Endeca RAD Toolkit for ASP.NET

Developer's Guide

Version 2.1.3 • March 2012

Contents

Copyright and disclaimer

Preface

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

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

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

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

About this guide

This guide describes the major tasks involved in developing an Endeca application using the Rapid Application Development (RAD) Toolkit for ASP.NET.

This guide assumes that you have read the *Oracle Endeca Guided Search Concepts Guide* and that you are familiar with Endeca's terminology and basic concepts.

This guide covers the main features of the RAD Toolkit for ASP.NET. This guide is not a replacement for the *Endeca Basic Development Guide* or the *Endeca Advanced Development Guide*.

Who should use this guide

This guide is intended for ASP.NET developers who are building Endeca applications using the Rapid Application Development (RAD) Toolkit for ASP.NET.

This guide assumes that you are familiar with C#, the Microsoft .NET Framework, Microsoft Visual Studio, and Web development concepts.

If you are a new user of Oracle Endeca Guided Search and you are not familiar with developing Endeca applications, read this guide in conjunction with the *Endeca Basic Development Guide* and the *Endeca Advanced Development Guide*, available for download from the Oracle Technology Network (OTN). Those guides provide detailed feature descriptions that are not fully covered in this version of the *Endeca Developer's Guide for the RAD Toolkit for ASP.NET.*

If you are an existing user of Oracle Endeca Guided Search and you are familiar with developing Endeca applications, this guide should provide enough information to help you build a new application using the RAD Toolkit for ASP.NET.

Also, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*. In a default installation, this is located in the `C:\Endeca\RADToolkits\`*`version`*`\ASP.NET\doc` directory. This file contains reference information for the RAD API, the Endeca data source, and Endeca user interface controls.

Conventions used in this guide

This guide uses the following typographical conventions:

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

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

Contacting Oracle Endeca Customer Support

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

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

Chapter 1

Installing and uninstalling the RAD Toolkit for ASP.NET

This section provides system prerequisites, installation procedures, and deployment procedures for the Endeca RAD Toolkit for ASP.NET.

Prerequisites for installing the RAD Toolkit for ASP.NET

This topic describes the software prerequisites that must be in place before you install the RAD Toolkit for ASP.NET.

Software compatibility

To determine the compatibility of the RAD Toolkit for ASP.NET with other Endeca installation packages, see the *Oracle Endeca Guided Search Compatibility Matrix* available on the Oracle Technology Network.

Operating system requirements

The RAD Toolkit for ASP.NET is supported on Windows Server 2003 and Windows Server 2008.

Software requirements

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

- Microsoft .NET Framework. The RAD Toolkit for ASP.NET supports the following versions of the Microsoft .NET Framework (both 32-bit and 64-bit): 2.0 with SP1, 3.0, and 3.5.
- Microsoft Visual Studio 2005 with Service Pack 1 (any edition) or Visual Studio 2008.
- On machines running the RAD Toolkit for ASP.NET reference application, you need Internet Information Services (IIS) 5.1 or later.

Note: The Endeca Presentation and Logging APIs for .NET are part of the RAD Toolkit installation and are installed into `<installation path>\Endeca\RADToolkits\<version>\ASP.NET\bin`.

Unsupported software

Microsoft .NET Framework version 1.1 is not supported.

Optional software

If you want to use the Analytics capabilities of the Endeca data source controls, refer to *Enabling Endeca Analytics*.

Installing the RAD Toolkit for ASP.NET

This topic describes how to install the Endeca RAD Toolkit for ASP.NET.

Before starting this procedure, see "Prequisites for installing the RAD Toolkit for ASP.NET".

To install the RAD Toolkit for ASP .NET:

1. In your local environment, locate the installation file for the RAD Toolkit for ASP.NET software that you downloaded from the Oracle Software Delivery Cloud.
2. Double click the appropriate 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. Oracle recommends that you accept the default installation path of `C:\Endeca\RADToolkits\<version>\ASP.NET`.
4. On the **Ready to Install** screen, click **Install**.
5. Click **Finish**.

After performing this task, go on to "Adding the RAD Toolkit for ASP.NET to Visual Studio".

Adding the RAD Toolkit for ASP.NET 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 DLL files.

Note: This procedure describes how to add the RAD Toolkit to Visual Studio 2005. The procedure may be slightly different in Visual Studio 2008.

To add the RAD Toolkit for ASP.NET to Visual Studio:

1. Start Visual Studio and open your Web site.
2. In the **Toolbox** window, right-click the **Standard** tab and select **Add Tab**.
3. Type `Endeca RAD Toolkit`.
4. Right-click in the grey box under the entry for the RAD Toolkit.
   Note: The grey box contains the text "There are no usable controls in this group..."
5. Select **Choose Items...**
6. On the **.NET Framework Components** tab of the **Choose Toolbox Items** dialog, click **Browse...**.
7. Browse to `<installation path>\Endeca\RADToolkits\version\ASP.NET\bin\Endeca.Web.dll` and click **Open**.
8. Browse to `<installation path>\Endeca\RADToolkits\version\ASP.NET\bin\Endeca.Web.UI.WebControls.dll` and click **Open**.
9. Click **OK**.

The Endeca data source controls and the Endeca user interface controls display in the **Toolbox** under Endeca RAD Toolkit.

10. In the **Solution Explorer** window, right-click your Web site.
11. Select **Add Reference...**.
12. In the **Add Reference** dialog, select the **Browse** tab.
13. Browse to the Endeca DLL files for the RAD Toolkit installation. In a default installation, these are in `<installation path>\Endeca\RADToolkits\version\ASP.NET\bin`.
14. Select `Endeca.Navigation.dll`, `Endeca.Navigation.AccessControl.dll`, `Endeca.Data.dll`, and `Endeca.Logging.dll`. You do not need to select `Endeca.Web.dll` or `Endeca.Web.UI.WebControls.dll`.
    Visual Studio copies the directory's contents into the Web site project.

Uninstalling the RAD Toolkit for ASP.NET

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

To uninstall the RAD Toolkit for ASP.NET:

1. From the Windows Control Panel, select **Add or Remove Programs**.
2. Select Endeca RAD Toolkit .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 RAD Toolkit** tab and select **Delete Tab**.

Deploying the RAD Toolkit for ASP.NET reference application

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 the header of the 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.

Note: Make sure that you have enabled the ASP.NET 2.0 Web Service Extension in IIS before deploying the reference application.

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

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...**.

Note: If you are using IIS 7, you should create an **Application** rather than a **Virtual Directory**.

4. Complete the **Virtual Directory Creation Wizard** as follows:
   a) Click **Next**.
   b) Type an alias name such as RAD.
   c) Click **Next**.
   d) In the **Web Site Content Directory** screen, click **Browse** and locate the reference application that is packaged with the RAD Toolkit for ASP.NET. This is in `<installation path>\Endeca\RADToolkits\version\ASP.NET\reference\RadAspNetRef`.
   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 RAD that you created in the step above.
6. Right click RAD 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.
9. Select the **ASP.NET** tab and from the **ASP.NET version** list, select 2.0.x or later.
10. Click **OK**.
11. Restart IIS. (Right click the machine name icon in IIS. Select **All Tasks** from the submenu and select **Restart IIS**.)
12. On your Windows machine, start Internet Explorer and navigate to `http://hostname/RAD/GuidedNavigation.aspx`.
13. 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**.

Chapter 2

Overview of the RAD Toolkit for ASP.NET

This section provides an overview of the RAD Toolkit for ASP.NET including its data source controls, user interface controls, the reference application, and life cycle events.

Components in the RAD Toolkit for ASP.NET

The Rapid Application Development (RAD) Toolkit for ASP.NET provides a set of Web server controls to build Endeca applications and also provides the RAD API to the Endeca MDEX Engine.

The RAD Toolkit for ASP.NET is made up of the following components:

- RAD API for ASP.NET
- A set of data source controls
- A set of user interface controls
- A reference application
- Documentation

The RAD API for ASP.NET to the Endeca MDEX Engine

The RAD API for ASP.NET provides a simplified interface to the Endeca MDEX Engine and makes programming more friendly to the typical .NET developer.

Endeca assemblies

The RAD API is made up of the following assemblies:

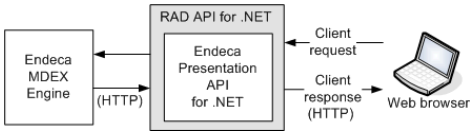
- `Endeca.Data.dll` -- This is the RAD API itself. It is an abstraction of the Endeca Presentation API that includes both input and output types and presents a standard interface for .NET developers.
- `Endeca.Web.dll` -- This contains the Endeca data source controls.
- `Endeca.Web.UI.WebControls.dll` -- This contains the Endeca user interface controls and Endeca Workbench instrumentation controls.

These assemblies must reside in your Web site project with `Endeca.Navigation.dll`, `Endeca.Navigation.AccessControl.dll`, and `Endeca.Logging.dll`.

Interaction of the RAD API for .NET and the Presentation API for .NET

Endeca is not yet deprecating the Presentation API for .NET or the *Endeca Basic Development Guide* and *Endeca Advanced Development Guide*. Endeca is adding the RAD API for ASP.NET as a layer on top of the Presentation API for .NET to make programming more friendly to the typical .NET developer. This layer of abstraction provides a simplified interface. The RAD API for ASP.NET is intended for use by all .NET customers.

The following diagram illustrates the interaction of the RAD API for .NET layer on top of the Presentation API for .NET:



Class diagrams

The documentation includes two class diagrams that illustrate the input and output types of the RAD API for .NET. These diagrams are `<installation path>\Endeca\RADToolkits\version\ASP.NET\doc\InputTypes.png` and `OutputTypes.png`.

Endeca data source controls

Endeca provides data source controls that are available from the **Toolbox** window of Visual Studio. Each data source control provides design-time views of source data in the MDEX Engine. This data can be bound to by other ASP.NET controls in your application.

Note: Endeca *data source controls* are built as *ASP.NET data source server controls*. For simplicity, this guide uses the more brief form of *data source control*.

The data source controls provide representative MDEX Engine data by performing round trip queries to the MDEX Engine at design time.

You drag and drop any of following data source controls into your project and configure the data source controls for your application:

Data source control	Description
AggregateRecordDetailsDataSource	Provides details about a single aggregate record to other controls in your Web site.
CompoundDimensionSearchDataSource	Provides records produced by compound dimension search to other controls in your Web site.
DimensionSearchDataSource	Provides results produced by dimension search to other controls in your Web site.
MetadataDataSource	Provides record metadata such as search keys, sort keys, aggregation keys and so on.
NavigationDataSource	Provides navigation query information, including dimension refinements, record results, aggregated record results,

Data source control	Description
	Analytics results, breadcrumbs information, metadata, and supplemental objects.
RecordDetailsDataSource	Provides details (e.g. property values and dimension values) about a single record to other controls in your Web site.

Endeca user interface controls

Endeca provides a number of custom user interface controls that are available from the **Toolbox** window of Visual Studio. Each control provides a convenient user interface for common Endeca page elements in an application.

Note: Endeca *user interface controls* are built as *ASP.NET server controls*. For simplicity, this guide refers to them using the more brief form of *user interface controls*.

You drag and drop the following user interface controls into your project and configure them as appropriate for your application.

Web server control	Description
Breadcrumbs	Displays the search navigation descriptors for the current query. Descriptors include dimension values, search terms, and range filters. For example, a breadcrumb made up of dimension values might look like this: `Wine Type > Red > Merlot > 1999 [X]`
GuidedNavigation	Renders dimension groups, dimensions, and dimension value names on a Web page.
Pager	Displays paging choices to page through the record set, when the number of search results are sufficient to create multiple pages. For example a paging control typically looks like this: `<< < 2, 3, 4, > >>`
TagCloud	Renders the most commonly used tags in records stored in an MDEX Engine. Tags for frequently occurring query results display in a larger font. Selecting a tag within a tag cloud produces a collection of query results that are associated with that tag.

Endeca Workbench instrumentation controls

Endeca provides two instrumentation controls to help integrate a custom application into Endeca Workbench.

Note: Endeca *Workbench instrumentation controls* are built as *ASP.NET server controls*. For simplicity, this guide uses the more brief form of *instrumentation controls*.

You drag and drop the following instrumentation controls into any page that requires instrumentation and configure the controls as appropriate for your application. See the *Oracle Endeca Workbench Administrator's Guide* for more information about instrumenting an application.

Web server control	Description
WebStudioNavigationInstrumentation	Instruments a Web page to provide navigation state to Endeca Workbench.
WebStudioRecordInstrumentation	Instruments a Web page to provide record details state to Endeca Workbench.

Reference application

The RAD Toolkit for ASP.NET provides a reference application that can be run in Postback mode, URL mode, RAD Toolkit Server Controls Postback mode, or RAD Toolkit Server Controls URL mode. Like other Endeca reference applications, the RAD Toolkit reference application provides a simple front-end interface that allows you to connect to an MDEX Engine and examine a record set.

Installation locations

By default, the installer places the reference application in `C:\Endeca\RADToolkits\`*`version`*`\ASP.NET\reference\RadAspNetRef`. This directory includes the files for all modes of the reference application. The start page for the Postback mode is `GuidedNavigation.aspx`. The start page for the URL parameter mode is `GuidedNavigationUrl.aspx`.

Differences between the reference application modes

The Postback mode of the reference application uses HTTP POST requests, not HTTP GET requests for each query in an application.

The URL parameter mode uses URL query parameters to generate an HTTP GET request for each change in the navigation state. This means that any change to the navigation state (i.e. any search or navigation request) is reflected in the URL itself and all information about the navigation state displays in the URL.

Both modes of the reference application have a RAD Toolkit Server Controls mode. The RAD Toolkit Server Controls modes use the Endeca user interface and data source controls that are included with the RAD Toolkit for ASP.NET.

Deployment

You can deploy the reference application by following the instructions in "Deploying the RAD ASP.NET reference application".

Page life cycle events for RAD Toolkit controls

First, by way of disclaimer, this topic does not go into detail about the core concepts of ASP.NET such as general page life cycle stages or life cycle events for data bound controls. For background information about page life cycle stages and page events, you can refer to the MSDN article titled "ASP.NET Page Life Cycle Overview" at `http://msdn2.microsoft.com/en-us/library/ms178472.aspx`.

This topic assumes you have read the MSDN information and goes on to describe where the Endeca data source controls plug into the ASP.NET page life cycle events.

RaisePostbackEvent

This is the first event in the life cycle that typically affects your page and the state of the queries on it. All postback events such as button clicks and drop downs selected are handled at this stage. It is common practice to programmatically update parameters on your data source in this event.

PreRender event

The Endeca data source controls perform a number of operations in the `PreRender` event of the page event life cycle. The data source executes a query during this event only if another control on the page requests data from the data source. When a control on the page is bound to a data source, it defines a `DataSourceView` from which it gets its data. If the `DataSourceView` is not yet populated with fresh data (based on the current filter set of the query), it calls `ExecuteSelect` to query the MDEX Engine for fresh data.

Any control that is bound to an Endeca data source, for example, a **Repeater** control, a **GuidedNavigation** control, and so on, also fetches its data during the `PreRender` page event.

Record class generation utility

The RAD Toolkit includes a utility for code-generating a strongly-typed record class.

The `ASP.NET\bin\recgen.exe` utility queries an MDEX Engine for records and creates a class featuring the record's properties as strongly typed members. If you provide the class's type name as an argument to either the `NavigationCommand.TypeNames.RecordTypeName` or the `NavigationDataSource.RecordTypeName` property, the RAD Toolkit then serializes records into this class instead of its default `Endeca.Data.Record` class.

Type `recgen /?` at the command line for more information.

Chapter 3

Creating an application using Endeca server controls

This section provides an overview of how to create an Endeca front-end application using the Endeca data source controls and the Endeca user interface controls.

High level process of creating an application with server controls

The basic workflow is as follows:

1. Create (or open) your ASP.NET Web site in Visual Studio.
2. Add Endeca data source controls to the Web site so you can bind other controls to MDEX Engine query results.
3. Add Endeca user interface controls to the Web site so you can provide search and navigation features such as Guided Navigation controls, paging controls, breadcrumb controls, and so on.
4. Add other Visual Studio .NET controls to build search boxes, display record results, and so on.
5. Test and deploy your Web site.

Adding Endeca data source controls to a Web page

This section describes how to create and configure the Endeca data source controls available in the RAD Toolkit for ASP.NET. Other ASP.NET controls can bind to the Endeca data source controls and access Endeca query results.

On each data source, you can check the **Preview Endeca data** option to populate a control with representative data from the MDEX Engine. This option gives you a design time view of the data that the control provides to other controls on the page. If this option is unchecked, the user interface controls populate themselves with placeholder data.

Creating a NavigationDataSource control

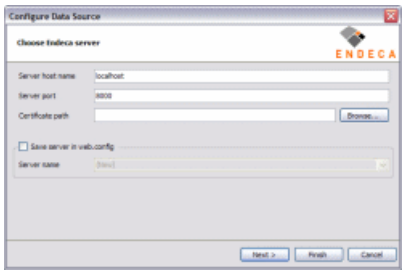
You create and configure a **NavigationDataSource** control in order to provide MDEX Engine records to other controls in your Web site.

In other words, the **NavigationDataSource** provides design time functionality to populate the other controls (e.g. user interface controls) with all the properties in an Endeca record.

Note: In addition to configuring the **NavigationDataSource** according to the procedure below, you can also configure the **NavigationDataSource** using the Properties window of Visual Studio.

