protected PagerItemsType getPagerItemsType()
      {
         INavigationRecords records =
(INavigationRecords)ContentItem.Properties["navigation_records"].Value;
         if (!String.IsNullOrEmpty(records.AggregationKey))
         {
            return PagerItemsType.AggregateRecords;
         }
         return PagerItemsType.Records;
      }

      protected object getRecordsResult()
      {
        INavigationRecords records = (INavigationRecords)ContentItem.Prop¬
erties["navigation_records"].Value;
         object result = records.RecordsResult;
         if (!String.IsNullOrEmpty(records.AggregationKey))
         {
            result = records.AggregateRecordsResult;
         }
         return new object[] { result };
      }
```

Tag cloud

The following is a typical example in the RAD Toolkit for ASP.NET:

```
<cc1:TagCloud
  ID="tagcloud"
  runat="server"
  DataSourceID="dsNav"
  DimensionId="8">
</cc1:TagCloud>
```

The same example with the Content Assembler:

```
<cc1:TagCloud
  ID="tagcloud"
  runat="server"
  NavigationDataSourceID="dsNav"
  DataSource='<%# new object[] {ContentItem.Properties["refinements"].Value}
 %>'
  DimensionId="8">
</cc1:TagCloud>
```

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*.

Chapter 3

Extending the Content Assembler with Tag Handlers

The Content Assembler uses tag handlers to transform content XML into an object representation of a dynamic page. Tag handlers can be written by the Endeca community (including Endeca Professional Services, partners, or customers) in order to customize or extend the Content Assembler to process custom content XML and integrate with third-party systems.

About tag handlers in the Content Assembler

A tag handler enables you to define your own processing logic for the content that is configured by content administrators in Page Builder.

When your application queries the MDEX Engine using the Content Assembler API, the corresponding landing page configuration is returned as part of the response in the form of content XML. The Content Assembler processes this XML, executing additional queries as needed, and returns a tree of `ICon¬tentItem` objects and their associated properties.

Each of the standard property types in Page Builder is represented by an element in XML, such as `<String>`, `<RecordList>`, or `<ContentItem>`. For each of the standard types, the Content Assembler has a standard tag handler associated with that element that processes the element into an object.

You can take advantage of the same mechanism to write a tag handler that processes a specific element in the content XML and returns objects to the application. Community tag handlers process elements outside of the Endeca content XML namespaces (that is, `http://endeca.com/schema/con¬tent/2008` and `http://endeca.com/schema/content-tags/2008`). These elements may be either pass-through XML defined in the template or custom XML generated by a community editor. For more information on pass-through XML and community editors, refer to the *Page Builder Developer's Guide*.

The combination of custom XML and a community tag handler enables you to extend the query processing logic in the Content Assembler — for example, by executing additional queries against the MDEX Engine, or interfacing with a third-party system to return data — before returning the results to the application. Use cases for community tag handlers include the following:

- Given some XML that specifies a rollup key for a navigation query or aggregated record query, pass this key with the query to the MDEX Engine to return records for Content Spotlighting.
- Implement A/B testing for Content Spotlighting by executing different queries to the MDEX Engine for identical requests. The results of the queries are then transparently passed on to the applicaton.

- Query a third-party source for information to display on a product detail page. Examples include RSS feeds, content stored in another repository (such as a CMS), inventory information, or a recommendation engine.

It is not necessary to implement a tag handler to use custom XML. If no tag handler is registered to handle a particular element, the Content Assembler passes the XML through to the application as a `string`, which can then be parsed into XML and handled by your rendering logic. A tag handler provides a mechanism to encapsulate any processing you need to do for a particular element and abstract this processing from the rendering code.

Scenarios for extending Page Builder and the Content Assembler

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

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

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

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

Life cycle of a Content Assembler query

This section describes the sequence of events that occur when the Content Assembler processes a query.

The following sequence of events occurs when the query is executed:

1. The Content Assembler sends the query to the MDEX Engine and retrieves the `ContentResource` from the query results.

   This is the content XML that represents the landing page configuration created in Page Builder. The content XML is retrieved from the properties of the first rule returned in the zone that you specified when you configured the `ContentNavigationDataSource`.

2. The Content Assembler calls `ContentAssembler.Assemble()`.

   This method creates an `XmlReader` that reads the `ContentResource`, then calls `ContentAssembler.Evaluate()`, passing in a `ContentContext` that contains the relevant resources for the current query and the `XmlReader` that provides access to the content XML.

3. `ContentAssembler.Evaluate()` calls the `Evaluate()` method of the appropriate tag handler (in the case of the root element, this is `Endeca.Data.Content.Assembler.TagHandlers.ContentItemTagHandler`).

   This method takes two arguments: the current `ContentContext`, and an `XmlReader` positioned at the element to be evaluated.

4. The tag handler marshals the XML element into an object.

As part of the `Evaluate()` method, the tag handler may execute additional queries against an MDEX Engine or a third-party system. The Content Assembler also provides a mechanism for a tag handler to invoke additional tag handlers.

For example, `ContentItemTagHandler` invokes `PropertyTagHandler`, which in turn invokes tag handlers for specific property types to populate the property values.

When the Content Assembler has finished processing the content XML, it has transformed the XML tree into a tree of `IContentItem` objects with properties and nested `IContentItem` objects. This object tree is then returned to the application for rendering.

Class overview

The `Endeca.Data.Content.Assembler` namespace contains the classes and interfaces that make up the core Content Assembler implementation and enable extension of Content Assembler functionality through tag handlers.

Class	Description
`ContentAssembler`	Used to transform an `IContentResource` into an object model representation of its content item.
`ContentContext`	Provides access to resources that are shared across tag handlers.
`IContentResource`	The byte representation of the content XML returned in the MDEX query results. The `ContentAssembler` creates an `XmlReader` to read an `IContentResource` in order to enable tag handlers to process the content XML.
`IContentResourceLocator`	Used internally to fetch an `IContentResource` object.
`ITagHandler`	Transforms a specific element in the content XML into an object. The Content Assembler ships with several standard tag handlers. You can implement your own tag handlers to process custom XML elements.

Related Links

Resources managed by the ContentContext object on page 39
The `ContentContext` object provides access to resources that are shared across tag handlers.

Implementing the tag handler interface

A tag handler takes an element in the content XML and transforms it into an object. In the typical use case, you write a tag handler to return the value of a particular property.

Your tag handler can do as much or as little processing as desired during the course of marshaling XML into objects, including executing one or more queries to an MDEX Engine or another third-party system.

Important:

All tag handlers are instantiated once, then reused for each element that the Content Assembler processes. Because multiple invocations of a tag handler may be executed concurrently, tag handlers must be reentrant.

For performance reasons, tag handlers should not contain large blocks of synchronized code.

To implement the tag handler interface:

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

