

UI for ASP.NET AJAX

Table of Contents

1.	Introduction	1
1.1.	Important Information	1
1.2.	Who Should Read This Courseware	1
1.3.	What Do You Need To Have Before You Read This Courseware?	1
1.4.	What Do You Need To Know Before Reading This Courseware?	1-2
1.5.	How This Courseware Is Organized	2-6
1.6.	Introducing RadControls	6-13
1.7.	Before You Begin...	13
2.	Navigation Controls	14
2.1.	Objectives	14
2.2.	Introduction	14-17
2.3.	Getting Started	17-22
2.4.	Designer Interface	22-29
2.5.	Server-Side Programming	29-37
2.6.	Control Specifics	37-42
2.7.	Summary	42-43
3.	Input Controls	44
3.1.	Objectives	44
3.2.	Introduction	44-46
3.3.	Getting Started	46-49
3.4.	Designer Interface	49-57
3.5.	Server-Side Programming	57-60
3.6.	Client-Side Programming	60-64
3.7.	How To	64-69
3.8.	RadInputManager	69-79
3.9.	Summary	79
4.	Client-Side API	80
4.1.	Objectives	80

4.2. Introduction	80
4.3. Referencing RadControl Client Objects	80-82
4.4. Using RadControl Client Properties and Methods	82-83
4.5. JavaScript Intellisense	83-85
4.6. Naming Conventions	85-86
4.7. Using Client Events	86-91
4.8. Client Events Walk Through	91-95
4.9. JSON: Fat-Free Data Interchange	95-98
4.10. MS AJAX Library	98
4.11. Summary	98
5. User Interface and Information Controls	99
5.1. Objectives	99
5.2. Introduction	99-100
5.3. Getting Started	100-105
5.4. Designer Interface	105-110
5.5. Server-Side Programming	110-113
5.6. Client Side Programming	113-117
5.7. How To	117-119
5.8. Summary	119
6. RadRotator	120
6.1. Objectives	120
6.2. Introduction	120
6.3. Getting Started	120-124
6.4. Designer Interface	124-126
6.5. Client-Side Programming	126-127
6.6. Client-Side Items Management	127-128
6.7. Control Specifics	128-129
6.8. Coverflow mode	129-132
6.9. Carousel mode	132-134
6.10. Summary	134

7.	Ajax	135
7.1.	Objectives	135
7.2.	Introduction	135-136
7.3.	Getting Started	136-141
7.4.	Designer Interface	141-145
7.5.	Server-Side Programming	145-148
7.6.	Client-Side Programming	148-162
7.7.	Page vs MasterPage vs UserControl	162-163
7.8.	Page Lifecycle	163-165
7.9.	Dynamic User Controls for Ajax-Enabling Entire Page	165-174
7.10.	Using RadAjaxManagerProxy	174-176
7.11.	Summary	176
8.	ActiveSkill: Getting Started	177
8.1.	Objectives	177
8.2.	Introduction	177
8.3.	Setup ActiveSkill Project Structure	177-178
8.4.	Setting Up the Database	178-181
8.5.	ASP.NET Membership	181-188
8.6.	Create the ActiveSkill Login Page	188-192
8.7.	Create Registration Page	192-201
8.8.	Implement the Registration Page	201-202
8.9.	The CreateUserWizardWrapper Code-Behind	202-204
8.10.	The CreateUserWizardWrapperUI	204-210
8.11.	Create the Billing Control Code-Behind	210-214
8.12.	Create the BillingControl User Control	214-216
8.13.	Add Utility Classes	216-224
8.14.	Configure the Profile	224-225
8.15.	Summary	225
9.	Screen "Real Estate" Management	226
9.1.	Objectives	226

9.2. Introduction	226-231
9.3. Getting Started	231-236
9.4. Designer Interface	236-242
9.5. Control Specifics	242-246
9.6. Server-Side Programming	246-256
9.7. Client-Side Programming	256-265
9.8. How To	265-280
9.9. Summary	280
10. Skinning	281
10.1. Objectives	281
10.2. Introduction	281-282
10.3. Getting Started	282-283
10.4. Registering and Assigning Skins	283-288
10.5. Understanding the Skin CSS File	288-292
10.6. Creating a Custom Skin	292-296
10.7. Summary	296
11. Databinding	297
11.1. Objectives	297
11.2. Introduction	297-298
11.3. Getting Started	298-309
11.4. Binding Hierarchical Data	309-315
11.5. Server-Side Programming	315-322
11.6. Binding to Business Objects	322-330
11.7. Binding to Linq	330-337
11.8. Summary	337
12. Templates	338
12.1. Objectives	338
12.2. Introduction	338-340
12.3. Getting Started	340-346
12.4. Binding Expressions	346-350

12.5. Designer Interface	350-353
12.6. Server-Side Programming	353-360
12.7. Client-Side Programming	360-361
12.8. Summary	361
13. ActiveSkill: Admin Page	362
13.1. Objectives	362
13.2. Introduction	362
13.3. Build the Admin Page	362-368
13.4. Create User Controls	368-372
13.5. Create ActiveSkill Skin	372-374
13.6. Summary	374
14. RadAsyncUpload	375
14.1. Objectives	375
14.2. Introduction	375-376
14.3. Getting Started	376-379
14.4. Important Properties	379
14.5. Upload Modules	379-380
14.6. Server-Side Programming	380-381
14.7. Client-Side Programming	381-382
14.8. Summary	382
15. RadComboBox	383
15.1. Objectives	383
15.2. Introduction	383-384
15.3. Getting Started	384-386
15.4. Designer Interface	386-390
15.5. Control Specifics	390-402
15.6. Server-Side Programming	402-408
15.7. Client-Side Programming	408-414
15.8. How To	414-427
15.9. Summary	427-428

16. RadTreeView	429
16.1. Objectives	429
16.2. Introduction	429-430
16.3. Getting Started	430-432
16.4. Designer Interface	432-437
16.5. Control Specifics	437-447
16.6. Server Side Programming	447-458
16.7. Client-Side Programming	458-463
16.8. How To	463-465
16.9. Performance	465-469
16.10. Summary	469-470
17. RadFileExplorer	471
17.1. Objectives	471
17.2. Introduction	471
17.3. Getting Started	471-474
17.4. Thumbnails Mode	474-475
17.5. Server-Side Programming	475-478
17.6. Client-Side Programming	478-479
17.7. How To	479-482
17.8. Summary	482
18. RadSiteMap	483
18.1. Objectives	483
18.2. Introduction	483-484
18.3. Getting started	484-486
18.4. Designer Interface	486-491
18.5. Server Side Programming	491-492
18.6. How To	492-495
18.7. Summary	495
19. RadGrid	496
19.1. Objectives	496

19.2. Introduction	496-497
19.3. Getting Started	497-503
19.4. Using the Design Time Interface	503-514
19.5. Server Side Code	514-525
19.6. Client Side Code	525-537
19.7. Summary	537-538
19.8. Columns	538-545
19.9. Rows	545-546
20. RadEditor	547
20.1. Objectives	547
20.2. Introduction	547-548
20.3. Getting Started	548-551
20.4. Designer Interface	551-562
20.5. Using the NewLineMode Property	562-563
20.6. Customizing Content Area	563-566
20.7. Configuring the ToolsFile	566-567
20.8. RibbonBar and Editor	567-570
20.9. Server-Side Programming	570-573
20.10. Client-Side Programming	573-578
20.11. How To	578-587
20.12. Summary	587
21. RadBarcode	588
21.1. Objectives	588
21.2. Introduction	588
21.3. Barcode types	588-589
22. RadButton	590
22.1. Objectives	590
22.2. Introduction	590-591
22.3. Getting Started	591-593
22.4. Specifying RadButton Icons	593-594

22.5. RadButton as an Image Button	594-596
22.6. RadButton as a Toggle Button	596-599
22.7. Important Properties	599-601
22.8. Creating a single click button	601-602
22.9. Bigger Icons and Buttons	602
22.10. Confirm postback with RadButton	602-604
22.11. Specifying the content of a RadButton	604-605
23. RadBinaryImage	606
23.1. Objectives	606
23.2. Introduction	606-607
23.3. Getting Started	607-608
24. RadFilter	609
24.1. Objectives	609
24.2. Introduction	609
24.3. Getting Started	609-610
24.4. Events	610
24.5. Summary	610
25. RadImageEditor	611
25.1. Objectives	611
25.2. Introduction	611
25.3. Smart Tag	611-612
25.4. Getting Started	612-613
25.5. Configuring the Toolbar	613-615
25.6. Localization	615-616
25.7. Creating a Custom Tool	616-617
25.8. Save a Thumbnail	617-620
26. RadListView	621
26.1. Objectives	621
26.2. Introduction	621
26.3. Getting Started	621-625

26.4. Using the design Time Interface	625-628
26.5. Server Side Code	628-637
26.6. RadDataPager	637-639
26.7. Summary	639
27. RadNotification	640
27.1. Objectives	640
27.2. Introduction	640
27.3. Getting Started	640-642
27.4. Notification Menu	642-643
27.5. Embedded Icons	643
27.6. Different Ways to Show A Notification	643-645
27.7. Populating Plain Text And Rich Content	645-647
27.8. Callback Support	647-648
27.9. How To Combine Properties	648-649
27.10. Auto Save RadEditor's content and notify the user	649-650
28. RadCompression	651
28.1. Objectives	651
28.2. Introduction	651
28.3. Using RadCompression	651-652
28.4. Summary	652-653
29. RadCaptcha	654
29.1. Objectives	654
29.2. Introduction	654-655
29.3. Getting Started	655-657
29.4. Important Properties	657-658
29.5. Optimize for Maximum Security	658
29.6. Configure RadCaptcha audio	658-660
30. RadXmlHttpPanel	661
30.1. Objectives	661
30.2. Introduction and Overview	661

30.3. Supported Scenarios	661-662
30.4. Configuring the XmlHttpPanel	662-667
30.5. Client-Side Programming	667-671
30.6. Server-Side Programming	671
30.7. Known Issues	671
31. RadTagCloud	672
31.1. Objectives	672
31.2. Introduction	672
31.3. Getting Started	672-674
31.4. Important Properties	674-675
31.5. Databinding	675-678
31.6. Filtering and Sorting of the TagCloud Items	678-679
31.7. Generating TagCloud from External Sources	679
31.8. Client-Side Data Binding	679-687
32. RadRating	688
32.1. Objectives	688
32.2. Introduction	688
32.3. Getting Started	688-689
32.4. Server-Side Programming	689-691
32.5. Client-Side Programming	691-692
32.6. Summary	692
33. RadRibbonBar	693
33.1. Objectives	693
33.2. Introduction	693-695
33.3. Getting Started	695-696
33.4. Server-Side Programming	696-697
33.5. Client-Side Programming	697-698
33.6. How -to	698-699
33.7. Summary	699
34. RadOrgChart	700

34.1. Objectives	700
34.2. Introduction	700-701
34.3. Getting Started	701-702
34.4. Control Specifics	702-703
34.5. Server-Side Programming	703
34.6. How-to	703
34.7. Summary	703-704
35. RadPivotGrid	705
35.1. Objectives	705
35.2. Introduction	705
35.3. Getting Started	705-706
35.4. RadPivotGrid Fields	706-710
35.5. Summary	710
36. RadSocialShare	711
36.1. Objectives	711
36.2. Introduction	711
36.3. Button Types And Button Collections	711-712
36.4. Important Properties	712
36.5. Using The Configurator	712-714
36.6. First Steps	714-717
36.7. Controlling the URL and the Title	717-718
36.8. Using Third Party Buttons	718-719
37. RadTreeList	720
37.1. Objectives	720
37.2. Introduction	720
37.3. Getting-Started	720-729
37.4. Using the design-time interface	729-735
37.5. Data Editing	735-736
37.6. Appearance and Styling	736-739
37.7. Summary	739

37.8. Scrolling	739-740
37.9. Items Drag and Drop	740-741
37.10. Load On Demand	741-742
37.11. Columns	742-744
38. ActiveSkill: Database Maintenance	745
38.1. Objectives	745
38.2. Introduction	745-747
38.3. Building the Categories Tree Control	747-758
38.4. Implement Categories Control	758-776
38.5. Implement Questions Control	776-789
38.6. Implement CreateExams Control	789-803
38.7. Summary	803-804
39. ActiveSkill: User Functionality	805
39.1. Objectives	805
39.2. Build the User Home Page	805-812
39.3. Build the Choose Exam Control	812-816
39.4. Build the Exam Question Control	816-838
39.5. Summary	838
40. RadChart	839
40.1. Objectives	839
40.2. Introduction	839
40.3. Getting Started	839-857
40.4. Designer Interface	857-865
40.5. Control Specifics	865-867
40.6. Server-Side Programming	867-882
40.7. Client-Side Programming	882-886
40.8. How To	886-890
40.9. Summary	890
41. RadHtmlChart	891
41.1. Objectives	891

41.2. Introduction	891
41.3. Getting Started	891-896
41.4. Chart Types	896-900
41.5. Databinding	900-902
42. ActiveSkill: Building the Exam Finish Control	903
42.1. Objectives	903
42.2. Building the Exam Finish Page	903-913
42.3. Summary	913
43. Date, Time, Calendar and Scheduling	915
43.1. Objectives	915
43.2. Date-Time and Calendar Controls Getting Started	915-918
43.3. Tour of Date-Time and Calendar Controls	918-919
43.4. Date-Time and Calendar Controls Designer Interface	919-928
43.5. Date-Time and Calendar Controls Server-Side Programming	928-933
43.6. Date-Time and Calendar Controls Server-Side Walk-through	933-936
43.7. Date-Time Picker Validation	936-938
43.8. Date-Time and Calendar Controls Client-Side Programming	938-944
43.9. Getting Started with RadScheduler	944-955
43.10. Scheduler Resources	955-959
43.11. Custom Attributes	959
43.12. Scheduler Designer Interface	959-963
43.13. Scheduler Server-Side Programming	963-966
43.14. Scheduler Server-Side Events	966-972
43.15. Scheduler Client-Side Programming	972-978
43.16. Using Scheduler Templates	978-981
43.17. Summary	981-982
44. ActiveSkill: Exam Scheduling	983
44.1. Objectives	983
44.2. Defining the Markup	983-986
44.3. Handling the Drag and Drop Client-Side	986-987

44.4. Handle Server-Side Events	987-991
44.5. Integrate the Exam Scheduler	991
44.6. Summary	991

1 Introduction

1.1 Important Information

This courseware is for .NET developers who are starting their journey with Telerik UI for ASP.NET AJAX. It contains step-by-step instructions on how to use the Telerik controls and how to create applications.

 NOTE: The courseware was last updated on 3/28/2009. It may contain outdated information.

Most of the information from the courseware has already been transferred to the documentation in the form of Getting Started articles. The **documentation** (<http://www.telerik.com/help/aspnet-ajax/introduction.html>) is the most complete and up to date place to find information about Telerik UI for ASP.NET AJAX. We urge you to check them out. In addition, you can also go through the **live demos** (<http://demos.telerik.com/aspnet-ajax/>) to see the controls in action.

If you encounter issues with this courseware or its related projects, feel free to **submit a support ticket** (<http://www.telerik.com/account/support-tickets/available-support-list.aspx>) and explain the problem you have encountered. We will do our best to help you.

1.2 Who Should Read This Courseware

You should read this courseware if:

- You have never used AJAX or any of the Microsoft AJAX controls and want to learn what it's all about.
- You have used AJAX or some kind of AJAX based controls and want to learn the Telerik approach using RadControls for ASP.NET AJAX.
- You have used previous versions of RadControls and want to learn how to use RadControls for ASP.NET AJAX.
- You have used RadControls for ASP.NET AJAX and want to make your knowledge more comprehensive.

1.3 What Do You Need To Have Before You Read This Courseware?

Computer Setup

- Windows XP Professional or newer
- Microsoft .NET Framework 3.5
- Internet Information Services 5+
- Internet Explorer 7+
- Microsoft Visual Studio 2010
- Microsoft SQL Server Express or Microsoft SQL Server 2005 or above.
- RadControls for ASP.NET AJAX. You can purchase RadControls for ASP.NET AJAX from:

<http://www.telerik.com/purchase/purchase-online.aspx>

or download the trial from:

<http://www.telerik.com/products/aspnet-ajax/download.aspx>

Learn more about **system requirements for RadControls for ASP.NET AJAX** here (<http://www.telerik.com/products/aspnet-ajax/system-requirements.aspx>).

1.4 What Do You Need To Know Before Reading This Courseware?

This courseware assumes that you are familiar with ASP.NET using either VB.NET or C# code. You will also need a basic understanding of the differences between server and client code. The courseware uses Visual Studio 2008 and assumes you know your way around this environment. You should be able to navigate the basic functional areas of the IDE (e.g. Solution Explorer, Properties, design/source for web pages, etc.) and be able to run and debug web applications.

1.5 How This Courseware Is Organized

Courseware Chapter Organization

The courseware chapters fall into these categories:

- The courseware has chapters on groups of RadControls where there are similarities between the controls. For example, all of the navigation controls are more alike than different when it comes to the API and the design-time environment. This allows you to leverage a common set of skills between controls.
- There are separate chapters for controls that don't fit together in a category with other controls, or are larger and more involved, such as the grid, editor or chart controls.
- We have also added steps on how to create a demonstration application "ActiveSkill". These chapters leverage your knowledge from preceding sections to see how the controls are used together in a closer-to-real-world setting. ActiveSkill is quite a bit smaller than a production application, but also larger than your typical demo application that may only use one or two controls at a time.

Each chapter contains:

- A list of the objectives to be accomplished in the chapter.
- A brief introduction to orientate you to the "why and where" each control should be used.
- A "Getting Started" tutorial to get your feet wet with the control.
- A tour of the design-time interface and a brief overview of significant control properties or groups of properties.
- A guide to the server-side capabilities of the control with the focus on important properties, collections and methods.
- A review of the client-side API that demonstrates how to get references to the control's client object, methods and events.
- A brief review of the objectives that were accomplished.

The "ActiveSkill" chapters will only have the objectives and summary. The body of these chapters will be the steps to build the ActiveSkill application.

Chapter Summary

Navigation Controls

This chapter tours "navigation" related RadControls so you can become familiar with how and where each of these controls are used. You will see some of the important properties, methods and events that are common between navigation controls. You will create a simple application that uses the menu, tab strip and tool bar controls. This chapter shows common server-side tasks such as add/edit/delete, iterating items in a collection and locating items based on various criteria (i.e. text, value or attribute). This chapter also shows some control-specific tasks such as working with the tab strip and Multi-Page together and using the context menu.

Input Controls

This chapter tours "input" related RadControls. The chapter shows significant properties and notes common properties shared by input controls. You will build a simple application that uses all four types of input control and makes use of common properties such as labels and empty messages. You will learn how to use the server-side API to respond to user input and to create input controls dynamically. The chapter demonstrates how to

perform common client-side tasks such as enabling and disabling some controls based on the responses to others, restricting input as the user types, and handling parsing errors. The chapter also shows how to use input controls with other controls such as an ASP.NET validator or RadSpellCheck.

Client-Side API

This chapter demonstrates basic techniques used to obtain RadControl object references in client code, how to call client methods and use properties of the client objects. You will learn the consistent naming conventions used throughout the RadControls client API so that you can re-apply that knowledge on new controls. The chapter shows how to implement client side event handlers and how to add and remove event handlers on-the-fly. Finally, you will put your knowledge to work by building a tabbed interface that displays a breadcrumb trail as the mouse hovers each tab.

RadRotator

This chapter explores the RadRotator control and some of the ways it can display a stream of changing content. You will become familiar with significant properties for configuring the rotator and will create a simple application displaying data taken from an XML file. The chapter demonstrates how to start and stop the rotator using the client-side api. The chapter also shows how to add items explicitly when the rotator is not bound to a data source.

User Interface and Information Controls

This chapter tours the user interface and information controls RadFormDecorator, RadToolTipManager, and RadToolTip. You will create a simple application that demonstrates how these controls change the look-and-feel of standard ASP.NET controls and tool tips. You will become familiar with the design-time support for these controls and will review their most important properties. This chapter demonstrates how the server-side API supplies content for customized tool tips. You will learn how the client-side API handles tool tip visibility and work with client properties to perform other functions in your Web pages. The chapter also shows how to add client-side IDs to an image map so that it can be used with RadToolTip.

AJAX

In this chapter we take a tour of the AJAX related RadControls, paying particular attention to the powerful and flexible RadAjaxManager. You will build a simple AJAX-enabled application that first uses RadAjaxPanel, then substitute RadAjaxManager to see how the two mechanisms contrast. You will also leverage RadAjaxLoadingPanel to provide better user feedback during AJAX requests.

You will learn how to define AJAX settings programmatically at run-time and at design-time using the RadAjaxManager Property Builder dialog to configure settings. Later you will use RadAjaxManagerProxy to perform the same settings configuration within a user control.

In this chapter you will build an application that "deals" cards to demonstrate how AJAX requests can be triggered on the client and handled on the server. You will code client-only functions to access common RadAjaxManager properties, e.g. configuration settings, enabling AJAX, canceling requests. You will also handle RadAjaxManager client events that let you set and restore state at the beginning and conclusion of AJAX requests.

The chapter also looks at design decisions regarding AJAX-enabling applications. In the process we will take a walk through the ASP.NET page lifecycle and its impact on dynamically created user controls, and finally put this information to use in a Winform-like UI demonstrating dynamic user controls together with AJAX.

You will see how RadAjaxManagerProxy provides visibility to RadAjaxManager settings in complex container-ship scenarios.

Finally, you will see how RadScriptBlock and RadCodeBlock handle common script + markup related issues.

Screen "Real Estate" Management

This chapter introduces the "real estate" management controls, showing how they can help organize web page content into flexible content areas that can be moved, resized, or hidden. You will create an application that

uses dock zones and dock windows, a splitter, and some pop-up windows managed by a window manager. You will also create simple applications to become familiar with minimize zones and sliding zones.

This chapter demonstrates how to use the server-side API with the RadDock family of controls, adding content in the code-behind, implementing custom commands, and preserving dock layout in a cookie. You will learn how to perform common client-side tasks such as responding to layout changes, implementing custom commands, manipulating windows, printing the panes of a splitter, and using the customizable alert, confirm, and prompt dialogs.

Finally, you will learn techniques that are important to some of the more common applications that use the "real estate" management controls, including implementing tool windows and modal dialogs, creating a desktop-like window by filling the entire Web page with a splitter, and creating dockable windows dynamically.

Skinning

Learn how to use built-in skins to provide a coherent, consistent style to your applications. The chapter explores the general makeup of the skin CSS files and how the styles interact with the controls rendered in the browser. You will learn multiple techniques for registering and assigning skins. You can use the included pre-defined skins for these controls, or design your own skins for a completely custom look.

Databinding

This chapter introduces the interfaces that RadControls can bind to and the task specific Data Source controls that can be used to bind declaratively. You will build a simple declarative data binding example using RadToolBar with SqlDataSource. This chapter covers in more detail how the data binding properties are used and how to bind to multiple data sources at one time.

In server-side code you see how simple arrays and lists, hierarchical data, business objects and LINQ data are bound. You will also handle data binding related server events.

Templates

This chapter shows the general techniques for working with templates as used by RadControls. First you will build a simple application that uses templates and data binding to elements within the templates. We will explore the details of binding expressions, starting with public server methods and working through Container, DataItem, Eval() and Bind() methods. You will also learn how to find controls within templates using both server and client code.

RadComboBox

This chapter examines the RadComboBox control and the powerful features it provides. You will create a simple application that populates one combo box with statically declared items and another with items loaded from a data source.

The chapter will review the design time support for the combo box and explore many of the properties and groups of properties you can use to configure the combo box at design time. You will learn about the different types of templates you can use with a combo box, and how to work with combo box custom attributes. You will also learn about the load-on-demand mechanism and how it can be used with virtual scrolling or a "More Results" box to improve performance.

The chapter reviews some of the server-side properties and methods, especially those for working with the items in the drop-down list. You will look at some of the important server-side events, such as responding to selected text changes or that service the load-on-demand mechanism. The chapter also covers when and how to sort the drop-down list in server-side code.

You will explore some of the client-side methods for working with the items collection, and use important client-side events, including those for responding to selection changes, opening and closing the drop-down list, and the events surrounding the load-on-demand mechanism.

Finally, you will learn some advanced techniques, including implementing custom sort criteria, keeping the drop-down list open when an item template includes input controls, controlling when the load-on-demand mechanism fetches items, enabling virtual scrolling when not allowing custom text, and creating a Web service

for loading items on demand.

RadTreeView

This chapter reviews the very useful RadTreeView control and how you can add the functionality of a desktop TreeView to your Web applications. You will create a simple application that populates a tree view with statically declared items and another with items loaded from a data source. In the process you will become familiar with important tree view and tree node properties.

We will look at design time support for the tree view and review many of the properties and groups of properties you can use to configure the tree view and its nodes at design time. You will discover how to use special features of RadTreeView, including node editing, check boxes, drag-and-drop, and node context menus.

You will learn some of the server-side properties and methods, and will learn how to propagate changes to all ancestors or descendants of a node. You will build a node hierarchy dynamically in server-side code, and see how this can be used to populate a tree view with data from multiple tables. You will also learn about several of the tree view server-side events.

This chapter explores client-side methods for working with the tree node and tree view objects, how to implement the 'radio button' pattern for state changes on nodes, and how to attach an event handler directly to the tree view's DOM object when the tree view first loads. This chapter also shows a few "tricks" for working with the tree view, such as getting the text of nodes to wrap and how to add controls directly to tree nodes without using templates.

Finally, you will see how the load-on-demand feature improves performance for large tree views, expanding nodes using either a postback, a callback, or a Web Service.

RadGrid

This chapter explores the versatile and powerful RadGrid control. You will create a simple application that binds the grid to live data and manipulates the auto-generated columns. You will also explore the most fundamental features of the RadGrid such as Sorting, Filtering, Grouping and Paging.

You worked with an example of implementing add, edit and delete operations manually in server-side code. You will learn how to access data values and manipulate the appearance of a column in server-side code, implement powerful new client-side databinding feature of the RadGrid and finally, use advanced client-side coding techniques, including accessing data values, manipulating appearance and binding to client-side events to make a responsive and flashy interface.

RadEditor

In this chapter we explore RadEditor's rich feature set, learn how to configure RadEditor for the runtime environment and look at the editor's design-time interface. You will learn how to manipulate RadEditor using client-side code including how to reference the editor, the document and the current selection, as well as responding to editor client events. Finally, you will learn some of the editor's customization possibilities, how to optimize RadEditor for multiple instances and how to localize RadEditor for a specific language.

RadChart

This chapter explores the rich functionality and data presentation capabilities of the innovative RadChart control. In this chapter you will build a simple chart with static items and also learn how to bind data to the chart. We will take a tour of the basic RadChart elements as well as the types of charts that are available. You will use the tools in the designer to help navigate the many RadChart capabilities. You will also learn about some of the latest RadChart features, including zooming and scrolling. You will create and configure many of the chart elements programmatically, including the chart series, items, legend and chart title. You will also learn how to bind to database data and respond to events on the server side.

Date, Time, Calendar and Scheduling

This chapter explores the features of the date/time picker, the calendar and the scheduler controls. You will create some simple applications to become familiar with the controls, review their design time interfaces and

use the server-side API to work with the major objects that make up each control. In particular, we will set calendar special days, add scheduler appointments, add scheduler resources, schedule recurrence and handle client-side events. You will also learn how to validate date and time picker control entries and how to use scheduler templates.

Building ActiveSkill - Chapter Summary

Getting Started

In this chapter you will build the initial framework for a demonstration application that uses many of the RadControls for ASP.NET AJAX. You will set up the project structure, learn how to set up and use ASP.NET Membership and finally use RadFormDecorator and RadInput controls.

Building the Admin Page

In this chapter we build the Admin Home page, starting with the general layout and adding the code-behind required to swap user controls dynamically. We will create each of the user controls and test the dynamic swapping behavior. Finally, we will create a new custom skin (called ActiveSkill) that is based on the standard Telerik "Black" skin and then configure the application to use that skin.