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

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **NavigationDataSource** 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, do the following:
   a) Specify the host and port on which the MDEX Engine is running. These are the only two required values to finish the wizard. If you do not need any additional configuration, click **Finish**.
   For example, the most basic configuration looks like this:



   b) If you are connecting to the MDEX Engine via SSL, click **Browse...** to locate the Web application's private certificate file (typically a PEM file).
   c) Check **Save server in web.config** if you want the data source to refer to the `web.config` file for host and port information. If unchecked, the host and port information is set as a property of the data source.
   d) If you checked **Save server in web.config**, specify the name of a server on which the file is stored.
   This option adds `<configSections>`, `<expressionBuilders>`, and `<endeca>` to your `web.config` file as shown in this example:

```
<configuration>
   <configSections>
      <sectionGroup name="endeca"
      type="Endeca.Data.Configuration.EndecaSectionGroup,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" >
      <section name="servers"
      type="Endeca.Data.Configuration.ServersSection,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
      </sectionGroup>
   </configSections>

<!-- (elements removed to show only what Endeca adds) -->
```

```
      <expressionBuilders>
         <add expressionPrefix="EndecaConfig"
         type="Endeca.Web.UI.Design.EndecaConfigurationExpression¬
Builder,
         Endeca.Web, Version=2.1.0.0, Culture=neutral,
         PublicKeyToken=6d02be8724ca751c" />
      </expressionBuilders>

<!-- (elements removed to show only what Endeca adds) -->

   <endeca>
      <servers>
         <add name="MDEXServer" hostName="localhost" port="8000"
         certificatePath="" />
      </servers>
   </endeca>
</configuration>
```

Furthermore, this option sets several property values in your ASPX file using ASP.NET expressions, as shown in this example snippet:

```
<cc1:NavigationDataSource ID="NavigationDataSource1" runat="server"
   MdexCertificatePath='<%$ EndecaConfig:Servers.Servers["MDEXServ¬
er"].CertificatePath %>'
   MdexHostName='<%$ EndecaConfig:Servers.Servers["MDEXServer"].HostName
 %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["MDEXServer"].Port %>'>
</cc1:NavigationDataSource>
```

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. On the **Edit data source parameters** screen, do the following:
   a) To add all parameters available to the **NavigationDataSource** control, click **Add All Parameters**. These are all the parameters that can be passed to the MDEX Engine. An application will likely only use a small number of these parameters; it is not required to include them all if not desired.
   b) To add only specific parameters, you can add one parameter at a time. Click **Add Parameter** and specify a name.
   c) To indicate where the value of a parameter comes from, select an option from the **Parameter** source list.
   d) To specify the value of a property, select the property and provide a value in **DefaultValue**.
   e) If desired, you can use the **Up**, **Down**, and **Delete** buttons to change the display position of parameters and delete parameters.
9. 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 specify optional Analytics query information. Adding an Analytics query makes the results of that query available to the data source and other controls on the page.
10. On the **Configure analytics query** screen, do the following:
    a) Specify the syntax of one or more Analytics queries to run against the MDEX Engine. Here is a simple example query that averages a property and groups the results by the `P_Winery` property:

```
RETURN AvgPriceTable
AS SELECT AVG(@AvgPrice) AS AvgPrice
GROUP BY P_Winery
```

b) If you want the data source to refer to the `web.config` file to read the Analytics query, check **Would you like to save this query in web.config?** If unchecked, the Analytics query is set in the **AnalyticsExpression** property of the data source.
c) If checked, specify a name for the Analytics query.
This option adds `<queries>` to your `web.config` file as shown in this example:

```
<queries>
   <add name="QueryWithAveragePrice">
     <![CDATA[RETURN AvgPriceTable AS SELECT AVG(@AvgPrice) AS AvgPrice

      GROUP BY P_Winery]]>
   </add>
</queries>
```

d) Click **Next**.
e) In the **Analytics Parameter Values** dialog box, specify a value for each parameter in the Analytics query. Before submitting the Analytics query to the MDEX Engine, the data source performs token replacement on the parameters using the values you specify.
For example, for the query above, you specify a value for the `AvgPrice` parameter. In this example, the value is the `P_Price` property, which is determined at runtime.

11. On the **Configure analytics result schema** screen, do the following:
a) If necessary, modify the data type of each parameter and query result to correspond appropriately. For example, if a query produces an `AverageScore` result with values such as 83.7500, 86.0000 and so on, you would change the default data type from **String** to **Double**. That way, the data type of the parameter corresponds to its actual values.
This data type information is saved in the schema for the Analytics query.
b) If you want the data source to refer to the `web.config` file to reuse this schema, check **Would you like to save this schema in web.config?** If unchecked, the schema information is set in the **AnalyticsSchema** property of the data source.
This option adds `<schemas>` to your `web.config` file as shown in this example:

```
<schemas>
    <add name="SchemaQueryWithAverageScore">
     <xs:schema id="SchemaQueryWithAverageScore" xmlns=""
xmlns:xs="http://www.w3.org/2001/XMLSchema"
      xmlns:msdata="urn:schemas-microsoft-com:xml-msdata">
      <xs:element name="SchemaQueryWithAverageScore" msdata:Is¬
DataSet="true"
       msdata:Locale="en-US">
       <xs:complexType>
        <xs:choice minOccurs="0" maxOccurs="unbounded">
         <xs:element name="AvgScoreTable" msdata:Locale="en-US">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="P_Winery" type="xs:string" minOccurs="0"
 />
            <xs:element name="AvgScore" type="xs:double" minOccurs="0"
 />
           </xs:sequence>
          </xs:complexType>
         </xs:element>
        </xs:choice>
       </xs:complexType>
      </xs:element>
     </xs:schema>
    </add>
</schemas>
```

12. Click **Finish**.
13. In the **Properties** window of Visual Studio, modify the following properties for the data source control if necessary. Many of these properties are set when you run the **Configure Data Source...** wizard.

Property	Description
AnalyticsExpressionParameters	Optional. Invokes a wizard where you specify the parameters used in an Analytics expression.
MDEXCertificatePath	Optional. Specifies a path to the Web application's private certificate file (typically a PEM file) if you are connecting to the MDEX Engine via SSL.
MDEXHostName	Required. Specifies the host on which the MDEX Engine is running.
MDEXPort	Required. Specifies the port on which the MDEX Engine is running.
RecordTypeName	Optional. Specifies a custom type for use by records returned by this data source. The default value for `RecordTypeName` is `Endeca.Data.Record`.
SelectParameters	Optional. Specifies a collection of parameters to use when the data source queries the MDEX Engine.
PermanentRefinementConfigs	Optional. Specifies one or more dimensions, by ID, to always expose and keep exposed on the page. A dimension cannot be collapsed if a user clicks it. Click the **Browse...** option to invoke the **RefinementConfig Collection Editor**. This editor allows you to specify a `DimensionValueId` for each dimension you want to expose.
AnalyticsExpression	Optional. Specifies either the syntax of an Analytics expression or specifies a property that points to the expression in the `web.config` file. Typically, you create an Analytics expression in the **Configure analytics query** screen of the configuration wizard.
AnalyticsSchema	Optional. Specifies the attributes of the schema for the `AnalyticsExpression`.
BusinessRulesFilter	Optional. Specifies the filter that evaluates which dynamic business rules the MDEX Engine runs.
BusinessRulesPreviewTime	Optional. Specifies the time used to determine whether the MDEX Engine runs dynamic business rules that have a time trigger.
EnableComputeAlternativePhrasing	Optional. Specifies whether the MDEX Engine should compute alternative phrasings for the current record search query. The default value for `EnableComputeAl¬ternativePhrasing` is `False`.
EnableDidYouMean	Optional. Specifies whether a full-text search should turn on the Did You Mean? feature. Only used if a full-text search query is being made. The default value for `En¬ableDidYouMean` is `False`.

Property	Description
EnableExposeAllRefinements	Optional. Specifies whether to expose (open) all dimensions on the page. Setting this property to `True` exposes all dimensions.
EnableRewriteWithAlternativePhrasing	Optional. Specifies whether the MDEX Engine uses one of the alternative phrasings it has computed instead of the end user's original query when computing the set of documents to return. `True` instructs the MDEX Engine to use a computed alternative phrasing, while `False` (the default) instructs it to use the user's original query. The `EnableComputeAlternativePhrasing` must be set to `True` when using `EnableRewriteWithAl¬ternativePhrasing`.
RefinementConfigs	Optional. Specifies one or more dimensions, by ID, to initially expose (open) on the page. Unlike **PermanentRefinementConfigs**, the dimensions you specify for the **RefinementConfigs** property can be collapsed if a user clicks them. Click the **Browse...** option to invoke the **RefinementConfig Collection Editor**. This editor allows you to specify a `DimensionValueId` for each dimension you want to expose.

Note: This is not a comprehensive list of the properties available in the Properties window. There may also be additional properties available to the control from the code behind (the .aspx.cs file for the control) that do not display in the Properties window but are necessary to fully configure the data source. For details about the properties for each data source control, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*.

NavigationDataSouce example code

The markup generated on the **Source** page of Visual Studio is similar to the following:

```
<cc1:NavigationDataSource ID="NavigationDataSource1" runat="server" Mdex¬
Port="7900"
   MdexCertificatePath="" MdexHostName="smith-690">
   <PermanentRefinementConfigs>
      <end:RefinementConfig DimensionValueId="3">
      </end:RefinementConfig>
   </PermanentRefinementConfigs>
</cc1:NavigationDataSource>
```

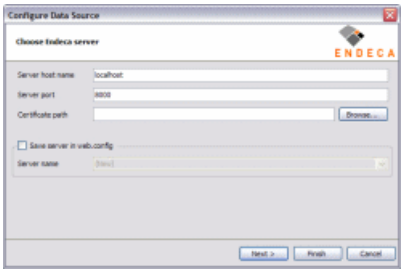
Creating a MetadataDataSource control

You create and configure a **MetadataDataSource** control in order to provide metadata such as the available search keys, sort keys, aggregation keys, and so on, to other controls in your Web site. For example, you might bind a drop-down list control to the available search keys from a **MetadataDataSource** control.

When you create the control, the control looks up the metadata once and stores it but does not look it up again unless you modify the caching properties.

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

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **MetadataDataSource** on to the **Design** tab of your Web page.
4. Select the **MetadataDataSource** and click the **Smart Tag**.
5. Check **Preview Endeca data** to populate other controls you add later with representative data from the MDEX Engine
6. Select **Configure Data Source...**.
7. On the **Choose Endeca server** screen, do the following:
   a) Specify the host and port on which the MDEX Engine is running. These are the only two required values to finish the wizard. If you do not need any additional configuration, click **Finish**.
   For example, the most basic configuration looks like this:



   b) If you are connecting to the MDEX Engine via SSL, click **Browse...** to locate the Web application's private certificate file (typically a PEM file).
   c) Check **Save server in web.config** if you want the **MetadataDataSource** to refer to the `web.config` file for host and port information. If unchecked, the host and port information is set as a property of the **MetadataDataSource**.
   d) If you checked **Save server in web.config**, specify the name of the server on which the file is stored.
   This option adds `<configSections>`, `<expressionBuilders>`, and `<endeca>` to your `web.config` file as shown in this example:

```
<configuration>
   <configSections>
      <sectionGroup name="endeca"
      type="Endeca.Data.Configuration.EndecaSectionGroup,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" >
      <section name="servers"
      type="Endeca.Data.Configuration.ServersSection,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
      </sectionGroup>
   </configSections>

<!-- (elements removed to show only what Endeca adds) -->

      <expressionBuilders>
         <add expressionPrefix="EndecaConfig"
         type="Endeca.Web.UI.Design.EndecaConfigurationExpression¬
Builder,
         Endeca.Web, Version=2.1.0.0, Culture=neutral,
```

```
         PublicKeyToken=6d02be8724ca751c" />
      </expressionBuilders>

<!-- (elements removed to show only what Endeca adds) -->

   <endeca>
      <servers>
         <add name="MDEXServer" hostName="localhost" port="8000"
         certificatePath="" />
      </servers>
   </endeca>
</configuration>
```

Furthermore, this option sets several property values in your ASPX file using ASP.NET expressions, as shown in this example snippet:

```
<cc1:MetadataDataSource ID="MetadataDataSource1" runat="server" Mdex¬
Port="7900"
  MdexCertificatePath='<%$ EndecaConfig:Servers.Servers["MetadataDSCon¬
nection"].
    CertificatePath %>'
   MdexHostName='<%$ EndecaConfig:Servers.Servers["MetadataDSConnec¬
tion"].
    HostName %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["MetadataDSConnec¬
tion"].Port %>' />
</cc1:MetadataDataSource>
```

8. Click **Finish**.
9. In the **Properties** window of Visual Studio, modify the following properties for the **MetadataDataSource** control if necessary:

Property	Description
CacheDuration	Optional. Specifies the length of time, in seconds, that data retrieved by this data source is cached. The default value for `CachingDuration` is `0`.
CacheExpirationPolicy	Optional. Specifies the cache expiration behavior that, when combined with the duration, describes the behavior of the cache that the data source control uses. The default value for `CacheExpirationPolicy` is `Sliding`.
CacheKeyDependency	Optional. Specifies a user-defined key dependency that is linked to all data items cached by this control. The default value is empty.
EnableCaching	Optional. Specifies whether the data retrieved by this data source should be fetched once and cached in the ASP.NET cache object rather than fetched each time from the MDEX Engine. Settting this property to `True` fetches the data once from the MDEX Engine and then stores that data in the ASP.NET cache. Setting this value to `False` fetches the data from the MDEX Engine each time. The default value for `EnableCaching` is `False`.

Note: This is not a comprehensive list of the properties available in the Properties window. There may also be additional properties available to the control from the code behind (the .aspx.cs file for the control) that do not display in the Properties window but are necessary to fully configure the data source. For details about the properties for each data source control, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*.

MetadataDataSource example code

The markup generated on the **Source** page of Visual Studio is similar to the following:

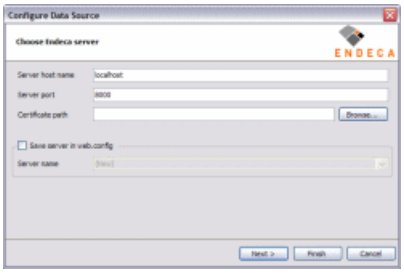
```
<cc1:MetadataDataSource ID="MetadataDataSource1" runat="server"
  MdexCertificatePath="" />
```

Creating a DimensionSearchDataSource control

You create and configure a **DimensionSearchDataSource** control in order to provide results produced by dimension search to other controls in your Web site.

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

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **DimensionSearchDataSource** 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, do the following:
   a) Specify the host and port on which the MDEX Engine is running. These are the only two required values to finish the wizard. If you do not need any additional configuration, click **Finish**.
   For example, the most basic configuration looks like this:



   b) If you are connecting to the MDEX Engine via SSL, click **Browse...** to locate the Web application's private certificate file (typically a PEM file).
   c) Check **Save server in web.config** if you want the data source to refer to the `web.config` file for host and port information. If unchecked, the host and port information is set as a property of the data source.
   d) If you checked **Save server in web.config**, specify the name of a server on which the file is stored.
   This option adds `<configSections>`, `<expressionBuilders>`, and `<endeca>` to your `web.config` file as shown in this example:

```
<configuration>
   <configSections>
      <sectionGroup name="endeca"
      type="Endeca.Data.Configuration.EndecaSectionGroup,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" >
      <section name="servers"
      type="Endeca.Data.Configuration.ServersSection,
```

```
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
      </sectionGroup>
   </configSections>

<!-- (elements removed to show only what Endeca adds) -->

      <expressionBuilders>
         <add expressionPrefix="EndecaConfig"
         type="Endeca.Web.UI.Design.EndecaConfigurationExpression¬
Builder,
         Endeca.Web, Version=2.1.0.0, Culture=neutral,
         PublicKeyToken=6d02be8724ca751c" />
      </expressionBuilders>

<!-- (elements removed to show only what Endeca adds) -->

   <endeca>
      <servers>
         <add name="MDEXServer" hostName="localhost" port="8000"
         certificatePath="" />
      </servers>
   </endeca>
</configuration>
```

Furthermore, this option sets several property values in your ASPX file using ASP.NET expressions, as shown in this example snippet:

```
<cc1:DimensionSearchDataSource ID="DimensionSearchDataSource1"
runat="server"
  MdexCertificatePath='<%$ EndecaConfig:Servers.Servers["DSDSConnec¬
tion"].
   CertificatePath %>'
  MdexHostName='<%$ EndecaConfig:Servers.Servers["DSDSConnection"].Host¬
Name %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["DSDSConnection"].Port
%>'>
</cc1:DimensionSearchDataSource>
```

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. Adding data source parameters makes them available to other controls on the page.
8. On the **Edit data source parameters** screen, do the following:
   a) To add all parameters available to the data source control, click **Add All Parameters**. These are all the parameters that can be passed to the MDEX Engine. An application will likely only use a small number of these parameters; it is not required to include them all if not desired.
   b) To add only specific parameters, you can add one parameter at a time. Click **Add Parameter** and specify a name.
   c) To indicate where the value of a parameter comes from, select an option from the **Parameter** source list.
   d) To specify the value of a property, select the property and provide a value in **DefaultValue**.
   e) If desired, you can use the **Up**, **Down**, and **Delete** buttons to change the display position of parameters and delete parameters.
9. Click **Finish**.
10. In the **Properties** window of Visual Studio, modify the following properties for the data source control if necessary. Many of these properties are set when you run the **Configure Data Source...** wizard.