```
using System;
using System.Xml;
using Endeca.Data.Content;
using Endeca.Data.Content.Assembler;
```

2. Specify the element name and namespace that the tag handler is intended to process by defining the appropriate properties. For example:

```
  public string TagName { get { return "Integer"; } }
  public string TagNamespace { get { return "http://endeca.com/sample-
schema/2010"; } }
```

Note: To avoid conflicts between tag handlers, ensure that each tag handler processes an element that has a unique qualified name.

3. Implement the `Evaluate()` method.

   This method takes an `XmlReader` positioned at the XML element to process and a `ContentCon¬text`, and returns an `Object` to the tag handler that invoked it (typically, a `PropertyTagHandler`).

Resources managed by the ContentContext object

The `ContentContext` object provides access to resources that are shared across tag handlers.

A new `ContentContext` is instantiated for each search or navigation query executed by a `Content¬NavigationDataSource`. The `ContentContext` class includes the following properties:

Property	Description
`ContentAssembler`	A reference to the active `ContentAssembler`. You can use the `ContentAssembler` to invoke additional tag handlers by calling its `Evaluate()` method.

Property	Description
`LocalPropertyName`	Returns the name of the `IProperty` on which processing has most recently begun. This property is set by the `PropertyTagHandler` when it begins to process a `<Property>` element.
`ContentItems`	A reference to the stack of `IContentItem` objects currently being assembled. When the `ContentItemTagHandler` begins to process a `<ContentItem>` element, it adds the `IContentItem` to this stack. The tag handler pops the `IContentItem` from the stack when it is done.  **Note:** Other tag handlers should not modify the content item stack.
`ContentResourceLocator`	A reference to the active `ContentResourceLocator`. Typically, you do not need to use the `ContentResource¬ Locator` unless you want to retrieve a `ContentResource` from a zone other than the one that you specified on the `ContentNavigationDataSource`. You can pass a `Con¬ tentResource` to the `ContentAssembler.Assemble()` method to transform the content XML into an `IContentItem` object (which may contain other `IContentItem` objects).
`ContentResources` or `ContentResourceStack`	Provides access to the `ContentResource` objects being assembled. Typically, there is only one `ContentResource` for any given query, and you do not need to access it directly. Rather, tag handlers work on the XML that is passed through the `Eval¬ uate()` method.

Related Links

Class overview on page 38

The `Endeca.Data.Content.Assembler` namespace contains the classes and interfaces that make up the core Content Assembler implementation and enable extension of Content Assembler functionality through tag handlers.

About invoking other tag handlers

You can write tag handlers that invoke other tag handlers (either standard tag handlers or other community tag handlers).

You invoke another tag handler by calling `ContentAssembler.Evaluate()` and passing in an `XmlReader` positioned at the element to be processed, along with a reference to the current `Content¬Context`.