Database Maintenance

In this chapter you will build maintenance functionality for categories, questions and exam related tables. You will use RadGrid heavily to leverage its powerful CRUD handling abilities, creating both master-detail in a single grid and in two related grids. You will use RadControls within a standard ASP.NET FormView along with Eval() and Bind() binding expressions. You will also build a user control that combines RadComboBox with RadTreeView for reuse throughout the application.

User Functionality

In this chapter you will build functionality for the central purpose of the application, the taking of exams. The work is heavily weighted to the client where you will consume a web service to bring back the exam data, use your own JavaScript objects to encapsulate the exam, navigate through the exam and summarize the exam results. You will bind a client exam responses object directly to the RadGrid using client code only. You will also use LINQ to SQL within the web service to consume Exam database data.

Building the Exam Finish Control

In this chapter you will implement the "finish" page of ActiveSkill. This page will display the test results and a chart showing results by question category. In the process you will learn how about serializing JSON and passing JSON between client and server. You will add HTML controls to display exam results and also add and configure a RadChart. You will bind the RadChart to a generic List of objects and display the data in a stacked bar format with two series of data.

Exam Scheduling

In this chapter you will implement the scheduling for ActiveSkill. You will learn how to configure RadTreeView and RadScheduler for drag and drop, how to handle scheduler events to create new appointments and modify the attributes of existing appointments based on commands set within the appointment template. You will also learn how to format appointments as they are created based on the logged in user role and appointment attribute data.

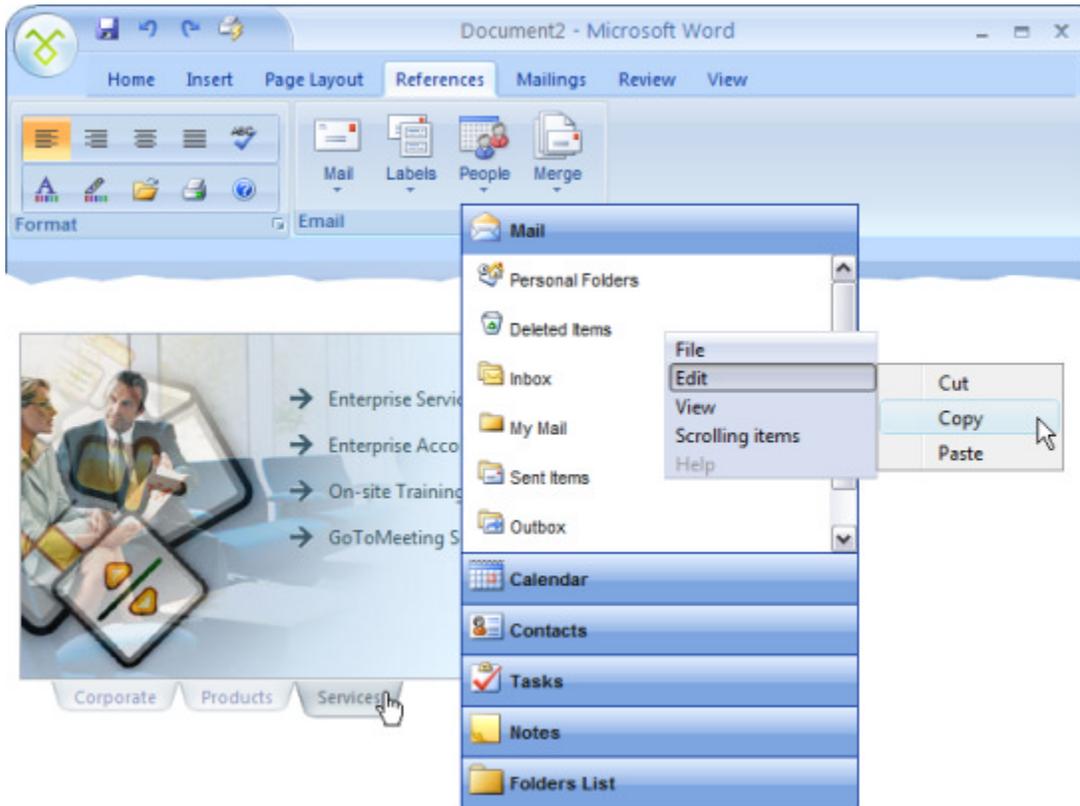
1.6 Introducing RadControls

Navigation Controls

RadControls for ASP.NET AJAX comes with a full set of powerful, flexible controls that help you express your user interface with tab strips, tool bars, menus, panel bar and the ability to combine these into the Office

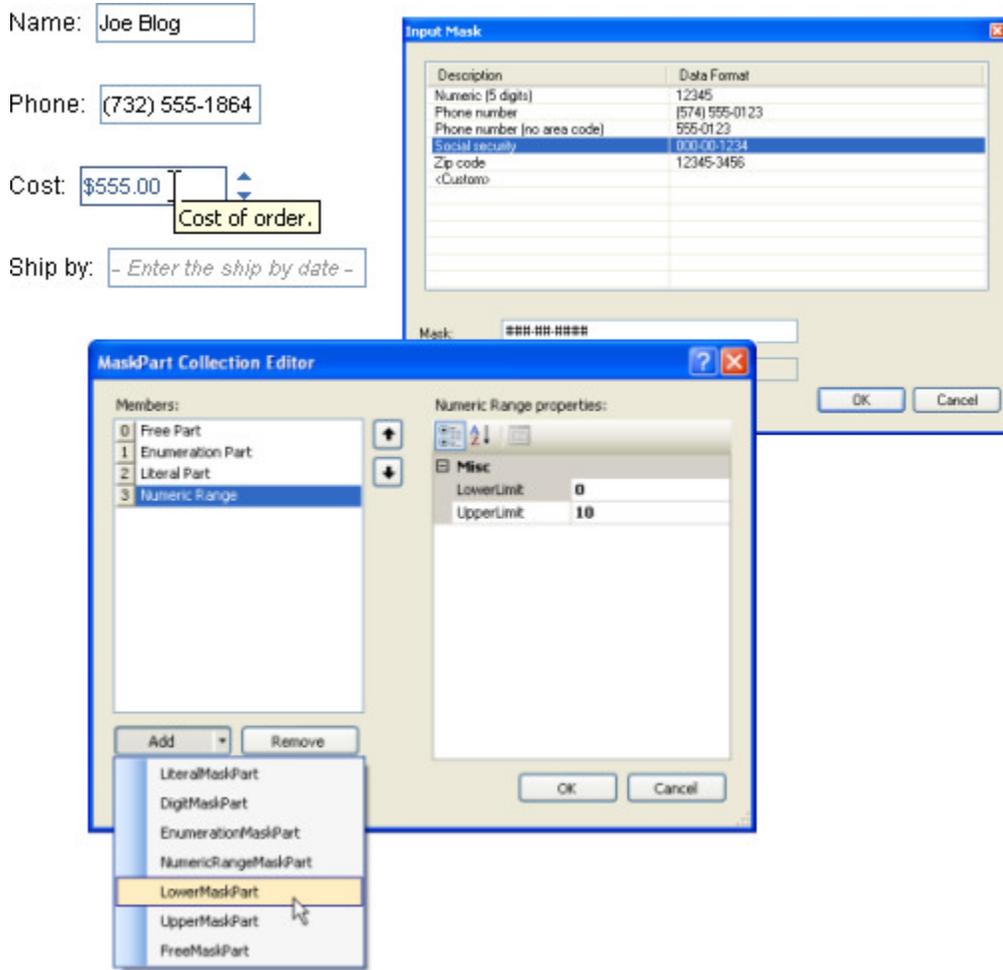
UI for ASP.NET AJAX

"Ribbon Bar" style interface. These controls let your user navigate throughout your site and to trigger your custom server or client-side code. As with all the RadControls, these controls come with a set of pre-defined skins or design your own skin for a completely custom look.



Input Controls

The input controls make it easy to collect information from users, whether it is generic text or typed data such as numbers and dates. You can choose from several types of input controls, RadTextBox, RadMaskedTextBox, RadNumericTextBox and RadDateInput. Extensive support for built-in and custom masks make it easier for your user to make valid entries.



User Interface and Information Controls

RadFormDecorator and RadToolTip let you extend the skin-based look and feel to standard ASP.NET elements such as check boxes, radio buttons, command buttons and tool tips.

Music Styles

Media

- Classical
- Classic Rock
- Jazz and Fusion
- Rhythm and Blues

- CD
- Tape

See Listing



RadRotator lets you display and scroll images and data vertically or horizontally, either as a continuous stream

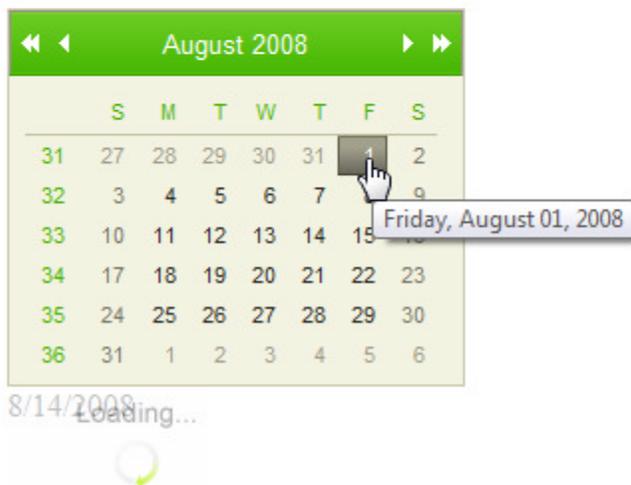
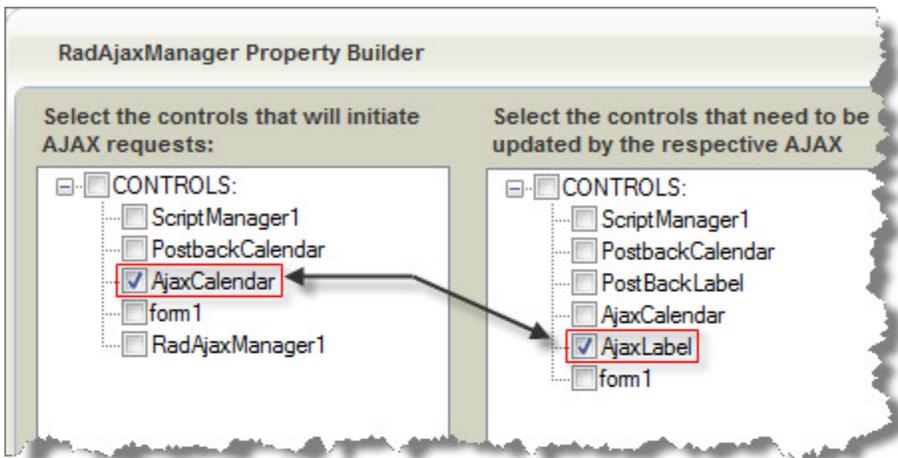
UI for ASP.NET AJAX

or as a slide show. Because this control works with templates, you have complete flexibility and control over the layout.



RadAjax

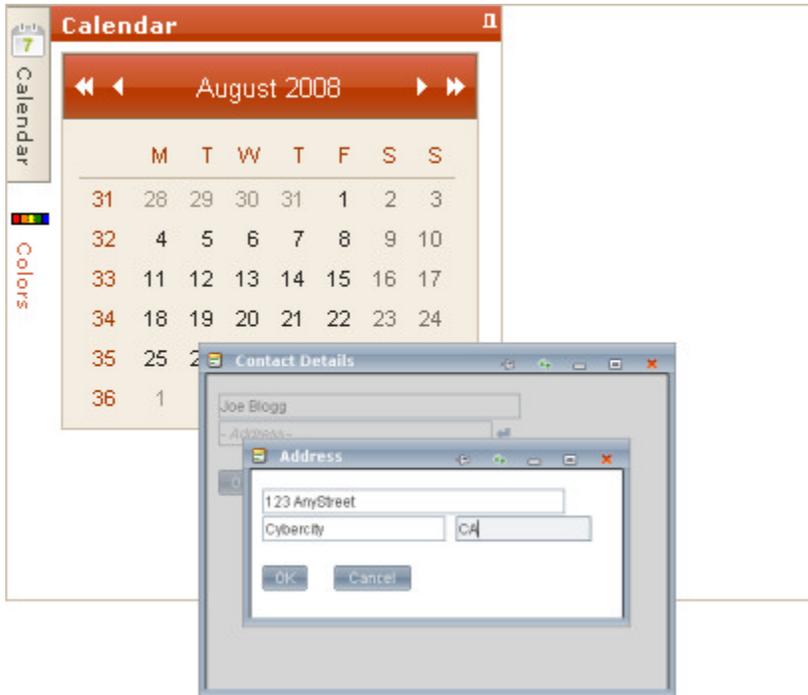
The RadAjax family of controls let you instantly AJAX-enable your application with little programming and configuration effort on your part. RadAjaxPanel AJAX-enables everything that sits on the panel and is an easy way to get started. For more control and potential performance benefit, RadAjaxManager lets you AJAX-enable specific parts of your application. Both routes let you display a "spiny" graphic during long running processes using the RadAjaxLoadingPanel. With RadAjax controls you can get startling performance and that Windows desktop look-and-feel.



Screen Real Estate Management Controls

The controls that let you manage screen "real estate" define regions of the Web page that can be moved around the screen, minimized or hidden away. By using these "real estate" controls, you can organize your Web pages and add flexibility that lets your users configure the layout in an individualized way. These controls include

windowing, docking, splitter bars and sliding zones.



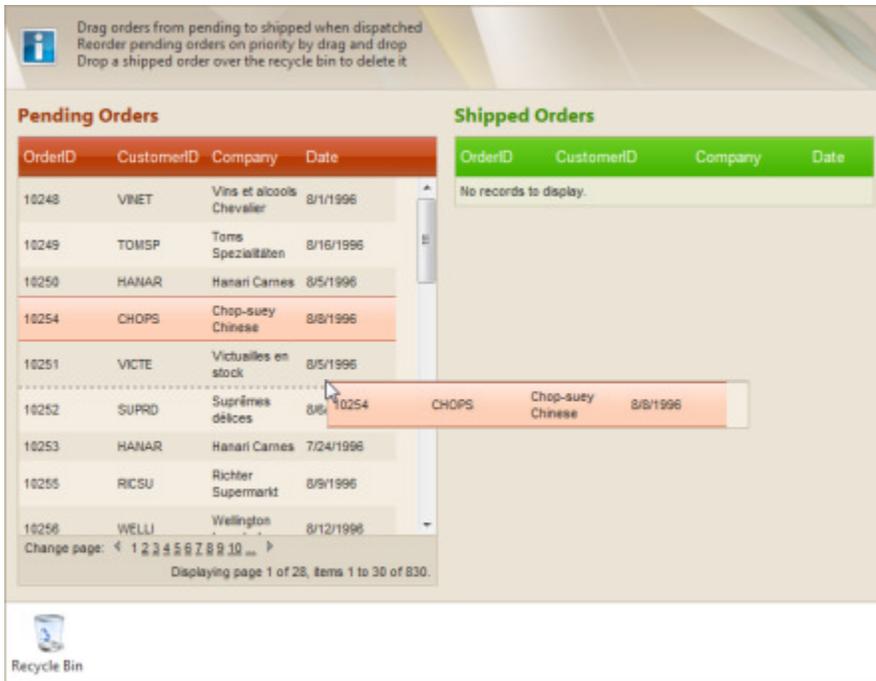
RadComboBox

RadComboBox is an amped up version of a standard drop down list that lets you add images, animated effects, and is templated for complete control over the layout. Unlike the ASP.NET DropDownList control, which restricts users to selecting only items from the list, RadComboBox can optionally allow users to type in their own entries. RadComboBox also works well for very long lists of items. The auto-complete feature automatically scrolls the list and highlights matches, or you can use the filtering capability to limit items to currently entered text. You can even configure the combo box to load on demand.



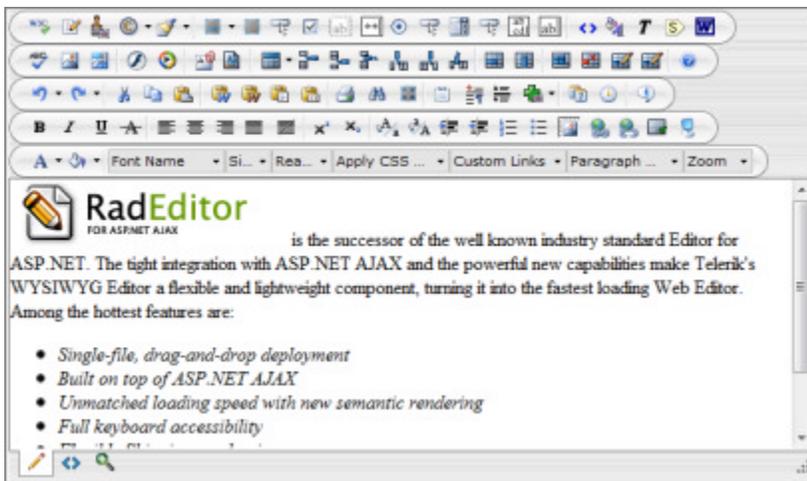
RadGrid

RadGrid for ASP.NET AJAX is the fastest and most feature-rich Datagrid for ASP.NET, designed to provide desktop-like user experience with minimum HTML output. RadGrid provides real-time performance as well as almost codeless development experience for a rich variety of features.



RadEditor

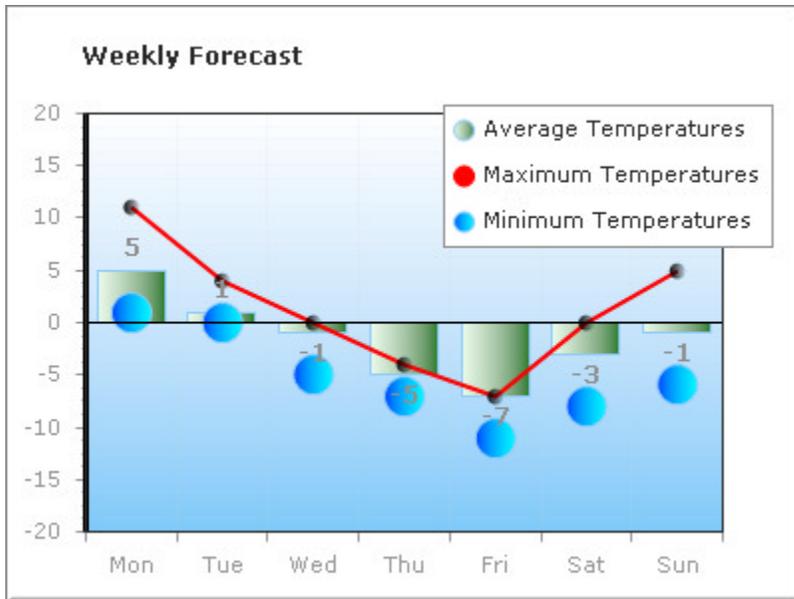
RadEditor is a powerful but lightweight editor control you can use in your web applications when you need a full-featured editor. It comes loaded with lots of built-in goodies like pre-defined buttons, drop down lists and context menus that perform any tasks you are likely to need. If the built-in tools don't fill the bill, RadEditor can be extensively customized.



RadChart

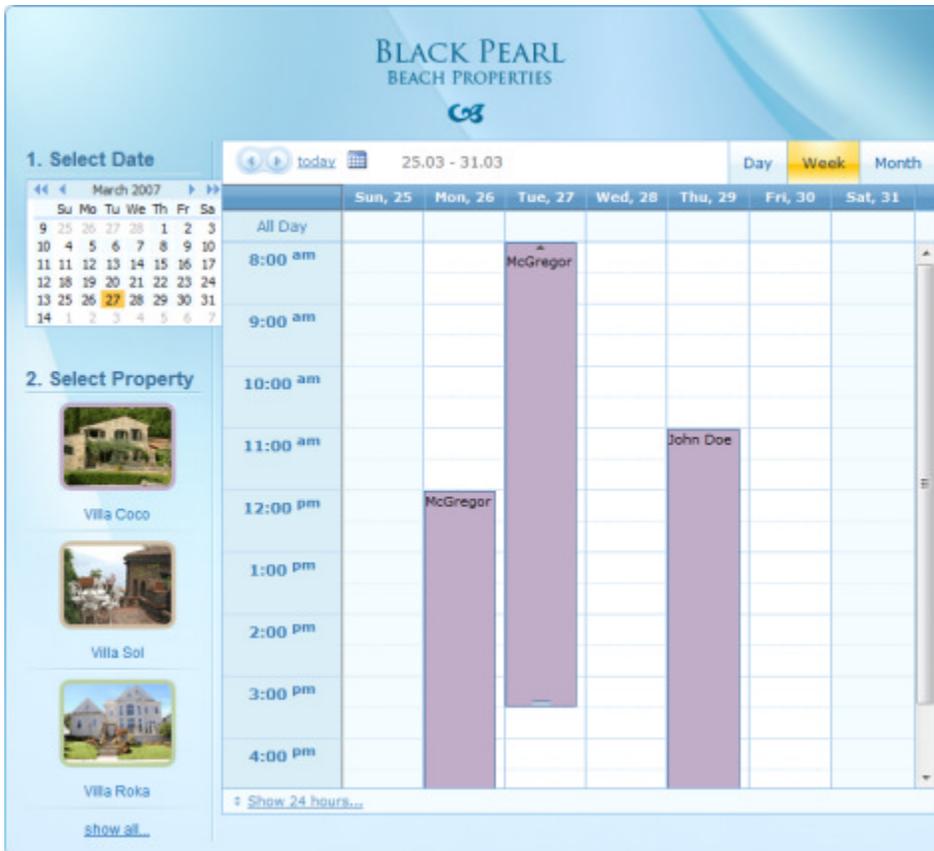
RadChart is a powerful business data presentation tool that can show your data off with striking impact. RadChart comes with many customizable chart types and skins to tailor the behavior and look of each chart.

You can choose to exercise fine-tune control over all aspects of your chart or use the automatic layout, automatic text wrapping and intelligent labeling functions to handle the details. At design time you get quick access to critical properties with the Smart Tag, convenient groups of important properties in the RadChart wizard, or control all RadChart settings from the Properties Window.



Date, Time, Calendar and Scheduling Controls

RadControls comes with a full set of date and time and calendar related controls. Date and time picker controls let the user enter directly or choose using mouse-only in a pop-up dialog. With RadCalendar you have virtually unlimited control over appearance and formatting and can define special days for individual display of holidays and appointments. For complete support of scheduling, RadScheduler provides automatic support of appointment inserts/updates/deletions, multiple views (day, week, month, time-span and more), ability to drag appointments, custom attributes, resources, inline and advanced views of appointments and many customization options.



1.7 Before You Begin...

The projects in this learning guide will assume the following:

1. You will add the following "Imports" (VB) or "uses" (C#) statements to your projects to reference the Telerik.Web.UI namespace:

[VB] Including the Telerik.Web.UI Namespace

```
Imports Telerik.Web.UI
```

[C#] Including the Telerik.Web.UI Namespace

```
using Telerik.Web.UI;
```

2. RadControls for ASP.NET AJAX requires a ScriptManager before any of the controls on the page. You may instead use the RadScriptManager although it is not required. RadScriptManager has some optimization capabilities that can be used for maximum performance.
3. Example projects can be found in \VS Projects\\<CS or VB>\<project name>. For example, the Navigation Controls ServerTags project for C# can be found at \VS Projects\Navigation Controls\CS\ServerTags.

2 Navigation Controls

2.1 Objectives

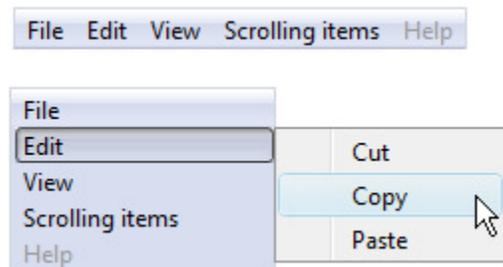
- Inventory the "navigation" related RadControls. Explore how and where these navigation controls are used.
- See how each of the navigation controls are similar so you can leverage the same knowledge with each control.
- Create a simple application to get confidence in using each of the controls.
- Explore the design time interface for each of the navigation controls, again taking special notice of where the controls are similar. You will learn how to access properties and methods through Smart Tag, Properties Window and Property Builder.
- Explore principal properties and groups of properties where 80% of the functionality is found.
- Learn server-side coding techniques, starting with an exploration of important methods and events. You will also learn how to perform common server-side tasks (e.g. add/edit/delete items in a collection) and control-specific tasks (e.g. set a context menu target).

2.2 Introduction

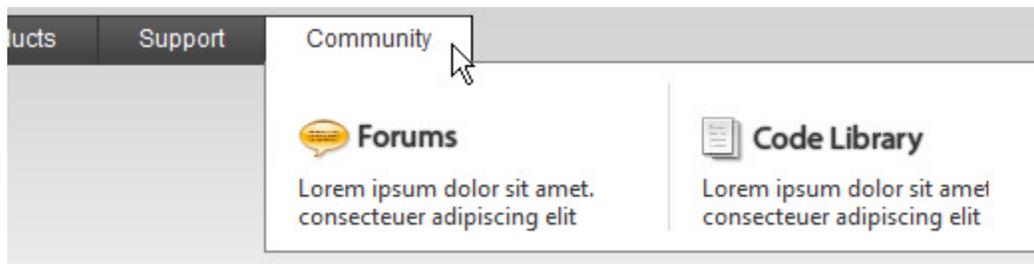
Go to any popular web site and you expect to see menus and tab strips across the top and along the sides. Traversing web pages is after all the bread-and-butter of many web sites. For web applications that mimic full featured Windows applications you expect to see Outlook-like panel bars for organizing functionality, tool bars for taking direct actions and context menus for making intuitive choices within the user's own data. RadControls have you covered with a versatile set of navigation controls for building compelling user interfaces easily:

RadMenu

RadMenu is an advanced navigation control that allows you to build lightweight and search-engine-friendly menu systems. Menus can be displayed horizontally or vertically and have multiple levels. Child items can open up (as shown in the screenshot below) or can automatically scroll.



Menu items can display text, images or both. And because RadMenu items can be templated, you can add virtually any content that suits your purpose:



RadContextMenu

UI for ASP.NET AJAX

RadContextMenu is similar to RadMenu but is designed to popup over a "target" control where the user right-clicks. Context menus can also be triggered by other events and popped up programmatically (either client or server side).



RadTabStrip

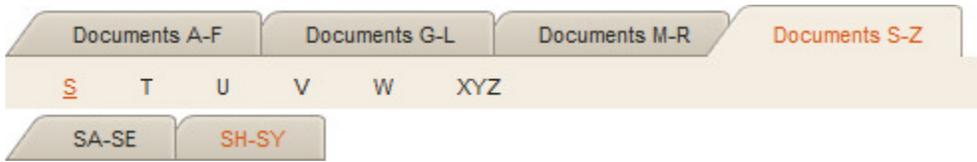
Use RadTabStrip to build tabbed interfaces for URL navigation or making choices based on tab selection. RadMultiPage is a related control that manages content of pages that can be automatically selected by RadTabStrip. Tabs can be located on the top or bottom (see screenshot below), left or right side of your web page.



RadTabStrip has a number of options for customizing layout and appearance including:

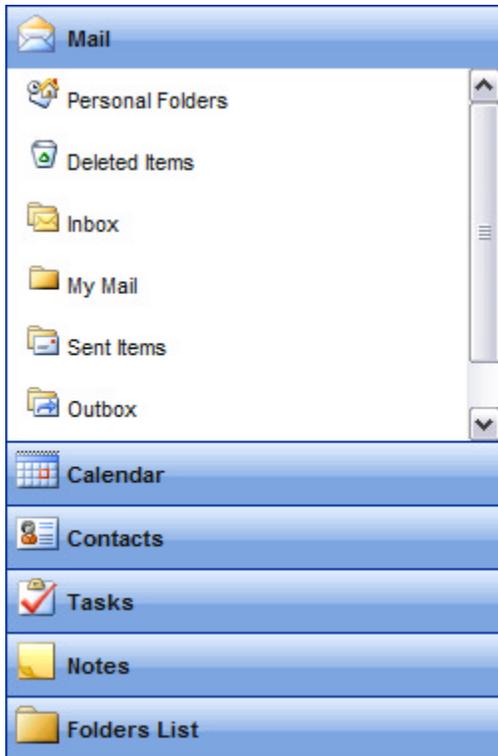
- Tabs can be aligned left, center, right or justified.
- Tabs can appear on multiple rows and can "break" at any tab to form a new row.
- Tabs can scroll for better management of your screen real-estate.