Property	Description
MDEXCertificatePath	Optional. Specifies a path to the Web application's private certificate file (typically a PEM file) if you are connecting to the MDEX Engine via SSL.
MDEXHostName	Required. Specifies the host on which the MDEX Engine is running.
MDEXPort	Required. Specifies the port on which the MDEX Engine is running.
SelectParameters	Optional. Specifies a collection of parameters to use when the data source queries the MDEX Engine.
DimensionValuesPerDimension	Optional. Specifies the number of results to return for the dimension search.
EqlExpression	Optional. Specifies the Endeca Query Language expression for the dimension search.
RecordFilter	Optional. Specifies the record filter for the dimension search.
SearchMode	Specifies a search mode for each dimension search operation in a query. Valid search modes are the following: All, AllAny, AllPartial, Any, Boolean, Partial, PartialMax. See "Valid search modes" in the *Endeca Basic Development Guide* for a definition of each mode.
SearchTerms	Required. Specifies the search terms for the dimension search.

Note: This is not a comprehensive list of the properties available in the Properties window. There may also be additional properties available to the control from the code behind (the .aspx.cs file for the control) that do not display in the Properties window but are necessary to fully configure the data source. For details about the properties for each data source control, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*.

DimensionSearchDataSouce example code

The markup generated on the **Source** page of Visual Studio is similar to the following:

```
<cc1:DimensionSearchDataSource ID="DimensionSearchDataSource1"
runat="server" MdexPort="7900"
   MdexCertificatePath="" MdexHostName="smith-690">
</cc1:DimensionSearchDataSource>
```

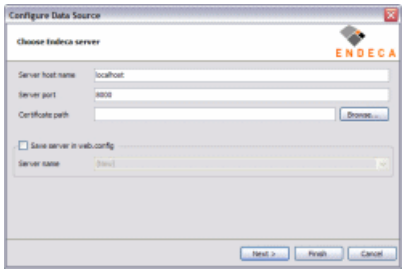
Creating a CompoundDimensionSearchDataSource control

You create and configure a **CompoundDimensionSearchDataSource** control in order to provide results produced by compound dimension search to other controls in your Web site.

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

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **CompoundDimensionSearchDataSource** 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, do the following:
   a) Specify the host and port on which the MDEX Engine is running. These are the only two required values to finish the wizard. If you do not need any additional configuration, click **Finish**.
   For example, the most basic configuration looks like this:



   b) If you are connecting to the MDEX Engine via SSL, click **Browse...** to locate the Web application's private certificate file (typically a PEM file).
   c) Check **Save server in web.config** if you want the data source to refer to the `web.config` file for host and port information. If unchecked, the host and port information is set as a property of the data source.
   d) If you checked **Save server in web.config**, specify the name of a server on which the file is stored.
   This option adds `<configSections>`, `<expressionBuilders>`, and `<endeca>` to your `web.config` file as shown in this example:

```
<configuration>
   <configSections>
      <sectionGroup name="endeca"
      type="Endeca.Data.Configuration.EndecaSectionGroup,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" >
      <section name="servers"
      type="Endeca.Data.Configuration.ServersSection,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
      </sectionGroup>
   </configSections>

<!-- (elements removed to show only what Endeca adds) -->

      <expressionBuilders>
         <add expressionPrefix="EndecaConfig"
         type="Endeca.Web.UI.Design.EndecaConfigurationExpression¬
Builder,
         Endeca.Web, Version=2.1.0.0, Culture=neutral,
         PublicKeyToken=6d02be8724ca751c" />
      </expressionBuilders>

<!-- (elements removed to show only what Endeca adds) -->

   <endeca>
      <servers>
         <add name="MDEXServer" hostName="localhost" port="8000"
```

```
         certificatePath="" />
      </servers>
   </endeca>
</configuration>
```

Furthermore, this option sets several property values in your ASPX file using ASP.NET expressions, as shown in this example snippet:

```
<cc1:CompoundDimensionSearchDataSource ID="CompoundDimensionSearchData¬
Source1"
   runat="server"
   MdexCertificatePath='<%$ EndecaConfig:Servers.Servers["CmpdDim¬
Search"].CertificatePath %>'
  MdexHostName='<%$ EndecaConfig:Servers.Servers["CmpdDimSearch"].Host¬
Name %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["CmpdDimSearch"].Port
%>'>
</cc1:CompoundDimensionSearchDataSource>
```

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. Adding data source parameters makes them available to other controls on the page.
8. On the **Edit data source parameters** screen, do the following:
   a) To add all parameters available to the data source control, click **Add All Parameters**. These are all the parameters that can be passed to the MDEX Engine. An application will likely only use a small number of these parameters; it is not required to include them all if not desired.
   b) To add only specific parameters, you can add one parameter at a time. Click **Add Parameter** and specify a name.
   c) To indicate where the value of a parameter comes from, select an option from the **Parameter** source list.
   d) To specify the value of a property, select the property and provide a value in **DefaultValue**.
   e) If desired, you can use the **Up**, **Down**, and **Delete** buttons to change the display position of parameters and delete parameters.
9. Click **Finish**.
10. In the **Properties** window of Visual Studio, modify the following properties for the data source control if necessary. Many of these properties are set when you run the **Configure Data Source...** wizard.

Property	Description
MDEXCertificatePath	Optional. Specifies a path to the Web application's private certificate file (typically a PEM file) if you are connecting to the MDEX Engine via SSL.
MDEXHostName	Required. Specifies the host on which the MDEX Engine is running.
MDEXPort	Required. Specifies the port on which the MDEX Engine is running.
SelectParameters	Optional. Specifies a collection of parameters to use when the data source queries the MDEX Engine.
DimensionValuesPerDimension	Optional. Specifies the number of results to return for the compound dimension search.
EqlExpression	Optional. Specifies the Endeca Query Language expression for the compound dimension search.

Property	Description
RecordFilter	Optional. Specifies the record filter for the compound dimension search.
SearchMode	Optional. Specifies a search mode for each compound dimension search operation in a query. Valid search modes are the following: All, AllAny, AllPartial, Any, Boolean, Partial, PartialMax. See "Valid search modes" in the *Endeca Basic Development Guide* for a definition of each mode.
SearchTerms	Required. Specifies the search terms for the dimension search.

Note: This is not a comprehensive list of the properties available in the Properties window. There may also be additional properties available to the control from the code behind (the .aspx.cs file for the control) that do not display in the Properties window but are necessary to fully configure the data source. For details about the properties for each data source control, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*.

CompoundDimensionSearchDataSouce example code

The markup generated on the **Source** page of Visual Studio is similar to the following:

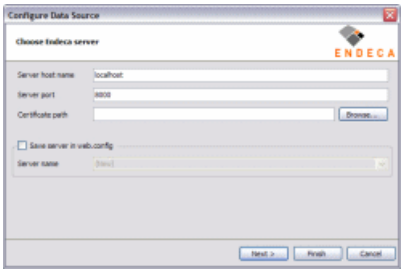
```
<cc1:CompoundDimensionSearchDataSource ID="CompoundDimensionSearchData¬
Source1" runat="server"
   MdexCertificatePath='<%$ EndecaConfig:Servers.Servers["CmpdDim¬
Search"].CertificatePath %>'
   MdexHostName='<%$ EndecaConfig:Servers.Servers["CmpdDimSearch"].HostName
 %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["CmpdDimSearch"].Port %>'>
</cc1:CompoundDimensionSearchDataSource>
```

Creating a RecordDetailsDataSource control

You create and configure a **RecordDetailsDataSource** control in order to provide record detail information (property values and dimension values) about a single record to other controls in your Web site.

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

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **RecordDetailsDataSource** 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, do the following:
   a) Specify the host and port on which the MDEX Engine is running. These are the only two required values to finish the wizard. If you do not need any additional configuration, click **Finish**.
   For example, the most basic configuration looks like this:



b) If you are connecting to the MDEX Engine via SSL, click **Browse...** to locate the Web application's private certificate file (typically a PEM file).
c) Check **Save server in web.config** if you want the data source to refer to the `web.config` file for host and port information. If unchecked, the host and port information is set as a property of the data source.
d) If you checked **Save server in web.config**, specify the name of a server on which the file is stored.
This option adds `<configSections>`, `<expressionBuilders>`, and `<endeca>` to your `web.config` file as shown in this example:

```
<configuration>
   <configSections>
      <sectionGroup name="endeca"
      type="Endeca.Data.Configuration.EndecaSectionGroup,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" >
      <section name="servers"
      type="Endeca.Data.Configuration.ServersSection,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
      </sectionGroup>
   </configSections>

<!-- ...(elements removed to show only what Endeca adds)... -->

      <expressionBuilders>
         <add expressionPrefix="EndecaConfig"
         type="Endeca.Web.UI.Design.EndecaConfigurationExpression¬
Builder,
         Endeca.Web, Version=2.1.0.0, Culture=neutral,
         PublicKeyToken=6d02be8724ca751c" />
      </expressionBuilders>

<!-- ...(elements removed to show only what Endeca adds)... -->

   <endeca>
      <servers>
         <add name="MDEXServer" hostName="localhost" port="8000"
         certificatePath="" />
      </servers>
   </endeca>
</configuration>
```

Furthermore, this option sets several property values in your ASPX file using ASP.NET expressions, as shown in this example snippet of a **NavigationDataSource** control:

```
<cc1:RecordDetailsDataSource ID="RecordDetailsDataSource1"
   runat="server" MdexCertificatePath='<%$
    EndecaConfig:Servers.Servers["RDDSConnection"].CertificatePath %>'
```

```
  MdexHostName='<%$ EndecaConfig:Servers.Servers["RDDSConnection"].Host¬
Name %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["RDDSConnection"].Port
%>'>
</cc1:RecordDetailsDataSource>
```

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. Adding data source parameters makes them available to other controls on the page.
8. On the **Edit data source parameters** screen, do the following:
   a) To add all parameters available to the data source control, click **Add All Parameters**. These are all the parameters that can be passed to the MDEX Engine. An application will likely only use a small number of these parameters; it is not required to include them all if not desired.
   b) To add only specific parameters, you can add one parameter at a time. Click **Add Parameter** and specify a name.
   c) To indicate where the value of a parameter comes from, select an option from the **Parameter source** list.
   d) To specify the value of a property, select the property and provide a value in **DefaultValue**.
   e) If desired, you can use the **Up**, **Down**, and **Delete** buttons to change the display position of parameters and delete parameters.
9. Click **Finish**.
10. In the **Properties** window of Visual Studio, modify the following properties for the data source control if necessary. Many of these properties are set when you run the **Configure Data Source ...** wizard.

Property	Description
MDEXCertificatePath	Optional. Specifies a path to the Web application's private certificate file (typically a PEM file) if you are connecting to the MDEX Engine via SSL.
MDEXHostName	Required. Specifies the host on which the MDEX Engine is running.
MDEXPort	Required. Specifies the port on which the MDEX Engine is running.
SelectParameters	Optional. Specifies a collection of parameters to use when the data source queries the MDEX Engine.
Identifier	Required. Specifies the identifier for a record whose details you want to display.

Note: This is not a comprehensive list of the properties available in the Properties window. There may also be additional properties available to the control from the code behind (the .aspx.cs file for the control) that do not display in the Properties window but are necessary to fully configure the data source. For details about the properties for each data source control, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*.

RecordDetailsDataSouce example code

The markup generated on the **Source** page of Visual Studio is similar to the following snippet. This example sets the Identifier when a user clicks a record in a ListBox control.

```
<cc1:RecordDetailsDataSource ID="RecordDetailsDataSource1" runat="server"
```

```
   MdexHostName="smith-690">
   <SelectParameters>
      <asp:ControlParameter ControlID="LinkButton1" Name="Identifier"
PropertyName="Text" />
   </SelectParameters>
</cc1:RecordDetailsDataSource>
```

Creating an AggregateRecordDetailsDataSource control

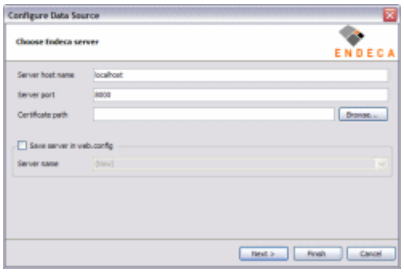
You create and configure an **AggregateRecordDetailsDataSource** control in order to provide details about a single aggregate record to other controls in your Web site.

You specify an `AggregationKey` property and any other properties necessary to describe the set of aggregate records to return from the MDEX Engine. These properties may include `SelectedDimen¬sionValueIds`, `EqlExpression`, `RecordFilter`, and so on. Together these properties build up the navigation state of an aggregate record query submitted to the MDEX Engine.

The MDEX Engine returns an aggregate record in an `AggregateRecordsResult` object. The `Ag¬gregateRecordsResult` object contains the `AggregateRecord` object, which contains a collection of `Record` objects and the properties and dimensions of a single representative record.

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

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **AggregateRecordDetailsDataSource** 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, do the following:
   a) Specify the host and port on which the MDEX Engine is running. These are the only two required values to finish the wizard. If you do not need any additional configuration, click **Finish**.
   For example, the most basic configuration looks like this:



   b) If you are connecting to the MDEX Engine via SSL, click **Browse...** to locate the Web application's private certificate file (typically a PEM file).
   c) Check **Save server in web.config** if you want the data source to refer to the `web.config` file for host and port information. If unchecked, the host and port information is set as a property of the data source.
   d) If you checked **Save server in web.config**, specify the name of a server on which the file is stored.

This option adds <configSections>, <expressionBuilders>, and <endeca> to your web.config file as shown in this example:

```
<configuration>
   <configSections>
      <sectionGroup name="endeca"
      type="Endeca.Data.Configuration.EndecaSectionGroup,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" >
      <section name="servers"
      type="Endeca.Data.Configuration.ServersSection,
      Endeca.Data, Version=2.1.0.0, Culture=neutral,
      PublicKeyToken=6d02be8724ca751c" />
      </sectionGroup>
   </configSections>

<!-- ...(elements removed to show only what Endeca adds)... -->

      <expressionBuilders>
         <add expressionPrefix="EndecaConfig"
         type="Endeca.Web.UI.Design.EndecaConfigurationExpression¬
Builder,
         Endeca.Web, Version=2.1.0.0, Culture=neutral,
         PublicKeyToken=6d02be8724ca751c" />
      </expressionBuilders>

<!-- ...(elements removed to show only what Endeca adds)... -->

   <endeca>
      <servers>
         <add name="MDEXServer" hostName="localhost" port="8000"
         certificatePath="" />
      </servers>
   </endeca>
</configuration>
```

Furthermore, this option sets several property values in your ASPX file using ASP.NET expressions, as shown in this example snippet:

```
<cc1:AggregateRecordDetailsDataSource ID="AggregateRecordDetailsData¬
Source1"
  runat="server"
   MdexCertificatePath='<%$ EndecaConfig:Servers.Servers["AGRDSConnec¬
tion"].
    CertificatePath %>'
   MdexHostName='<%$ EndecaConfig:Servers.Servers["AGRDSConnec¬
tion"].HostName %>'
   MdexPort='<%$ EndecaConfig:Servers.Servers["AGRDSConnection"].Port
 %>'>
</cc1:AggregateRecordDetailsDataSource>
```

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. Adding data source parameters makes them available to other controls on the page.
8. On the **Edit data source parameters** screen, do the following:
   a) To add all parameters available to the data source control, click **Add All Parameters**. These are all the parameters that can be passed to the MDEX Engine. An application will likely only use a small number of these parameters; it is not required to include them all if not desired.

b) To add only specific parameters, you can add one parameter at a time. Click **Add Parameter** and specify a name.
c) To indicate where the value of a parameter comes from, select an option from the **Parameter source** list.
d) To specify the value of a property, select the property and provide a value in **DefaultValue**.
e) If desired, you can use the **Up**, **Down**, and **Delete** buttons to change the display position of parameters and delete parameters.

9. Click **Finish**.
10. In the Properties window of Visual Studio, modify the following properties for the data source control if necessary. Many of these properties are set when you run the **Configure Data Source...** wizard.

Property	Description
AggregationKey	Required. Specifies the property or dimension by which records should be aggregated.
EqlExpression	Optional. Specifies the Endeca Query Language expression for the aggregate record's record list.
Identifier	Required. Specifies the identifier for an aggregate record whose details you want to display.
RecordFilter	Optional. Specifies the record filter for the aggregate record list.
MDEXCertificatePath	Optional. Specifies a path to the Web application's private certificate file (typically a PEM file) if you are connecting to the MDEX Engine via SSL.
MDEXHostName	Required. Specifies the host on which the MDEX Engine is running.
MDEXPort	Required. Specifies the port on which the MDEX Engine is running.
SelectParameters	Optional. Specifies a collection of parameters to use when the data source queries the MDEX Engine.

Note: This is not a comprehensive list of the properties available in the Properties window. There may also be additional properties available to the control from the code behind (the .aspx.cs file for the control) that do not display in the Properties window but are necessary to fully configure the data source. For details about the properties for each data source control, see the *Endeca API Reference for the RAD Toolkit for ASP.NET*.

AggregateRecordDetailsDataSouce example code

The markup generated on the **Source** page of Visual is similar to the following snippet:

```
<cc1:AggregateRecordDetailsDataSource ID="RecordDetailsDataSource1"
runat="server"
   MdexHostName="smith-690">
</cc1:AggregateRecordDetailsDataSource>
```

Refreshing a data source control

If you update the records in the MDEX Engine, you can refresh the data source connection to pick up the updates.

To refresh an Endeca data source control:

1. Select the Endeca data source and click the **Smart Tag**.
2. Click **Refresh Schema**.

Working with Endeca user interface controls

This section describes how to create and configure the Endeca user interface controls (**Breadcrumbs**, **GuidedNavigation**, **Pager**, and **TagCloud**) available in the RAD Toolkit for ASP.NET.

Creating a GuidedNavigation control

You create and configure a **GuidedNavigation** control in order to render dimension groups, dimensions, and dimension values on a Web page.