```
  return pContext.ContentAssembler.Evaluate( pContext, pReader );
```

The `ContentAssembler.Evaluate()` method identifies the appropriate tag handler for the element, if one exists, and calls its `Evaluate()` method.

It is not necessary to invoke other tag handlers from within your own tag handler, even if you have nested elements within your custom XML. There are two cases in which this may be especially useful.

Multiple combinations of valid child elements

You may have optional elements or different possible combinations of elements within your custom XML. In such a case, rather than adding logic to check for each element that your tag handler may have to process, you can write separate tag handlers for each possible child element. The parent tag handler can simply iterate through the child elements and call `ContentAssembler.Evaluate()` on each child.

For example, the standard `RecordList` element can contain either a `RecordQuery` (for featured records) or a `NavQuery` (for dynamic records). The `RecordListTagHandler` invokes either the `RecordQueryTagHandler` or the `NavQueryTagHandler` to perform a query against the MDEX Engine that returns the records for a Content Spotlighting cartridge.

Same element nested under more than one parent

Your use of custom XML within your application may produce a structure similar the following:

```
<Property>
  <TagA>
    <TagC/>
  </TagA>
</Property>
<Property>
  <TagB>
    <TagC/>
  </TagB>
</Property>
```

In this case, you can write separate tag handlers for `<TagA>` and `<TagB>` that each invoke a third handler for `<TagC>`. This ensures consistent handling of `<TagC>` regardless of its parent element.

Integrating a tag handler into the Content Assembler

The Content Assembler API provides a simple interface for registering tag handlers.

This procedure assumes that you have already written a tag handler class that implements `Endeca.Da¬ta.Content.Assembler.ITagHandler`.

To integrate a tag handler with the Content Assembler for the RAD Toolkit for ASP.NET:

1. Register your tag handler by updating your application configuration. You can specify the configuration in the `app.config` or `Web.config` file for your application.
2. Package the tag handler as an assembly (DLL).
3. Install the tag handler by copying the assembly into the `\bin` directory of your application.

4. Restart IIS.

In order for the Content Assembler to make use of the new tag handler, the content XML must contain the element that the tag handler is intended to process. You can achieve this in one of the following ways:

- Specify pass-through XML in a page template or cartridge template.
- Specify a custom property type in a template and bind it to an editor that generates custom XML.

Registering a tag handler

You register a tag handler by specifying it in your application's `app.config` or `Web.config` file.

To register a tag handler:

1. Open the `app.config` or `Web.config` file for your application.
2. Define the tag handler configuration section as in the following example:

```
<configSections>
  <sectionGroup name="endeca.content"
    type="Endeca.Data.Content.Configuration.EndecaContentSectionGroup,
    Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
    PublicKeyToken=6d02be8724ca751c" >
    <section name="tagHandlers"
      type="Endeca.Data.Content.Configuration.TagHandlersSection,
      Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
  </sectionGroup>
</configSections>
```

3. Specify the assembly-qualified type name for each tag handler as in the following example:

```
<endeca.content>
  <tagHandlers>
    <add handlerType="Endeca.Data.Content.Sample.IntegerTagHandler,
      Endeca.Data.Content.Sample" />
  </tagHandlers>
</endeca.content>
```

As the Content Assembler processes the content XML that represents a landing page, it invokes the appropriate tag handler for each element. In the sample content XML excerpt below, the Content Assembler would call the `Evaluate()` method of the `Endeca.Data.Content.Sample.Inte¬gerTagHandler` handler to process the `<Integer>` element. In this case, it returns an `IProperty` object with an `int` value of 17.