- Tabs can be structured in a hierarchy for more complex relationships (see screenshot below).



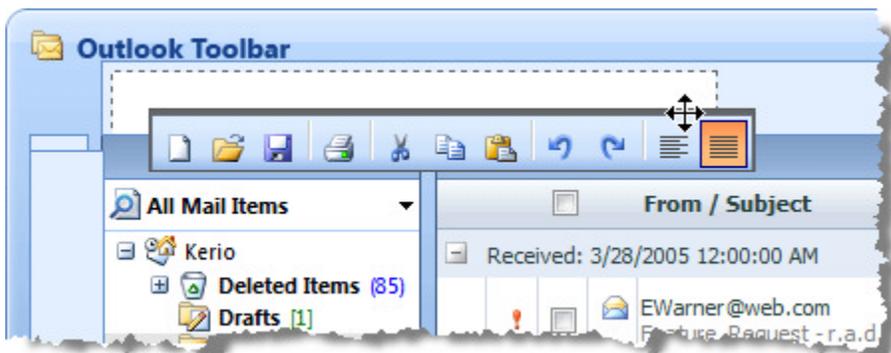
RadPanelBar

Use RadPanelBar to create collapsible vertical menus or Outlook style panels. You can use templates to create a tool bar or simple entry form area within panels. RadPanelBar can be configured to open only one panel at a time, or multiple panels at one time.



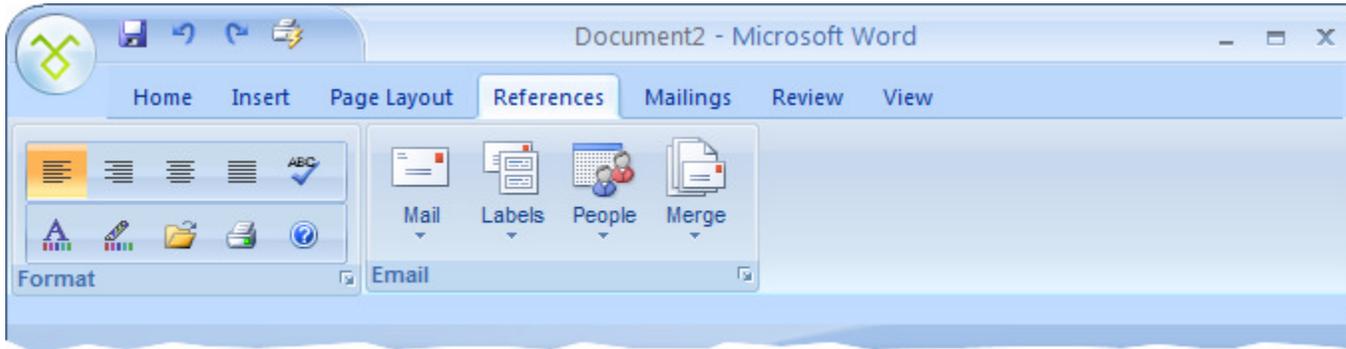
RadToolBar

Tool strips are used in most web applications to allow quick access to common tools. RadToolBar mimics the flexibility of desktop toolbars which can be floating, dockable, re-orderd and can be oriented vertically or horizontally. RadToolBar can be used in conjunction with RadDock to creating a docking toolbar:



All Together Now...

Also know that navigation controls can be combined to create an Office "Ribbon Bar" style interface.

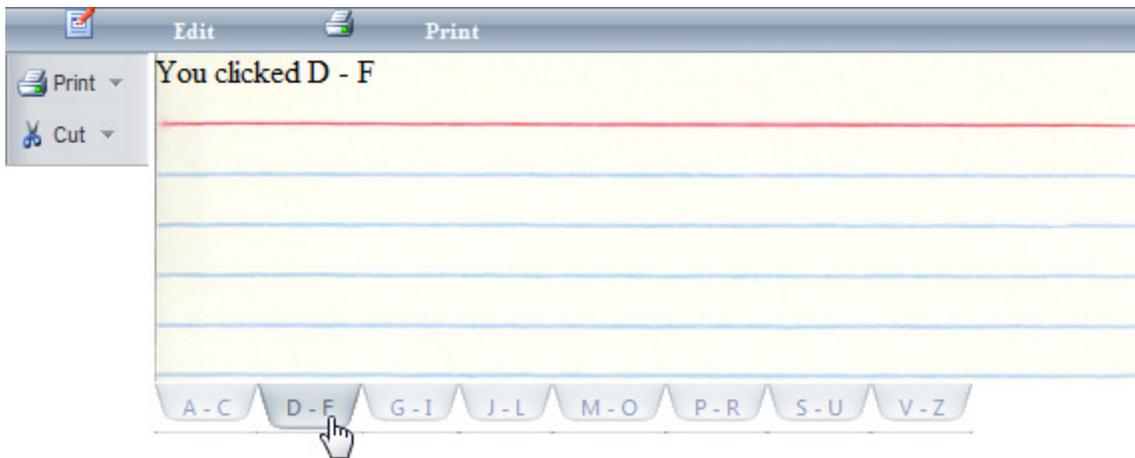


Each of the navigation controls...

- Uses "semantic rendering" for minimal HTML markup. Semantic rendering avoids costly HTML tables and instead uses Cascading Style Sheets (CSS) to handle appearance and placement. The HTML output is significantly reduced resulting in dramatic performance improvement.
- Can be populated at design-time, in markup, through data binding, in server code or in client code. You can jump ahead to the chapter on Data Binding for detailed information on hooking up all kinds of database and object data to your RadControls.
- Can be skinned for a great visual appearance that's consistent with your entire web application. Each control comes with a standard set of matched skins (e.g. "Outlook", "Vista", "Black", "Telerik", etc.) that can be simply selected from a list or you can create your own custom skin. You can skip ahead to the chapter on skinning for details on building your own custom skins.
- Includes full keyboard support for navigating and activating items.
- Right-to-left support to allow your application to be internationalized.
- Except for the tool bar, the navigation controls can be animated so that visual actions such as menu expansion uses one of several predefined effects. Animation can be disabled. Delay and duration for the animation effect can be specified in milliseconds.
- Each item has a special Attributes collection that can contain any custom name/value pairs you might need. Attributes can be defined declaratively and accessed in code (either client or sever-side).
- Includes a rich, consistent client-side API for adding/deleting items on-the-fly, locating/changing items and monitoring events. All these tasks can be performed with best performance right on the client browser.
- Supports templates so that portions of your RadControl can contain any arbitrary arrangement of HTML including ASP.NET controls, RadControls or anything else that can be entered into markup.

2.3 Getting Started

In this walk-through you will become familiar with the menu, tab strip and tool bar controls (more on panel bar in the server-side section upcoming). These controls will produce the card catalog style interface you see below.



Setup the project structure

1. Create a new Web Application and add a ScriptManager to the default page.
2. In the solution explorer, create a new \Images folder.
3. Copy images from the <Courseware projects folder> \ Navigation \ CS \ GettingStarted \ GettingStarted \ images library to the projects \images directory: CopyHS.png, CutHS.png, PasteHS.png, EditInformationHS.png, PrintHS.png and PrintPreviewHS.png. *These images will be associated with the buttons in the navigation controls.*
4. Copy the image "3X5Card.png" from the <Courseware projects folder> \ Navigation \ CS \ GettingStarted \ GettingStarted \ images folder to your projects \images folder. *This image will form the background of your interface.*
5. Add the following styles to the <head> tag of the ASP.NET markup. These styles will position the card image and the tab strip underneath the card.

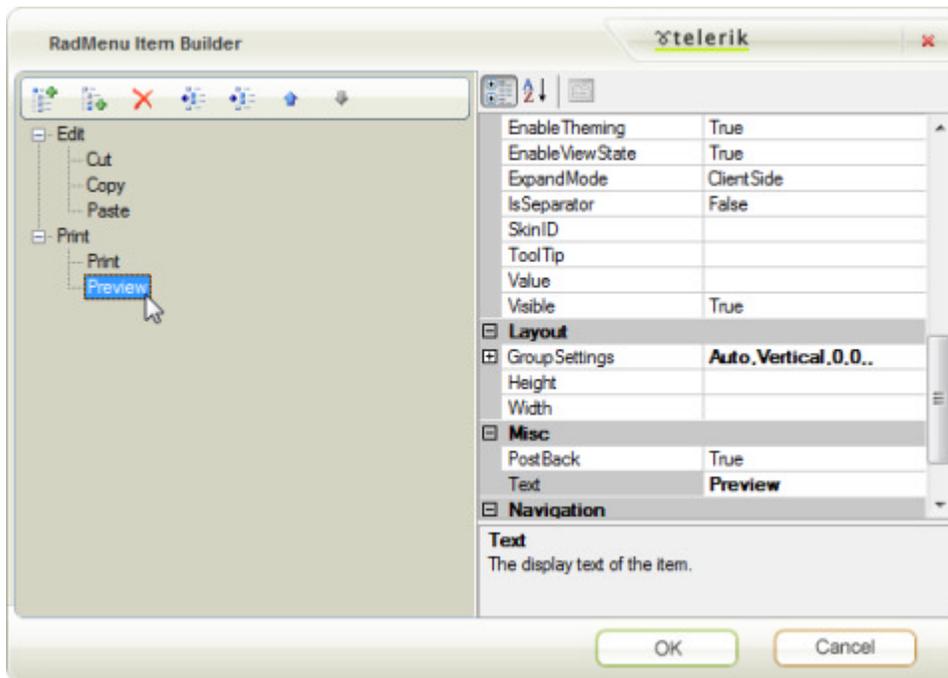
[ASP.NET] Setting Styles

```
<head runat="server">
  <title>Getting Started</title>
  <style type="text/css" media="screen">
    #content
    {
      background-image: url( 'images/3X5card.png' );
      height: 165px;
      width: 500px;
      margin-left: 75px;
      vertical-align: top;
    }
    #tabs
    {
      margin-left: 75px;
      width: 500px;
      vertical-align: bottom;
    }
  </style>
</head>
```

Add the RadMenu

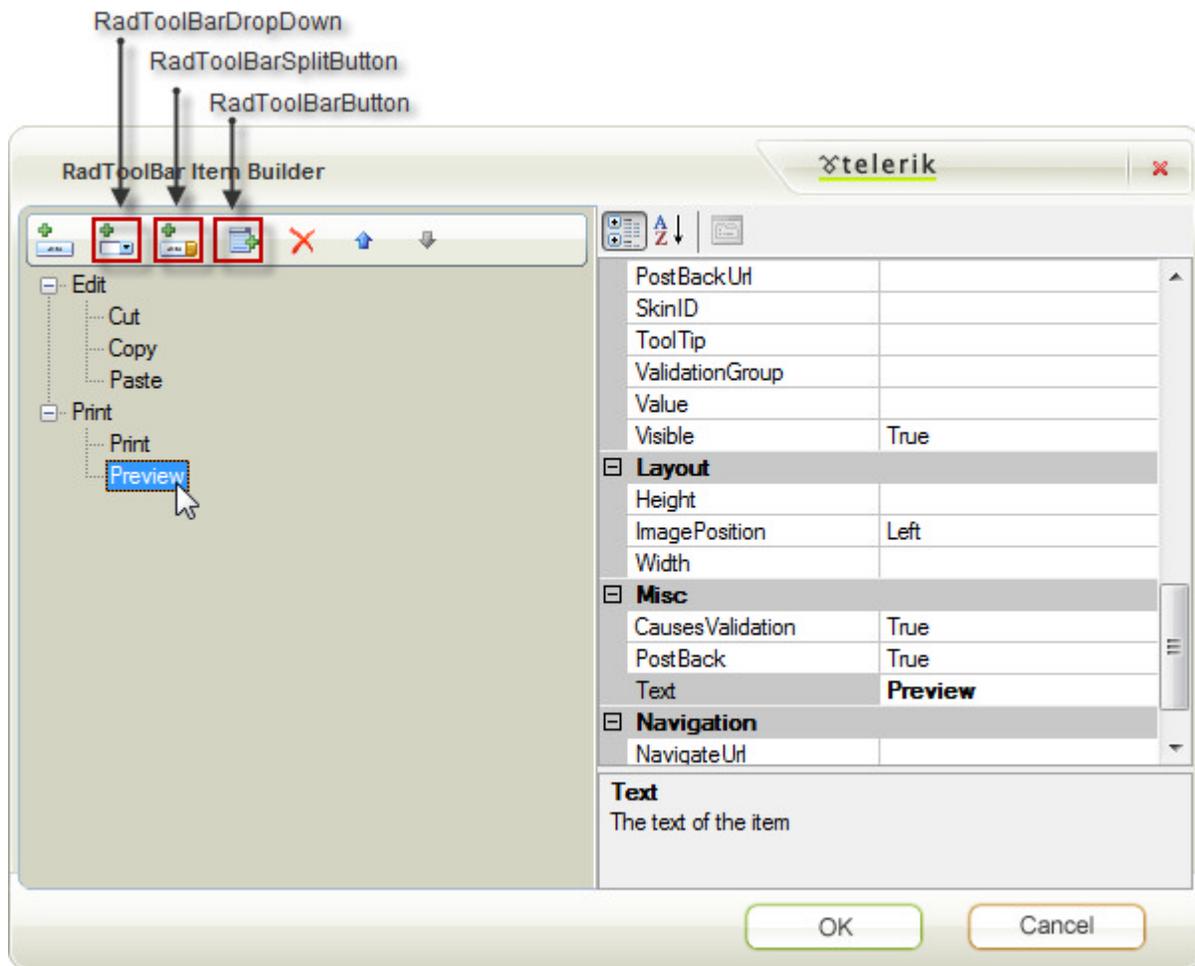
UI for ASP.NET AJAX

1. Add a **RadMenu** to the top of the web page. Set the **Skin** property to "WebBlue". Set the **Width** property to "100%".
2. Open the RadMenu Smart Tag and select the **Build RadMenu...** option.
3. Add two root level items with text "Edit" and "Print". Set the **ImageUrl** properties for these items to "\images>EditInformationHS.png" and "\images\PrintHS.png" respectively.
4. Select the "Edit" item and add three child items with Text "Cut", "Copy" and "Paste". Set the ImageUrl properties for these items to "\images\CutHS.png", "\images\CopyHS.png" and "\images\PasteHS.png" respectively.
5. Select the "Print" item and add two child items "Print" and "Preview". Set the ImageUrl properties for these items to "\images\PrintHS.png" and "\images\PreviewHS.png" respectively.



Add the RadToolBar

1. Add a **RadToolBar** underneath the tab strip. Set the **Skin** property to "WebBlue". Set the **Width** property to "70px" and the **Orientation** property to **Vertical**.
2. Open the RadToolBar Smart Tag and select the **Build RadToolBar...** option.
3. Add a **RadToolBarDropDown** and a **RadToolBarSplitButton** with text "Edit" and "Print". Set the **ImageUrl** properties for these items to "\images>EditInformationHS.png" and "\images\PrintHS.png" respectively.
4. Select the "Edit" item and add three **RadToolBarButton** child items with **Text** "Cut", "Copy" and "Paste". Set the ImageUrl properties for these items to "\images\CutHS.png", "\images\CopyHS.png" and "\images\PasteHS.png" respectively.
5. Select the "Print" item and add two **RadToolBarButton** child items with Text "Print" and "Preview". Set the ImageUrl properties for these items to "\images\PrintHS.png" and "\images\PreviewHS.png" respectively.



Add Content Area Divs

1. Add a <div> tag with id "content" to contain the 3x5 card graphic and text.
2. Inside the "content" div add a standard ASP.NET Label control with ID "lblContent".

[ASP.NET] Adding the Content Div

```
<div id="content" runat="server">
  <asp:Label ID="lblContent" runat="server" Text="Label"></asp:Label>
</div>
```

3. Under the "Content" div tag add another div with ID "tags".
4. Inside the "tags" div add a RadTabStrip. Set the **Orientation** to "HorizontalBottom", the **Skin** to "WebBlue" and the **Width** to "500px". **Note:** We will add items to the tab strip later in server-side code.

[ASP.NET] Adding the Tabs Div

```
<div id="tags" runat="server">
  <telerik:RadTabStrip ID="RadTabStrip1" runat="server" Orientation="HorizontalBottom"
    Skin="WebBlue" OnTabClick="RadTabStrip1_TabClick" SelectedIndex="0" Width="500px">
  </telerik:RadTabStrip>
</div>
```

Adding Server-Side Code

1. Use the Properties window, Events  button to create event handlers for the following:
 1. RadMenu **ItemClick**
 2. RadToolBar **ButtonClick**
 3. RadTabStrip **TabClick**
2. Populate the event handlers with the code below:

[VB] Handling Navigation Control Click Events

```
Protected Sub RadMenu1_ItemClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadMenuEventArgs)
    lblContent.Text = "You clicked " + e.Item.Text
End Sub
Protected Sub RadToolBar1_ButtonClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadToolBarEventArgs)
    lblContent.Text = "You clicked " + e.Item.Text
End Sub
Protected Sub RadTabStrip1_TabClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadTabStripEventArgs)
    lblContent.Text = "You clicked " + e.Tab.Text
End Sub
```

[C#] Handling Navigation Control Click Events

```
protected void RadMenu1_ItemClick(object sender, Telerik.Web.UI.RadMenuEventArgs e)
{
    lblContent.Text = "You clicked " + e.Item.Text;
}
protected void RadToolBar1_ButtonClick(object sender, Telerik.Web.UI.RadToolBarEventArgs
{
    lblContent.Text = "You clicked " + e.Item.Text;
}
protected void RadTabStrip1_TabClick(object sender, Telerik.Web.UI.RadTabStripEventArgs e)
{
    lblContent.Text = "You clicked " + e.Tab.Text;
}
```

-  Notice that the event handlers are substantially similar except for the specific event argument object passed in. Each has just an "Item" property (except for TabClick which has a "Tab" property) that refers to the item clicked by the user. Use the e.Item or e.Tab properties to get at the Text, Value, Attributes, ImageUrl and other properties for the item.

Populate the TabStrip

1. In the Page_Load event handler, add the following code:

[VB] Populating the TabStrip

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        Dim letters As String() = "ABC|DEF|GHI|JKL|MNO|PQR|STU|VWXYZ".Split("|"C)
```

```

For Each chunk As String In letters
    Dim tab As New RadTab(chunk(0) + " - " + chunk(chunk.Length - 1))
    RadTabStrip1.Tabs.Add(tab)
Next
End If
End Sub

```

[C#] Populating the TabStrip