You must have created a **NavigationDataSource** control before performing this task.

To create a **GuidedNavigation** control:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **GuidedNavigation** control on to the **Design** tab of your Web page.
4. Select the **GuidedNavigation** control and click the **Smart Tag**.
5. From the **Choose Data Source** list, select a **NavigationDataSource** control.
   If you checked **Preview Endeca Data** on the **NavigationDataSource**, then the **GuidedNavigation** control displays actual dimension and dimension value names rather than the placeholder names.
6. In the **Properties** window of Visual Studio, modify the following properties for the **GuidedNavigation** control:

Property	Description
DataMember	Required. This is set to `DimensionStatesResult` and is not available to modify.
UrlManagerId	Optional. Specifies the name of the `UrlManager` that builds URLs for the control.
ExclusionDimensionIds	Optional. Excludes the dimension values you specify by ID and displays all other dimension values. Do not use this property in conjunction with the `InclusionDimensionIds` property; use one or the other but not both. By default, the **GuidedNavigation** control includes all dimensions for display.
InclusionDimensionIds	Optional. Displays only the dimension values you specify by ID and no other dimension values. Do not use this property in conjunction with the `ExclusionDimensionIds` property; use one or the other but not both. By default, the **GuidedNavigation** control includes all dimensions for display.
ShowImplicitDimensionValues	Optional. Specifies whether any implicit dimension values display in the control. Choosing `True` displays implicit dimensions. See the *Endeca Basic Development Guide* for more information about implicit dimensions. The default value for `ShowImplic¬itDimensionValues` is `False`.

Property	Description
ShowRefinements	Optional. Specifies whether any dimension values display in the control. Choosing `False` means the control does not display any dimension values. The default value for `ShowRefinements` is `True`.
ShowSelectedDimensionValues	Optional. Specifies whether a selected dimension value displays in the control. Choosing `True` displays the selected dimension values along with an [X] next to the value and also closes the parent dimension. Clicking the [X] removes the dimension value from the query. (This removal feature is essentially an inline breadcrumb for the **GuidedNavigation** control.) Choosing `False` does not display the selected dimension value in the control. The default value for is `False`.

GuidedNavigation example code

The markup generated on the **Source** page is similar to the following:

```
<cc2:GuidedNavigation ID="GuidedNavigation1" runat="server"
   DataSourceID="NavigationDataSource1" ShowRefinements="True">
</cc2:GuidedNavigation>
```

Here is an example **GuidedNavigation** control as rendered on a Web page:

GuidedNavigation control templates

The **GuidedNavigation** control supports several templates that allow you to modify the default presentation of the control. Each template in the control has default markup that is part of the `Ende¬ca.Web.UI.WebControls` assembly. You can over ride the default presentation as necessary by providing your own markup in the templates.

Template name	Description

`DimensionStateGroupTem¬plate`	This template defines how dimension groups display. The default markup for the template is:
	`<DimensionStateGroupTemplate>`
	`  <div class="DimensionGroupName">`
	`  <%# Eval("Name") %>`
	`  </div>`
	`</DimensionStateGroupTemplate>`

Template name	Description

`DimensionStateTemplate`	This template defines how dimensions display. The default markup for the template is:
	`<DimensionStateTemplate>`
	`  <asp:LinkButton runat="server" Command¬`
	`Name="ToggleDimensionStateExposure"`
	`  Visible='<%# Eval("IsRefinable") %>'>`
	`  <%# Eval("Dimension.DisplayName") %>`
	`  </asp:LinkButton>`
	`  <asp:Label runat="server" Visible='<%#`
	`!(bool)Eval("IsRefinable") %>'>`
	`  <%# Eval("Dimension.DisplayName") %>`
	`  </asp:Label>`
	`</DimensionStateTemplate>`

`RefinementDimensionVal¬`
`ueTemplate`
	This template defines how dimension values display. The default markup for the template is:
	`<RefinementDimensionValueTemplate>`
	`  <asp:LinkButton runat="server" CommandName="Se¬`
	`lectDimensionValue">`
	`  <%# Eval("DisplayName") %>`
	`  </asp:LinkButton><asp:Label runat="server"`
	`class="RecordCount"`
	`  Visible='<%# Container.DataItem.Record¬`
	`Count.HasValue %>'>`
	`  (<%# Eval("RecordCount") %>)`
	`  </asp:Label>`
	`</RefinementDimensionValueTemplate>`

`SelectedDimensionValueTem¬`
`plate`
	This template defines how selected dimension values display. The default markup for the template is:
	`<SelectedDimensionValueTemplate>`
	`  <asp:Label runat="server"><%# Eval("Display¬`
	`Name") %></asp:Label>`
	`  <asp:LinkButton runat="server" CommandName="Re¬`
	`moveDimensionValue">[x]</asp:LinkButton>`
	`</SelectedDimensionValueTemplate>`

`ImplicitDimensionValueTem¬`
`plate`
	This template defines how implicit dimension values display (if they are enabled for display). The default markup for the template is:
	`<ImplicitDimensionValueTemplate>`
	`  <asp:Label runat="server"><%# Eval("Display¬`
	`Name") %></asp:Label>`
	`</ImplicitDimensionValueTemplate>`

`ShowMoreDimensionValuesTem¬`
`plate`
	This template defines how the default **More...** link displays when additional dimension values can be selected. The default markup for the template is:
	`<ShowMoreDimensionValuesTemplate>`
	`  <asp:LinkButton runat="server" Command¬`
	`Name="ShowMore">More...</asp:LinkButton>`
	`</ShowMoreDimensionValuesTemplate>`

Creating a Breadcrumbs control

You create and configure a **Breadcrumbs** control in order to display the search and navigation descriptors for a query. Descriptors include dimension value paths (including inert dimensions), search terms, and range filters.

You must have created a **NavigationDataSource** control before performing this task.

The **Breadcrumbs** control supports two modes to render bread crumbs -- standard mode and alternate mode. Standard mode separates search and navigation descriptors with the > sign, and you remove all search and navigation descriptors (the full path) by clicking the **Remove** link in a breadcrumb.

In alternate mode, each search and navigation descriptor after the root dimension is separated by an `X` (the separator is configurable as text or an image of your choice), and you remove search and navigation descriptors by clicking the `X` for a particular search or navigation descriptor in a query. For example, in a breadcrumb such as `Wine Type > Red (X) > Merlot (X)`, you click `X` to delete one navigation descriptor at a time.

To create a **Breadcrumbs** control:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **Breadcrumbs** control on to the **Design** tab of your Web page.
4. Select the **Breadcrumbs** control and click the **Smart Tag**.
5. From the **Choose Data Source** list, select an **NavigationDataSource** on the page.
6. You can run the **Configure Data Source** wizard to modify the data source you selected above. This step is typically unnecessary.
7. In the **Properties** window of Visual Studio, modify the following properties for the **Breadcrumbs** control:

Property	Description
DataMember	Required. This is set to `AppliedFiltersRe¬sult` and cannot be modified.
UrlManagerId	Optional. Specifies the name of the `UrlManager` that builds URLs for the control.
Title	Optional. Specifies a title for the control if breadcrumbs exist. If there are no breadcrumbs to display the value of `TitleWhenEmpty` displays. The default value for `Title` is an empty string.
TitleWhenEmpty	Optional. Specifies a title for the control to display if breadcrumbs do not exist. The default value for `TitleWhenEmpty` is an empty string.
ExclusionDimensionIds	Optional. Excludes the dimension values you specify by ID and displays all other dimension values in a breadcrumb. Do not use this property in conjunction with the `InclusionDimension¬Ids` property; use one or the other but not both. By default, the **Breadcrumbs** control includes all dimensions for display in a breadcrumb path.
InclusionDimensionIds	Optional. Displays only the dimension values you specify by ID and no other dimension values in a breadcrumb. Dimension values display in the

Property	Description
	breadcrumb in the same order you specify them in the array. Do not use this property in conjunction with the `ExclusionDimensionIds` property; use one or the other but not both. By default, the **Breadcrumbs** control includes all dimensions for display.
SuggestedSearchAdjustmentCaption	Optional. Specifies text that precedes the list of Did You Mean options (i.e. search adjustments).
RemoveImageUrl	Optional. Use only if `UseAlternateRendering` is set to `True`. Browse to an image file you want the user to click to remove a breadcrumb. If you specify **RemoveImageUrl** and **RemoveLinkText**, the image takes precedence over the link text.
RemoveLinkText	Optional. Use only if `UseAlternateRendering` is set to `True`. Specify text that you want to use to for the link to remove a breadcrumb. (This is often simply an X.)
Separator	Optional. Specifies what separator to use between navigation descriptors in the control. The default value for `Separator` is the > sign.
ShowSelectedDimensionValues	Optional. Specifies whether dimension values display in breadcrumbs. `True` enables display and `False` disables display. The default value for `ShowSelectedDimensionValues` is `True`. If this is `False` the **ShowFullyInertDimensions** and **ShowPartiallyInertDimensions** settings do not apply.
ShowFullyInertSelectedDimensionValues	Optional. Specifies whether inert dimensions display in breadcrumbs. A fully inert dimension shows every dimension value selected as part of the breadcrumb. (Typically clicking on an inert dimension changes the dimension tree but does not change record results. Therefore inert dimensions usually do not appear in breadcrumbs. However, there may be cases where you want to set `ShowFullyInertSelectedDimensionVa¬lues` to `True` to show each inert dimension selected in a breadcrumb trail.) The default value for `ShowFullyInertSelectedDimensionVa¬lues` is `True`.
ShowPartiallyInertSelectedDimensionValues	Optional. Specifies whether inert dimensions display in breadcrumbs. A partially inert dimension shows only the leaf dimension value in the breadcrumb rather than the entire path to the leaf. In other words, the first dimensions in the path are normal dimensions and the leaf dimension is inert. The default value for `ShowPartiallyInertSe¬lectedDimensionValues` is `True`.

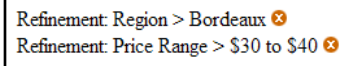
Property	Description
ShowRangeFilters	Optional. Specifies whether range filters display in breadcrumbs. `True` enables display and `False` disables display. The default value for `ShowRangeFilters` is `True`.
ShowSearches	Optional. Specifies whether the search terms in a query display in breadcrumbs. `True` enables display and `False` disables display. The default value for `ShowSearches` is `True`.
UseAlternateRendering	Optional. Specifies whether to use standard mode breadcrumbs or alternate mode breadcrumbs. `True` enables alternate mode. If you set this to `True`, you can also specify a value for `RemoveImageUrl` or `RemoveLinkText`. If you specify neither property, the default separator for alternate rendering is an X. The default value for `UseAlternateRendering` is `False`.

Breadcrumbs example code

The markup generated on the **Source** page is similar to the following

```
<cc2:Breadcrumbs ID="Breadcrumbs1" runat="server" DataSourceID="NavigationDataSource1"
   RemoveImageUrl="~/Resources/x.gif" UseAlternateRendering="True">
</cc2:Breadcrumbs>
```

When navigating the sample wine data set, the control in the above example renders on a Web page like this:



Breadcrumbs control templates

The **Breadcrumbs** control supports several templates that allow you to modify the default presentation of the control. Each template in the control has default markup that is part of the `Endeca.Web.UI.WebControls` assembly. You can over ride the default presentation as necessary by providing your own markup in the templates.

Template name	Description

`FirstDimensionValueTemplate`	This template defines how the first dimension value in a breadcrumb displays. The formatting you specify applies only to the first dimension in the breadcrumb path. The default markup for the template is:
	`<FirstDimensionValueTemplate>`
	`<asp:Button runat="server" Text="Remove"`
	`CommandName="RemoveDimensionValue" />`
	`<%# Eval("DisplayName") %>`
	`</FirstDimensionValueTemplate>`

Template name	Description

	For example:
	`<FirstDimensionValueTemplate>`
	`  <asp:LinkButton runat="server" Text="[Remove]"`
	`  CommandName="RemoveDimensionValue" />`
	`  <%# Eval("DisplayName") %>`
	`</FirstDimensionValueTemplate>`

`IntermediateDimensionValueTemplate`	This template defines how dimension values display that occur between the first and last dimension values (the intermediates). The formatting you specify applies to all the intermediate dimensions. You write the template once and it can be rendered any number of times for each intermediate dimension in a breadcrumb path. The default markup for the template is:
	`<IntermediateDimensionValueTemplate>`
	`> <asp:LinkButton runat="server"`
	`  CommandName="SelectDimensionValue" >`
	`  <%# Eval("DisplayName") %>`
	`  </asp:LinkButton>`
	`</IntermediateDimensionValueTemplate>`

`LastDimensionValueTemplate`	This template defines how the last dimension value in a breadcrumb displays. The formatting you specify applies only to the last dimension in the breadcrumb path. The default markup for the template is:
	`<LastDimensionValueTemplate>`
	`  > <%# Eval("DisplayName") %>`
	`</LastDimensionValueTemplate>`
	For example:
	`<LastDimensionValueTemplate>`
	`  <%# Eval("DisplayName")%>`
	`  <asp:LinkButton runat="server" Text="(x)" CommandName="SelectParentDimensionValue" />`
	`</LastDimensionValueTemplate>`

`RangeBreadcrumbTemplate`	This template defines how range values display. For example, this template replaces the default display with boundaries of the selected range.
	`<RangeBreadcrumbTemplate>`
	`  <asp:Button runat="server" Text="Remove" CommandName="RemoveRangeFilter" />`
	`  <span class="Caption">Range: </span>`
	`  <asp:PlaceHolder runat="server" Visible='<%#`
	`  Container.DataItem.Operator ==`
	`  RangeFilterOperator.Between %>'>`
	`  <%# Eval("Bound1") %> <= <%# Eval("PropertyName") %>`
	`  <= <%# Eval("Bound2") %>`
	`  </asp:PlaceHolder>`
	`  <asp:PlaceHolder runat="server" Visible='<%#`
	`Container.DataItem.Operator`
	`  == RangeFilterOperator.LessThan %>'>`

Template name	Description
	<%# Eval("PropertyName") %> < <%# Eval("Bound1") %> <asp:PlaceHolder> <asp:PlaceHolder runat="server" Visible='<%# Container.DataItem.Operator == RangeFilterOperator.LessThanOrEqual %>'> <%# Eval("PropertyName") %> <= <%# Eval("Bound1") %> </asp:PlaceHolder> <asp:PlaceHolder runat="server" Visible='<%# Container.DataItem.Operator == RangeFilterOperator.GreaterThan %>'> <%# Eval("PropertyName") %> > <%# Eval("Bound1") %> </asp:PlaceHolder> <asp:PlaceHolder runat="server" Visible='<%# Container.DataItem.Operator == RangeFilterOperator.GreaterThanOrEqual %>'> <%# Eval("PropertyName") %> => <%# Eval("Bound1") %> </asp:PlaceHolder> </RangeBreadcrumbTemplate>
SearchBreadcrumbTemplate	This template defines how search terms display. In this example, the template replaces the default display with an auto correction option; however, it does not include a Did You Mean option. <SearchBreadcrumbTemplate> <asp:Button runat="server" Text="Remove" Com¬ mandName="RemoveSearch" /> Search: <%# Eval("Search.Terms") %> <asp:Repeater runat="server" DataSource='<%# Eval ("AppliedSearchAdjustments") %>'> <ItemTemplate> > Corrected To: <%# Eval("Terms") %> </ItemTemplate> </asp:Repeater> </SearchBreadcrumbTemplate>
SuggestedSearchAdjust¬ mentTemplate	This template defines how Did You Mean options display. The default markup for the template is: <SuggestedSearchAdjustmentTemplate> <span runat="server" class="Caption" visi¬ ble='<%# Container.DataItemIndex > 0 %>'> or <asp:LinkButton runat="server" Text='<%# Eval("Terms") %>'

Template name	Description
	CommandName="ApplyAdjustment" /> </SuggestedSearchAdjustmentTemplate>

Creating a Pager control

You create and configure a **Pager** control in order to provide paging controls on a Web page. You can set the number of results to display per page.

You must have created a **NavigationDataSource** control before performing this task.

By default, a **NavigationDataSource** returns a maximum of ten records for display on a Web page. Additional results can be displayed on Next or Previous pages as provided by the control. You modify the items per page value on the **NavigationDataSource**, not the **Pager** control. The **Pager** control should provide an items per page choice that is equal to the default items per page property of the NavigationDataSource.

By default, the **Pager** control displays choices of 10, 20, or 30 items per page. To override this setting, use the `ItemsPerPageChoices` property. For example, if you change `ItemsPerPageChoices` to `50,100,150` then the user can choose to display a maximum of 50 records, 100 records, or 150 records on a results page.

To create a **Pager** control:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **Pager** control on to the **Design** tab of your Web page.
4. Select the **Pager** and click the **Smart Tag**.
5. From the **Choose Data Source** list, select a **NavigationDataSource** control.
6. In the Properties window of Visual Studio, modify the following properties for the **Paging** control:

Property	Description
ItemsPerPageChoices	Optional. Specifies the number of results per page available to the user. Clicking one of the page sizes displays that number of results per page. To modify the default number of results per page, change `10,20,30` to page size values of your choice. For example, changing this property to `50,100,150` creates three page size options in increments of 50, 100, and 150 records per page.
ItemsPerPageChoicesLabel	Optional. Specifies a caption before listing the items per page choices, for example, `Results Per Page`.
OffsetChoicesLabel	Optional. Specifies a caption before listing the offset choices, for example, `Page`.
RenderInTable	Optional. Specifies whether to render the control in a table format. The default value is `False`.
NextBlockText	Optional. Specifies the marker the user clicks for next block of pages. The default value for `NextBlockText` is >>. For example, if the `PageButtonCount` is `5`, then clicking >> moves the set of pages from 1-5 to 6-11, and so on.