```
<ContentItem type="PageTemplate">
  <TemplateId>IntegerTagSample</TemplateId>
  <Name>Integer demo</Name>
  <Property name="humbug">
    <Integer xmlns="http://endeca.com/sample-schema/2010">17</Integer>
  </Property>
</ContentItem>
```

Standard tag handlers in the Content Assembler

The Content Assembler API package includes several tag handlers associated with the standard property types.

The following handlers are provided for the standard Page Builder content types:

XML element	Tag handler implementation
`http://endeca.com/schema/con¬tent/2008:Boolean`	`Endeca.Data.Content.Assembler.TagHan¬dlers.BooleanTagHandler`
`http://endeca.com/schema/con¬tent/2008:ContentItem`	`Endeca.Data.Content.Assembler.TagHan¬dlers.ContentItemTagHandler`
`http://endeca.com/schema/con¬tent/2008:ContentItemList`	`Endeca.Data.Content.Assembler.TagHan¬dlers.ContentItemListTagHandler`
`http://endeca.com/schema/content-tags/2008:DimensionList`	`Endeca.Data.Content.Navigation.TagHan¬dlers.DimensionListTagHandler`
`http://endeca.com/schema/content-tags/2008:NavigationRecords`	`Endeca.Data.Content.Navigation.TagHan¬dlers.NavigationRecordsTagHandler`
`http://endeca.com/schema/content-tags/2008:NavigationRefinements`	`Endeca.Data.Content.Navigation.TagHan¬dlers.NavigationRefinementsTagHandler`
`http://endeca.com/schema/content-tags/2008:NavigationResult`	`Endeca.Data.Content.Navigation.TagHan¬dlers.NavigationResultTagHandler`
`http://endeca.com/schema/content-tags/2008:NavQuery`	`Endeca.Data.Content.Navigation.TagHan¬dlers.NavQueryTagHandler`
`http://endeca.com/schema/con¬tent/2008:Property`	`Endeca.Data.Content.Assembler.TagHan¬dlers.PropertyTagHandler`
`http://endeca.com/schema/con¬tent/2008:RecordList`	`Endeca.Data.Content.Navigation.TagHan¬dlers.RecordListTagHandler`
`http://endeca.com/schema/content-tags/2008:RecordQuery`	`Endeca.Data.Content.Navigation.TagHan¬dlers.RecordQueryTagHandler`
`http://endeca.com/schema/content-tags/2008:Sort`	`Endeca.Data.Content.Navigation.TagHan¬dlers.SortTagHandler`
`http://endeca.com/schema/con¬tent/2008:String`	`Endeca.Data.Content.Assembler.TagHan¬dlers.StringTagHandler`
`http://endeca.com/schema/content-tags/2008:Supplement`	`Endeca.Data.Content.Navigation.TagHan¬dlers.SupplementTagHandler`

About the sample tag handler

The Content Assembler API package includes a sample tag handler implementation.

The sample is located in `ContentAssemblerAPIs\RAD Toolkit for ASP.NET\`*`version`*`\reference\tag_handlers` and includes the following:

File	Description
`Endeca.Data.Content.Sample.dll`	Contains a sample tag handler that transforms the contents of an `<Integer>` element into an `int`.
`IntegerTagHandler.cs`	The source code for the `IntegerTagHandler`.

File	Description
`PageTemplate-IntegerTagSample.xml`	A sample page template that contains an `<Integer>` property.

The sample package is intended to demonstrate the integration points between tag handlers and the Content Assembler. It does not include any rendering code for the reference application to make use of the integer property returned by the Content Assembler.

Installing the sample tag handler

The sample tag handler is provided as an assembly along with a simple page template that defines an `<Integer>` property with a default value.

To install the sample tag handler:

1. Register the tag handler by editing the `app.config` or `Web.config` file for your application.
   a) Define the config section as follows:

```
<configSections>
  <sectionGroup name="endeca.content"
    type="Endeca.Data.Content.Configuration.EndecaContentSectionGroup,

    Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
    PublicKeyToken=6d02be8724ca751c" >
    <section name="tagHandlers"
      type="Endeca.Data.Content.Configuration.TagHandlersSection,
      Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
  </sectionGroup>
</configSections>
```

   b) Specify the tag handler as follows:

```
<endeca.content>
  <tagHandlers>
    <add handlerType="Endeca.Data.Content.Sample.IntegerTagHandler,
      Endeca.Data.Content.Sample" />
  </tagHandlers>
</endeca.content>
```

2. Copy the assembly into the `\bin` directory of your application.
3. Restart IIS.
4. Copy the sample template to your local templates directory and upload it using the emgr_update utility. For example:

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

   The template does not define any editors associated with the integer property. The `<Integer>` element is treated as pass-through XML.

About extending the Content Assembler to validate custom XML

You can configure the Content Assembler to validate content XML, including custom XML.

Recall that you can enable XML validation in Content Assembler by setting the `ContentValidation` property of the `ContentNavigationDataSource` to true.

If validation is enabled, the Content Assembler performs schema validation as it processes the content XML. By default, the Content Assembler validates any elements within the Endeca content XML namespaces (`http://endeca.com/schema/content/2008` and `http://ende¬ca.com/schema/content-tags/2008`) that are defined in the associated schemas.

You can specify additional schemas that the Content Assembler uses to validate content XML by editing the application configuration. You can specify additional schemas in either the `Web.config` or the `app.config` file, as in the following example:

```
<!--specify the config section as follows; the tagHandlers section should
     already have been added to specify the tag handlers themselves-->
<configSections>
  <sectionGroup name="endeca.content"
    type="Endeca.Data.Content.Configuration.EndecaContentSectionGroup,
    Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
    PublicKeyToken=6d02be8724ca751c" >
    <section name="tagHandlers"
      type="Endeca.Data.Content.Configuration.TagHandlersSection,
      Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
    <section name="schemas"
      type="Endeca.Data.Content.Configuration.SchemasSection,
      Endeca.Data.Content, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
  </sectionGroup>
</configSections>

<!-- additional elements not included in this example -->

<endeca.content>
  <tagHandlers>
    <add handlerType="Endeca.Data.Content.Sample.IntegerTagHandler,
      Endeca.Data.Content.Sample" />
  </tagHandlers>
<!-- specify additional schemas as in the following example -->
  <schemas>
    <add namespace="http://endeca.com/sample-schema/2010"
      uri="Endeca.Data.Content.Sample.Resources.integer_tag_handler.xsd"
      assembly="Endeca.Data.Content.Sample"/>
  </schemas>
</endeca.content>
```

At runtime, the Content Assembler matches the namespace of each element in the content XML against the namespaces defined in the configuration file. If the associated schema file defines the element being processed, the Content Assembler validates that element against the schema.

Note: Validation can be useful in a testing environment for debugging purposes, particularly if you are working with a community editor that generates custom XML. 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.

Index

C

cartridges
  and user controls 22
  building 21
  rendering code 22
  rendering custom navigation refinements 24
  rendering custom results lists 24
class overview
  com.endeca.content.assembler 38
  ContentContext object 39
community editors
  scenarios 36
community tag handlers
  scenarios 36
Content Assembler API
  and RAD API 32
  components 10
Content Assembler API for the RAD Toolkit for ASP.NET
  and the RAD Toolkit for ASP.NET 17
Content Assembler reference application controls, using 22
content items and Content Assembler API 21
content properties, accessing 21
content query
  and content XML validation 21
  executing 32
  results 32
ContentNavigationDataSource and content XML validation 21
ContentNavigationDataSourceControl 17
custom navigation refinements, rendering 24
custom results lists
  additional considerations 25
  rendering 24
custom trigger conditions
  filtering based on rule properties 19
  overview 18
  using hidden dimensions 19
  with rule zones 20
  with user profiles 20

D

dynamic content 11
DynamicContentPlaceHolder 22
  introduced 28
  using 28, 29

E

Endeca Content Assembler API
  dynamic content 11
  overview 9
Endeca Content Assembler reference application
  cartridges 14
  CSS 15
  host, changing 15
  overview 11
  port, changing 15
  skinning 15
  templates 14

I

IContentControl 22

R

RAD Toolkit for ASP.NET server controls 31

S

see-all links 27
server controls 31

T

tag handlers
  about 35
  implementing 39
  in life cycle of Content Assembler query 37
  integrating with Content Assembler 41
  invoking from other tag handlers 41
  list of standard tag handlers 43
  registering 42
  sample 43, 44

X

XML validation
  extending 45