```

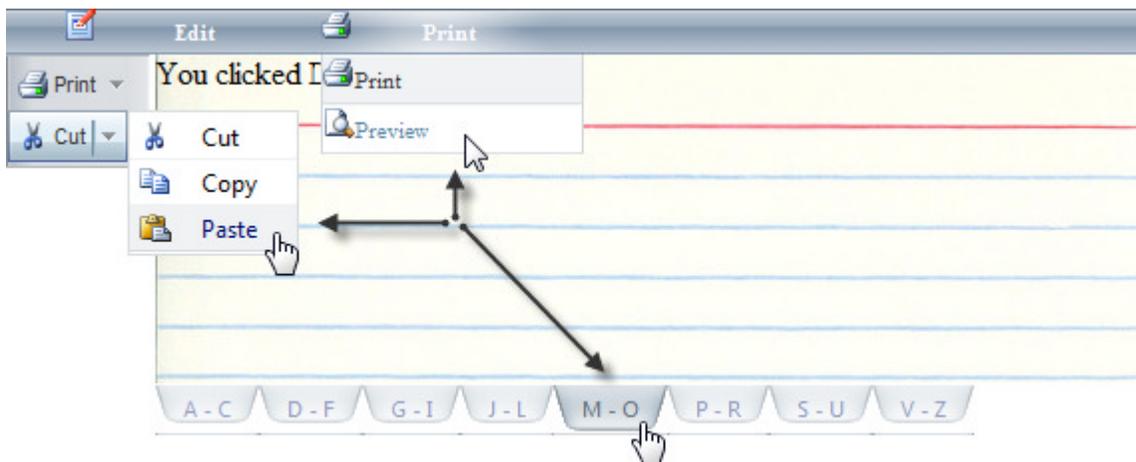
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        string[] letters = "ABC|DEF|GHI|JKL|MNO|PQR|STU|VWXYZ".Split('|');
        foreach (string chunk in letters)
        {
            RadTab tab = new RadTab(chunk[0] + " - " + chunk[chunk.Length -1]);
            RadTabStrip1.Tabs.Add(tab);
        }
    }
}

```

When the page first loads, a string representation of the alphabet is broken up into chunks and fed into an array of string. The first and last letter of each chunk is formatted and placed into the text for a newly created RadTab and added to the Tabs collection.

✍ Don't forget to add Telerik.Web.UI to the "Imports" (VB) or "uses" (C#) section of code.

2. Press Ctl-F5 to run the application. You should be able to click any option on the the menu above or from the tool bar on the side. In addition you should be able to click on any of the tabs. The clicked on item will be reflected in the label sitting on the 3x5 card image.



2.4 Designer Interface

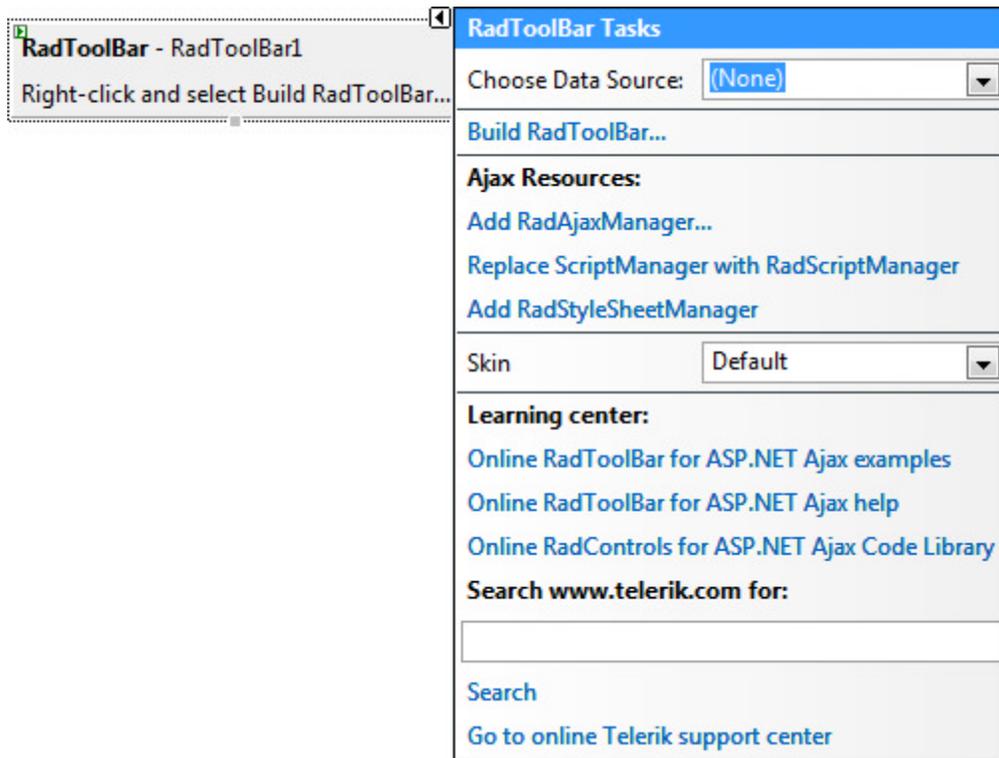
Each of the navigation controls has a similar designer interface with a few exceptions:

Smart Tag

The Smart Tag provides easy access to frequently needed design tasks. To display the Smart Tag, click the small left-pointing arrow located in the upper right of the control or choose "Show Smart Tag" from the context

UI for ASP.NET AJAX

menu. The screenshot below shows the RadToolBar Smart Tag, but the Smart Tag for other navigation controls is substantially similar (except where noted).



Tasks

- **Choose Data Source** lets you bind the control declaratively by selecting a data source from a drop-down list of all available ASP.NET 2.0 data source components. You can also select **<New Data Source...>** to display the standard Windows Data Source Configuration Wizard.
- **Build** displays an Item Builder dialog where you can create and configure statically defined items for your navigation control.

Ajax Resources

- **Add RadAjaxManager...** adds a RadAjaxManager component to your web page and displays a RadAjaxManager configuration settings dialog.
- **Replace ScriptManager with RadScriptManager** swaps out the standard ScriptManager for a RadScriptManager. RadScriptManager is not required for RadControls for ASP.NET AJAX but does include the ability to combine scripts for greater efficiency.
- **Add RadStyleSheetManager** adds a RadStyleSheetManager component to your web page. RadStyleSheetManger combines style sheets to reduce page load time and traffic.

Skin

Use the Skin drop-down to preview and select built-in skins.

Learning Center

Navigate directly to examples for the control, find help or use the code library. You can also search the Telerik

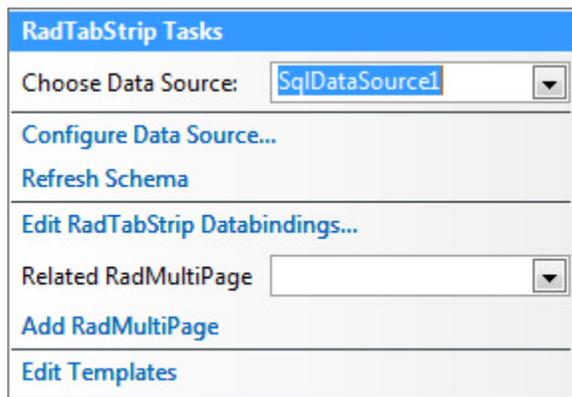
web site for a given string.

Edit Templates

Click the Edit Templates link to display the template design surface. Here you can create or edit templates used by your control. You can jump ahead to the Templates chapter for more details on how templates are used in RadControls to build custom interfaces.

Smart Tag when Data Bound...

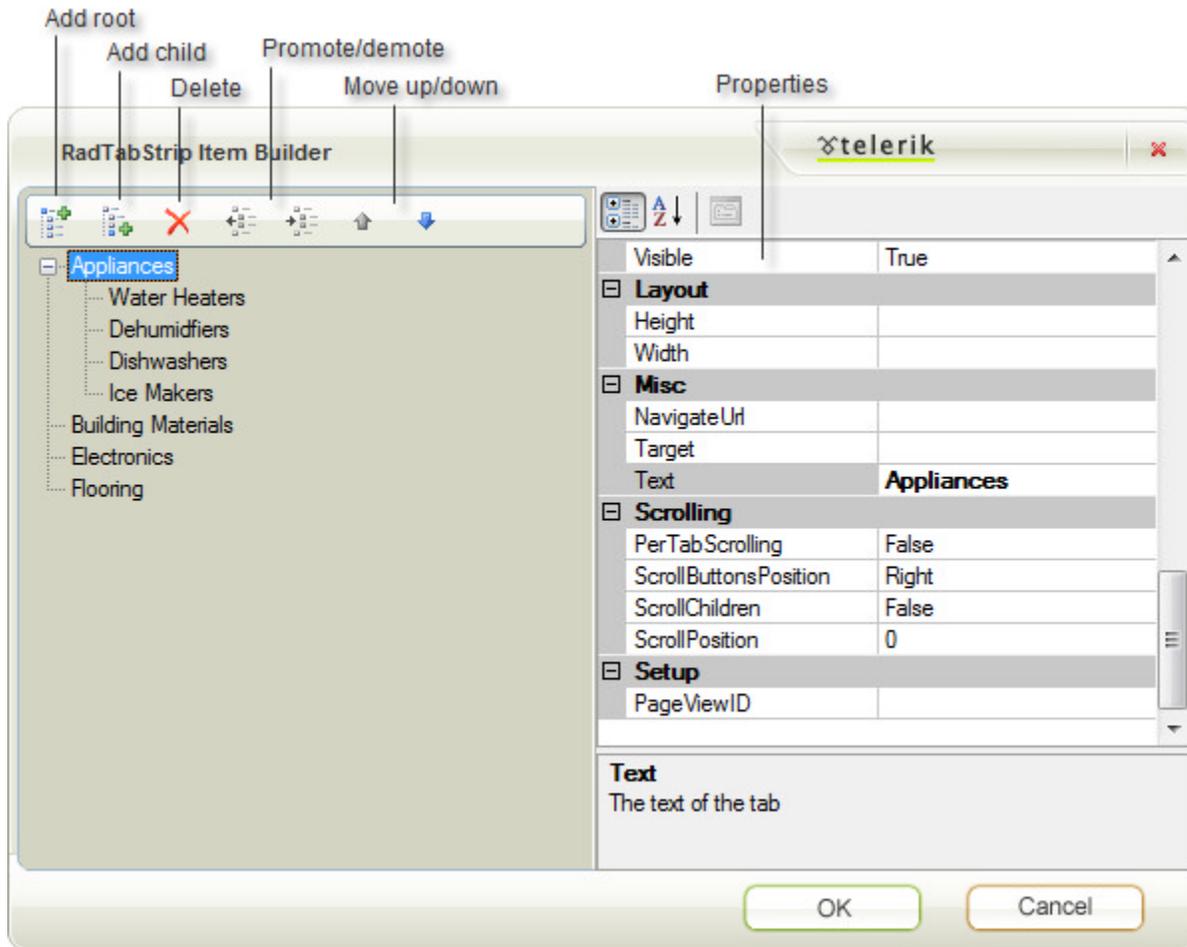
The Smart Tag changes when the control is bound to a data source. You can now select to **Configure the Data Source** to re-execute the Windows Data Source Configuration Wizard or **Refresh Schema** if the underlying data changes and you need the data source to reflect the new schema.



Property Builder

Each of the navigation controls displays a Property Builder dialog specific to that control. Display the builder dialog either from the Smart Tag or clicking the **Items** property ellipses in the Properties Window (the property is called **Tabs** for the RadTabStrip). The property builder will look substantially the same for all controls except RadToolBar. RadMenu, RadPanelBar and RadTabStrip all support hierarchies of multiple levels, while RadToolBar has a relatively flat structure.

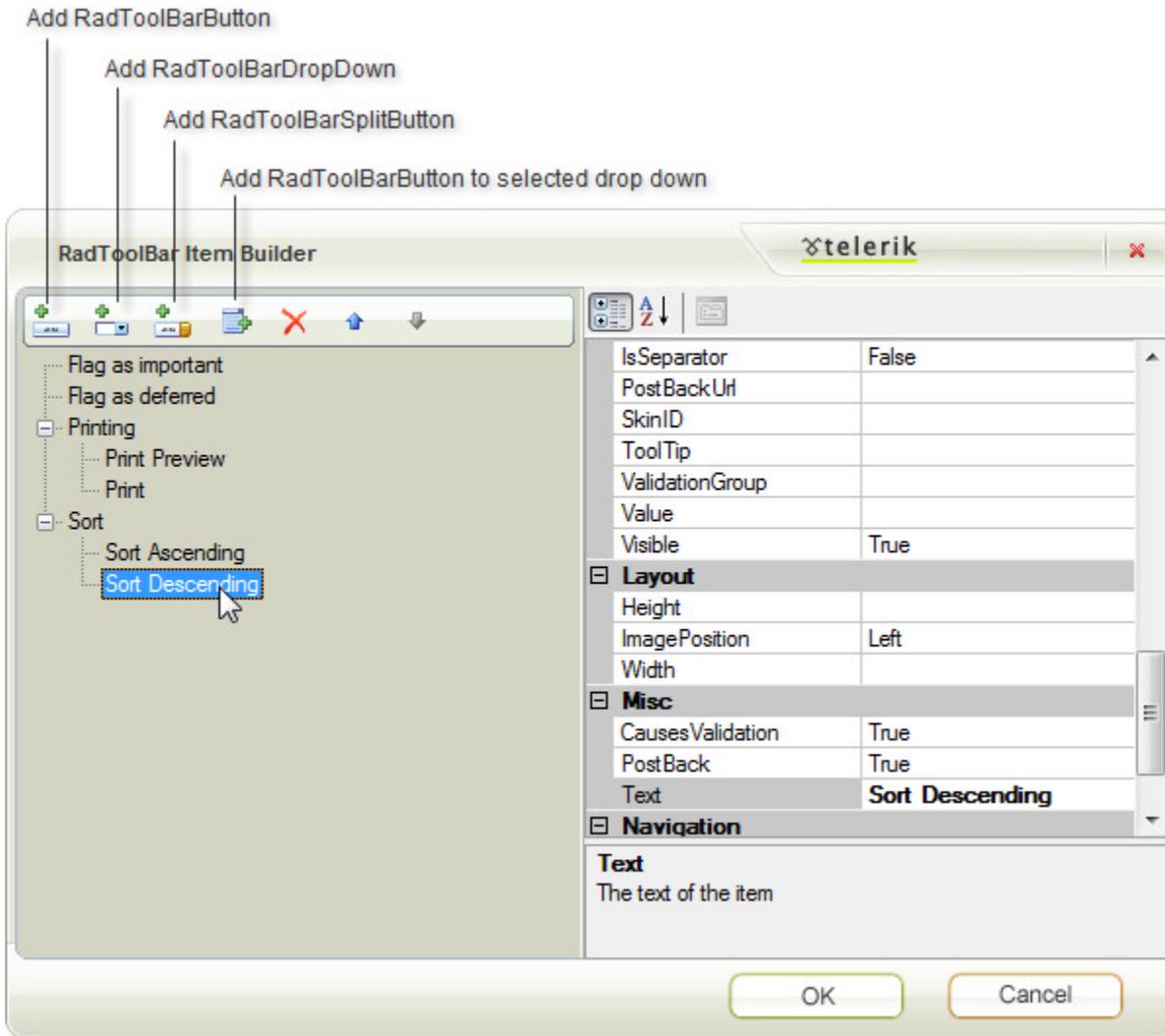
Below is a screen shot of the property builder for RadTabStrip items. Use the buttons on the upper left to add root and child level items. You can use the button labeled "Promote" shown below to make a child item a sibling of its parent. Use the "demote" button to make an item a child of the preceding sibling. To edit the text of an item in-line, select it with the mouse, then click it a second time. You can select any of the items and set item properties using the list on the right of the dialog. Typically, you will set the **Text** property first.



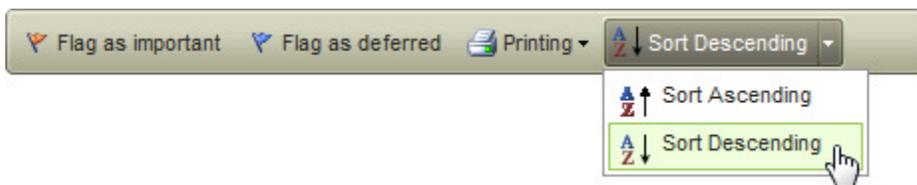
The RadToolBar Item Builder has an essentially flat structure, although a second level of buttons is allowed for drop down and split buttons. RadToolBar does not have an unlimited number of levels and so does not have promote or demote buttons. Also, there are several types of buttons you can add:

- RadToolBarButton: Executes some immediate command, or can be configured to have a state and work like a check box or radio button. You can add RadToolBarButton as a root level item or add it under a drop down or split button.
- RadToolBarDropDown: This button acts as a drop down list of commands when clicked.
- RadToolBarSplitButton: This button acts much like the RadToolBarDropDown, but has a default command, i.e. the last button in the list you clicked. The split button works well when one of the commands is used all of the time.

The screenshot below shows the possible combinations:

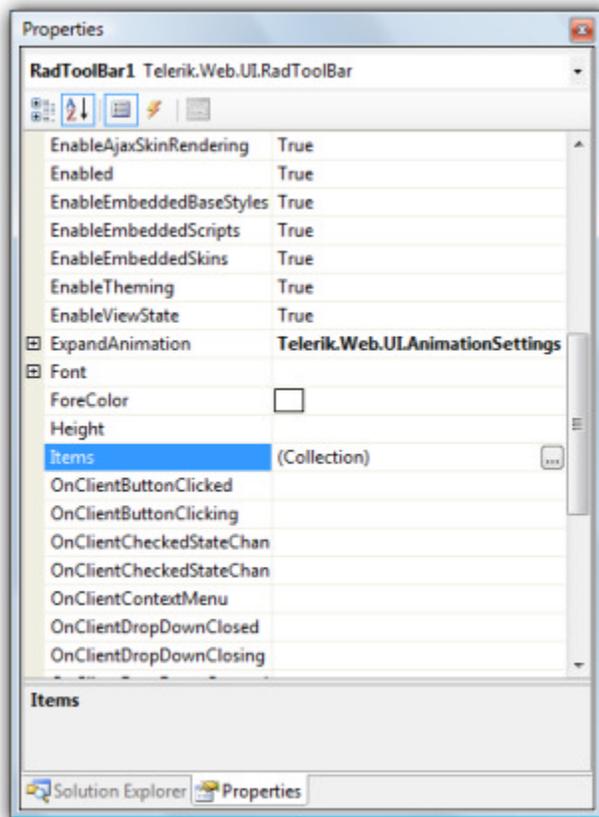


The resulting tool bar running in the browser looks something like this:



Properties Window

The superset of properties available to the control are found in the Properties window. We will use the 80/20 rule here; that is, locate the most important properties and groups of properties common between navigation controls that are used constantly.



The single most important property of the navigation controls is the **Items** collection (or **Tabs** in the case of RadTabStrip). Items make up the content of the navigation control. You can populate your navigation control items...

- Statically, using the Items property or using the Item Builder dialog.
- Defining items in the ASP.NET markup. For example, the ASP.NET fragment below shows RadMenu with two levels of items defined.

[ASP.NET] Defining RadMenu Items

```
<telerik:RadMenu ID="RadMenu1" runat="server" >
  <Items>
    <telerik:RadMenuItem runat="server" Text="File">
      <Items>
        <telerik:RadMenuItem runat="server" Text="Exit">
        </telerik:RadMenuItem>
      </Items>
    </telerik:RadMenuItem>
    <telerik:RadMenuItem runat="server" Text="Edit">
      <Items>
        <telerik:RadMenuItem runat="server" Text="Cut">
        </telerik:RadMenuItem>
        <telerik:RadMenuItem runat="server" Text="Copy">
        </telerik:RadMenuItem>
        <telerik:RadMenuItem runat="server" Text="Paste">
        </telerik:RadMenuItem>
      </Items>
    </telerik:RadMenuItem>
  </Items>
</telerik:RadMenu>
```

```
</telerik:RadMenu>
```

- Adding programmatically on the server or client side. We will cover the details in the server and client side programming sections upcoming.
- Data Binding. We will cover data binding thoroughly in a later chapter. For now, know that the data binding specific properties are: **DataSource**, **DataSourceID**, **DataMember**, **DataTextField**, **DataTextFormatString**, **DataValueField** and **AppendDataBoundItems**. Multi-level hierarchies are implemented (for navigation controls other than RadToolBar) with the use of **DataFieldID**, **DataFieldParentID** and **MaxDataBindDepth** properties. Also, all navigation controls other than RadToolBar have a **DataNavigateUrlField** that lets you bind to a column that contains a URL.

Each item within the Items collection has its own set of properties. **Text** is the string that displays in the UI for an item, **ImageUrl** is a path to an image file that will display next to the Text and **NavigateUrl** is a URL that will be navigated to when the item is clicked. With just these three properties alone, you can do quite a bit of web site building. Use the **NavigateUrl** property together with **Target** to specify the target window or frame to display the **NavigateUrl** web page content. **Target** can be specified as **_blank** (target URL will open in a new window), **_self** (target URL will open in the same frame), **_parent** (target URL will open in the parent frameset) and **_top** (target URL will open in the topmost frame).

If your purpose is not to navigate URLs, but to make choices within a web application, the **Value** property is a useful place to store codes, record IDs or any arbitrary string. The Value can be retrieved in both client and server code.

To craft the look of individual items, look for property names ending in "ImageUrl". Depending on the particular item type you will see **DisabledImageUrl**, **ExpandedImageUrl**, **HoverImageUrl**, **SelectedImageUrl**, etc. Also look for properties ending in "CssClass". These properties specify CSS classes used to style the item during particular states, e.g. **ClickedCssClass**, **DisabledCssClass**, **ExpandedCssClass**, **FocusedCssClass**, etc. Some of these classes may be pre-populated with class names from the control's skin (see the chapter on Skinning for details on working with RadControls skins).

Use separators to visually group items into two or more categories. Set the **IsSeparator** property of an item to true; that item will not respond to user clicks or keyboard actions. The RadMenu screenshot below shows a separator defined for an item between the "Save" and "Exit" items.



The markup for this example looks like this:

[ASP.NET] Using IsSeparator

```
<telerik:RadMenuItem runat="server" Text="File">
  <Items>
    <telerik:RadMenuItem runat="server" Text="New">
    </telerik:RadMenuItem>
    <telerik:RadMenuItem runat="server" Text="Open">
    </telerik:RadMenuItem>
    <telerik:RadMenuItem runat="server" Text="Save">
    </telerik:RadMenuItem>
    <telerik:RadMenuItem runat="server" IsSeparator="True" Text="My Separator">
    </telerik:RadMenuItem>
    <telerik:RadMenuItem runat="server" Text="Exit">
    </telerik:RadMenuItem>
  </Items>
</telerik:RadMenuItem>
```

```
</telerik:RadMenuItem>
```

2.5 Server-Side Programming

Adding Items

The general pattern for adding items to a navigation control Items collection is:

- Create an item instance for the particular type of collection.
- Populate the instance properties.
- Add the item to the Items collection.

When you add to the Items collection and type in the open parenthesis, IntelliSense will display code completion with the specific item type (or press ctrl-shift-spacebar to invoke IntelliSense).

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        RadMenu1.Items.Add(|
    }
}
void Telerik.Web.UI.RadMenuItemCollection.Add(Telerik.Web.UI.RadMenuItem item)
item:
The Telerik.Web.UI.RadMenuItem to append to the end of the current Telerik.Web
```

Notice that the RadMenuItem type in the screenshot above is in the Telerik.Web.UI namespace. Save some time and typing effort by adding a using statement to include this reference. The remainder of this courseware will assume that you have included the Telerik.Web.UI namespace.

The example below uses a ScriptManager (or RadScriptManager), a RadAjaxManager and a RadMenu. The RadAjaxManager has a nifty little Alert() method that automatically pops up a client-side alert dialog. You can skip ahead to the chapter "AjaxPanel, AjaxManager and AjaxProxy" for an exploration of these important components. In the Page_Load event handler, two RadMenuItem instances are created. The first RadMenuItem is assigned Text, NavigateUrl and Target properties, then added to the RadMenu.Items collection. The Target property is set to "_blank" so that a second browser will pop up.

 **Gotcha!** When you populate the NavigateUrl property, be sure to type the entire URL including the "http://" and avoid the "Resource not found" error.

The second RadMenuItem is not given a NavigateUrl but instead gets a Value property. Finally, the ItemClick event handler is hooked up. When the user clicks the first item, a second browser window pops up to display the Telerik web site. When the second menu item is clicked, a JavaScript alert dialog displays "The value for clicked item is: 123".

Find the code for this project in \VS Projects\Navigation\ServerSide.

[VB] Adding Items

```
Imports Telerik.Web.UI
Namespace ServerSide
Public Partial Class _Default
Inherits System.Web.UI.Page
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
If Not IsPostBack Then
' 1) create the item instance
Dim menuItem As New RadMenuItem()
```

```

' 2) populate properties
menuItem.Text = "Visit the Telerik Web Site"
menuItem.NavigateUrl = "http://www.telerik.com"
menuItem.Target = "_blank"
' add the instance to the Items collection
RadMenu1.Items.Add(menuItem)
' Add a second item that has a value, but no NavigateUrl
Dim menuItem2 As New RadMenuItem()
menuItem2.Text = "Display a value"
menuItem2.Value = "123"
RadMenu1.Items.Add(menuItem2)
' Hook up the ItemClick server side event handler.
AddHandler RadMenu1.ItemClick, AddressOf RadMenu1_ItemClick
End If
End Sub
Sub RadMenu1_ItemClick(ByVal sender As Object, ByVal e As RadMenuEventArgs)
' Retrieve the Item object returned in RadMenuEventArgs and
' display the associated Value property
RadAjaxManager1.Alert("The value for clicked item is: " + e.Item.Value)
End Sub
End Class
End Namespace

```

[C#] Adding items

```

using Telerik.Web.UI;
namespace ServerSide
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
            {
                // 1) create the item instance
                RadMenuItem menuItem = new RadMenuItem();
                // 2) populate properties
                menuItem.Text = "Visit the Telerik Web Site";
                menuItem.NavigateUrl = "http://www.telerik.com (http://www.telerik.com/)";
                menuItem.Target = "_blank";
                // add the instance to the Items collection
                RadMenu1.Items.Add(menuItem);
                // Add a second item that has a value, but no NavigateUrl
                RadMenuItem menuItem2 = new RadMenuItem();
                menuItem2.Text = "Display a value";
                menuItem2.Value = "123";
                RadMenu1.Items.Add(menuItem2);
                // Hook up the ItemClick server side event handler.
                RadMenu1.ItemClick += new RadMenuEventHandler(RadMenu1_ItemClick);
            }
        }
        void RadMenu1_ItemClick(object sender, RadMenuEventArgs e)
        {
            // Retrieve the Item object returned in RadMenuEventArgs and
            // display the associated Value property
            RadAjaxManager1.Alert("The value for clicked item is: " + e.Item.Value);
        }
    }
}

```

```
    }  
  }  
}
```

Using Server Tags

You can use server tags to keep values in markup synchronized with your server code. Say you have a class called "MyConstants" that defines all the actions your navigation control will take. Notice that in the example below the namespace is called "ServerTags".

Find the code for this project in \VS Projects\Navigation\ServerTags.

[VB] Defining Constants

```
Namespace ServerTags  
  Public Class MyConstants  
    Public Const PURCHASE_TICKETS As String = "PURCHASE_TICKETS"  
    Public Const PRINT_ITINERARY As String = "PRINT_ITINERARY"  
    Public Const CHANGE_FLIGHTS As String = "CHANGE_FLIGHTS"  
  End Class  
End Namespace
```

[C#] Defining Constants

```
namespace ServerTags  
{  
  public class MyConstants  
  {  
    public const string PURCHASE_TICKETS = "PURCHASE_TICKETS";  
    public const string PRINT_ITINERARY = "PRINT_ITINERARY";  
    public const string CHANGE_FLIGHTS = "CHANGE_FLIGHTS";  
  }  
}
```

Now you can refer to these constants in both the markup and the server code. Here is markup for a RadTabStrip. Notice that the constants are emitted from the server using <%= %>.

[ASP.NET] Markup using server tags

```
<%@ Import Namespace="ServerTags" %>  
<telerik:RadTabStrip ID="RadTabStrip1" runat="server"  
  ontabclick="RadTabStrip1_TabClick" SelectedIndex="1">  
  <Tabs>  
    <telerik:RadTab runat="server" Text="Purchase Tickets"  
      Value="<%=MyConstants.PURCHASE_TICKETS%>">  
    </telerik:RadTab>  
    <telerik:RadTab runat="server" Text="Print Itinerary"  
      Value="<%=MyConstants.PRINT_ITINERARY%>" Selected="True">  
    </telerik:RadTab>  
    <telerik:RadTab runat="server" Text="Change Flights"  
      Value="<%=MyConstants.CHANGE_FLIGHTS%>">  
    </telerik:RadTab>  
  </Tabs>  
</telerik:RadTabStrip>
```

In order for this to work notice that you need to import the "ServerTags" namespace to the markup. In the server code you can use the MyConstants class to determine which item was clicked on:

[VB] Using the constants in server code

```
Protected Sub RadTabStrip1_TabClick(ByVal sender As Object, ByVal e As
```

```

Telerik.Web.UI.RadTabStripEventArgs)
Select Case e.Tab.Value
Case MyConstants.PURCHASE_TICKETS
' do something...
RadAjaxManager1.Alert("You clicked on " + e.Tab.Text)
Exit Select
Case Constants.PRINT_ITINERARY
' do something...
Exit Select
Case Constants.CHANGE_FLIGHTS
' do something...
Exit Select
End Select
End Sub

```

[C#] Using the constants in server code

```

protected void RadTabStrip1_TabClick(object sender, Telerik.Web.UI.RadTabStripEventArgs e)
{
switch (e.Tab.Value)
{
case MyConstants.PURCHASE_TICKETS:
{
// do something...
RadAjaxManager1.Alert("You clicked on " + e.Tab.Text);
break;
}
case Constants.PRINT_ITINERARY:
{
// do something...
break;
}
case Constants.CHANGE_FLIGHTS:
{
// do something...
break;
}
}
}
}

```

Adding Multiple Levels

For navigation controls that allow multiple levels of hierarchy, each item has its own Items collection. The following example demonstrates adding an "Edit" menu item, with "Cut", "Copy" and "Paste" items beneath the "Edit". The example shows how to associate images with items. The sample project has an \Image directory where the images are stored.



You can find sample images to work with in the Visual Studio 2008 directory: \Microsoft Visual Studio 9.0 \Common7\VS2008ImageLibrary\1033\VS2008ImageLibrary.zip

The example populates the ImageUrl property using image paths within the project. First the Edit item is

UI for ASP.NET AJAX

created, populated and added to the RadMenu Items collection. Then child items are created and added to the top level menu item, Items collection using the **Add()** method. **Note:** *You can also use the **Insert(index, RadMenuItem)** method to place a menu item anywhere in the collection.*

Find the code for this project in \Navigation\ServerSide2.

[VB] Adding Multiple Level Items

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Create, populate and add the top level menu item to
    ' the RadMenu Items collection
    Dim editItem As New RadMenuItem("Edit")
    editItem.ImageUrl = "\\Images\EditInformationHS.png"
    RadMenu1.Items.Add(editItem)
    ' Create, populate and add child menu items to the edit
    ' menu item Items collection.
    Dim cutItem As New RadMenuItem("Cut")
    cutItem.ImageUrl = "\\Images\CutHS.png"
    editItem.Items.Add(cutItem)
    Dim copyItem As New RadMenuItem("copy")
    copyItem.ImageUrl = "\\Images\copyHS.png"
    editItem.Items.Add(copyItem)
    Dim pasteItem As New RadMenuItem("paste")
    pasteItem.ImageUrl = "\\Images\pasteHS.png"
    editItem.Items.Add(pasteItem)
End Sub
```

[C#] Adding Multiple Level Items

```
protected void Page_Load(object sender, EventArgs e)
{
    // Create, populate and add the top level menu item to
    // the RadMenu Items collection
    RadMenuItem editItem = new RadMenuItem("Edit");
    editItem.ImageUrl = "\\Images\\EditInformationHS.png";
    RadMenu1.Items.Add(editItem);
    // Create, populate and add child menu items to the edit
    // menu item Items collection.
    RadMenuItem cutItem = new RadMenuItem("Cut");
    cutItem.ImageUrl = "\\Images\\CutHS.png";
    editItem.Items.Add(cutItem);
    RadMenuItem copyItem = new RadMenuItem("copy");
    copyItem.ImageUrl = "\\Images\\copyHS.png";
    editItem.Items.Add(copyItem);
    RadMenuItem pasteItem = new RadMenuItem("paste");
    pasteItem.ImageUrl = "\\Images\\pasteHS.png";
    editItem.Items.Add(pasteItem);
}
```

Deleting Items

To delete an item, call the navigation control Remove() method and pass the item object to be deleted, or call RemoveAt() and pass the index of the item to be removed. The example below shows two different ways to remove the first item in the collection.

[VB] Deleting an Item

```
RadToolBar1.Items.Remove(RadToolBar1.Items(0))
RadToolBar1.Items.RemoveAt(0)
```

[C#] Deleting an Item

```
RadToolBar1.Items.Remove(RadToolBar1.Items[0]);
RadToolBar1.Items.RemoveAt(0);
```

In the case of RadPanelBar, use the Remove() method of the RadPanelBarItemCollection object that contains it (see the "Locating Items" example coming up next).

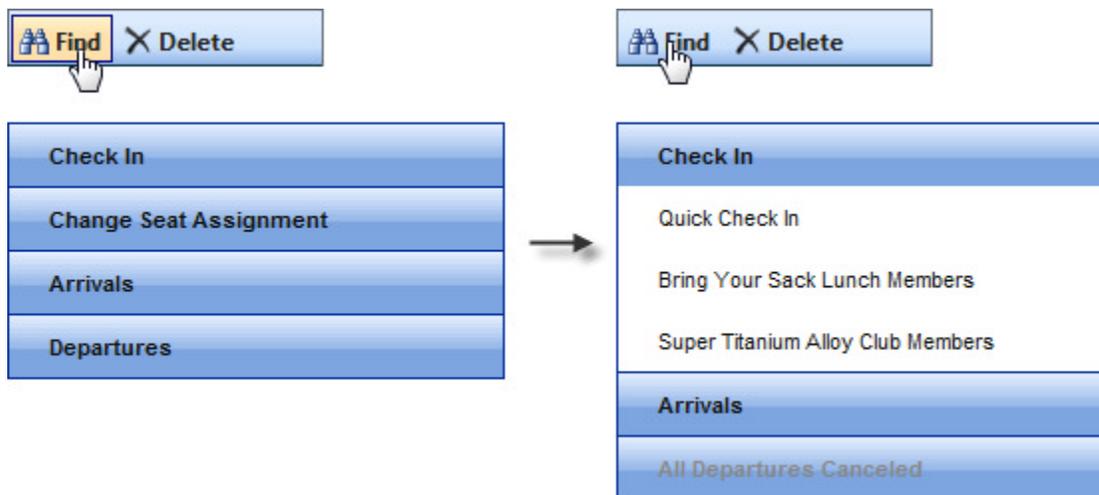
Locating Items

Each navigation controls Items collection comes with a rich set of server-side methods for locating items. All three methods pass back an item instance of found (e.g. RadMenuItem, RadPanelBarItem, etc) or null if not found.

- **FindItemByText(string text):** Pass a string that matches the Text property of an item.
- **FindItemByValue(string value):** Pass a string that matches the Value of an item.
- **FindItemByAttribute(string attributeName, string value):** This one is a little trickier. You can add arbitrary attributes to an item's markup and this method searches by the name and value of the attribute. For example, you could give a RadMenuItem a custom attribute and value, for example 'Category="Clothing"'. then call FindItemByAttribute("Category", "Clothing").

You can find this next project at \VS Projects\navigation\ServerLocatingItems.

The example uses a RadToolBar to initiate the find, looking for items in a RadPanelBar by Text, Value and Attribute. Items are then expanded, hidden and disabled. The screenshots below show the before and after state of the PanelBar.



Review the markup below and notice that each PanelBar item is populated with Text, a unique Value and a "Priority" attribute. The custom attribute "Priority" may be "Low", "Medium" and "High". Also notice that the RadToolBar has an OnButtonClick event handler defined.

[ASP.NET] PanelBar Items Markup

```
<telerik:RadToolBar ID="RadToolBar1" Runat="server"
onbuttonclick="RadToolBar1_ButtonClick" Skin="Outlook" Width="155px">
<Items>
<telerik:RadToolBarButton runat="server" ImageUrl="~/Images/FindHS.png"
Text="Find">
</telerik:RadToolBarButton>
<telerik:RadToolBarButton runat="server" ImageUrl="~/Images/DeleteHS.png"
Text="Delete">
</telerik:RadToolBarButton>
</Items>
```