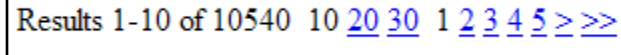
Property	Description
NextPageText	Optional. Specifies the marker the user clicks for the next page of results. The default value for `NextPageText` is >.
PageButtonCount	Optional. Specifies the number of pages that display when a user clicks >> (indicated by the `NextBlockText` property) or the user clicks << (indicated by the `PreviousBlockText` property). The default value for `PageButtonCount` is 5.
PreviousBlockText	Optional. Specifies the marker the user clicks for previous block of pages. The default value for `PreviousBlockText` is <<. For example, if the `PageButtonCount` is 5, then clicking >> moves the set of pages from 1-5 to 6-11, and so on. The default value for `PreviousBlockText` is <<.
PreviousPageText	Optional. Specifies the marker the user clicks for the previous page of results. The default value for `PreviousPageText` is <.
ItemsType	Required. Select `RecordsResult` if you want the number of results per page to be based on record count. Select `AggregateRecordsResult` if you want the number of results per page to be based on aggregate records.
UrlManagerId	Optional. Specifies the name of the `UrlManager` that builds URLs for the control.
NextBlockImageUrl	Optional. Specifies an image that the user clicks for next block of pages. Setting an image for `NextBlockImageUrl` overrides the value of `NextBlockText`.
NextPageImageUrl	Optional. Specifies an image that the user clicks for the next page of results. Setting an image for `NextPageImageUrl` overrides the value of `NextPageText`.
PreviousBlockImageUrl	Optional. Specifies an image that the user clicks for previous block of pages. Setting an image for `PreviousBlockImageUrl` overrides the value of `PreviousBlockText`.
PreviousPageImageUrl	Optional. Specifies an image that the user clicks for the previous page of results. Setting an image for `PreviousPageImageUrl` overrides the value of `PreviousPageText`.

Pager example code

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

```
<cc2:Pager ID="Pager1" runat="server" DataMember="RecordsResult"
   DataSource ID="NavigationDataSource1"
   ItemsPerPageChoices="10,20,30">
</cc2:Pager>
```

Here is an example **Pager** control as rendered on a Web page:



Pager control templates

The **Pager** control supports several templates that allow you to modify the default presentation of the control. Each template in the control has default markup that is part of the `Endeca.Web.UI.WebCon¬trols` assembly. You can override the default presentation as necessary by providing your own markup in the templates.

Template name	Description

`ItemsPerPage¬ ChoiceTemplate`	This template defines the page size choices available to the user. The default markup for the template is:
	`<ItemsPerPageChoiceTemplate>`
	`  <asp:LinkButton runat="server" CommandArgument="<%#`
	`  Container.DataItem.ItemsPerPage %>" Command¬ Name="ChangeItemsPerPage"`
	`  Visible='<%# !Container.DataItem.IsSelected %>'>`
	`  <%# Container.DataItem.ItemsPerPage %></asp:LinkButton>`
	`  <asp:Label ID="Label2" runat="server" Text="<%#`
	`  Container.DataItem.ItemsPerPage %>" Visible='<%#`
	`  Container.DataItem.IsSelected %>'/>`
	`</ItemsPerPageChoiceTemplate>`

`NextBlockTemplate`	This template defines the user interface element the user clicks for next block of pages. The default markup for the template is:
	`<NextBlockTemplate>`
	`  <asp:LinkButton runat="server" CommandArgument="<%#`
	` Container.Offset %>"`
	`  CommandName="ChangeOffset" >`
	`  <%# Container.Text %></asp:LinkButton>`
	`</NextBlockTemplate>`
	In this example, this template replaces the default element of >> with linked text that displays `Next Block`.
	`<NextBlockTemplate>`
	`  <asp:LinkButton runat="server" Text="Next Block"`
	`  CommandName="ChangeOffset"`
	`  CommandArgument="<%# Container.Offset %>">`
	`  </asp:LinkButton>`
	`</NextBlockTemplate>`

`NextPageTemplate`	This template defines the user interface element the user clicks for the next page of results. The default markup for the template is:
	`<NextPageTemplate>`
	`  <asp:LinkButton runat="server" CommandArgument="<%#`
	`  Container.Offset %>" CommandName="ChangeOffset" >`
	`  <%# Container.Text %></asp:LinkButton>`
	`</NextPageTemplate>`
	In this example, this template replaces the default element of > with linked text that displays `Next Page`.
	`<NextPageTemplate>`
	`  <asp:LinkButton ID="NextPageId1" runat="server"`
	`Text="Next Page"`

Template name	Description
	CommandName="ChangeOffset" CommandArgument="<%# Container.Offset %>"> </asp:LinkButton> </NextPageTemplate>
PreviousBlockTem¬plate	This template defines the user interface element the user clicks for previous block of pages. The default markup for the template is: <PreviousBlockTemplate> <asp:LinkButton runat="server" CommandArgument="<%# Container.Offset %>" CommandName="ChangeOffset" > <%# Container.Text %></asp:LinkButton> </PreviousBlockTemplate> In this example, this template replaces the default element of << with linked text that displays Previous Block. <PreviousBlockTemplate> <asp:LinkButton runat="server" Text="Previous Block" CommandName="ChangeOffset" CommandArgument="<%# Container.Offset %>"> </asp:LinkButton> </PreviousBlockTemplate>
PreviousPageTem¬plate	This template defines the user interface element the user clicks for the previous page of results. The default markup for the template is: <PreviousPageTemplate> <asp:LinkButton runat="server" CommandArgument="<%# Container.Offset %>" CommandName="ChangeOffset" > <%# Container.Text %></asp:LinkButton> </PreviousPageTemplate> In this example, this template replaces the default element of < with linked text that displays Previous Page. <PreviousPageTemplate> <asp:LinkButton ID="NextPageId1" runat="server" Text="Previous Page" CommandName="ChangeOffset" CommandArgument="<%# Container.Offset %>"></asp:LinkButton> </PreviousPageTemplate>
NoResultsTemplate	This template defines what to display when no results are returned. The default markup for the template is: <NoResultsTemplate> 0 Matching Records </NoResultsTemplate> In this example, the template replaces "0 matching records" with "No results". <NoResultsTemplate> No results </NoResultsTemplate>

Template name	Description

`OffsetChoiceTem¬ plate`	This template defines the list of pages to display. This is typically `< 1, 2, 3, 4, >` and so on. The default markup for the template is:
	`<OffsetChoiceTemplate>`
	`  <asp:LinkButton runat="server" CommandArgument="<%#`
	`Container.DataItem.Offset`
	`  %>" CommandName="ChangeOffset" Visible='<%# !Contain¬`
	`er.DataItem.IsSelected %>'>`
	`  <%# Container.DataItem.DisplayValue`
	`%></asp:LinkButton>`
	`  <asp:Label ID="Label1" runat="server"`
	`  Text='<%# Container.DataItem.DisplayValue %>' Vis¬`
	`ible='<%#`
	`  Container.DataItem.IsSelected %>'/>`
	`</OffsetChoiceTemplate>`
	For example:
	`<OffsetChoiceTemplate>`
	`  <asp:LinkButton runat="server" Visible="<%# !Contain¬`
	`er.DataItem.IsSelected %>"`
	`  Text="<%# Container.DataItem.DisplayValue %>"`
	`CommandName="ChangeOffset"`
	`  CommandArgument="<%# Container.DataItem.Offset`
	`%>"></asp:LinkButton>`
	`  <asp:Label runat="server" Visible="<%# Contain¬`
	`er.DataItem.IsSelected %>"`
	`  Text="<%# Container.DataItem.DisplayValue`
	`%>"></asp:Label>`
	`</OffsetChoiceTemplate>`

`PagerResultsTem¬ plate`	This template defines the three data items that indicate the first index of page results, last index page, and total number of results. The default markup for the template is:
	`<PagerResultsTemplate>`
	`  Results <asp:Label runat="server"`
	`  Text="<%# Container.PagingInfo.FirstIndexOfPage %>"`
	`/> - <asp:Label`
	`  runat="server" Text="<%# Container.PagingInfo.LastIn¬`
	`dexOfPage %>" />`
	`  of <asp:Label ID="Label3" runat="server" Text="<%#`
	`  Container.PagingInfo.TotalItemCount %>" />`
	`</PagerResultsTemplate>`
	In this example, this information displays similar to `Results 1 - 10 of 596` where 1 is the first index, 10 is the last index and 596 is the total number of results. For example:
	`<PagerResultsTemplate>`
	`  <span>Results:</span>`
	`  <asp:Label ID="firstIndexId1" runat="server"`
	`Text="<%# Container.PagingInfo.FirstIndexOfPage %>"/>`
	`  <span>-</span>`
	`  <asp:Label ID="lastIndexId1" runat="server"`
	`Text="<%# Container.PagingInfo.LastIndexOfPage %>"/>`
	`  <span>of</span>`
	`  <asp:Label ID="totalRecordsId1" runat="server"`
	`Text="<%#`

Template name	Description
	`Container.PagingInfo.TotalItemCount %>"/>` `</PagerResultsTemplate>`

Working with a tag cloud

A **TagCloud** control renders the most frequently used tags or most relevant tags in a dimension as a Web page control. When an application user clicks a tag, the control produces query results that correspond with the selected tag, and the control produces a new set of tags to refine subsequent queries within the dimension.

Dimension requirements

There are several requirements for the dimension that you associate with a **TagCloud** control:

- The dimension must be exposed. To expose the dimension, either a user can select the dimension, or you, the developer, can set the dimension to be exposed. If a dimension is not exposed, the tag cloud does not have any tags to render on the page and outputs the `EmptyTemplate`.
- The dimension must be identified to the **TagCloud** control.
- The dimension must have **Compute refinement statistics** enabled to calculate the correct tag size. If this is not enabled, all tags render as the same size. You enable the **Compute refinement statistics** option on the Dimension editor of Developer Studio. See *Endeca Developer Studio Help* for details.

Typical use case for a tag cloud

In a typical use case for a tag cloud, you create a dimension of extracted terms that spans the entire record set for your application and associate that dimension of extracted terms with the **TagCloud** control.

In addition to the items listed in the "Dimension requirements" section, this use case relies on the following additional tasks:

- Create a dimension of extracted terms. For more information, see the "Implementing Term Discovery and Cluster Discovery" chapter of the *Endeca Relationship Discovery Guide*.
- Exclude the dimension of extracted terms from the navigation menu of the Web page. This dimension is simply a container for the extracted terms and not the kind of dimension a user would click on to navigate. As such, you exclude it from the navigation menu.

Creating a TagCloud control

You create a **TagCloud** control and associate it with one dimension. The **TagCloud** control creates tags based on the dimension values in that dimension.

You must have created a **NavigationDataSource** control before performing this task.

To create a **TagCloud** control:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Endeca RAD Toolkit** tab.
3. Drag the **TagCloud** control on to the **Design** tab of your Web page.
4. Select the **TagCloud** control and click the **Smart Tag**.
5. From the **Choose Data Source** list, select a **NavigationDataSource** on the page.

6. In the **Properties** window of Visual Studio, modify the following properties for the **TagCloud** control:

Property	Description
DataMember	Required. You can set this to `DimensionStatesResult` and specify a number for the `DimensionID` property. Alternatively, you can identify the dimension for the tag cloud by specifying the `DataMember` as `DataMember='DimensionStatesResult.RefinableDimension¬States["Name"]'` where `Name` is the actual name of the dimension. For example, `DataMember='DimensionStatesResult.Refin¬ableDimensionStates["Region"]'` bases the tag cloud on the Region dimension in the sample wine application.
DimensionID	Required if you set `DataMember` to `DimensionStateResult`. Specify the ID of the dimension you want to create the control for. (You can determine a dimension ID by looking at the Dimensions View of Developer Studio or you can examine your `dimensions.xml` file after running Forge.)
UrlManagerId	Optional. Specifies the name of the `UrlManager` that builds URLs for the control.
MaximumNumberOfTags	Optional. Specify a number to restrict the total number of tags that display in the tag cloud. For example, if you specify `7`, then the tag cloud contains up to a maximum of `7` tags. It is good practice to restrict the number of tags if the dimension has many dimension values.
SelectAction	Optional. Choose an option to specify what happens when a tag is clicked. Choose `Navigate` to indicate that clicking the tag is the same as navigating to the dimension value of the same name. Choose `Search` to indicate that clicking the tag performs a search for the text of the tag name within the dimension. If you set `SelectAction` to `Search`, also set `SearchKey` and `SearchMode` to specify what search mode is used when a search is performed. The default value for `SelectAction` is `Navigate`.
SearchKey	Optional. Use when `SelectAction` is set to `Search`. Specify the name of the Endeca property or Endeca search interface to search on. The default value for `SearchKey` is `All`.
SearchMode	Optional. Use when `SelectAction` is set to `Search`. Choose one of the standard Endeca match modes for the tag cloud. For more information about match modes, see the *Endeca Basic Development Guide*.
Sizing	Optional. Specify what the display of tags is based on within a dimension. Select `Frequency` to display tags based on record count. The greater the number of records that contain a tag, the larger the tag size on a Web page. Or select `Relevancy` to calculate the relevance of a tag based on its frequency in the current result set as compared to its frequency across the entire dataset (an implementation of the F Measure algorithim). The default value for `Sizing` is `Frequency`.
SortType	Optional. Choose either `Alphabetical` to list tags alphabetically, or choose `Default` to list tags in the order returned from the MDEX Engine. The default value for `SortType` is `Alphabetical`.
TagSizeCount	Optional. Specify the number of distinct sizes for tags in the controls. You can choose a value from 1 to 9. For example, if you specify `7` , you

Property	Description
	can have up to 7 different sized tags in the tag cloud. If you want a value of 10 or greater, you have to customize your page's CSS to support that sizing. The default value for `TagSizeCount` is 6.

TagCloud example code

In a simple case, the mark up generated on the **Source** page of Visual Studio is similar to the following (your dimension ID varies):

```
<cc2:TagCloud runat="server" DataSourceID="NavigationDataSource1"
   DimensionId="3">
</cc2:TagCloud>
```

Here is an example **TagCloud** control as rendered on a Web page:

TagCloud control templates

The **TagCloud** control supports several templates that allow you to modify the default presentation of the control. Each template in the control has default markup that is part of the `Endeca.Web.UI.WebControls` assembly. You can override the default presentation as necessary by providing your own markup in the templates.

Template name	Description

`DimensionValueTemplate`	This template defines the dimension value name with the contents of the template. The default markup for the template is:
	`<DimensionValueTemplate>`
	`  <asp:LinkButton runat="server"`
	`  CssClass='<%# "Tag" + Container.TagSize + " " + (Container.DataItemIndex`
	`  % 2 == 0 ? "Item" : "AlternatingItem") %>'`
	`  CommandName='<%# TagCloud.SelectAction == TagCloudSelectAction.Navigate ?`
	`  "SelectDimensionValue" : "SearchForDimensionValue" %>'`
	`  Text='<%# Eval("DisplayName") %>' />`
	`</DimensionValueTemplate>`
	Here is an example that creates a tag cloud with dimension value names and their corresponding number of records in parentheses:
	`<DimensionValueTemplate>`
	`  <asp:LinkButton runat="server" CssClass ='<%# "Tag" + Container.TagSize %>'`
	`  OnClick="Navigate" CommandArgument='<%# Eval("Id") %>'>`
	`  <%# Container.RecordCount%>`
	`  </asp:LinkButton>`
	`  <span>(<%# Container.DataItem.RecordCount %>)</span>`
	`</DimensionValueTemplate>`

Template name	Description

	This example relies on you defining a method for `Navigation`:
	`protected void Navigate(object sender, EventArgs e)`
	`{`
	`NavigationDataSource navDatasource = BindingUtili¬ty.FindDataSource<NavigationDataSource>("dsNav", this);`
	`LinkButton link = (LinkButton)sender;`
	`navDatasource.SelectedDimensionValueIds.Add(link.Com¬mandArgument);`
	`}`

`SeparatorTemplate`	This template defines the default tag separator (a space) with the contents of the template. The default markup for the template is:
	`<SeparatorTemplate> </SeparatorTemplate>`
	Here is an example that replaces the space with | between tags:
	`<SeparatorTemplate>|</SeparatorTemplate>`

`EmptyTemplate`	This template defines the content to display if the tag cloud is empty. The default markup for the template is:
	`<EmptyTemplate></EmptyTemplate>`
	Here is an example that adds the text `No tags available.` to an otherwise empty control area:
	`<EmptyTemplate>No tags available.</EmptyTemplate>`

Setting a dimension to be exposed on a Web page

In some scenarios, you may want to force a dimension to be exposed (open) on a Web page. This is necessary, for example, when using a **TagCloud** control.

To set a dimension to be exposed on a Web page:

1. Open your Web site in Visual Studio.
2. Select the **NavigationDataSource** control on the page.
3. In the **Properties** window of Visual Studio, locate the **PermanentRefinementConfigs** property.
4. In its value field, click **Browse...**.
   The **RefinementConfig Collection Editor** displays.
5. In the **RefinementConfig Collection Editor**, click **Add**.
6. Specify a value for the **DimensionValueId** property that identifies the dimension you want to be exposed.
7. Click OK.

Open dimension example code

The mark up generated on the **Source** page of Visual Studio is similar to the following (your dimension varies):

```
<PermanentRefinementConfigs>
   <end:RefinementConfig DimensionValueId="3">
   </end:RefinementConfig>
</PermanentRefinementConfigs>
```

Excluding a dimension from the navigation menu

In some scenarios, you may want to exclude a dimension from the navigation menu of a Web page. This is necessary, for example, if you create a dimension of extracted terms for use with a **TagCloud** control. In this case, the extracted terms span the entire data set and the dimension serves as a container for the terms; you do not want application users to navigate to that dimension.