```
</telerik:RadToolBar>
<br />
<br />
<br />
<telerik:RadPanelBar ID="RadPanelBar1" runat="server" Skin="Outlook" >
  <Items>
    <telerik:RadPanelItem Text="Check In" Value="1" Priority="False" >
      <Items>
        <telerik:RadPanelItem Text="Quick Check In" Value="11" Priority="Low" >
        </telerik:RadPanelItem>
        <telerik:RadPanelItem Text="Bring Your Sack Lunch Members" Value="12" Priority="Low"
        >
          </telerik:RadPanelItem>
        <telerik:RadPanelItem Text="Super Titanium Alloy Club Members" Value="13"
Priority="High" >
          </telerik:RadPanelItem>
        </Items>
      </telerik:RadPanelItem>
      <telerik:RadPanelItem Text="Change Seat Assignment" Value="2" Priority="Low"
></telerik:RadPanelItem>
      <telerik:RadPanelItem Text="Arrivals" Value="3" Priority="Medium">
</telerik:RadPanelItem>
      <telerik:RadPanelItem Text="Departures" Value="4" Priority="Medium"
></telerik:RadPanelItem>
    </Items>
  </telerik:RadPanelBar>
```

When the "Find" button is clicked, the FindItemByText() method looks for a RadPanelItem with text "Check In". If the item is found, the item is expanded to expose three other child items. A second search is performed looking for a top level attribute called "Priority" with a value of "Low" and if found, makes the item invisible. By the way, this search only looks at the top level nodes. What if you want to search all items, at all levels of the hierarchy? We will get to a solution to that problem in a minute. The last search looks for a top level item with a Value of "4" and disables it.

[VB] Finding Items

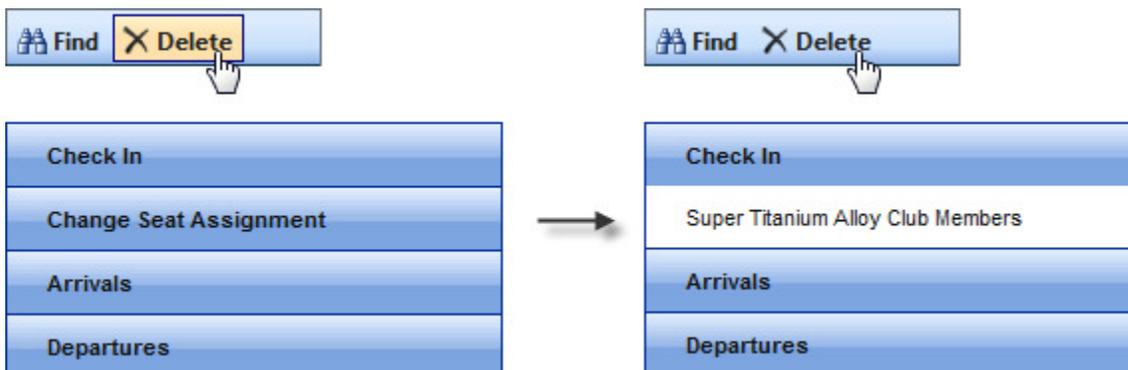
```
Protected Sub RadToolBar1_ButtonClick(ByVal sender As Object, ByVal e As
RadToolBarEventArgs)
  Select Case e.Item.Text
    Case "Find"
      ' locate the top level item with text "Check In" and expand it
      Dim checkInItem As RadPanelItem = RadPanelBar1.Items.FindItemByText("Check In")
      If checkInItem <> Nothing Then
        checkInItem.Expanded = True
      End If
      ' locate the top level item with an attribute "Priority", value "Low" and hide it
      Dim lowPriorityItem As RadPanelItem = RadPanelBar1.Items.FindItemByAttribute("Priority",
"Low")
      If lowPriorityItem <> Nothing Then
        lowPriorityItem.Visible = False
      End If
      ' locate a top level item with a value of "4", change its text and disable it.
      Dim departuresItem As RadPanelItem = RadPanelBar1.Items.FindItemByValue("4")
      If departuresItem <> Nothing Then
        departuresItem.Text = "All Departures Canceled"
        departuresItem.Enabled = False
      End If
```

```

Exit Select
...
[C#] Finding Items
protected void RadToolBar1_ButtonClick(object sender, RadToolBarEventArgs e)
{
    switch (e.Item.Text)
    {
        case "Find":
            {
                // locate the top level item with text "Check In" and expand it
                RadPanelItem checkInItem = RadPanelBar1.Items.FindItemByText("Check In");
                if (checkInItem != null)
                {
                    checkInItem.Expanded = true;
                }
                // locate the top level item with an attribute "Priority", value "Low" and hide it
                RadPanelItem lowPriorityItem = RadPanelBar1.Items.FindItemByAttribute("Priority",
"Low");
                if (lowPriorityItem != null)
                {
                    lowPriorityItem.Visible = false;
                }
                // locate a top level item with a value of "4", change its text and disable it.
                RadPanelItem departuresItem = RadPanelBar1.Items.FindItemByValue("4");
                if (departuresItem != null)
                {
                    departuresItem.Text = "All Departures Canceled";
                    departuresItem.Enabled = false;
                }
            }
            break;
        }
    }
}
...

```

A more flexible way to work on Items is to iterate the collection. Let's say we want to delete all items that are "low priority". The screenshot shows the before and after state of the PanelBar. The first item "Check In" is collapsed but contains three items. After the "Delete" button is clicked, the two "Low priority" items are deleted, and the top level "Change Seat Assignment" item is deleted.



First we need to get all items, not just root level items for a given Items collection. To do this, call the navigation control's **GetAllItems()** method. This will return a generic **IList** collection containing all items, at all levels. You can then iterate your **IList** and perform operations on each item.

[VB] Deleting Items

UI for ASP.NET AJAX

```
Protected Sub RadToolBar1_ButtonClick(ByVal sender As Object, ByVal e As
RadToolBarEventArgs)
    Select Case e.Item.Text
        '...
        Case "Find"
        Case "Delete"
            ' get all the items in the panel bar Items collection
            Dim allItems As System.Collections.Generic.IList(Of RadPanelItem) =
RadPanelBar1.GetAllItems()
            ' iterate all items
            For Each item As RadPanelItem In allItems
                item.Expanded = True
                ' remove all "low priority" items, i.e. that have a "Priority" attribute with a value of
"Low"
                If item.Attributes("Priority").Equals("Low") Then
                    ' To remove a panel item, use the Remove method of the RadPanelItemCollection
                    ' object that contains it
                    item.Owner.Items.Remove(item)
                End If
            Next
        Exit Select
    End Select
End Sub
```

[C#] Deleting Items

```
protected void RadToolBar1_ButtonClick(object sender, RadToolBarEventArgs e)
{
    switch (e.Item.Text)
    {
        case "Find":
        {
            //...
        }
        case "Delete":
        {
            // get all the items in the panel bar Items collection
            System.Collections.Generic.IList<RadPanelItem> allItems = RadPanelBar1.GetAllItems();
            // iterate all items
            foreach (RadPanelItem item in allItems)
            {
                item.Expanded = true;
                // remove all "low priority" items, i.e. that have a "Priority" attribute with a
value of "Low"
                if (item.Attributes["Priority"].Equals("Low"))
                {
                    // To remove a panel item, use the Remove method of the RadPanelItemCollection
                    // object that contains it
                    item.Owner.Items.Remove(item);
                }
            }
            break;
        }
    }
}
```

2.6 Control Specifics

PageView and Multi-Page

A typical tabbed interface lets the user click a tab to see content that corresponds to the tab text. For example, a home building supplies online store would have tabs for "Appliances", "Tools" and "Building Materials". When the user clicks "Appliances", a list of appliance descriptions, images, and links displays. RadMultiPage used with RadTabStrip makes this kind of interface easy to build.

Use the **RadMultiPage** control to organize content of tabbed pages. RadMultiPage acts as a container for **RadPageView** controls, where you typically have a RadPageView holding content of a page associated with a **RadTabStrip** tab. RadMultiPage is a completely separate control from RadTabStrip and can be positioned anywhere on the page.

Even though RadMultiPage and RadTabStrip can be used independently of each other, these controls are best used together. To automatically synchronize tabs with corresponding pages, set the **MultiPageID** property of RadTabStrip to the ID of a RadMultiPage control. By default, the tabs and pages will correspond based on index. When the user clicks on the first tab, the first page view displays; when the second tab is clicked, the second page view displays, and so on. If you don't want this default behavior and instead want to link particular tabs to page views, use the tab **PageViewID** property to link specific page views.

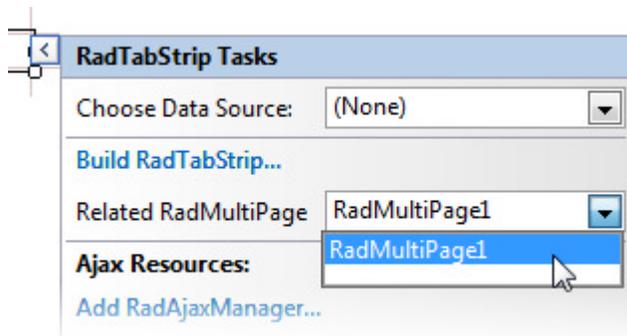


- The order of tabs is "depth first"; that is, the children of the first tab are before the second root level tab.
- If there are more page views than tabs, the last page views are ignored.
- If there are more tabs than page views, the last tabs do not display a page view.

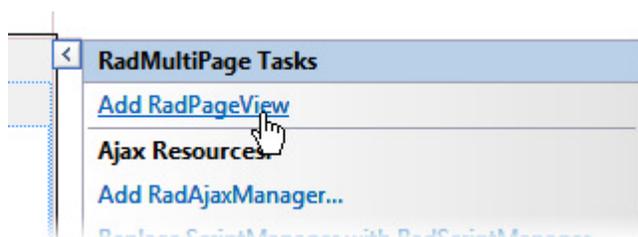
TabStrip and MultiPage Walk-through

You can find this project in \VS Projects\navigation\MultiPage.

1. Create a Web Application and add a ScriptManager to the default page..
2. Add a RadTabStrip to the default page. Set the **Skin** property to "Black".
3. Add a RadMultiPage to the default page. **Note:** *The multipage control is a container only and has no Skin property.*
4. From the RadTabStrip Smart Tag, select the RadMultiPage from the "Related RadMultiPage" drop down list.

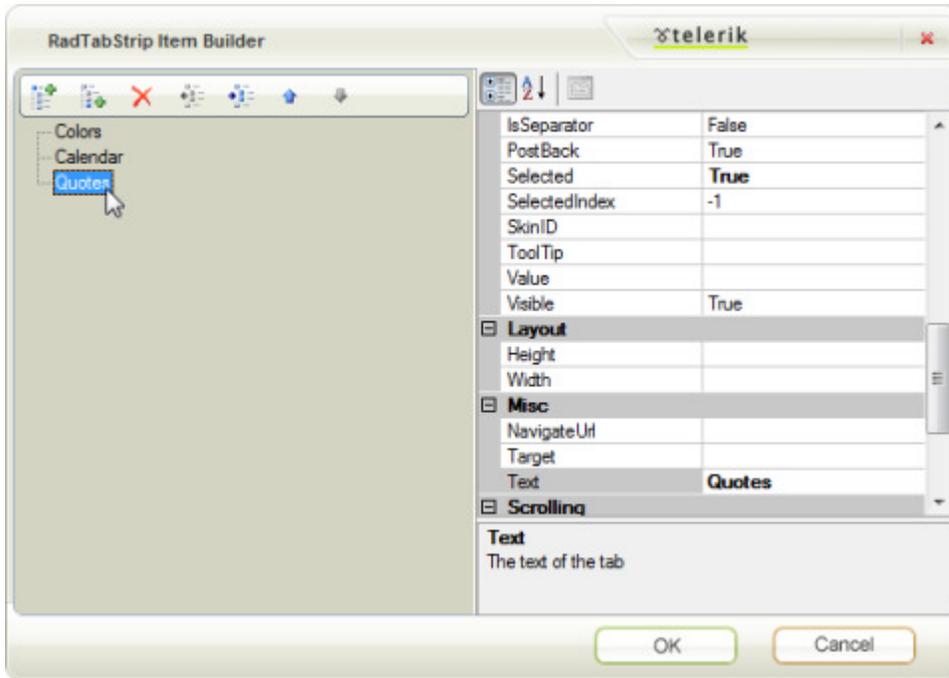


5. From the RadMultiPage Smart Tag, select the "Add RadPageView" link twice. RadMultiPage starts with a single PageView by default, so you should have three PageViews at this point.

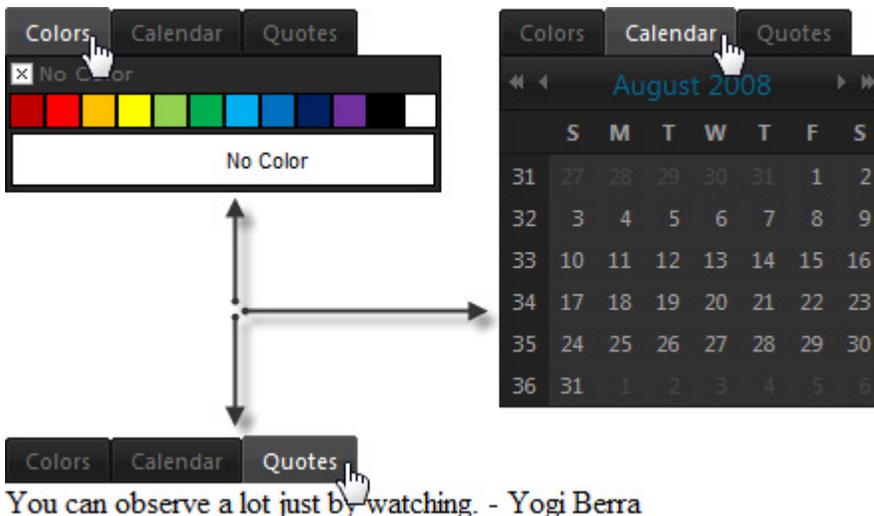


UI for ASP.NET AJAX

- From the RadTabStrip Smart Tag, select the **Build RadTabStrip...** link.
- Add three root level tabs and set the **Text** properties to "Colors", "Calendar" and "Quotes". Click **OK** to close the dialog.



- In the designer, drop a RadColorPicker into the first PageView, a RadCalendar control to the second PageView and enter the quote "You can observe a lot just by watching - Yogi Berra" directly into the last PageView. Set the Skin property for the RadCalendar and RadColorPicker to "Black". In addition, set the *Width* property of RadColorPicker to "220px" and *Preset* property to "Standard".
- Press **Ctrl-F5** to run the application.

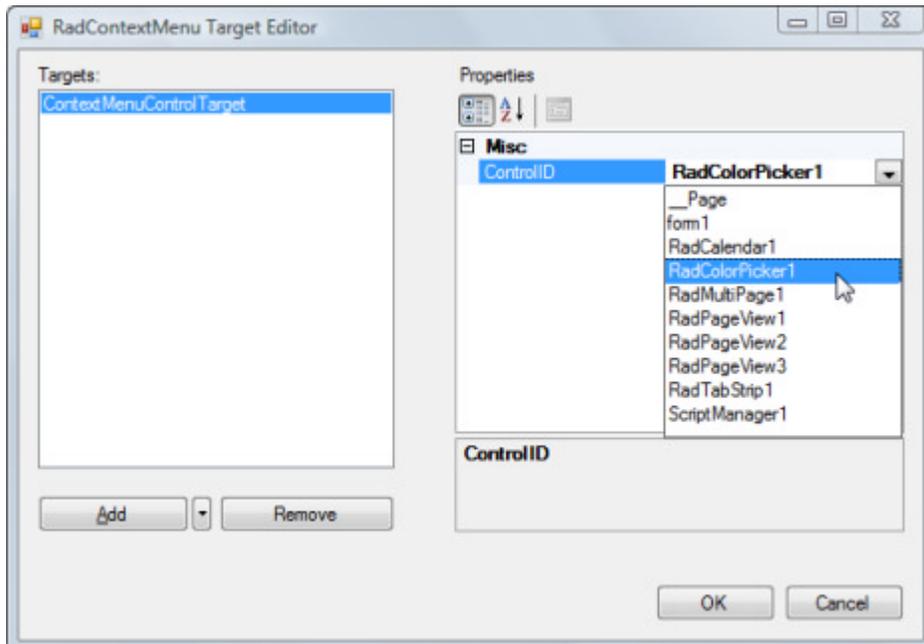


⚠ That's a lot of functionality from just dragging and dropping. But be aware that all the content is present on the page whether it is visible or not. For better performance and scalability, larger applications will need to bring in content dynamically. We will talk about one way to do this with AJAX and user controls in the upcoming "AjaxPanel, AjaxManager and AjaxProxy".

Context Menus

RadContextMenu is similar to RadMenu but has some unique aspects. The menu isn't visible when the page first loads but is launched by client code or by specifying a "target". The target is some item in the ASP.NET markup. When the user right-clicks that item, the context menu is displayed. The target can be a HTML element, the document element (the user right-clicks the page to show the menu), a control or a tag name.

1. Starting with the "MultiPage" project you created in the last chapter, add a RadContextMenu control to the default web page.
2. From the context menu Smart Tag select the **Build RadContextMenu...** option. Add a single root level item with Text "Colors".
3. From the context menu Smart Tag select the **Edit RadContextMenu Target's** option to display the RadContextMenu Target Editor. **Note:** you can also get to this dialog from the Properties Window using the *Targets* property ellipses.
4. Click the **Add** button to create a ContextMenuControlTarget. **Note:** Use the downward pointing arrow next to the Add split button for other target types, i.e. control, element, tag name and document.
5. Click the drop down arrow on the **ControlID** property in the Properties window and select the RadColorPicker. Click the **OK** button to close the dialog.



6. In the Properties window for the RadContextMenu, select the Events button  and double click the **ItemClick** event to create an event handler. Add the following code to the ItemClick and the Page_Load event handlers:

[VB] Handling the Page_Load and ItemClick Events

```
Imports Telerik.Web.UI
Namespace MultiPage
Public Partial Class _Default
Inherits System.Web.UI.Page
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
If Not IsPostBack Then
' get the parent menu item
```

```

Dim colorsItem As RadMenuItem = RadContextMenu1.Items.FindItemByText("Colors")
' iterate the color picker color presets
For Each preset As ColorPreset In [Enum].GetValues(GetType(ColorPreset))
' add color preset names as child items
colorsItem.Items.Add(New RadMenuItem(preset.ToString()))
Next
End If
End Sub
Protected Sub RadContextMenu1_ItemClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadMenuEventArgs)
' look only at child items
If e.Item.Level = 1 Then
' find child items under "Colors" parent item
If (TryCast(e.Item.Parent, RadMenuItem)).Text.Equals("Colors") Then
' set the color picker preset to the selected preset
RadColorPicker1.Preset = DirectCast([Enum].Parse(GetType(ColorPreset), e.Item.Text),
ColorPreset)
End If
End If
End Sub
End Class
End Namespace

```

[C#] Handling the Page_Load and ItemClick Events

```

using Telerik.Web.UI;
namespace MultiPage
{
public partial class _Default : System.Web.UI.Page
{
protected void Page_Load(object sender, EventArgs e)
{
if (!IsPostBack)
{
// get the parent menu item
RadMenuItem colorsItem = RadContextMenu1.Items.FindItemByText("Colors");
// iterate the color picker color presets
foreach (ColorPreset preset in Enum.GetValues(typeof(ColorPreset)))
{
// add color preset names as child items
colorsItem.Items.Add(new RadMenuItem(preset.ToString()));
}
}
}
protected void RadContextMenu1_ItemClick(object sender, Telerik.Web.UI.RadMenuEventArgs
e)
{
// look only at child items
if (e.Item.Level == 1)
{
// find child items under "Colors" parent item
if ((e.Item.Parent as RadMenuItem).Text.Equals("Colors"))
{
// set the color picker preset to the selected preset
RadColorPicker1.Preset = (ColorPreset)Enum.Parse(typeof(ColorPreset), e.Item.Te

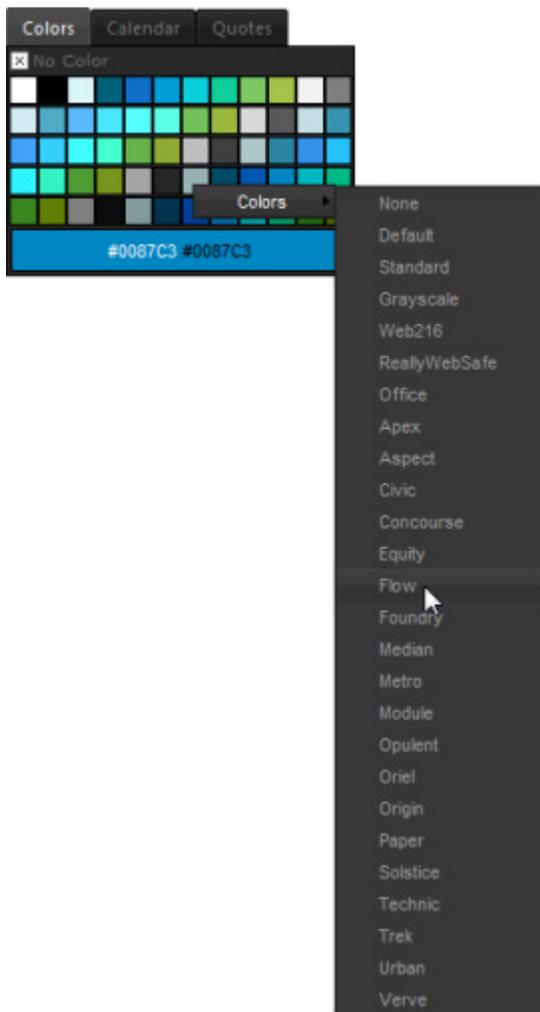
```

```
}
}
}
}
```

The Page_Load first gets a reference to the "Colors" menu item. This is overkill for a single menu item, but you will need to find items when the number and complexity of menu items grows. The RadColorPicker has a ColorPreset enumeration that defines all available preset color groups. You can use the Enum.GetValues() static method to walk through the enumeration and add children to the "Colors" menu item.

In the ItemClick event handler we look only at child menu items and make sure that the parent is the "Colors" menu item. Then you can use the Enum.Parse() static method to convert the preset name to its actual ColorPreset value and assign it back to the RadColorPicker Preset property.

7. Press Ctl-F5 to run the application. Right click the color picker control to display the context menu.



2.7 Summary

In this chapter you took a tour of the "navigation" related RadControls and became familiar with how and where each of these controls are used. You saw some of the important properties, methods and events that are common between navigation controls. You created a simple application that used the menu, tab strip and tool bar controls. You learned some common server-side tasks such as add/edit/delete, iterating items in a

UI for ASP.NET AJAX

collection and locating items based on various criteria (i.e. text, value or attribute). You also learned some control-specific tasks such as working with the tab strip and Multi-Page together and using the context menu.

3 Input Controls

3.1 Objectives

- Inventory the "input" related RadControls. Explore how and where these input controls are used.
- See how each of the input controls are similar so you can leverage the same knowledge with each control.
- Create a simple application to get confidence in using each of the controls.
- Explore the design time interface for each of the input controls, again noting where the controls are similar. You will learn how to access properties and methods through Smart Tag, Properties Window and control-specific dialogs.
- Explore principal properties and groups of properties where 80% of the functionality is found.
- Learn how to perform common server-side tasks such as creating controls dynamically, setting values, and responding to changed values.
- Learn how to perform common client-side tasks such as enabling and disabling, restricting input as the user types, and handling parsing errors.
- Learn to use the input controls with other controls such as RadSpellCheck or ASP.NET validator controls.

3.2 Introduction

Often, you want to create a Web application that collects data from the users who visit your Web site. This data can be anything from details for shipping and billing to an elaborate survey form. RadControls make it easy to collect information from users, whether it is generic text or typed data such as numbers and dates. You can choose from several types of input controls, depending on what type of data you want users to enter:

RadTextBox

RadTextBox is a highly configurable input control that lets users enter arbitrary text values. Users can enter any type of character into RadTextBox (alphabetic, numeric, and symbols). RadTextBox supports three different modes:

Single-line mode lets users enter short values that fit on a single line.

SingleLine:

Multi-line mode lets users enter longer values that can take up several lines:

MultiLine:

Password mode hides the characters that users type so that it can be used for entering sensitive information such as passwords:

Password:

From Q2 2011 on the RadTextBox control offers a password strength checking feature. The same can be used inside a TextBoxSetting created with a RadInputManager.

The feature allows you to specify your custom criteria for password strength and visualize an indicator to inform the user how strong the typed password is according to this criteria.

RadMaskedTextBox

RadMaskedTextBox is similar to RadTextBox, allowing both single- or multi-line modes. However, it is designed to restrict user input to values that conform to a strict format. The input format is controlled by a special string called a mask. You can select from a variety of built-in masks for common patterns such as phone numbers or social security numbers, or you can construct your own custom masks.

RadMaskedTextBox prompts the user to enter data in the required format by displaying a prompt character of your choosing for all text the user should enter, along with literal parts that the mask supplies. In the screenshot below, which shows RadMaskedTextBox using a mask for a telephone number, the prompt character is an underscore ('_').

Phone: () -

RadNumericTextBox

RadNumericTextBox restricts users to entering numeric values. This control supports a wide variety of formatting options; you can rely on the local culture setting to format number, currency, or percentage values, or you can supply your own detailed formatting specifications.

Number: 1.23
Currency: \$1.23
Percent: 1.23 %

While users can always type numbers into RadNumericTextBox, you can also let them change the current value by simply incrementing or decrementing it. You can let users increment or decrement the current value in any or all of the following ways:

- Spin buttons can be added to the right or left of the input area.
- Mouse wheel support can be enabled to let users change the value using the mouse wheel when the numeric text box has focus.
- Arrow key support can be enabled to let users change the value using the up and down arrow keys.

RadDateInput

Use RadDateInput to let users enter date and time values. RadDateInput is a free-form date and time input control. That is, it has a built-in parsing engine that can recognize date and time values in a wide variety of valid formats, so that you do not need to restrict users to a limited format in order to interpret values. The parsing engine is culture-sensitive, so that you can easily localize your Web application.

Like RadNumericTextBox, RadDateInput lets you control how values are formatted for display. You can specify the format using standard ASP.NET date and time format strings. You can also set the culture to control how RadDateInput interprets the culture-specific parts of those format strings (such as the names of months or days).

Date and Time: Tuesday, 12 August 2008 2:34 p.m.

Also like RadNumericTextBox, you can let users increment or decrement the current value by enabling mouse wheel or arrow key support. (You can't add spin buttons to RadDateInput, however.)

Common Features

Each of the input controls...

- Supports interaction with the clipboard, including built-in shortcut keys for cut, copy, and paste.
- Displays a built-in context menu when the user right clicks to invoke common editing tasks such as

clipboard functions or undoing the last edit.

- Can be skinned for a great visual appearance that's consistent with your entire web application. You can choose from a standard set of matched skins (e.g. "Outlook", "Vista", "Black", "Telerik", etc.) or you can create your own custom skin.
- Lets you add an integrated label and/or button on the left or right of the input area. (On RadNumericTextBox, you can also add a set of spin buttons).
- Supports tool tips that can give the user additional information about the value to be entered.
- Lets you specify the position of the caret and whether the text is selected when the input control gets focus. This lets you control how the value changes when the user first starts typing.
- Distinguishes between edit mode (when the control has focus) and display mode. Except for RadTextBox, you can specify different formatting options for edit and display mode. In display mode, you can also specify a string that appears when the value has not yet been set (even for RadTextBox).
- Can be set to ReadOnly mode when you want to use it for display purposes only.
- Supports limitations on the range of valid values. The type of range depends on the type of input control: RadTextBox lets you set the maximum length; RadMaskedTextBox lets you specify a range on parts of the mask; RadNumericTextBox and RadDateInput let you specify minimum and maximum values.
- Includes a rich, consistent client-side api for managing the value range, selection, and caret position of the input control, as well as a wide range of client events for responding to client input quickly on the browser without the need for postbacks.
- Can be optionally set to trigger postbacks when the value changes and to trigger ASP.NET validation of other controls on the page when that postback occurs.

3.3 Getting Started

In this walk-through you will become familiar with the text box, masked text box, numeric text box, and date input controls. When you are finished, your project should match the one supplied in \VS Projects\Input\GettingStarted. The input controls will produce the entry form you see below:

Name:

Phone:

Cost: 

Ship by:

Set up the project structure

1. Create a new ASP.NET Web Application.
2. In the designer, drag a ScriptManager from the AJAX extensions section of the tool box onto your page.

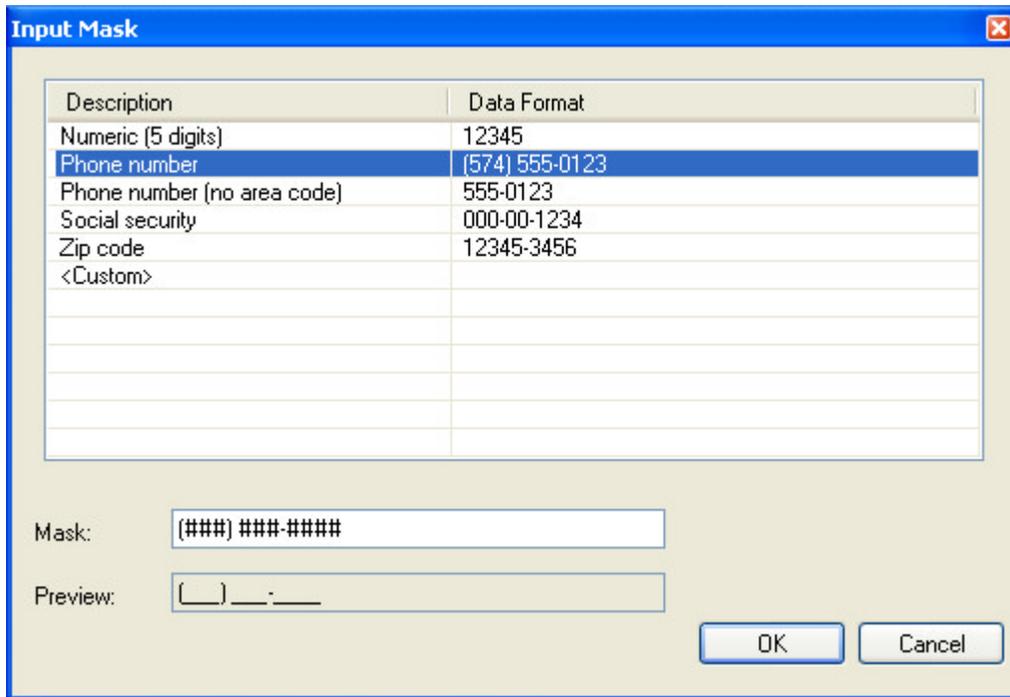
Add the RadTextBox

1. Add a RadTextBox to your web page. In the Smart Tag, set the **Empty Message** to "- Enter your name -" and select "Office2007" from the **Skin** drop-down.
2. In the Appearance section of the Properties Window, set the **Label** property to "Name: ".

3. In the Behavior section of the Properties Window, set the **MaxLength** property to 100, the **SelectionOnFocus** property to "CaretToEnd", and the **ToolTip** property to "Name to which item should be shipped."

Add the RadMaskedTextBox

1. Add a few line breaks after the RadTextBox, and then add a **RadMaskedTextBox** underneath the text box.
2. In the Smart Tag, set the **Skin** property to "Office2007" and then click the link labeled **SetMask**.
3. The Input Mask Dialog appears. Select the row for Phone Number to select a pre-defined mask, and hit OK:



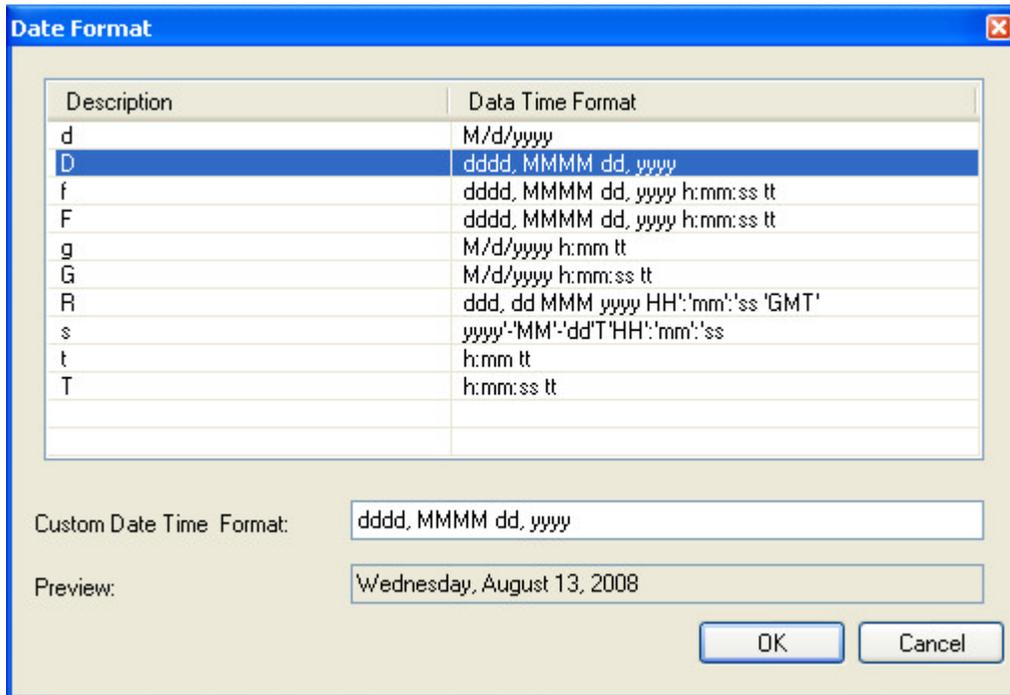
4. In the Appearance section of the Properties Window, set the **Label** property to "Phone: ".
5. In the Behavior section of the Properties Window, set the **EmptyMessage** property to "- Enter phone number -" and the **HideOnBlur** property to **True**. Because the **Mask** property is set, the masked text box displays the mask when no text has been entered. By setting the **HideOnBlur** property, you cause the masked text box to show the value of **EmptyMessage** instead when the control is in display mode.
6. Set the **SelectionOnFocus** property to "CaretToBeginning" and the **ToolTip** property to "Phone number of contact."

Add the RadNumericTextBox

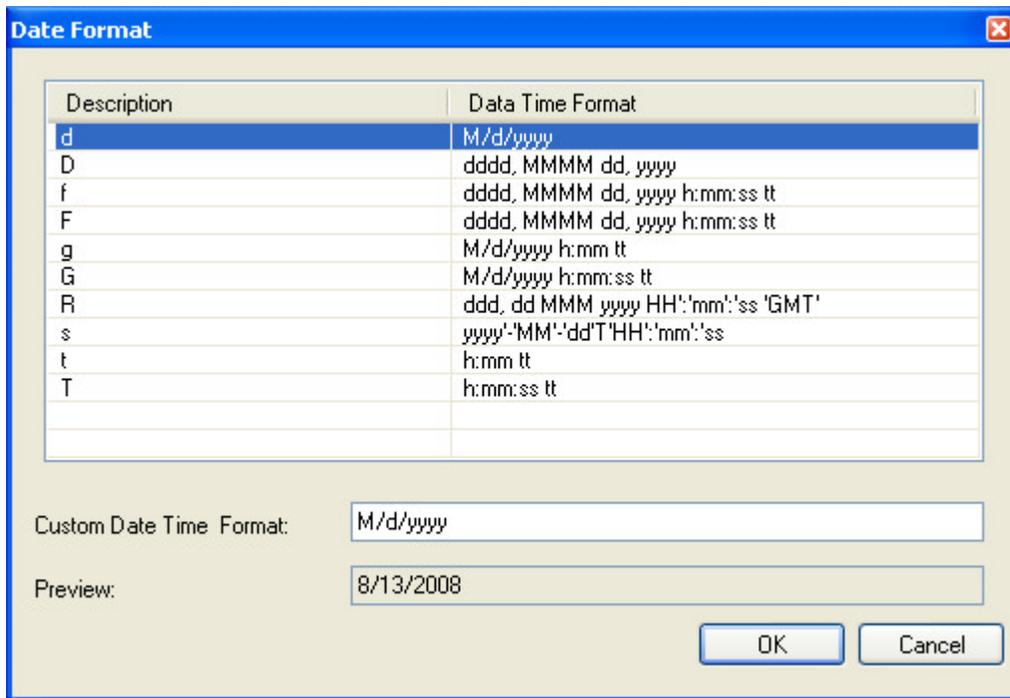
1. Add a few more line breaks after the RadMaskedTextBox, then add a **RadNumericTextBox** underneath the masked text box.
2. In the Smart Tag, set the **Numeric Type** to "Currency" and the **Skin** to "Office2007".
3. In the Appearance section of the Properties Window, set the **Label** property to "Cost: " and the **ShowSpinButtons** property to true.
4. In the Behavior section of the Properties Window, set the **EmptyMessage** property to "- Enter cost -", the **SelectionOnFocus** property to "CaretToEnd", and the **ToolTip** property to "Cost of order."
5. Set the **MinValue** property to 0 and the **MaxValue** property to 10000.

Add the RadDateInput

1. Add a few more line breaks after the RadNumericTextBox, and then add a **RadDateInput** underneath the numeric text box.
2. In the Smart Tag, set the **Skin** to "Office2007", and then click the **Set Display Date Format** link.
3. The Date Format Dialog appears. Select the row for the long date format ("D") and then hit OK:



4. In the Smart Tag again, click the **Set Date Format** link to bring up the Date Format Dialog again. This time, the date format is for edit mode. Select the row for the short date format ("d") and then hit OK:



5. In the Appearance section of the Properties Window, set the **Label** property to "Ship by: ".
6. In the Behavior section of the Properties Window, set the **EmptyMessage** property to "- Enter the ship by date -", the **ToolTip** property to "The last date the order can be shipped." and the **SelectionOnFocus** property to "SelectAll".

Run the application

1. You have just created an entry form without writing a single line of code! Press Ctl-F5 to run the application. Note that the empty messages appear for all the input controls you entered.
2. Tab around the form and enter some values. Note the differences in where the caret appears when each control gets focus, based on the **SelectionOnFocus** property. Note the tool tips that appear when the mouse hovers over an item. Note that the range you specified for the cost field is enforced.

Name:

Phone:

Cost:
Cost of order.

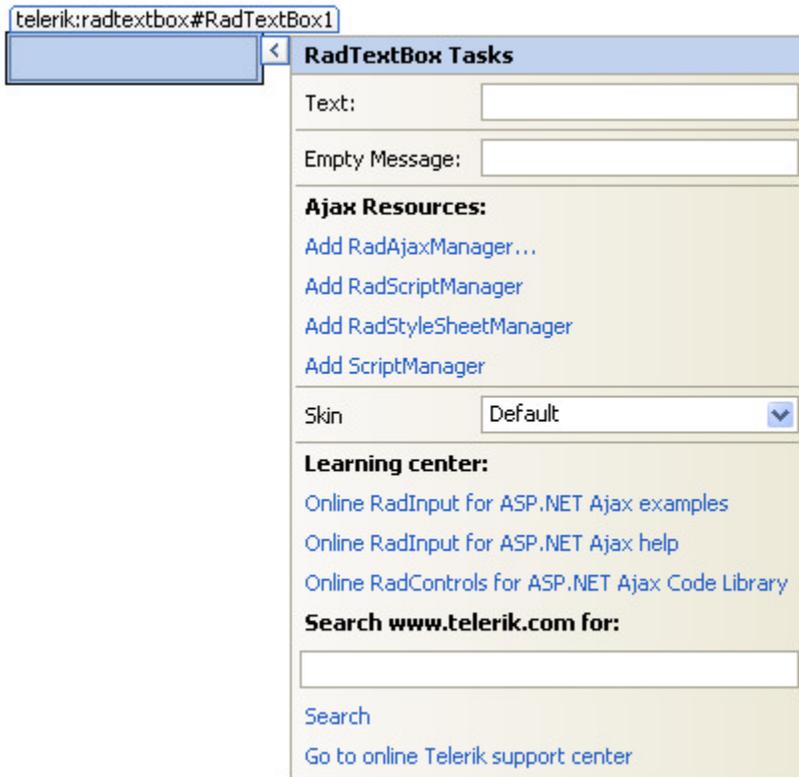
Ship by:

3.4 Designer Interface

In the Visual Studio designer, you can configure all of the input controls using the Smart Tag and the Properties Window. In addition, some of the input controls have special dialogs for specifying how you want the control to format its value.

Smart Tag

The Smart Tag provides easy access to frequently needed design tasks. You can display the Smart Tag using the small left-pointing arrow in the upper right of the control or choose "Show Smart Tag" from the context menu, just as with all other RadControls. The screenshot below shows the RadTextBox Smart Tag. As you can see, like the Smart Tags for the navigation controls, this one has some tasks at the top that are specific to the control (RadTextBox in this case), followed by Ajax Resources, Skin, and Learning center.



You have already seen the Ajax Resources, Skin selection, and Learning center when looking at the Smart Tag for the navigation controls. This time, we will just look at the Tasks that are specific to the individual input control types.

Tasks

The top portion of the Smart Tag for each type of input control lists a different set of tasks you can perform. The **RadTextBox** Smart Tag lists two tasks at the top:

- **Text** lets you set the initial value of the text box. Any string you enter here appears in the input area as the value of the text box, which the user can subsequently edit.
- **Empty Message** lets you specify a message that appears in the input area when no value has been set. Using an empty message is a convenient way to provide a prompt to the user about what data should be entered or to provide feedback that the value has not been set (as distinct from a value that is set to an empty string).

The **RadMaskedTextBox** Smart Tag lists only one task at the top, but it is an important one:

- **Set Mask** brings up the Input Mask Dialog, where you can assign the mask that the text box uses to restrict input. This dialog is described in more detail below. When you assign a mask using the Set Mask option, the mask you assign controls the data the user can enter and the way it is formatted. By default, this mask

UI for ASP.NET AJAX

controls the appearance of the text box in both edit and display modes. You can, however, assign a second mask to the `DisplayMask` property in the Properties Window, which is then used for formatting the text box's value in display mode only.

The `RadNumericTextBox` Smart Tag lists two tasks at the top:

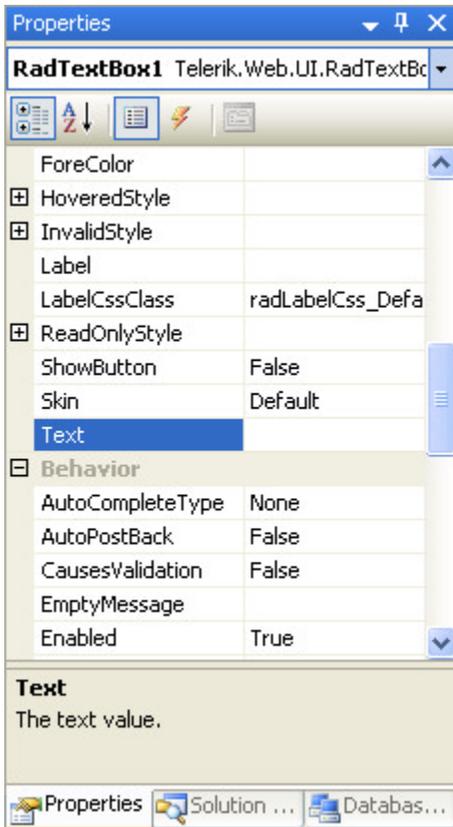
- **Numeric Type** lets you specify the type of numeric value that the numeric text box is to represent. This can be Number, Currency, or Percent. The numeric type affects the way the value is formatted when the numeric text box is in display mode. (In edit mode, the number is always formatted as a number only.)
- **Value** lets you specify the initial value of the numeric text box.

The `RadDateInput` Smart Tag lists two tasks at the top:

- **Set Display Date Format** brings up the Date Format Dialog, where you can specify the format string that is used in display mode.
- **Set Date Format** also brings up the Date Format Dialog, but this time the format string you specify is used to format the value when the text box has focus (when the user is editing its value). If you specify this format string, but not a display date format, this string is always used to format values, even when the text box does not have focus.

Properties Window

All of the properties available to the control are found in the Properties window. As before, we use the 80/20 rule here; that is, locate the most important properties and groups of properties of the input controls.



Properties for the value

Probably the most important property of any input control is the one that holds its value. While you may not always want to initialize this property at design time, you will certainly want to read the value that the user

entered when the form is posted back. Each of the different input controls uses a different property for its value:

- **RadTextBox** uses the **Text** property.
- **RadMaskedTextBox** is a little more complicated, because you may want to consider several values: with or without the literal characters of the mask, and with or without the prompt characters in the mask. As a result, there are four separate properties for the value:
 - **Text** is the value without any prompt characters or literal characters from the mask. This is the value you can set to provide an initial value.
 - **TextWithPrompt** is just what the name implies: the text plus prompt characters for any un-entered parts of the mask, but without literal characters from the mask. It is read-only.
 - **TextWithLiterals** is the text plus the literal characters from the mask (but no prompt characters). This is not read-only, so that the control can be data-bound to a source that stores values which include literals.
 - **TextWithPromptAndLiterals** has the text, plus prompt characters and literal characters from the mask. This is again read-only.
- **RadNumericTextBox** uses the **Value** property for its value. Value is a double rather than a string, so that your application does not need to worry about converting the value. If you are using the numeric text box as part of a data-bound custom control, you can use the **DbValue** property instead, so that the control can handle null values.
- **RadDateInput** uses the **SelectedDate** property. SelectedDate is, of course, a DateTime value. Like the DbValue property of RadNumericTextBox, RadDateInput has a **DbSelectedDate** property that can handle null values.

Properties for common features

The four types of input control have a lot of features in common, and these are reflected by a common set of properties. The **EmptyMessage** property, which we have already encountered on the RadTextBox Smart Tag, is available for all four types of input control. For RadMaskedTextBox, however, this property only has an effect if the **HideOnBlur** property is set to true. Other important properties include **ToolTip**, which lets you supply a message that appears when the mouse hovers over the control, **Label**, which lets you supply a text label that appears to the left of the input area, **SelectionOnFocus**, which determines the default placement of the caret and selection of text when the control gets focus, and **ReadOnly**, which lets you limit the control to display mode.

Both RadTextBox and RadMaskedTextBox let you set the **InputMode** property to SingleLine, MultiLine, or Password. When InputMode is MultiLine, the **Rows** and **Columns** properties determine the number of rows displayed, and the number of characters in each row. The **Wrap** property specifies whether text wraps when it exceeds the number of characters specified by Columns, or whether the control only honors line breaks and uses scroll bars for long lines.

RadNumericTextBox and RadDateInput let you set the **IncrementSettings** property to specify how the user can increment and decrement values. This is a composite property, with sub-properties for enabling arrow keys or mouse wheel, and for specifying the step size for each increase or decrease.

Properties governing look-and-feel

Like most RadControls, you can use the **Skin** property to set the general appearance of the input controls to match the other controls in your Web application. Predefined skins can be selected from a list or you can skip ahead to the chapter on skinning for details on building your own.

You can further craft the appearance of your input control for when it appears in different states by using the various "Style" properties. These include **EnabledStyle**, **DisabledStyle**, **EmptyMessageStyle**, **FocusedStyle**, **HoveredStyle**, **InvalidStyle**, and (in the case of RadNumericTextBox) **NegativeStyle**. Also look for properties

ending in "CssClass". These properties specify CSS classes used to style parts of the control: **CssClass** for the input area, **LabelCssClass** for the label, and **ButtonCssClass** if you have added a button to the control. On **RadNumericTextBox**, you can also find **SpinUpCssClass** and **SpinDownCssClass** for the up and down spin buttons. **LabelCssClass** may be pre-populated with a class name from the control's skin (see the chapter on Skinning for details on working with RadControls skins).

Important Properties for specific input types

Because each input control handles data of a specific type, some properties that affect the data are unique to each type of input control.

RadTextBox handles any type of input, so it does not have many of these idiosyncratic properties. There are only two important properties to mention here: The **MaxLength** property lets you limit the number of characters that users can enter when the **InputMode** is **SingleLine**. The **AutoCompleteType** property lets you make use of the AutoComplete feature of certain browsers. AutoComplete is only available for certain browsers, and usually must be enabled in the browser itself. When enabled, the browser "remembers" values that the user has entered, and when it encounters an input control with the same **AutoCompleteType** as one that was previously entered, it provides a list of previous responses for the user to select.

RadMaskedTextBox has a number of properties to let you specify the mask and how it is applied. The **Mask** property specifies the mask that is used for edit mode, while the **DisplayMask** property specifies the mask to use for display mode. If you only set the **Mask** property, it is used for both edit and display modes. In the Properties Window, you can click on the ellipsis button for these two properties to bring up the Input Mask Dialog (described in the next section), which lets you select a pre-defined mask or generate a custom mask. As an alternate approach to defining the mask, you can build up a mask part by part using the **MaskParts** and **DisplayMaskParts** properties. The ellipsis button for these two properties brings up the MaskPart Collection Editor (also described in a following section), which lets you define each mask part using properties rather than requiring you to remember the special characters used in mask strings. The **PromptChar** property lets you specify the character that is used to prompt the user for unentered data in the mask. Finally, three properties govern the way numeric ranges and enumerated values that make up part of a mask are applied.

AllowEmptyEnumerations determines whether enumerated mask parts can be left empty,

ZeroPadNumericRanges determines whether numeric range parts use leading zeros to ensure all values are fixed length, and **NumericRangeAlign** determines whether numeric range parts are right- or left-aligned (if **ZeroPadNumericRanges** is false).

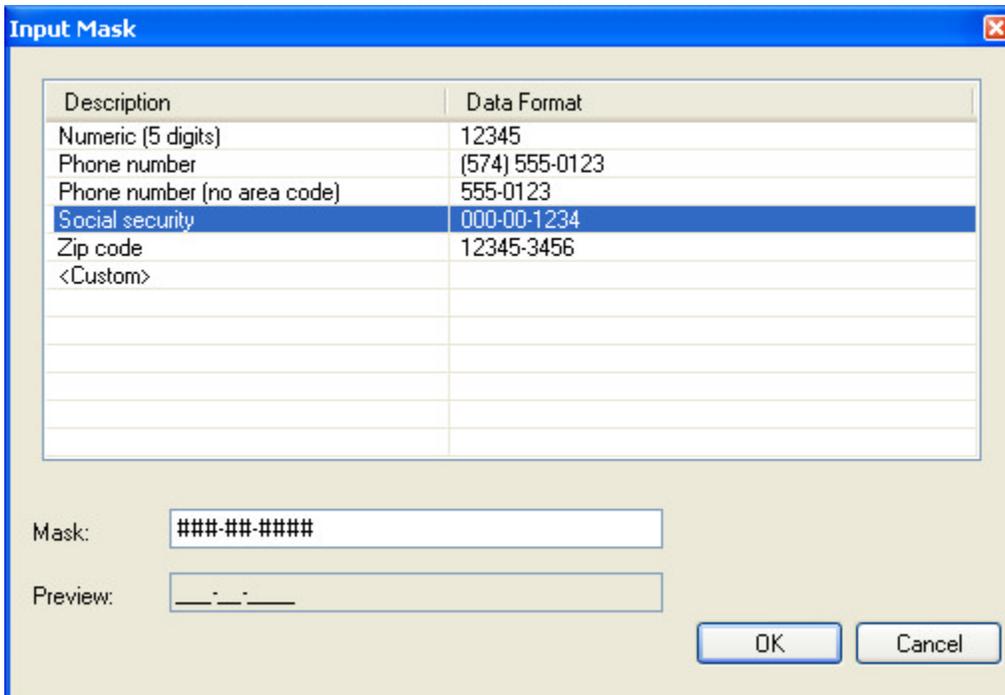
RadNumericTextBox has three properties that affect the way the value is formatted. You have already seen **Type** on the **RadNumericTextBox** Smart Tag. It lets you specify whether the value is a simple number, a currency value, or a percentage. **Culture** lets you assign a culture which influences how that type is applied, determining the currency symbol, decimal separator, and so on. **NumberFormat** lets you override the **Type** and **Culture** settings to completely control the format of values. Two properties, **MaxValue** and **MinValue**, let you set the range of valid values that the user can enter.

RadDateInput, like **RadNumericTextBox**, has a number of properties for the way values are formatted.

DateFormat and **DisplayDateFormat** specify the ASP.NET format strings for edit and display modes, respectively. You have already seen these properties on the **RadDateInput** Smart Tag. Two additional properties determine how the format string is applied: **Culture** lets you assign a culture that determines the value of culture-dependent strings such as month names (and also affects the way dates are parsed), and **ShortYearCenturyEnd** determines how two-digit year strings are interpreted. Again like **RadNumericTextBox**, there are two properties to set the range of valid values: **MaxDate** and **MinDate**.

Input Mask Dialog

The Input Mask Dialog is used to specify a mask for a **RadMaskedTextBox** control. You can display this dialog from the control's Smart Tag, or by clicking the ellipsis button next to the **Mask** or **DisplayMask** property in the Properties Window.



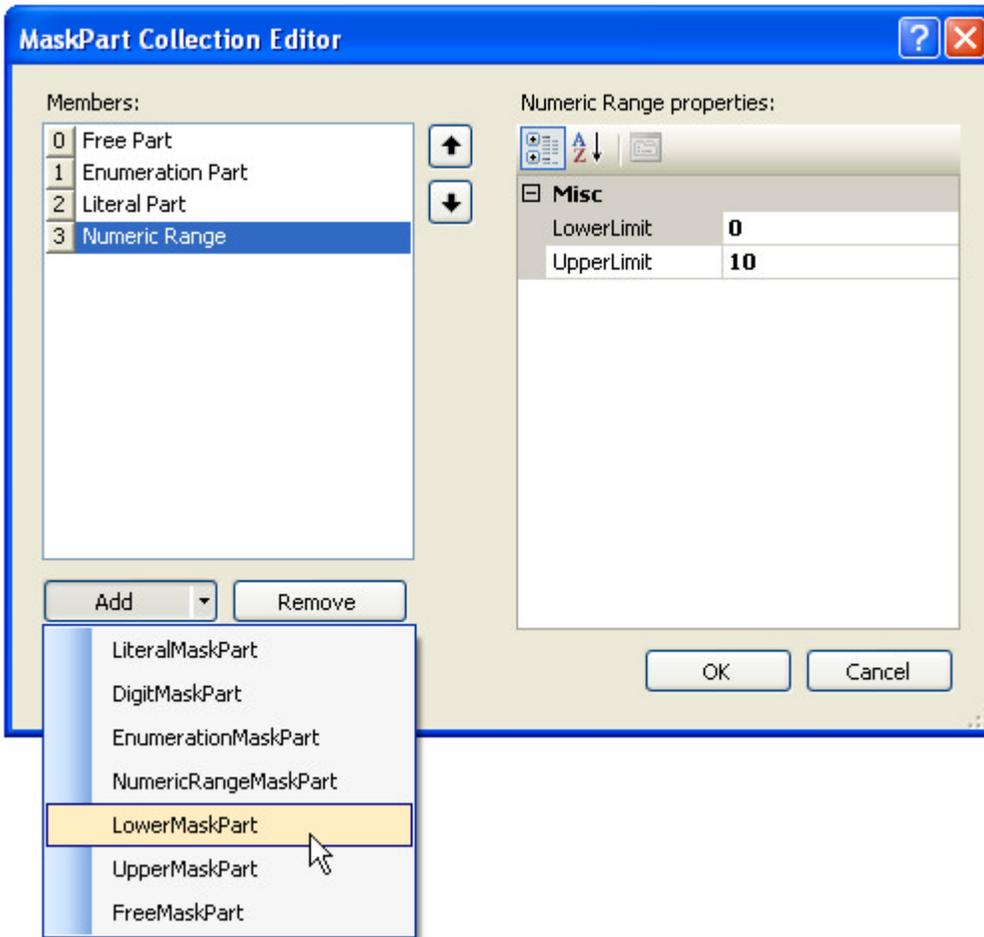
As shown in the preceding screen shot, you can choose from a selection of pre-defined masks. Just click on the row for a mask and the mask string automatically appears in the Mask text box, with a preview to show the prompts and literals below it. You can modify the pre-defined type by editing the string in the Mask text box. You can also define your own mask string from scratch by selecting the row labeled <Custom> and typing in a mask string. The preview updates as you type.

The mask string is made up of one or more parts, where each part represents a single (possibly optional) character or a value selected from a numeric range or set of enumerated strings. The following table lists the mask characters that correspond to each type of mask part:

Mask Element	MaskPart class	Description
a	FreeMaskPart	Accepts a single character. If this position is blank in the text, it is rendered as a prompt character.
L	UpperMaskPart	Uppercase letter (required). Restricts input to the ASCII letters A-Z.
l	LowerMaskPart	Lowercase letter (required). Restricts input to the ASCII letters a-z.
#	DigitMaskPart	Digit or space (optional). If this position is blank in the text, it is rendered as a prompt character.
<n..m>	NumericRangeMaskPart	range. Restricts the user to an integer in the declared numeric range. Numeric range mask parts can occupy multiple positions.
<Option1 Option2 Option3>	EnumerationPart	Restricts the user to one of a fixed set of options. The pipe (" ") serves as a separator between the option values.
\	N/A	Escape character, allowing the following character to act as literal text. For example "\a" is the character "a" rather than including a free mask part. "\\\" is the literal back slash character.
Any other characters	LiteralPart	All non-mask elements appear as themselves. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user.

MaskPart Collection Editor

If you are uncomfortable editing a mask string directly or trying to set up a particularly complicated mask, you can use the MaskPart Collection Editor rather than the Input Mask Dialog. The MaskPart Collection Editor lets you build up a mask part by part, setting the properties of each mask part. You can bring up the MaskPart Collection Editor by clicking on the ellipsis button next to the **MaskParts** or **DisplayMaskParts** property in the Properties Window for RadMaskedTextBox.



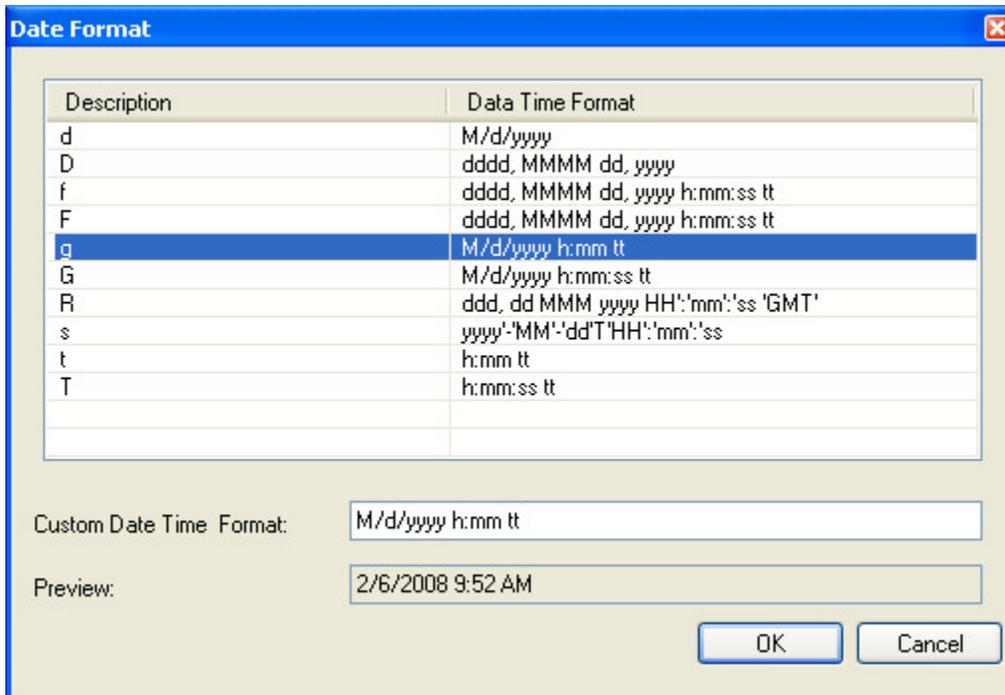
Use the MaskPart Collection Editor to build up a mask part by part. You can add parts to the mask by clicking the Add Button. Simply clicking the Add button adds a LiteralMaskPart. If you click on the drop-down arrow, you get a list of mask part types and can choose what type of part to add.

When a mask part in the collection is selected, the right side of the dialog shows the properties you can set for that type of mask part. In the screenshot above, a Numeric Range is selected, with properties for the maximum and minimum value in the range.

You can use the Remove button to remove parts from the mask you are building, and the arrow keys to rearrange the parts you have added. When you click the OK button to exit the dialog, the Mask or DisplayMask property is updated to reflect the new mask you built.

Date Format Dialog

The Date Format Dialog lets you specify the format strings that RadDateInput uses to format its value. You can bring up this dialog from the RadDateInput Smart Tag, or by clicking the ellipsis button next to the **DateFormat** or **DisplayDateFormat** property in the Properties Window.



In the Date Format Dialog, you can select from a set of standard format strings by selecting a row in the table. The expanded format string appears in the Custom Date Time Format text box, with a preview to show you how the string formats date and time values. You can then edit the string to get just the format you want, watching the preview update to reflect your changes.

⚠ RadDateInput uses standard ASP.NET date format strings with one exception: the one-character format strings listed in the table of the Date Format Dialog are always expanded to their constituent parts. As a result, if you change the Culture property, you must re-assign the DateFormat and DisplayDateFormat properties to ensure that the parts of the date format are expanded correctly.

The following table lists the format patterns to use when building a date format string:

Format Pattern	Description
d	The day of the month. Single-digit days have no leading zero. (Only if used in the context of a longer pattern. A single "d" on its own represents the Short date pattern.)
dd	The day of the month. Single-digit days have a leading zero.
ddd	The abbreviated name of the day of the week.
dddd	The full name of the day of the week.
M	The numeric month. Single-digit months have no leading zero. (Only if used in the context of a longer pattern. A single "M" on its own represents the Month day pattern.)
MM	The numeric month. Single-digit months have a leading zero.
MMM	The abbreviated name of the month.
MMMM	The full name of the month.
y	The year without the century. If the year without the century is less than 10, with no leading zero. (Only if used in the context of a longer pattern. A single "y" on its own represents the Month year pattern.)
yy	the year without the century. If the year without the century is less than 10, with a leading zero.
yyy	The year in four digits, including the century.
gg	The period or era (e.g. "A.D."). This pattern is ignored if the date to be formatted does not have an associated period or era.
h	The hour in a 12-hour clock. Single-digit hours have no leading zero.
hh	The hour in a 12-hour clock. Single-digit hours have a leading zero.
H	The hour in a 24-hour clock. Single-digit hours have no leading zero.

HH	The hour in a 24-hour clock. Single-digit hours have a leading zero.
m	The minute. Single-digit minutes have no leading zero. (Only if used in the context of a longer pattern. A single "m" on its own represents the Month day pattern)
mm	The minute. Single-digit minutes have a leading zero.
s	The second. Single-digit seconds have no leading zero. (Only if used in the context of a longer pattern. A single "s" on its own represents the sortable time pattern.)
ss	The second. Single-digit seconds have a leading zero.
t	The first character in the AM/PM designator. (Only if used in the context of a longer pattern. A single "t" on its own represents the short time pattern.)
tt	The AM/PM designator.

3.5 Server-Side Programming

Responding when the value changes

By default, the input controls do not cause a postback when the value changes. Typically, responding to changes, if at all, takes place in client-side code or when the form is submitted. However, there may be times when you want to respond dynamically to changed values, in spite of the performance hit of a postback. To accomplish this, you must do two things:

- Set the **AutoPostBack** property of the input control to true so that a postback occurs when the value of the control changes.
- Provide a handler for the **TextChanged** event that responds when the postback occurs.

The following example uses the **TextChanged** event to dynamically create new input controls based on the values of two input controls: a masked text box to specify the type of control to create, and a numeric text box to specify the number of new input controls to create.

Type:

Count:

newMaskedTextBox0

newMaskedTextBox1

newMaskedTextBox2

newMaskedTextBox3

newMaskedTextBox4

 This example uses a full postback for handling the **TextChanged** event. For a smoother response, you can look ahead to the chapter on **AJAXPanel**, **AjaxManager**, and **AjaxManagerProxy** to see how to handle the event in an asynchronous callback.

The masked text box has the mask "<TextBox|MaskedTextBox|NumericTextBox|DateInput>". This ensures that the user can only select one of the four input control types, and that the resulting selection is a known string. The numeric text box has **MinValue** and **MaxValue** set to 0 and 9, to limit the range of entries, and a **MaxLength** of 1 to prevent the user from trying to enter decimal values. The **NumberFormat** property sets **DecimalDigits** to 0 so that values are formatted as integers.

Both controls have the **AutoPostBack** property set to true. Because the **TextChanged** event for all input control types has the same signature, they can share the same event handler. The shared **TextChanged** event handler reads the values of the masked text box and numeric text box, and dynamically creates new input controls to reflect those values. The new controls are added to a **Placeholder**.



Gotcha! Be sure that the Placeholder control has its **EnableViewState** property set to false. Otherwise, you will get a runtime exception the second time the event handler is called because the viewstate will not match up.

The code for this project is in \VS Projects\Input\ServerSide.

[VB] Creating controls on TextChanged