You must have already created and configured a **GuidedNavigation** control before performing this task.

To exclude a dimension from a navigation menu:

1. Open your Web site in Visual Studio.
2. Select the **GuidedNavigation** control on the page.
3. In the **Properties** window of Visual Studio, locate the **ExclusionDimensionIds** property.
4. In its value field, click **Browse...**.
   The **String Collection Editor** displays.
5. Specify the dimension ID that you want to exclude from the navigation menu.
6. Click OK.

Exclude dimension example code

The markup generated on the **Source** page of Visual Studio is similar to the following (your dimension ID will vary):

```
<cc2:GuidedNavigation runat="server" DataSourceID="NavigationDataSource1"
 ExclusionDimensionIds="3">
</cc2:GuidedNavigation>
```

Renaming dimensions or dimension values before displaying them on a page

You can rename dimensions and dimension values before displaying them on a Web page. You rename a dimension in the `dimensions.xml` file and rename a dimension value using Developer Studio. In either case, you add a `DisplayName` property to the dimension or dimension value. This is part of your MDEX Engine configuration. If the property exists, the RAD Toolkit displays the new name in your application.

You may want to rename dimensions or dimension values before displaying them on a Web page in two cases:

- For internationalization, where you need one MDEX Engine to serve dimensions or dimension values with varying locale designations.
- As a workaround in cases where EQL places name restrictions on dimensions or dimension values that you do not want to affect your Web site (i.e. EQL does not allow spaces in names but you may need spaces in names).

To rename a dimension or dimension value before displaying it on a page:

1. To rename a dimension:
   a) Open the `dimensions.xml` file in a text editor. (This file is typically stored in the `forge_input` directory.)
   b) Locate the dimension you want to rename by its `DIMENSION` element.
   c) In the `DIMENSION` element, add a `PROP` element with a `NAME="DisplayName"` attribute as shown in the next step.
   d) Specify the new display name in the `PVAL` element. This example renames the Price Range dimension to the Price dimension.

```
<DIMENSION NAME="Price Range" SRC_TYPE="INTERNAL">
   <DIMENSION_ID ID="10"/>
   <DIMENSION_NODE>
      <DVAL TYPE="EXACT">
        <DVAL_ID ID="10"/>
        <SYN CLASSIFY="FALSE" DISPLAY="TRUE" SEARCH="FALSE">Price
Range</SYN>
      <PROP NAME="DisplayName">
         <PVAL>Price</PVAL>
      </PROP>
      </DVAL>

<!-- Other dimension values removed from the example. -->

</DIMENSION>
```

   e) Save and close the `dimensions.xml` file.

2. To rename dimension values, see the "Providing additional descriptive information for a dimension value" topic in the *Endeca Developer Studio Help*. When following the procedure in the *Endeca Developer Studio Help*, add a property called `DisplayName` to the dimension value with a name you want to display on the Web page.

Note: If you prefer to use a property with a name other than `displayName`, define the property in `web.config` using a different `propertyName` value as shown below. The example uses `displayName_en_us` instead of `displayName`.

```
<endeca>
      <displayName propertyName="displayName_en_us" />
</endeca>
```

3. Open the `web.config` file for your Web site and add the following `displayName` section:

```
<configSections>

   <!-- Other sections removed from the example. -->

   <section name="displayName" type="Endeca.Data.Configuration.Display¬
NameSection,
   Endeca.Data, Version=2.1.0.0, Culture=neutral, PublicKeyTo¬
ken=6d02be8724ca751c" />
```

```
   </sectionGroup>
</configSections>
```

4. Run a baseline update.

Generating the "More..." link

The Presentation API for .NET handles the "More..." link differently than the RAD API for ASP.NET.

The Presentation API for .NET generates the "More..." link in situations where all of the following occur:

- You select the **Enable dynamic ranking** option in the Dimension editor of Developer Studio.
- You select the **Generate "More..." dimension value** option in the Dimension editor of Developer Studio.
- The MDEX Engine returns query results with more dimension values than the cut off threshold you specified in **Maximum dimension values to return**.

However, the RAD API ignores the **Generate "More..." dimension value** option in Developer Studio and automatically generates a "More..." link when either:

- You select **Enable dynamic ranking** in Developer Studio and the MDEX Engine returns more dimension values than the cut off threshold allows.
- You use RefinementConfigs to set a limit on the number of dimension values to return, and the MDEX Engine returns more dimension values than the cut off threshold allows.

To control which dimensions will display a "More…" link, override the **ShowMoreDimensionValuesTemplate** in the **GuidedNavigation** control.

Note: In a future release of Developer Studio, the **Generate "More..." dimension value** option will be phased out.

Displaying query results on a Web page

Many Visual Studio controls allow you to display query results on a Web page. For example, using a **DataList** control to display results is one expedient way that is described here.

You must have already created and configured a **NavigationDataSource** control.

To display query results on a Web page:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Data** tab.
3. Drag a **DataList** control onto the **Design** page.
4. Select the **DataList** control and click the **Smart Tag**.
5. From the **Choose Data Source** list, select an Endeca data source control. Typically this is the **NavigationDataSource** on the page.
6. In the **Properties** window of Visual Studio, modify the following properties for the **DataList** control:

Property	Description
DataMember	Required. Set this to `RecordsResult.Records` to display Endeca records or set it to `AggregateRecordsResult.AggregateRecords` to display aggregated records.

7. Select the **DataList** control and click the **Smart Tag**.
8. Select **Refresh Schema**.

If you build the Web site at this point, the data items available to `RecordsResult.Records` or `AggregateRecordsResult.AggregateRecords` display in the **DataList** control. You can iterate over these collections to extract and display additional record information. For instructions, see "Displaying nested record properties".

Displaying nested record properties

An Endeca `Record` object contains dimension values and properties. You can iterate over these collections, using one or more **Repeater** controls, to extract and display additional `Record` properties. This topic describes one way to extract and display the nested properties.

To display nested record properties:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Data** tab.
3. Drag a **Repeater** control onto the **Design** page.
4. From the **Smart Tag**, select an Endeca data source control. The **RecordDetailsDataSource** returns a single Endeca record, so it is convenient for this example.
5. Select the **Repeater** control and then select the **Source** page of Visual Studio.
6. Add .NET Eval expressions as necessary to iterate over each `Record` property that you want to expose on the Web page. See the *Endeca API Reference for the RAD Toolkit for ASP.NET* for the available properties of an Endeca `Record`.

Displaying total record count

You can use common Visual Studio controls to display the total number of records that match a query.

You must have already created and configured a **NavigationDataSource** control.

To display total record count:

1. Open your Web site in Visual Studio.
2. In the **Toolbox** window, expand the **Data** tab.
3. Drag a **FormView** control onto the **Design** page.
4. Select the **FormView** control and click the **Smart Tag**.
5. From the **Choose Data Source** list, select an Endeca data source control. Typically this is the **NavigationDataSource** on the page.
6. In the **Properties** window of Visual Studio, modify the following properties for the **FormView** control:

Property	Description
DataMember	Required. Set this to `RecordsResult` to display Endeca records.

7. Select the **FormView** control and click**Edit Templates**.
8. Drag a **Label** control into the **ItemTemplate**.
9. Select the **Label** control and click **Edit Templates**.
10. In **Bindable** properties, select **Text**.
11. Click the **Field Binding** option, and from the **Bound to** list, select **TotalRecordCount**.
12. Click OK.
13. Next to the **Label** control, add descriptive text, such as, `Total records returned:`
14. On the **FormView** control, select **End Template Editing**.

When you build your Web site, you see the total record count displayed on the page.

Record count example code

The markup generated on the **Source** page of Visual Studio is similar to the following:

```
<asp:FormView ID="FormView1" runat="server" DataSourceID="NavigationData¬
Source1"
   DataMember="RecordsResult">
   <ItemTemplate>
      <asp:Label ID="Label1" runat="server" Text='<%# Eval("TotalRecord¬
Count") %>'></asp:Label>
      Total Record Count
   </ItemTemplate>
</asp:FormView>
```

Chapter 4

Creating an application based on the reference application

This section provides a brief overview of how to create an Endeca front-end application using the RAD ASP.NET reference application as a starting point. In this scenario, the reference application acts as a template that you copy and modify as needed until it meets your application requirements.

High level process of creating an application from a reference application

The basic workflow is as follows:

1. Copy the reference application from `<installation path>\Endeca RAD Toolkit .NET\reference\RadAspNet` to a staging area.
2. Apply your custom cascading style sheet (CSS) to the Web pages.
3. Modify the pages and controls for your requirements. This is the bulk of the work. In this step, you modify the refence application's search and navigation features such as dimension controls, paging controls, breadcrumb controls, and so on.
4. Delete any pages and controls that are not necessary in your application.
5. Test and deploy your Web site.

Using reference application controls in your application

In cases where you need only a few user controls, you probably do not want to create an Endeca application based on the reference application. For example, you might already have a partial application running and you simply want to copy individual user controls from the reference application and reuse them in your application.

Endeca provides user controls in the reference application for search boxes, record details, range filters, dynamic business rules, and so on that can be reused.

To use reference application controls in your application:

1. Open Windows Explorer and go to one of the following locations:
   - If your application uses HTTP postbacks, go to `<installation path>\Endeca\RADToolkits\version\ASP.NET\reference\RadAspNetRef\Resources\UserControls`.

- If your application uses URL query parameters, go to *<installation path>*\Endeca\RADToolkits*version*\ASP.NET\reference\RadAspNetRef\ResourcesUrl\UserControls.

2. Identify a user control you want and copy both the ASCX and ASCX.CS file for the control to a directory of your choice within your Web site directory structure. It is useful to create a \UserControls directory, or something similar, if you do not already have one in place.
3. Open your Web site in Visual Studio.
4. In the Solution Explorer, right-click your Web site and select **Add** > **Existing Item...**.
5. Browse to the ASCX and ASCX.CS file for the user control, select the files, and click **Open**. You may or may not get dependency errors for missing helper classes and methods. If you do, copy help code from *<installation path>*\Endeca\RADToolkits*version*\ASP.NET\reference\RadAspNetRef\App_Code

The user control displays in the Solution Explorer file list but not in the Toolbox window. Also, no design time data is sent to the user control. It displays as "databound" in the Design tab of Visual Studio.

Chapter 5

Overview of building URLs

The RAD Toolkit for ASP.NET provides classes to build URLs in your application. This can include links to dimensions, dimension values, records, aggregate records, and so on. The classes in `Ende¬ca.Web.Url` do this work, especially the `UrlManager` class and the `UrlBuilder` class. This section describes how to work with these classes.

About building URLs

This topic describes the basics of building and serializing URLs.

Note: This guide describes the basic tasks involved in building URLs using the RAD Toolkit for ASP.NET. If you are using the URL Optimization API with your application, there are some important additional considerations to ensure properly optimized URLs. For details, please refer to the *URL Optimization API for the RAD Toolkit for ASP.NET Developer's Guide*.

The process to build URLs

The high-level process to build a URL is as follows:

1. Create a `UrlManager` object and register the data sources and commands on your page. The primary function of `UrlManager` is to keep track of the commands that contribute to the URL that is getting built.
2. Initialize the `UrlManager` with the state encoded in a URL by calling `InitializeFrom(String)`. This state is setting up the commands properly to fetch the correct data to render on the page. You are taking the state encoded in a URL, copying that state, and modifying that state to build additional URLs.
3. Build a new URL by modifying the copies of commands registered with the `UrlManager`. This modification may include toggling dimensions, and selecting dimensions, dimension values, adding records, adding parameters, or any other change that represents a new URL in an application.

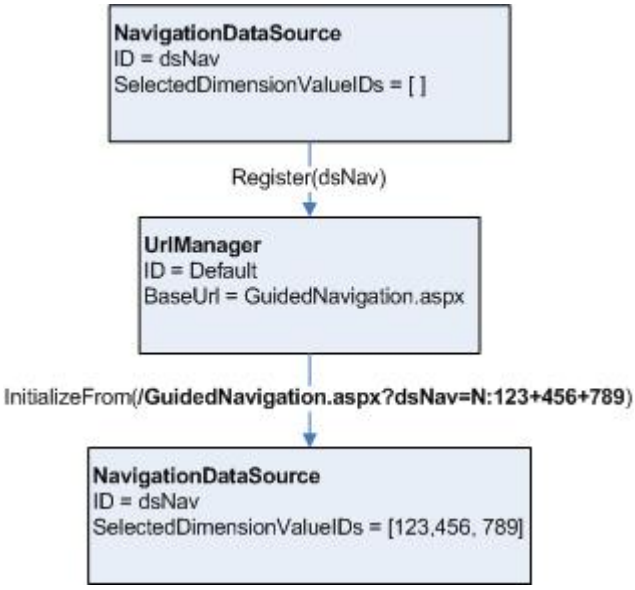
You can use the convenience methods in `UrlBuilder` to build URLs such as

- `SelectDimensionValue`
- `SelectParentDimensionValue`
- `SelectRecord`
- `SelectAggregateRecord`

You can use the more generic `BuildUrl` method. (Behind the scenes, the convenience methods call `BuildUrl`.)

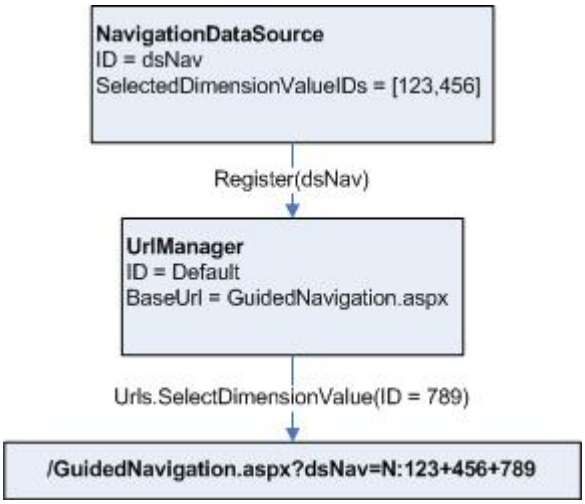
The following diagram illustrates steps 1 and 2.

- The first box represents an empty `NavigationDataSource` with an ID of dsNav.
- The second box represents the default `UrlManager` that registers the empty `NavigationData¬Source` and then initializes itself from the current base URL.
- The third box represents the data source populated with the parameters from the current URL



The following diagram illustrates step 3.

- The first box represents a data source populated with a navigation state of two dimension value IDs (123 and 456).
- The second box represents a `UrlManager` that builds a new URL based on a copy of the navigation state (123 and 456)and includes the addition of another dimension value ID (789).



About URLs in the reference application

The reference application located in *installation path*\Endeca\RADToolkits*version*\ASP.NET\reference\RadAspNetRef makes extensive use of BuildUrl and the convenience methods to build URLs.

Examine the files in *installation path*\Endeca\RADToolkits*version*\ASP.NET\reference\RadAspNetRef\Resources\Url\UserControls for typical URL examples.

About URL serialization

The UrlManager serializes each data source and command into a URL query string. To do this, the UrlManager relies on default classes provided with the API.

The UrlManager's default UrlProvider is the BasicUrlProvider. The BasicUrlProvider serializes each data source using, by default, UrlCommandSerializationProvider. These serialized data sources become the query parameters on the URL where the name of the parameter is the data source ID. The value is the serialized data source string.

To help illustrate which classes are responsible for each part of serialization, here is an example of an Endeca URL:

http://*localhost:port*/RadAspNetRef/Url.aspx?

dsNav1=Nrc:id-6200,N:8021&dsNav2=Nrc:id-2,N:4294960252

The section of the URL starting with the question mark, that is, ?dsNav1=Nrc:id-6200,N:8021&dsNav2=Nrc:id-2,N:4294960252, represents the EndecaCommand in the URL and it is serialized by UrlCommandSerializationProvider. The entire URL is then serialized by BasicUrlProvider.

Customizing URL serialization

URL serialization happens entirely as an implementation detail if you are using the default classes provided with the API.

If you want to customize how EndecaCommand objects are serialized within a URL, you can subclass and customize CommandSerializationProvider. Similiarly, if you want to customize the way the entire URL is serialized, you can subclass and customize UrlProvider and set the UrlManager.UrlProvider property. You can subclass and customize either one or both of these classes.

If you customize the CommandSerializationProvider, you specify the new provider in the application's Web.config file as shown in this example:

```
<commandSerialization defaultProvider="Postback">
   <providers>
       <add name="Postback"
         type="Endeca.Data.DefaultCommandSerializationProvider, Endeca.Data"
 />
       <add name="Url"
          type=MyNameSpace.MyCommandSerializationProvider" />
   </providers>
</commandSerialization>
```

If you customize the `UrlProvider`, you specify the assembly-qualified type name for the new provider in the application's `Web.config` file as shown in this example:

```
<urlProvider defaultType="MyNameSpace.MyUrlProvider, MyAssembly" />
```

You can also override this default setting with a line of code similar to the following:

```
UrlManager.Default.UrlProvider = new MyUrlProvider()
```

Creating a UrlManager and registering data sources and commands

You register data sources and commands with a `UrlManager` so the `UrlManager` can construct URLs based on the state of these objects and initialize these objects from the state encoded in a URL. The registration should happen early in a page's life cycle, such as during the `Page_Load` or `OnInit` methods.

The example below requires the following statements at the top of your code:

```
using System;
using Endeca.Data;
using Endeca.Web.UI;
using RadAspNetRef;
using Endeca.Web.Url;
```

This procedure assumes you have already created data sources or commands in your application and assigned them an ID.