```
Imports Telerik.Web.UI
Partial Public Class _Default
    Inherits System.Web.UI.Page

    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    End Sub

    Protected Sub AddInputControls(ByVal sender As Object, ByVal e As EventArgs) _
        Handles RadMaskedTextBox1.TextChanged, RadNumericTextBox1.TextChanged
        Placeholder1.Controls.Clear()
        Dim i As Integer = 0
        While i < RadNumericTextBox1.Value
            Select Case RadMaskedTextBox1.Text
            Case "TextBox"
                Dim newTextBox As New RadTextBox()
                newTextBox.ID = "newTextBox" + i.ToString()
                newTextBox.Label = newTextBox.ID
                newTextBox.Text = i.ToString()
                newTextBox.Skin = "Inox"
                Placeholder1.Controls.Add(newTextBox)
                Placeholder1.Controls.Add(New LiteralControl("<br/>"))
                Exit Select
            Case "MaskedTextBox"
                Dim newMaskedTextBox As New RadMaskedTextBox()
                newMaskedTextBox.ID = "newMaskedTextBox" + i.ToString()
                newMaskedTextBox.Label = newMaskedTextBox.ID
                newMaskedTextBox.Mask = "(###) ###-####"
                newMaskedTextBox.Text = "123456789" + i.ToString()
                newMaskedTextBox.Skin = "Inox"
                Placeholder1.Controls.Add(newMaskedTextBox)
                Placeholder1.Controls.Add(New LiteralControl("<br/>"))
                Exit Select
            Case "NumericTextBox"
                Dim newNumericTextBox As New RadNumericTextBox()
                newNumericTextBox.ID = "newNumericTextBox" + i.ToString()
                newNumericTextBox.Label = newNumericTextBox.ID
                newNumericTextBox.Value = i
                Placeholder1.Controls.Add(newNumericTextBox)
                Placeholder1.Controls.Add(New LiteralControl("<br/>"))
                Exit Select
            Case "DateInput"
                Dim newDateInput As New RadDateInput()
                newDateInput.ID = "newDateInput" + i.ToString()
                newDateInput.Label = newDateInput.ID
                newDateInput.DateFormat = "hh:mm:ss tt"
                newDateInput.SelectedDate = DateTime.Now
                Placeholder1.Controls.Add(newDateInput)
            End Select
            i += 1
        End While
    End Sub
End Class
```

```
        Placeholder1.Controls.Add(New LiteralControl("<br/>"))
        Exit Select
    End Select
    System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
End While
End Sub
End Class
```

[C#] Creating controls on TextChanged

```
using Telerik.Web.UI;
namespace ServerSide
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        protected void AddInputControls(object sender, EventArgs e)
        {
            Placeholder1.Controls.Clear();
            for (int i = 0; i < RadNumericTextBox1.Value; i++)
            {
                switch (RadMaskedTextBox1.Text)
                {
                    case "TextBox":
                        RadTextBox newTextBox = new RadTextBox();
                        newTextBox.ID = "newTextBox" + i.ToString();
                        newTextBox.Label = newTextBox.ID;
                        newTextBox.Text = i.ToString();
                        newTextBox.Skin = "Inox";
                        Placeholder1.Controls.Add(newTextBox);
                        Placeholder1.Controls.Add(new LiteralControl("<br/>"));
                        break;
                    case "MaskedTextBox":
                        RadMaskedTextBox newMaskedTextBox = new RadMaskedTextBox();
                        newMaskedTextBox.ID = "newMaskedTextBox" + i.ToString();
                        newMaskedTextBox.Label = newMaskedTextBox.ID;
                        newMaskedTextBox.Mask = "(###) ###-####";
                        newMaskedTextBox.Text = "123456789" + i.ToString();
                        newMaskedTextBox.Skin = "Inox";
                        Placeholder1.Controls.Add(newMaskedTextBox);
                        Placeholder1.Controls.Add(new LiteralControl("<br/>"));
                        break;
                    case "NumericTextBox":
                        RadNumericTextBox newNumericTextBox = new RadNumericTextBox();
                        newNumericTextBox.ID = "newNumericTextBox" + i.ToString();
                        newNumericTextBox.Label = newNumericTextBox.ID;
                        newNumericTextBox.Value = i;
                        Placeholder1.Controls.Add(newNumericTextBox);
                        Placeholder1.Controls.Add(new LiteralControl("<br/>"));
                        break;
                    case "DateInput":
                        RadDateInput newDateInput = new RadDateInput();
                        newDateInput.ID = "newDateInput" + i.ToString();
                        newDateInput.Label = newDateInput.ID;
```



```

<telerik:RadMaskedTextBox ID="MaritalStatus" Runat="server"
    Label="Marital Status"
    Mask="&lt;Single|Married|Separated|Divorced|Widowed&gt;"
    PromptChar=" " Width="200px" Text="Single" Skin="Outlook">
    <ClientEvents OnValueChanged="MaritalStatusChanged" />
</telerik:RadMaskedTextBox>
<br />
<telerik:RadNumericTextBox ID="NOfChildren" Runat="server"
    Label="Number of Children" MaxLength="2"
    MaxValue="99" MinValue="0"
    Width="200px" Skin="Outlook">
    <NumberFormat DecimalDigits="0" />
</telerik:RadNumericTextBox>
<br />
<telerik:RadDateInput ID="MarriageDate" Runat="server"
    Culture="English (United States)"
    DisplayDateFormat="MMMM dd, yyyy"
    Label="Date of Marriage" Enabled="False"
    Width="200px" Skin="Outlook">
    <DisabledStyle BackColor="#eeeeee" />
</telerik:RadDateInput>
</div>
</form>
</body>

```

Completing User Input

You can easily write a client-side function to implement a form of auto-complete. The following example illustrates how to accomplish this using the **OnValueChanging** client-side event. **OnValueChanging** is similar to the **OnValueChanged** event used in the previous example, but it occurs slightly earlier, and allows you to change the new value or prevent the edit that the user just made.

The event handler examines the new value, and if it represents a string that could be mapped to one of the expected responses, it performs that mapping using the **set_newValue()** method. If the event handler does not recognize the value that the user typed, it calls **set_cancel(true)**, which cancels the event so that the value of the text box is not changed.

[ASP.NET] Completing user input

```

<head runat="server">
    <title>Completing user input</title>
</head>
<body>
    <script type="text/javascript">
        function SetGender(sender, eventArgs) {
            // get the new value from the event arguments
            var newValue = eventArgs.get_newValue().trim();
            // any value that could represent 'male' is changed
            if (newValue == "m" || newValue == "M" ||
                newValue == "Male" ||
                newValue == "man" || newValue == "Man" ||
                newValue == "boy" || newValue == "Boy" ||
                newValue == "b" || newValue == "B")
                eventArgs.set_newValue("male");
            // any value that could represent 'female' is changed
            else if (newValue == "f" || newValue == "F" ||
                newValue == "Female" ||

```

```

        newValue == "woman" || newValue == "Woman" ||
        newValue == "w" || newValue == "W" ||
        newValue == "girl" || newValue == "Girl" ||
        newValue == "g" || newValue == "G")
        eventArgs.set_newValue("female");
    // any unrecognized value is rejected
    if (eventArgs.get_newValue().trim() != "male" &&
        eventArgs.get_newValue().trim() != "female")
        eventArgs.set_cancel(true);
}
</script>
<form id="form1" runat="server">
<div>
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>

    <telerik:RadTextBox ID="RadTextBox1" Runat="server"
        EmptyMessage="- Enter your sex -" Width="125px">
        <ClientEvents OnValueChanging="SetGender" />
    </telerik:RadTextBox>
</div>
</form>
</body>

```

Handling input errors

All of the input controls other than `RadTextBox` restrict the values that the user can enter. `RadMaskedTextBox` requires the user to enter a value that matches the mask, `RadNumericTextBox` requires the user to enter a number (possibly within a specified range), and `RadDateInput` requires users to enter a date and/or time value (again possibly within a specified range). If the user enters an invalid value, the client-side `OnError` event occurs.

The event arguments for `RadMaskedTextBox` are different than those for the other types of input control. Errors only arise when the input fails to match the mask. The event arguments have a `get_currentPart()` method to return the mask part that was not correctly matched. The `get_newValue()` method returns the text that would not match the current mask part.

In the case of `RadDateInput` and `RadNumericTextBox`, on the other hand, there are two types of error that can occur. The parser can fail to recognize the input as a valid value, or the value may be a recognizable date or number, but be out of range. The `get_reason()` method of the event arguments indicates which of these occurred. The `get_inputText()` method returns the new value that caused the problem, except in the case of parsing errors on numeric text box, where it returns the unedited value.

You can use the `OnError` event to implement your own parsing algorithm when the built-in parser fails, or to generate an error message. The following example illustrates generating an error message based on the information in the `OnError` event handler. The error handler for the masked text box displays an alert and moves the cursor to the part of the mask that failed. The error handlers for the numeric text box and date input controls indicate the type of error that occurred and the text that caused the problem (if available).

The source for this project is in `\VS Projects\Input\ClientErrorHandling`.

[ASP.NET] Error handling

```

<head runat="server">
    <title>Error Handling</title>
</head>
<body>
    <script type="text/javascript">

```

```
function HandleMaskError(sender, eventArgs) {
    // on masked text box, get_newValue() returns the problem value
    alert("Invalid value: " + eventArgs.get_newValue());
    // get_currentPart() returns the part that failed to match
    var part = eventArgs.get_currentPart();
    if (part) {
        // set the cursor on the problem part
        sender.set_cursorPosition(part.offset);
    }
    // we did not correct the error, so cancel the edit
    eventArgs.set_cancel(true);
}
function HandleNumericError(sender, eventArgs) {
    switch (eventArgs.get_reason()) {
        case 1: // Parsing error -- no invalid value available
            alert("Invalid character!");
            break;
        case 2: // Out of range
            alert("Value out of range: " + eventArgs.get_inputText());
            break;
    }
    // we did not correct the error, so cancel the edit
    eventArgs.set_cancel(true);
    // return focus to the numeric text box
    sender.focus();
}
function HandleDateError(sender, eventArgs) {
    switch (eventArgs.get_reason()) {
        case 1: // Parsing error
            alert("Value could not be parsed: " + eventArgs.get_inputText());
            break;
        case 2: // Out of range
            alert("Value out of range: " + eventArgs.get_inputText());
            break;
    }
    // we did not correct the error, so cancel the edit
    eventArgs.set_cancel(true);
    // return focus to the date input control
    sender.focus();
}
</script>
<form id="form1" runat="server">
    <div>
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <telerik:RadMaskedTextBox ID="RadMaskedTextBox1" Runat="server"
            EmptyMessage="- Enter SSN -" HideOnBlur="True" Mask="###-##-####">
            <ClientEvents OnError="HandleMaskError" />
        </telerik:RadMaskedTextBox>
        <br />
        <telerik:RadNumericTextBox ID="RadNumericTextBox1" Runat="server"
            Culture="English (United States)"
            EmptyMessage="-Enter cost below $5.00 -"
            MaxValue="5" MinValue="0.01" Type="Currency" Width="125px">
            <ClientEvents OnError="HandleNumericError" />
        </telerik:RadNumericTextBox>
    </div>
</form>
```

```

</telerik:RadNumericTextBox>
<br />
<telerik:RadDateInput ID="RadDateInput1" Runat="server"
  EmptyMessage="-Enter date in 1990's-"
  MaxDate="1999-12-31" MinDate="1990-01-01">
  <ClientEvents OnError="HandleDateError" />
</telerik:RadDateInput>
</div>
</form>
</body>

```

3.7 How To

You can enhance the functionality of RadTextBox by using it in combination with other ASP.NET controls.

How-to use ASP.NET validators with input RadControls

It is easy to use the input RadControls with ASP.NET validators: simply set the **ControlToValidate** property of the validator to the text box, masked text box, numeric text box, or date input control that you want to validate.

You can assign the input controls on your Web page to different validation groups so that the validators check them at different times. All you need do is set the **ValidationGroup** property of the validator to match the **ValidationGroup** property of the control that initiates the validation.

You can even use an input control to initiate validation. Just set the **CausesValidation** property to true, and it will initiate a validation every time its value changes.

The following example illustrates using validators with RadTextBox, although you can use them with any of the input controls. It demonstrates both the use of validation groups and the way an input control can trigger validation.

The form uses three validation groups: "LoginGroup", "SignUpGroup", and "PWGroup":

- The "LoginGroup" validation group is assigned to the two validators in the left-hand panel, and to the button in that panel which triggers validation on postback. Note that no special settings are required on the text boxes; they are validated simply because of the **ControlToValidate** property of the corresponding

validators.

[ASP.NET] "LoginGroup" controls and validators

```
<telerik:RadTextBox ID="LoginName" Runat="server"
  Skin="Outlook" Width="75%"
  Label="Name: " ToolTip="Enter your account name">
</telerik:RadTextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" runat="server"
  ControlToValidate="LoginName"
  ErrorMessage="You must enter your name to log in!"
  ValidationGroup="LoginGroup">
</asp:RequiredFieldValidator>
<br>
<telerik:RadTextBox ID="LoginPassword" Runat="server"
  Skin="Outlook" Width="75%" Label="Password: "
  TextMode="Password" ToolTip="Enter your password">
</telerik:RadTextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" runat="server"
  ControlToValidate="LoginPassword"
  ErrorMessage="You must enter your password to log in!"
  ValidationGroup="LoginGroup">
</asp:RequiredFieldValidator>
<br />
<br />
<asp:Button ID="Button1" runat="server" Text="Log In"
  BackColor="#99CCFF" ForeColor="#0000CC"
  CausesValidation="true" ValidationGroup="LoginGroup" />
```

- The "SignUpGroup" validation group is similar. It checks for required fields in the right-hand panel when the "Sign Up" button triggers a postback. The only thing new here is that this group includes a regular expression validator to check for valid email addresses as well as the required field validators.

[ASP.NET] "SignUpGroup" controls and validators

```
<telerik:RadTextBox ID="SignUpName" Runat="server"
  Skin="Outlook" Width="75%"
  Label="Name: " ToolTip="Enter a name for your account" >
</telerik:RadTextBox>
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" runat="server"
  ControlToValidate="SignUpName"
  ErrorMessage="You must supply an account name!"
  ValidationGroup="SignUpGroup">
</asp:RequiredFieldValidator>
<br />
<telerik:RadTextBox ID="SignUpPW" Runat="server"
  Skin="Outlook" Width="75%" Label="Password: "
  TextMode="Password" ToolTip="Enter the password you want to use">
</telerik:RadTextBox>
...
<asp:RequiredFieldValidator ID="RequiredFieldValidator4" runat="server"
  ControlToValidate="SignUpPW"
  ErrorMessage="You must supply a password!"
  ValidationGroup="SignUpGroup">
</asp:RequiredFieldValidator>
<br />
<telerik:RadTextBox ID="SignUpPWConfirm" Runat="server"
  Skin="Outlook" Width="75%" Label="Confirm Password: " TextMode="Password"
  ToolTip="Retype your password to confirm"
```

```

    CausesValidation="True" ValidationGroup="PWGroup">
</telerik:RadTextBox>
...
<asp:RequiredFieldValidator ID="RequiredFieldValidator5" runat="server"
    ControlToValidate="SignUpPWConfirm"
    ErrorMessage="You must confirm your password!"
    ValidationGroup="SignUpGroup">
</asp:RequiredFieldValidator>
<br />
<telerik:RadTextBox ID="SignUpEmail" Runat="server"
    Skin="Outlook" Width="75%" Label="Email: "
    Tooltip="Enter your email address.">
</telerik:RadTextBox>
<asp:RegularExpressionValidator ID="RegularExpressionValidator2" runat="server"
    ControlToValidate="SignUpEmail"
    ErrorMessage="Invalid email address!"
    ValidationExpression="^[\\w\\.\\-]+@[a-zA-Z0-9\\-]+(\\. [a-zA-Z0-9\\-]{1,})*(\\. [a-zA-Z]{2,3}){0,}
$"
    ValidationGroup="SignUpGroup">
</asp:RegularExpressionValidator>
<br />
<asp:Button ID="Button3" runat="server" Text="Sign Up"
    BackColor="#99CCFF" ForeColor="#0000CC"
    ValidationGroup="SignUpGroup" />

```

- The "PWGroup" validation group is a little different because it is not initiated by a postback. Instead, validators in this group are checked when the user enters a value in the password confirmation text box. To accomplish this, the password confirmation text box has its **CausesValidation** property set to true and its **ValidationGroup** property set to "PWGroup". When this validation group is checked, a regular expression validator ensures that the password is valid, and a compare validator checks that the confirmation matches. Note that the password confirmation text box is triggering validation on itself.

[ASP.NET] "PWGroup" controls and validators