For example, this snippet shows two data sources, each with an ID value:

```
<end:NavigationDataSource
  ID="dsNav"
  runat="server"
  MdexHostName="<%$ Snippet:_mdex.MdexHostName %>"
  MdexPort="<%$ Snippet:_mdex.MdexPort %>">
</end:NavigationDataSource>

<end:RecordDetailsDataSource
  ID="dsRecordDetails"
  runat="server"
  MdexHostName="<%$ Snippet:_mdex.MdexHostName %>"
  MdexPort="<%$ Snippet:_mdex.MdexPort %>">
</end:RecordDetailsDataSource>
```

There are two ways of creating a `UrlManager`. The first way is by calling `UrlManager.Get` and specifying a name as input. If a `UrlManager` by that name does not exist, the method creates it. The second way uses a convenience property named `Default` to create the `UrlManager` with a name of "Default". This is the equivalent to the first way of calling `UrlManager.Get("Default")`; however, the convenience property simply allows you to create the `UrlManager` in one line of code.

After creating the `UrlManager`, you register one or more data sources and commands with it. You register a data source or command by calling `UrlManager.Register`. The input to `Register` is an interface called `ICommandProvider` that allows access to both data sources and commands.

To register data sources and commands:

1. Create a `UrlManager` object using either of the ways described above.

2. Register each data source or command in the application by calling `Register` and specifying each data source ID.

Example of registering data sources

```
// Create a new UrlManager with the name Default if it
// does not exist already.
UrlManager.Get("Default");

// Registers a data source named dsNav with the UrlManager
// named Default.
UrlManager.Register(dsNav);

// Registers a data source named dsRecDetails with the
// UrlManager named Default.
UrlManager.Register(dsRecDetails);
```

This example is logically equivalent to the example above but shorter:

```
// The API provides a convenience method called Default
// to create a new instance of UrlManager.
// This creates a UrlManager and registers two data sources
// with it.
UrlManager.Default.Register(dsNav);
UrlManager.Default.Register(dsRecDetails);
```

Go on to "Initializing a UrlManager".

Initializing a UrlManager

After registering data sources and commands, you initialize the `UrlManager` by calling `Initialize¬From`. This call uses the current URL in an application to populate the data sources and commands registered with `UrlManager` with state encoded in a URL. The initialization should happen early in a page's life cycle, such as during the `Page_Load` or `OnInit` methods.

The example below requires the following statements in your code:

```
using System;
using Endeca.Data;
using Endeca.Web.UI;
using RadAspNetRef;
using Endeca.Web.Url;
```

To initialize a `UrlManager`:

Add code to call `InitializeFrom` and provide `Request.Url`.

Example of initializing a UrlManager

```
// The API provides a convenience method called Default
// to create a new instance of UrlManager.
// This creates a UrlManager and registers two data sources with it.
UrlManager.Default.Register(dsNav);
UrlManager.Default.Register(dsRecDetails);

// Initialize the UrlManager with state from the Request.Url.
UrlManager.Default.InitializeFrom(Request.Url);
```

Building a URL to toggle a dimension and display its dimension values

You can build a URL that toggles a dimension open or closed by calling `UrlBuilder.ToggleDimen¬sionStateExposure` and you can build URLs for the dimension values within it by calling `Select¬DimensionValue`.

The example below requires the following statements in your code:

```
<%@ Register TagPrefix="end" Assembly="Endeca.Web"
   Namespace="Endeca.Web.UI"  %>
<%@ Register TagPrefix="end" Assembly="Endeca.Web.UI.WebControls"
      Namespace="Endeca.Web.UI"  %>
<%@ Import Namespace="Endeca.Data" %>
<%@ Import Namespace="Endeca.Web.Url" %>
```

This procedure assumes you have already created data sources or commands in your application and assigned them an ID.

The `ToggleDimensionStateExposure` method builds a URL for the dimension you want to toggle and `SelectDimensionValue` builds a URL for each dimension value within that dimension.

To build a URL to toggle a dimension open or closed:

1. To build URLs for each dimension, create a repeater control with the following markup:
   a) Specify the data source for the dimension.
   b) Specify `DimensionStatesResult` as the `DataMember` property.
   c) Call `ToggleDimensionStateExposure` and identify the dimension to toggle. The method is overloaded, so you can identify the dimension either by `DimensionState` or by context path.
2. To build URLs for each dimension value, create a nested repeater control within the repeater that contains `ToggleDimensionStateExposure`, and add the following markup:
   a) Specify the data source for the dimension values.
   b) Call `SelectDimensionValue` and identify the dimension. The method is overloaded, so you can identify the dimension either by `DimensionValue` or by context path.

Example markup for toggleing a dimension and showing its dimension values

```
<asp:Repeater runat="server" DataSourceID="dsLeft"
   DataMember="DimensionStatesResult.RefinableDimensionStates">
   <ItemTemplate>
      <a
        href="<%# UrlManager.Default.Urls.ToggleDimensionStateExpo¬
sure(dsLeft.ID,
          Container.DataItem) %>">
        <%# Eval("Dimension.DisplayName") %>
      </a> <br />
    <asp:Repeater runat="server" DataSource='<%# Eval("Refinements") %>'>

        <ItemTemplate>
           <a
             style="margin-left: 10px"
             href="<%# UrlManager.Default.Urls.SelectDimensionVal¬
ue(dsLeft.ID,
               Container.DataItem) %>">
             <%# Eval("DisplayName") %>
           </a> <br />
        </ItemTemplate>
```

```
    </asp:Repeater>
  </ItemTemplate>
</asp:Repeater>
```

Building a URL to a record details page

You can build a URL for a record details page by calling the convenience method `UrlBuilder.SelectRecord`. This method requires a record details data source ID and a record object as input. You can build the URL on the current page of an application, for example, `Default.aspx`, or you can build a URL that links to another page such as `RecordDetails.aspx` by setting the page name in the `UrlManager.BaseUrl` property.

The example below requires the following statements in your code:

```
using System;
using Endeca.Data;
using Endeca.Web.UI;
using RadAspNetRef;
using Endeca.Web.Url;
```

This procedure also assumes you have already created data sources or commands in your application and assigned them an ID.

To build a URL to a record details page:

1. Create a repeater control and specify `RecordsResult` as the `DataMember` property.

   Note: You are not required to create a repeater to call `SelectRecord`. Also, steps 1 and 2 are in the procedure to provide context for a typical use case. If you do not need this context, you can skip to step 3.

2. Create a nested repeater control and specify the data source, in this case, `Records`.
3. Create a hyperlink, and within it, call `SelectRecord`.
4. Provide `SelectRecord` with input parameters that specify the ID of the record details data source and the Endeca record to build the URL for.
   See the first example below for the markup.
5. In the code behind for the markup, do the following:
   a) If you want to build a URL to a page different from the current page, specify a relative path to the application root in the `UrlManager.BaseUrl` property. This property value should begin with a tilde then slash (~/) followed by a path relative to the root of the Web application.
   b) Create a `UrlManager` on the record details page indicated by `UrlManager.BaseUrl`.
   c) Register a `RecordDetailsDataSource` on the record details page by calling `UrlManager.Register` and provide a record details data source (or command). This call is usually done as part of page configuration during a page's `Page_Load` or `OnInit` methods.
   d) Call `InitializeFrom` and provide the `Request.Url`.

   See the second example below for the code behind.
6. Register each data source in the application by calling `Register` and specifying each data source ID.

Example of building a record details page URL

This example shows the markup that calls `SelectRecord` to build a record details URL. Note that for the example, `recDetailsDataSource.ID` is pseudo-code. Your mark up should refer to a record details data source either as a string or by its ID.

```
<asp:Repeater ID="rptRecordResult" runat="server" DataMember="RecordsResult"

    DataSourceID="NavDataSource">
    <ItemTemplate>
        <asp:Repeater ID="rptRecord" runat="server" Data¬
Source='<%#Eval("Records") %>'>
           <ItemTemplate>
              <div>
                 <asp:HyperLink ID="HyperLink2" NavigateUrl='<%#
                 UrlManager.Get("recDetailsUrlManager").Urls.SelectRecord
                 (recDetailsDataSource.ID, Container.DataItem) %>'
runat="server">
                    <%# GetRecordTitle(Container.DataItem) %>
                 </asp:HyperLink>
              </div>
          </ItemTemplate>
       </asp:Repeater>
   </ItemTemplate>
</asp:Repeater>
```

This example shows the code behind that supports the mark up. This is in the page's `Page_Load` method. Note the use of `BaseUrl` that builds the link on the `RecordDetails.aspx` page.

```
recDetailsDataSource = new RecordDetailsDataSource();
recDetailsDataSource.ID = "recDetailsDataSource";

UrlManager.Get("recDetailsUrlManager").BaseUrl =
   "~/Records/RecordDetails.aspx";
UrlManager.Get("recDetailsUrlManager").Register(NavDataSource);
UrlManager.Get("recDetailsUrlManager").Register(recDetailsDataSource);

UrlManager.Get("DefaultPageManager").Register(NavDataSource);
UrlManager.Get("DefaultPageManager").InitializeFrom(Request.Url);
```

Building a URL to an aggregate record details page

You can build a URL for an aggregate record details page by calling the convenience method `Url¬Builder.SelectAggregateRecord`. This method requires an aggregate record details data source ID and an aggregate record object as input. You can build the URLthat links to the current page of an application, for example, `Default.aspx`, or you can build a URL that links to another page such as `AggRecordDetails.aspx` by setting the page name in the `UrlManager.BaseUrl` property.

The example below requires the following statements at the top of your code behind:

```
using System;
using Endeca.Data;
using Endeca.Web.UI;
using RadAspNetRef;
using Endeca.Web.Url;
```

This procedure also assumes you have already created data sources or commands in your application and assigned them an ID.

To build a URL to an aggregate record details page:

1. Create a repeater control and specify `AggregateRecordsResult` as the `DataMember` property.

   Note: You are not required to create a repeater to call `SelectAggregateRecord`. Also, steps 1 and 2 are in the procedure to provide context for a typical use case. If you do not need this context, you can skip to step 3.

2. Create a nested repeater control and specify the data source, in this case, `AggregateRecords`.
3. Create a hyperlink and in a data binding expression, assign `UrlManager.SelectAggregateRecord` to the `NavigateUrl` property.
4. Provide `SelectAggregateRecord` with input parameters that specify the ID of the aggregate record details data source, the navigation data source, and the Endeca aggregate record to build the URL for.
   See the first example below for the markup.
5. In the code behind for the markup, do the following:
   a) If you want to build a URL to a page different from the current the page, specify a relative path to the application root in the `UrlManager.BaseUrl` property. This property value should begin with a tilde then slash (~/) followed by a path relative to the root of the Web application.
   b) Create a `UrlManager` on the aggregate record details page indicated by `UrlManager.BaseUrl`.
   c) Register a `UrlManager` on the aggregate record details page by calling `UrlManager.Register` and provide an aggregate record details data source (or command). This call is usually done as part of page configuration during a page's `Page_Load` or `OnInit` methods.
   d) Call `InitializeFrom` and provide the `Request.Url`.

   See the second example below for the code behind.
6. Register each data source in the application by calling `Register` and specifying each data source ID.

Example of building an aggregate record details page URL

This example shows the mark up that calls `SelectAggregateRecord` to build an aggregate record details URL. Note that for the example, `aggRecDetailsDataSource.ID` and `NavDataSource.ID` is pseudo-code. Your markup should refer to an aggregate record details data source and navigation data source either as a string or by its ID.

```
<asp:Repeater ID="Repeater3" runat="server" DataMember="AggregateRecordsRe¬
sult"
   DataSourceID="NavDataSource">
   <ItemTemplate>
     <asp:Repeater ID="rptRecord" runat="server" DataSource='<%#Eval("Ag¬
gregateRecords") %>'>
         <ItemTemplate>
             <div>
                <asp:HyperLink ID="HyperLink1" NavigateUrl='<%# UrlManag¬
er.Get
                 ("aggRecDetailsUrlManager").Urls.SelectAggregateRecord
                 (aggRecDetailsDataSource.ID,NavDataSource.ID, Contain¬
er.DataItem)
                 %>' runat="server">
                     <%# GetRecordTitle(Container.DataItem) %>
                </asp:HyperLink>
             </div>
          </ItemTemplate>
```

```
      </asp:Repeater>
   </ItemTemplate>
</asp:Repeater>
```

This example shows the code behind that supports the mark up. This is in the page's `Page_Load` method. Note the use of `BaseUrl` that builds the link on the `RecordDetails.aspx` page.

```
aggRecDetailsDataSource = new AggregateRecordDetailsDataSource();
aggRecDetailsDataSource.ID = "aggRecDetailsDataSource";

UrlManager.Get("aggRecDetailsUrlManager").BaseUrl =
   "~/AggRecords/AggRecordDetails.aspx";
UrlManager.Get("aggRecDetailsUrlManager").Register(NavDataSource);
UrlManager.Get("aggRecDetailsUrlManager").Register(aggRecDetailsDataSource);

UrlManager.Get("DefaultPageManager").Register(NavDataSource);
UrlManager.Get("DefaultPageManager ").InitializeFrom(Request.Url);
```

Building a URL using the BuildUrl method

In addition to the convenience methods in `UrlBuilder` that build URLs, such as `SelectParentDi¬mensionValue`, `SelectDimensionValue`, `SelectRecord`, and so on, you can also use `Url¬Builder.BuildUrl` method to construct a URL. This method provides a flexible way to customize URLs if the convenience methods are not suitable.

The example below requires the following statements at the top of your code:

```
<%@ Register TagPrefix="end" Assembly="Endeca.Web"
   Namespace="Endeca.Web.UI"  %>
<%@ Register TagPrefix="end" Assembly="Endeca.Web.UI.WebControls"
      Namespace="Endeca.Web.UI"  %>
<%@ Import Namespace="Endeca.Data" %>
<%@ Import Namespace="Endeca.Web.Url" %>
```

This procedure assumes you have already created data sources or commands in your application and assigned them an ID.

The `UrlBuilder.BuildUrl` method takes a delegate that takes a `CommandActionInfo`. You write the body of the delegate to copy an `EndecaCommand`, using `CommandActionInfo.Get<T>` and then modify the copy so it can serve as the basis for a new URL.

To build a URL using `BuildUrl`:

1. Add code that calls `BuildUrl` on the `UrlManager`.
2. Create a delegate that takes `CommandActionInfo`.
3. In the delegate, add code to copy an `EndecaCommand` by calling `Get` and then specifying a data source ID as input.

   This copy becomes the basis of a new URL.
4. Add additional code to configure the URL as desired. There is a large amount of potential configuration you could perform at this point to build a URL. It may be useful to browse the examples in the reference application for typical use cases.
5. Optionally, add parameters to a URL. See the "Adding parameters ..." topics.

Example building a URL

This example shows the markup for a URL that clears all the dimensions and dimension values when an end user clicks **Reset**.

```
<a
   href="<%= UrlManager.Default.Urls.BuildUrl(delegate(CommandActionInfo
i) {
     NavigationCommand cmd = i.Get<NavigationCommand>( dsRight.ID );
     cmd.RefinementConfigs.Clear();
     cmd.SelectedDimensionValueIds.Clear(); }) %>">
   Reset
</a>
```

Adding parameters to all URLs produced by a UrlManager

You can add parameters to all the URLs produced by a particular instance of a `UrlManager`. This approach provides a way to add parameters to URLs produced by an instance of a `UrlManager` rather than adding parameters on a per-URL basis using the `UrlBuilder.BuildUrl`. The manner in which these parameters are encoded in the URL depends on the `UrlProvider` implementation associated with the `UrlManager`.

The example below requires the following statements in your code:

```
using System;
using Endeca.Data;
using Endeca.Web.UI;
using RadAspNetRef;
using Endeca.Web.Url;
```

This procedure also assumes you have already created data sources or commands in your application and assigned them an ID.

To add parameters to all URLs produced by a `UrlManager`:

1. Create a `UrlManager` and register data sources or commands with it. For details, see "Creating a UrlManager and registering data sources and commands".
2. Initialize the `UrlManager`. For details, see "Initializing a UrlManager".
3. Call the Add method and provide a string specifying the key-value pair for the parameter you want to add. Repeat for any additional parameters you want the `UrlManager` to create.

Example of adding parameters to all URLs

This example shows the code to check if a URL has a `SessionID` parameter. If not, it adds the `SessionID` to all URLs produced by the `UrlManager`. This code is in the page's `Page_Load` method.

```
// The API provides a convenience method called Default
// to create a new instance of UrlManager named "Default".
// This creates a UrlManager and registers two data sources
// with it.
UrlManager.Default.Register(dsLeft);
UrlManager.Default.Register(dsRight);

// Provide the UrlManager with state from the Request.Url.
UrlManager.Default.InitializeFrom(Request.Url);

// Logic to check if the SessionID parameter is
```

```
// in the collection of UrlManager parameters.
// If not, add a SessionID parameter to the UrlManager.
UrlManager.Default.Parameters[SessionID] = 1234;
```

This `BasicUrlProvider` produces a URL that encodes the contains the `SessionID` parameter as a query string parameter, as shown:

```
http://localhost:3932/RadAspNetRef/Url.aspx
?dsLeft=Nrc:id-7&amp;SessionID=1234
```

Adding parameters to a URL on a per-URL basis

You can add parameters on a per-URL basis by calling `UrlBuilder.BuildUrl` and specifying one or more key-value parameters as input.

The example below requires the following statements in your code:

```
<%@ Register TagPrefix="end" Assembly="Endeca.Web"
   Namespace="Endeca.Web.UI"  %>
<%@ Register TagPrefix="end" Assembly="Endeca.Web.UI.WebControls"
      Namespace="Endeca.Web.UI"  %>
<%@ Import Namespace="Endeca.Data" %>
<%@ Import Namespace="Endeca.Web.Url" %>
```

This procedure assumes you have already created data sources or commands in your application and assigned them an ID.

To add parameters to a URL on a per-URL basis:

1. Create a link object that calls `BuildUrl`.
   The example below does this in the markup rather than code behind.
2. Call the `Add` method and provide a string specifying the key-value pair for the parameter you want to add. Repeat for any additional parameters you want `BuildUrl` to create.

Example of adding parameters to a single URL

This example shows the code to add the `keystring` parameter to the URL produced by clicking on `LinkName`.

```
href="<%= UrlManager.Default.Urls.BuildUrl(delegate(CommandActionInfo i)
  {
  i.Parameters.[keystring]="valuestring";
  } ) %>">
LinkName
```

This code produces a URL for `LinkName` that contains the `keystring` parameter, as shown:

```
http://localhost:port/RadAspNetRef/Url.aspx?keystring=valuestring
```

Chapter 6

Using the RAD API for programmatic querying

This section provides examples of how to use the RAD API to programmatically query an MDEX Engine using all of the basic query types. The examples build each type of `Command` object and execute the command. The MDEX Engine returns a `NavigationResult` object to your application.

Executing a navigation command

This example code connects to an MDEX Engine, creates and executes a `NavigationCommand`, and examines various ways that the output types can be used. Note that the `NavigationCommand` object is the most efficient command to be used when seeking multiple result collections from a query.

This example requires 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;
```

To build a `NavigationCommand`:

Add code similar to following example:

```
// A PresentationApiConnection is an EndecaConnection that uses the
// legacy 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();
```

```
/// Example using Records output types
RecordsResult recordsRes = res.RecordsResult;

// How many total records there are for this navigation state
long totalNumberOfRecords = recordsRes.TotalRecordCount;
ReadOnlyCollection<Record> records = recordsRes.Records;
// loop through each record and get some variables that we might want to
 output
foreach (Record r in records)
{
   string name = r["P_Name"]; // get string value for property P_Name
   string vintage = r["Vintage"]; // works for dimensions (Vintage), too


   // This collection has all dimension values for the Flavors dimension

   ReadOnlyCollection<DimensionValue> dimvals = r.DimensionValues.ByDi¬
mensionName["Flavors"];
   foreach (DimensionValue dv in dimvals)
   {
      string dimensionValueName = dv.Name; // the name of each flavor

      // In order to select one of the DimensionValues simply add it to
 a NavigationCommand
      // cmd.SelectDimensionValue(dv);
   }
}

///
/// Example using DimensionState output types
///
DimensionStatesResult dimensionRes = res.DimensionStatesResult;
// IndexedCollection is just like a normal collection except it has lookup
 by Name
// or Id RefinableDimensionStates are DimensionStates that have refinements

// and are neither fully selected or fully implicit
IndexedCollection<DimensionState> dimensionStates = dimensionRes.Refin¬
ableDimensionStates;
foreach (DimensionState d in dimensionStates)
{
   string dimensionName = d.Dimension.DisplayName;

  // loop through available refinement dimension values for this dimension
 state
   // note that choices are only available if the dimension is expanded
   foreach (DimensionValue dv in d.Refinements)
   {
      string dimensionValueName = dv.Name;

      // In order to select one of the DimensionValues simply add it to
 a
      // NavigationCommand
      cmd.SelectDimensionValue(dv);
   }
}

///
/// Example using applied filters to output selected dimension values as
 breadcrumbs
```

```
///
AppliedFiltersResult filtersResult = res.AppliedFiltersResult;
// for this example, we only care about selected dimension values
DimensionValueCollection selectedDimensionValues = filtersResult.Dimen¬
sionValues;
// each dimension value knows its path from the root dimension value to
 itself
foreach (DimensionValue dimval in selectedDimensionValues)
{
   // for each dimension value, we'll use its path to get out the variables
 needed to make a
   // breadcrumb with [Remove] on the left that will remove the whole
breadcrumb,
   // and links under each dimension value in the middle to navigate back

   // up the hierachy to that location.
   // For example: "[Remove]  Wine Type > __Red__ > Beaujolais
   DimensionValuePath dimvalPath = dimval.DimensionValuePath;

   // We can use the encoded state of the DimensionValue to perform oper¬
ations
   // on the state such as removing, adding, and selecting ancestors

   // Remove in this example removes the last dimension value in the
breadcrumb
   string removeContext = dimvalPath.Last.ContextPath;

   // The removal can be performed by calling Remove on the Navigation¬
Command
   // with either the full DimensionValue or the context path.
   // The context path is useful when you will not have the full Dimen¬
sionValue object
   // cmd.RemoveDimensionValue(removeContext);
   //  or
   // cmd.RemoveDimensionValue(dimvalPath.Last);

   string rootName = dimvalPath.First.Name;
   // the intermediate dimension values are between the root and descriptor

   foreach (DimensionValue intermediate in dimvalPath.Intermediates)
   {
      string intermediateName = intermediate.Name;
     // when you click on the intermediate dimension value you'll navigate
 there
      string selectContext = intermediate.ContextPath;
      // cmd.SelectDimensionValue(selectContext);
      //  or
      // cmd.SelectDimensionValue(intermediate);
   }
   string descriptorName = dimvalPath.Last.Name;
}
```

Executing a records detail query

This example executes a query using a record ID as input and returns the corresponding single record.

This example requires 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;
```

To connect to an MDEX Engine and execute a records detail query:

Add code similar to following example:

```
PresentationApiConnection conn = new PresentationApiConnection("localhost",
 8000);
RecordDetailsCommand r = new RecordDetailsCommand(conn);

r.Identifier = "45728";

Record rec = r.Execute();
```

Executing an aggregated records query

This example executes an aggregated records query against the MDEX Engine and returns a single `AggregateRecord`.

This example requires 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;
```

To connect to an MDEX Engine and execute an aggregated records query:

Add code similar to following example:

```
PresentationApiConnection conn = new PresentationApiConnection("localhost",
 8000);
AggregateRecordDetailsCommand cmd = new AggregateRecordDetailsCom¬
mand(conn);

cmd.AggregationKey = "P_Winery";
cmd.Identifier = "43444";

AggregateRecord arec = cmd.Execute();
```

Executing a dimension search query

This example executes a dimension search query against an MDEX Engine and returns name, ID, group, and child dimension values.

This example requires 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;
```

To connect to an MDEX Engine and execute a dimension search query:

Add code similar to following example:

```
PresentationApiConnection conn = new PresentationApiConnection("localhost",
 8000);
DimensionSearchCommand dsc = new DimensionSearchCommand(conn);

// Search for any dimension values with either any of "red" or "drink"
in them
dsc.SearchTerms = "red drink";
dsc.SearchMode = SearchMode.Any;

DimensionSearchResult dsr = dsc.Execute();

// The results are in one large dimension value collection but can be
grouped
// by dimension name or dimension id
ReadOnlyDictionary<string,ReadOnlyCollection<DimensionValue>> byDimension
 =
dsr.DimensionValues.ByDimensionName;

// You can iterate over the dictionary and print out the name of the di¬
mension
// and all the results beneath it.
foreach (KeyValuePair<string, ReadOnlyCollection<DimensionValue>> pair
in byDimension)
{
   string dimensionName = pair.Key;
   foreach (DimensionValue dval in pair.Value)
   {
      string dimensionValueName = dval.Name;
      System.Diagnostics.Debug.WriteLine(string.Format("{0} -> {1}",
        dimensionName, dimensionValueName));
   }
}
```

Executing a compound dimension search query

This example executes a dimension search query against an MDEX Engine and returns the set of dimension values that match.

This example requires 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;
```

To connect to an MDEX Engine and execute a compound dimension search query:

Add code similar to following example:

```
PresentationApiConnection conn = new PresentationApiConnection("localhost",
 8000);
```

```
CompoundDimensionSearchCommand dsc = new CompoundDimensionSearchCom¬
mand(conn);

// Search for sets of dimension values with "red" and "drink" in them
dsc.SearchTerms = "red drink";

CompoundDimensionSearchResult dsr = dsc.Execute();

// The results are DimensionValueCollections where each value matches
one of the terms in the search
ReadOnlyCollection<DimensionValueCollection> compoundDimVals = dsr.Com¬
poundDimensionValues;

foreach(DimensionValueCollection dvcol in compoundDimVals)
{
   // The contents of this loop are a single compound result
   int count = 0;
   foreach (DimensionValue dval in dvcol)
   {
      System.Diagnostics.Debug.Write(string.Format("{0} -> {1} ",
dval.Dimension.Name, dval.Name));

      if (count++ < dvcol.Count-1)
      {
         System.Diagnostics.Debug.Write(" and ");
      }
   }
   System.Diagnostics.Debug.WriteLine("");
}
```

Executing a metadata query

This example executes a metadata query and returns key properties, search keys, sort keys and roll up keys (aggregation keys).

This example requires 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;
```

To connect to an MDEX Engine and execute a metadata query:

Add code similar to following example:

```
PresentationApiConnection conn = new PresentationApiConnection("localhost",
 8000);
MetadataCommand mdc = new MetadataCommand(conn);

MetadataResult mdr = mdc.Execute();
```

Chapter 7

Integrating a preview application into Oracle Endeca Workbench

This section describes how to integrate either the RAD ASP.NET reference application or a custom application into Oracle Endeca Workbench (known as Web Studio prior to version 6.0.x). Once integrated, an application becomes the preview application that you use to test dynamic business rules.

Integrating the RAD ASP.NET reference application into Endeca Workbench

In a default Oracle Endeca Guided Search installation, the JSP reference application is incorporated into the Rule Manager page of Endeca Workbench. The JSP reference application acts as the preview application for working with dynamic business rules. You can however substitute the RAD ASP.NET reference application in place of the JSP reference application and use the RAD reference application as the preview application.

Before starting the task below, make sure your reference application is running.

You integrate the RAD ASP.NET reference application into Endeca Workbench by changing the URL mappings on the Preview App Settings page of Endeca Workbench.

Note: The URL mode and RAD Toolkit Server Controls URL modes are both supported in Endeca Workbench. The Postback and RAD Toolkit Server Controls Postback modes are not supported in Endeca Workbench.

To integrate the RAD ASP.NET reference application into Endeca Workbench:

1. Log in to Endeca Workbench, and under **Application Settings**, select the **Preview App Settings** page.
2. Replace the URL Mapping values with the following URLs:

URL Type	Example URL Mapping
Search URL	`http://hostname:port/RAD/GuidedNavigationUrlServerControls.aspx?EndecaHostName=MDEXEngineHost&EndecaPort=MDEXEnginePort&dsNav=Ntk:${key}%7c${terms}%7c1%7c,Nmpt:${previewtime},Nmrf:${rulefilter}`

URL Type	Example URL Mapping
Navigation URL	`http://hostname:port/RAD/GuidedNavigationUrlServerControls.aspx?EndecaHostName=MDEXEngineHost&EndecaPort=MDEXEnginePort&dsNav=N:${nav},Nmpt:${previewtime},Nmrf:${rulefilter}`
Search and Navigation URL	`http://hostname:port/RAD/GuidedNavigationUrlServerControls.aspx?EndecaHostName=MDEXEngineHost&EndecaPort=MDEXEnginePort&dsNav=Ntk:${key}%7c${terms}%7c1%7c,N:${nav},Nmpt:${previewtime},Nmrf:${rulefilter}`
Record URL	`http://hostname:port/RAD/GuidedNavigationUrlServerControls.aspx?EndecaHostName=MDEXEngineHost&EndecaPort=MDEXEnginePort&dsRecordDetails=R:${record}`

Notes:

- If you copy the URLs, remove the line breaks when you paste them into Endeca Workbench.
- The syntax of the URL mappings is documented in the *Oracle Endeca Workbench Help*. See "Providing or confirming preview application URLS".
- In the examples above, the directory */RAD* refers to the virtual directory name of the RAD ASP.NET reference application. The page name for the reference application can be either GuidedNavigationUrlServerControls.aspx (as in the examples) or GuidedNavigationUrl.aspx.

3. You may also need to declare the Javascript domain in GuidedNavigationUrlServerControls.aspx or GuidedNavigationUrl.aspx (depending on which page you specified in the Preview App Settings page in Endeca Workbench). For details, see the *Oracle Endeca Workbench Administrator's Guide*.
4. On the **Preview App Settings** page, click **Save Changes**.
5. Navigate to the **Rule Manager** page.

The RAD ASP.NET reference application displays in the preview application pane.

Integrating a custom application into Endeca Workbench

In a default Oracle Endeca Guided Search installation, the JSP reference application is incorporated into the Rule Manager page of Endeca Workbench. The JSP reference application acts as the preview application for working with dynamic business rules. You can however substitute your own custom application in place of the JSP reference application and use it as the preview application.

You integrate a custom application into Endeca Workbench by changing the URL mappings in Endeca Workbench, by enabling `WebStudioCommandSerializationProvider`, and by adding instrumentation server controls.

Note: The URL mode and RAD Toolkit Server Controls URL modes are both supported in Endeca Workbench. The Postback and RAD Toolkit Server Controls Postback modes are not supported in Endeca Workbench.

To integrate a custom application into Endeca Workbench:

1. Navigate to the `reference\RadAspNetRef\App_Code\RadAspNetRefUrl` subdirectory of your RAD Toolkits installation.

   For example:
   `C:\Endeca\RADToolkits\version\ASP.NET\reference\RadAspNetRef\App_Code\RadAspNetRefUrl`

2. Copy the `WebStudioUrlCommandSerializationProvider.cs` file to the `App_Code` directory of your application.
3. Open the `Web.config` file for the custom application that you want to integrate into Endeca Workbench. In the `<configSections>` element, add the following snippet:

```
<configSections>
  <sectionGroup name="endeca" type="Endeca.Data.Configuration.EndecaSec¬
tionGroup,
   Endeca.Data, Version=2.1.0.0, Culture=neutral, PublicKeyTo¬
ken=6d02be8724ca751c" >
      <section name="commandSerialization"
      type="Endeca.Data.Configuration.CommandSerializationSection,
      Endeca.Data, Version=2.1.0.0, Culture=neutral, PublicKeyTo¬
ken=6d02be8724ca751c" />
      ...
```

4. Also in the `Web.config` file, add the following snippet immediately after the `<endeca>` element:

```
<endeca>
   <commandSerialization>
      <providers>
        <add name="Url" type="RadAspNetUrlRef.WebStudioUrlCommandSerial¬
izationProvider" />
      </providers>
   </commandSerialization>

<!-- Other elements removed from the example. -->

</endeca>
```

   For an example, see the `Web.config` file for the RAD ASP.NET reference application.
5. Open the home page for your custom application, or any page in your application that requires instrumentation, and do the following:
   a) For navigation instrumentation, drag the **WebStudioNavigationInstrumentation** control on to the **Design** tab of your page.
   b) From the **Smart Tag** of the **WebStudioNavigationInstrumentation** control, select the **NavigationDataSource** data source.
   c) For record instrumentation, drag the **WebStudioRecordInstrumentation** control on to the **Design** tab of your page.
   d) From the **Smart Tag** of the **WebStudioRecordInstrumentation** control, select the **RecordDetailsDataSource** data source.
   e) In the markup for the **WebStudioRecordInstrumentation** control, modify the `RecordSpecProp¬erty` to reflect the spec ID of records in your data set, and the `RecordNameProperty` properties to reflect the name.

Note: You may also need to declare the Javascript domain for the pages that are instrumented. For details, see the *Oracle Endeca Workbench Administrator's Guide*.

6. Log in to Endeca Workbench, and under **Application Settings**, select the **Preview App Settings** page.
7. Replace the URL Mapping values with the following URLs:

URL Type	Example URL Mapping
Search URL	`http://`*`hostname:port/AppName/PageName.aspx`*`?EndecaHostName=`*`MDEX¬ EngineHost`*`&EndecaPort=`*`MDEX¬`

URL Type	Example URL Mapping
	EnginePort&dsNav=Ntk:${key}%7c${terms}%7c1%7c,Nmpt:${preview¬ time},Nmrf:${rulefilter}
Navigation URL	http://*hostname:port/AppName/PageName.aspx*?EndecaHostName=*MDEX*¬ *EngineHost*&EndecaPort=*MDEXEnginePort*&dsNav=N:${nav},Nmpt:${pre¬ viewtime},Nmrf:${rulefilter}
Search and Navigation URL	http://*hostname:port/AppName/PageName.aspx*?EndecaHostName=*MDEX*¬ *EngineHost*&EndecaPort=*MDEX*¬ *EnginePort*&dsNav=Ntk:${key}%7c${terms}%7c1%7c,N:${nav},Nmpt:${pre¬ viewtime},Nmrf:${rulefilter}
Record URL	http://*hostname:port/AppName/PageName.aspx*?EndecaHostName=*MDEX*¬ *EngineHost*&EndecaPort=*MDEXEnginePort*&dsRecordDetails=R:${record}

Notes:

- If you copy the URLs, remove the line breaks when you paste them into Endeca Workbench.
- The syntax of the URL mappings are documented in the *Oracle Endeca Workbench Help*. See "Providing or confirming preview application URLS".
- In the examples above, the directory */AppName* refers to the virtual directory name of the custom application. The file *PageName.aspx* refers to the home page of your application.

8. On the **Preview App Settings** page, click **Save Changes**.
9. Navigate to the **Rule Manager** page of Endeca Workbench.

The custom application displays in the preview application pane.

Index

S

T

U

V

W