```

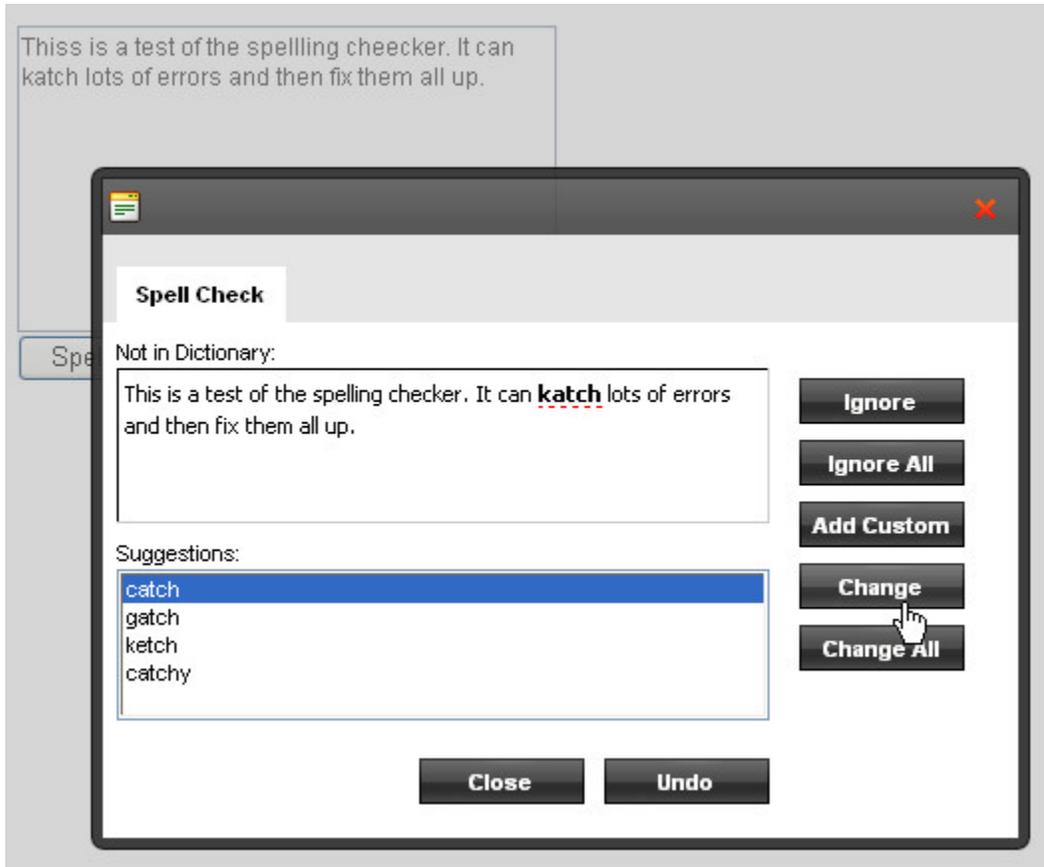
<telerik:RadTextBox ID="SignUpPW" Runat="server"
    Skin="Outlook" Width="75%" Label="Password: "
    TextMode="Password" Tooltip="Enter the password you want to use">
</telerik:RadTextBox>
<asp:RegularExpressionValidator ID="RegularExpressionValidator1" runat="server"
    ControlToValidate="SignUpPWConfirm"
    ErrorMessage="Password must be 6-10 characters, contain at least one digit and one number
and have no special characters!"
    ValidationExpression="(?!^[0-9]*$)(?!^[a-zA-Z]*$)^[a-zA-Z0-9]{6,10}$"
    ValidationGroup="PWGroup">
</asp:RegularExpressionValidator>
...
<telerik:RadTextBox ID="SignUpPWConfirm" Runat="server"
    Skin="Outlook" Width="75%" Label="Confirm Password: "
    TextMode="Password"
    Tooltip="Retype your password to confirm"
    CausesValidation="True" ValidationGroup="PWGroup">
</telerik:RadTextBox>
<asp:CompareValidator ID="CompareValidator1" runat="server"
    ControlToCompare="SignUpPW" ControlToValidate="SignUpPWConfirm"
    ErrorMessage="Password does not match!" ValidationGroup="PWGroup">
</asp:CompareValidator>

```

The complete code for this project is in \VS Projects\Input\HowToValidators.

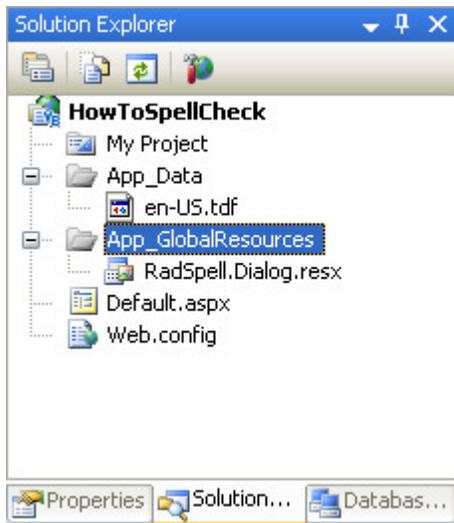
Spell checking text box values

Another useful control to use with RadTextBox is RadSpell. This control lets you easily enable spell checking so that the user can check the text after it is entered into the text box.

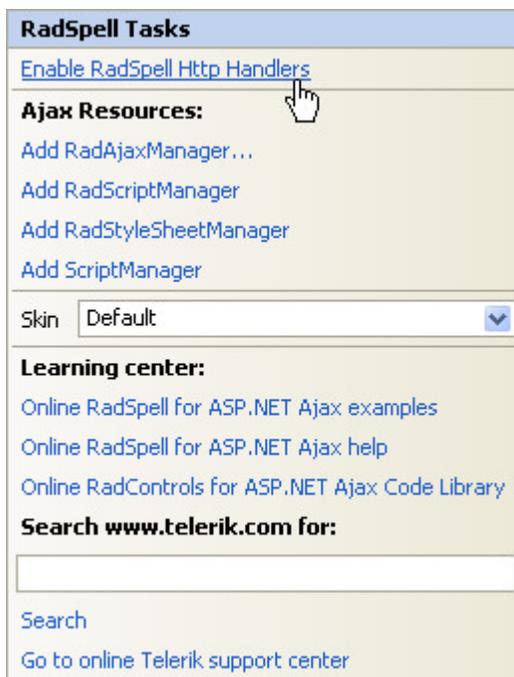


The following walk-through guides you through the process of linking up a spell checker with a multi-line text box. The code for this example can be found in \VS Projects\Input\HowToSpellCheck.

1. Create a new ASP.NET Web Application and add a ScriptManager onto the page from the AJAX extensions section of the tool box.
2. Locate the English dictionary that the spell checker uses. This file is called "en-US.tdf", and can be found in the App_Data\RadSpell folder inside the folder where you installed RadControls for ASPNET AJAX. Copy this file and paste it into the App_Data folder of your project (using the Project Explorer).
3. Right-click on your project in the Project Explorer and choose Add|Add ASP.NET Folder|App_GlobalResources from the context menu.
4. Locate the spell dialog resource, "RadSpell.Dialog.resx", in the App_GlobalResources folder inside the folder where you installed RadControls for ASPNET AJAX. Copy this file and paste it into the App_GlobalResources folder that you added in the last step. Your Project Explorer should now look something like the following:



5. Add a **RadTextBox** control to your Web page. Set its **Skin** property to "WebBlue", **TextMode** to "MultiLine", **Rows** to 10 and **Columns** to 50.
6. Add a **RadSpell** control to your Web page below the RadTextBox.
7. In the Smart Tag that appears automatically, click the **Enable RadSpell Http Handlers** link.



8. On the **RadSpell** control, set the **ControlToCheck** property to "RadTextBox1" and the **DictionaryPath** property to "App_Data".
9. Press Ctl-F5 to run the application. You can enter a lengthy value in the text box (with some spelling errors in it). Click the "Spell Check" button to invoke the spell checker. When you exit the dialog, any corrections you made in the dialog are reflected in the text box.

Password strength checking of RadTextBox in password mode

The feature allows you to specify your custom criteria for password strength and visualize an indicator to inform the user how strong is the typed password according to this criteria.

You can easily turn on the password strength check functionality by setting `PasswordStrengthSettings.ShowIndicator="true"`. This way the indicator will show and it will use its default values for password strength until you specify your own.

ASPX

```
<telerik:RadTextBox ID="RadTextBox1" runat="server" TextMode="Password">  
  <PasswordStrengthSettings ShowIndicator="true" />  
</telerik:RadTextBox>
```

Then you can specify your preferred options for the password which will be used for calculating its strength. The available properties are:

- **ShowIndicator** - enables/disables the indication. By default ShowIndicator is set to *false*. In order to show the indication set the property to *true*.
- **PreferredPasswordLength** - preferred length of the password.
- **MinimumNumericCharacters** - the number of minimum numeric characters that the user has to enter in order for his password to be considered as a strong one.
- **MinimumUpperCaseCharacters** - the number of minimum upper case characters expected.
- **MinimumLowerCaseCharacters** - the number of minimum lower case characters expected.
- **MinimumSymbolCharacters** - the number of minimum symbol characters expected.
- **CalculationWeightings** - a list of 4 semi-colon separated numeric values used to determine the weighting of a strength characteristic. The total of the 4 values should be 100. By default they are defined as 50;15;15;20. This means that **password length** will determine 50% of the strength calculation, **numeric criteria** is 15% of strength calculation, **casing criteria** is 15% of calculation, and **symbol criteria** is 20% of calculation. So the format is "A;B;C;D" where *A = length weighting*, *B = numeric weighting*, *C = casing weighting*, *D = symbol weighting*.
- **RequiresUpperAndLowerCaseCharacters** - specifies whether upper and lower case characters are required. By default the property is set to "true". When it is "false", the MinimumLowerCaseCharacters and MinimumUpperCaseCharacters properties do not affect the calculation of the password.
- **IndicatorElementID** - sets a div or span element to which the indication will be applied. If this property is not set, such element will be created automatically.
- **IndicatorElementBaseStyle** - the name of the CSS class that will be used for styling the indicator element.
- **TextStrengthDescriptions** - a list of five semi-colon separated strings which will be used as descriptions for the password strength. By default TextStrengthDescriptions is set to "Very Weak;Weak;Medium;Strong;Very Strong".
- **TextStrengthDescriptionStyles** - a list of six semi-colon separated CSS classes that will be applied depending on the calculated password strength. By default the property is "riStrengthBarL0;riStrengthBarL1;riStrengthBarL2;riStrengthBarL3;riStrengthBarL4;riStrengthBarL5;"



You can find the complete source for this project at:
\\VS Projects\Input\PasswordStrengthChecker

3.8 RadInputManager

The basics

Since Q3 2008, RadInput controls include a new member - the **RadInputManager**. This control is aimed to offer

two important features:

1. An easy and intuitive way to extend a standard **ASP.NET** text box, and without any extra custom code, introduce much functionality, normally related to a **RadInput** control. For example, a standard text box control offers no default functionality for text parsing and validation - this has to be done via custom code, either client or server side. This input validation is normally associated with **RadInput** controls. In addition the **RadInputManager** is very useful for formatting and styling of regular input controls. In other words, the **RadInputManager** transforms your standard text boxes into featured **RadInput** controls (However some limitations apply - one of the differences between text boxes extended via the input manager, and normal **RadInput** controls is that there are no additional buttons, such as a spin button.).
2. On the other hand, having a large number of input controls on the page may hurt performance. This is where the **RadInputManager** comes in. It automatically adds extra functionality to separate text boxes, or all text boxes nested in another control on the page, via a few settings added to the Input Manager. The main aspects of the better performance are:
 - Loading time - usually, having a large number of input controls on the page, each associated with a separate object and client events and handlers, would imply a performance hit. Introducing the **RadInputManager**, however, dramatically reduces the load time.
 - Maximum number of controls allowed on the page - local tests showed that with the help of **RadInputManager**, the number of input controls a standard application would allow can be increased up to ten times.
 - Footprint of the page - local tests showed that a standard page, with a total of 300 input controls, generates a footprint of approximately 400KB. On the other hand, extending 300 standard text boxes via the **RadInputManager**, to enhance their behavior to a **NumericInput** control, generates a footprint of approximately 100KB. This brings about faster loading and better responsiveness of the page.

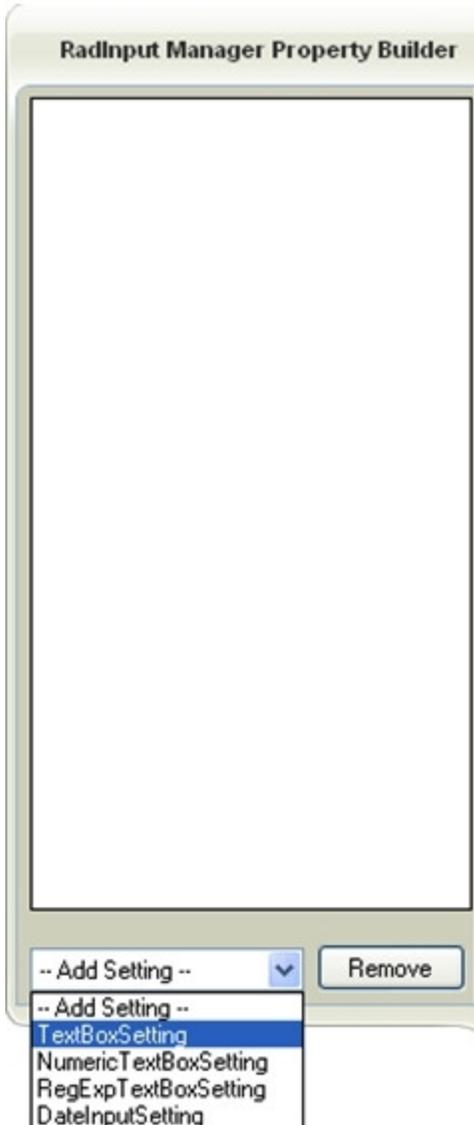
Performance

The core of the performance benefit of using a **RadInputManager** (as opposed to input controls) is in the following approach. A normal input control generates a client side object for each control instance. For example, if you declare 300 input controls on the page, you will have 300 client objects, once the page is compiled and run. On the other hand, when extending standard text boxes via the **RadInputManager** control, you will have a single client side object, which will dramatically improve performance, while at the same time providing enhanced data entry capabilities for user input validation.

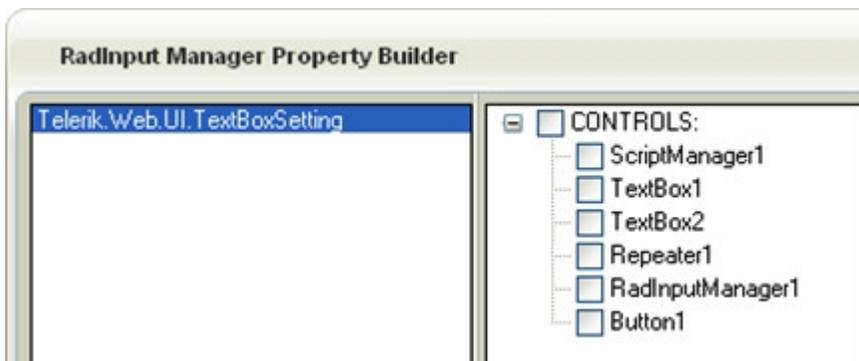
To summarize, the **RadInputManager** offers extended functionality to standard text boxes, with little overhead related to increased page footprint or extra coding.

Design-Time Support

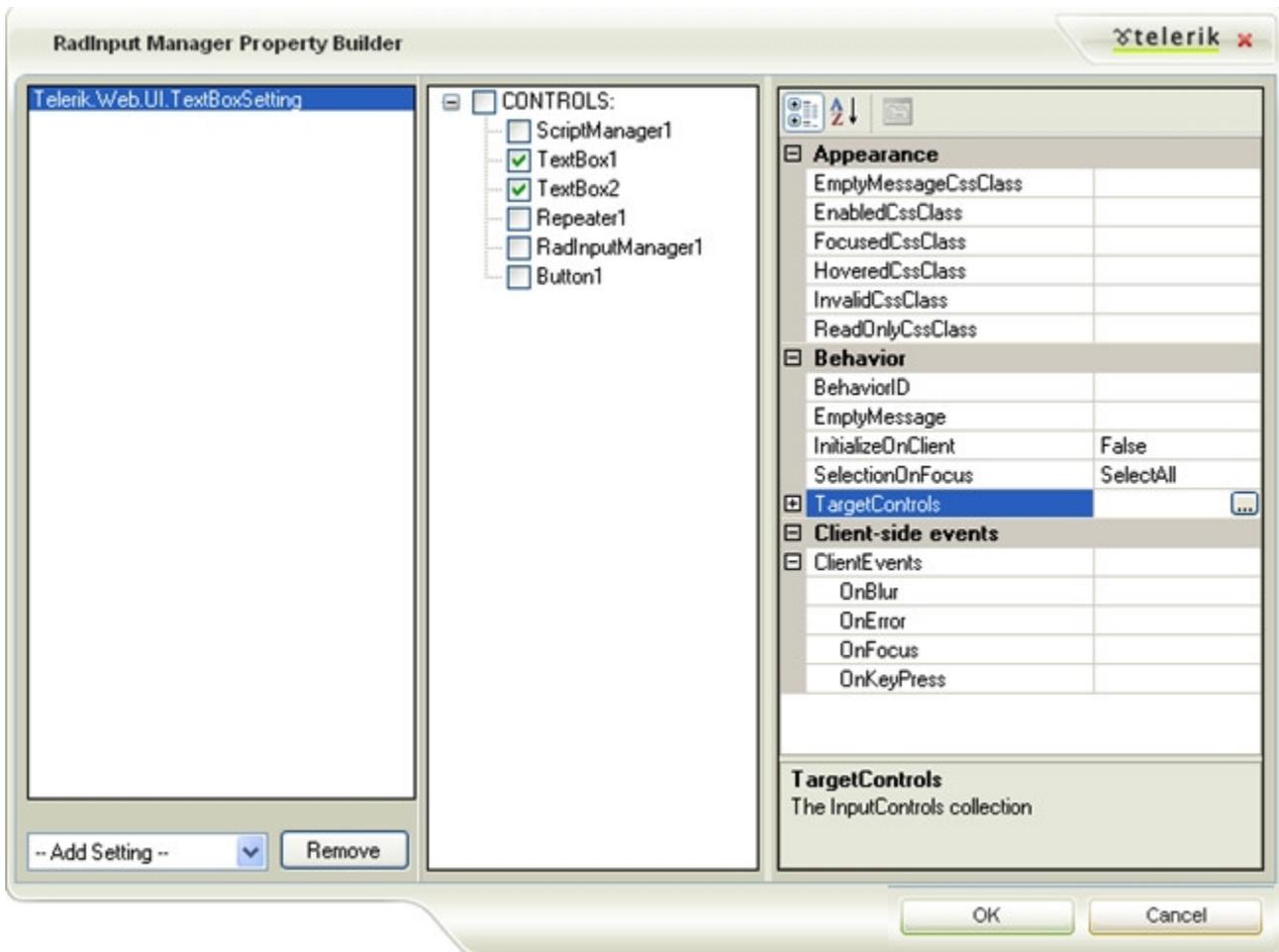
The most important aspect of the design time support for the control is the ability to configure it to determine which controls on the page will be extended through it. Essentially, the approach is similar to the one used for Ajax-enabled controls on the page via the **RadAjaxManager**. First, you select what type of setting you will be adding - **TextBoxSetting** / **NumericTextBoxSetting** / **RegExpTextBoxSetting** / **DateInputSetting**.



Once you have chosen one of the four possible options, you can choose which particular controls on the page you would like to extend - the right-hand side pane lists all the controls on the current page.



After you have chosen the setting, and the controls to be extended, you can set some of the most important properties of each setting, as shown in the screen shot below:



The properties which can be set include CSS classes for the different states (hover/enabled, etc), behavior settings such as **BehaviorID** and **EmptyMessage** and the client-side events (OnBlur, OnError, OnFocus, OnKeyPress).

☑ The **RadInputManager** property builder is also available in the Properties window through the **InputSettings** property.

Using the RadInputManager

Generally, there are two groups of controls which are extended using the **RadInputManager**. The first are controls which are located directly on the page, such as a normal text box somewhere on the form. The second are text boxes located in another control - for example, a text box nested in a repeater.

To extend the default functionality of the standard text box controls, **RadInputManager** defines four types of settings which could be set:

1. **telerik:TextBoxSetting** - the targeted text box will exhibit behavior like a normal **RadTextBox**.
2. **telerik:NumericTextBoxSetting** - the targeted text box will be accepting numeric input.
3. **telerik:DateInputSetting** - the targeted text box will be accepting input in a date format.
4. **telerik:RegExpTextBoxSetting** - the targeted text box will be accepting characters corresponding to a specified regular expression.

Below is a list of four tables, covering the most important properties of the elements discussed up to now:

TextBoxSetting

Property	Description
BehaviorID	A unique id for the settings related to a given text box.
ClientEvents-OnBlur	The name of the client side function which will be raised when the control loses focus.
ClientEvents-OnError	The name of the client side function which will be called when an error occurs - the user enters invalid input. This event is not raised for the text box control, since there is no input restriction.
ClientEvents-OnFocus	The name of the client side function which will be called when the control receives focus.
ClientEvents-OnKeyPress	The name of the client side function which will be called when the user presses a button, while the control has the focus.
EmptyMessage	The text which will be displayed before the user has entered any text.
InitializeOnClient	A property which indicates whether the client event handlers and css classes will be set on the client.
SelectionOnFocus	A property which is used to determine whether the text in the control will be selected once it receives focus.

NumericTextBoxSetting

Property	Description
AllowRounding	A setting which specifies whether the user input may be rounded.
BehaviorID	A unique id for the settings related to a given text box.
ClientEvents-OnBlur	The name of the client side function which will be raised when the control loses focus.
ClientEvents-OnError	The name of the client side function which will be called when an error occurs - the user enters invalid input.
ClientEvents-OnFocus	The name of the client side function which will be called when the control receives focus.
ClientEvents-OnKeyPress	The name of the client side function which will be called when the user presses a button, while the control has the focus.
DecimalDigits	Gets or sets the number of decimal places to use in numeric values.
DecimalSeparator	Gets or sets the string to use as the decimal separator in values.
EmptyMessage	The text which will be displayed before the user has entered any text.
ErrorMessage	Sets the message to be displayed when invalid value is entered.
GroupSeparator	Gets or sets the string that separates groups of digits to the left of the decimal in values.
GroupSizes	Gets or sets the number of digits in each group to the left of the decimal in values.
InitializeOnClient	A property which indicates whether the client event handlers and css classes will be set on the client.
MaxValue	The maximal numeric value which can be entered in the control.
MinValue	The minimal numeric value which can be entered in the control.
NegativePattern	Gets or sets the format pattern for negative values.
PositivePattern	Gets or sets the format pattern for positive values.
SelectionOnFocus	A property, which is used to determine whether the text in the control will be selected, once it receives focus.
Type	The type of the control - Currency/Number/Percent.

DateInputSetting

Property	Description
----------	-------------

BehaviorID	A unique id for the settings related to a given text box.
ClientEvents-OnBlur	The name of the client side function which will be raised when the control loses focus.
ClientEvents-OnError	The name of the client side function which will be called when an error occurs - the user enters invalid input.
ClientEvents-OnFocus	The name of the client side function which will be called when the control receives focus.
ClientEvents-OnKeyPress	The name of the client side function which will be called when the user presses a button, while the control has the focus.
DateFormat	Gets or sets the date and time format used by RadDateSetting.
DisplayDateFormat	Gets or sets the display date format used by RadDateSetting (Visible when the control is not on focus).
EmptyMessage	The text which will be displayed before the user has entered any text.
ErrorMessage	Sets the message to be displayed when invalid value is entered.
InitializeOnClient	A property which indicates whether the client event handlers and css classes will be set on the client.
MinDate	The minimal date which the user will be allowed to enter.
MaxDate	The maximal date which the user will be allowed to enter.
SelectionOnFocus	A property, which is used to determine whether the text in the control will be selected, once it receives focus.
ShortYearCenturyEnd	Gets or sets a value that indicates the end of the century that is used to interpret the year value when a short year (single-digit or two-digit year) is entered in the input.

RegExpTextBoxSetting

Property	Description
BehaviorID	A unique id for the settings related to a given text box.
ClientEvents-OnBlur	The name of the client side function which will be raised when the control loses focus.
ClientEvents-OnError	The name of the client side function which will be called when an error occurs - the user enters invalid input.
ClientEvents-OnFocus	The name of the client side function which will be called when the control receives focus.
ClientEvents-OnKeyPress	The name of the client side function which will be called when the user presses a button, while the control has the focus.
EmptyMessage	The text which will be displayed before the user has entered any text.
ErrorMessage	A message which is displayed if the regular expression matching fails.
InitializeOnClient	A property which indicates whether the client event handlers and css classes will be set on the client.
SelectionOnFocus	A property, which is used to determine whether the text in the control will be selected, once it receives focus.
ValidationExpression	A regular expression, representing the matching criteria.
ValidationGroup	The ValidationGroup to which the regular expression setting is assigned.

In addition to the properties above, all 4 settings support different CSS classes to be applied for the different states that a text box could currently be in. To use them just set the relevant `-CssClass` property:

- DisabledCssClass
- EmptyMessageCssClass

UI for ASP.NET AJAX

- EnabledCssClass
- FocusedCssClass
- HoveredCssClass
- InvalidCssClass
- ReadOnlyCssClass

Each one of the setting groups also allows for different behavior and contains a **BehaviorID** property, which is used to identify settings pertaining to a given text box. These can later be retrieved on the client, for example, and access a property such as the **EmptyMessage**.

 Note that each input setting must have at least one target control, otherwise it will not be serialized to the client and its client object would not be instantiated.

Give it a try

Set up the project structure

1. Create a new ASP.NET Web Application.
2. In the designer, drag a **ScriptManager** from the AJAX extensions section of the tool box onto your page.

Add the **RadInputManager** and **TextBox** controls.

1. Add a **RadInputManager** to your web page. In the Smart Tag, select "Vista" from the Skin drop-down.
2. Add a **TextBox** to your web page. In the Misc section of the Properties Window change its ID to "ExtendedTextBox".
3. Add a few line breaks after the **TextBox** and add a new **TextBox** to your web page. In the Misc section of the Properties Window change its ID to "ExtendedNumericTextBox".
4. Add a few line breaks after the **TextBox** and add a new **TextBox** to your web page. In the Misc section of the Properties Window change its ID to "ExtendedDateTextBox".
5. Add a few line breaks after the **TextBox** and add a new **TextBox** to your web page. In the Misc section of the Properties Window change its ID to "ExtendedRegExpTextBox".

Configure the **TextBoxSetting**.

1. From the **RadInputManager**'s smart tag, open the **Property Builder** dialog by clicking on the "Configure Input Manager" option.
2. Click the arrow in the **Add Setting** drop down in the bottom left corner of the dialog, choose "TextBoxSetting" and select the setting once it is added in the left hand-side pane.
3. In the Behavior section of the right-hand side pane set the **BehaviorID** property to "TextBoxSetting", the **EmptyMessage** property to "- Enter some text here -", and the **SelectionOnFocus** property to "SelectAll".
4. Check the check box in front of the "ExtendedTextBox" option in the middle pane of the dialog and click "OK" to add the setting to the manager's settings.

Configure the **NumericTextBoxSetting**.

1. From the **RadInputManager**'s smart tag, open the **Property Builder** dialog by clicking on the "Configure Input Manager" option.

2. Click the arrow in the **Add Setting** drop down in the bottom left corner of the dialog, choose "NumericTextBoxSetting" and select the setting once it is added in the left hand-side pane.
3. In the Behavior section of the right-hand side pane set the **BehaviorID** property to "NumericTextBoxSetting", the **EmptyMessage** property to "- Enter a number here -", the **SelectionOnFocus** property to "SelectAll", the **MinValue** property to "0", the **MaxValue** property to "10000" and the **Type** property to "Number". In the Misc section set the **AllowRounding** property to "False", the **DecimalDigits** property to "2", the **DecimalSeparator** property to ".", the **GroupSeparator** property to ",", and the **GroupSizes** property to "3".
4. Check the check box in front of the "ExtendedNumericTextBox" option in the middle pane of the dialog and click "OK" to add the setting to the manager's settings.

Configure the DateTimeSetting.

1. From the **RadInputManager**'s smart tag, open the **Property Builder** dialog by clicking on the "Configure Input Manager" option.
2. Click the arrow in the **Add Setting** drop down in the bottom left corner of the dialog, choose "DateTimeSetting" and select the setting once it is added in the left hand-side pane.
3. In the Behavior section of the right-hand side pane set the **BehaviorID** property to "DateTimeSetting", the **EmptyMessage** property to "- Enter a date here -", the **SelectionOnFocus** property to "SelectAll", the **MinDate** property to "2000-01-01", the **MaxDate** property to "2015-12-31", the **DateFormat** property to "dd.MM.yyyy", the **DisplayDateFormat** property to "dd.MMM.yyyy". In the Appearance section set the **ErrorMessage** property to "Invalid date!".
4. Check the check box in front of the "ExtendedDateTextBox" option in the middle pane of the dialog and click "OK" to add the setting to the manager's settings.

Configure the RegExpTextBoxSetting.

1. From the **RadInputManager**'s smart tag, open the **Property Builder** dialog by clicking on the "Configure Input Manager" option.
2. Click the arrow in the **Add Setting** drop down in the bottom left corner of the dialog, choose "RegExpTextBoxSetting" and select the setting once it is added in the left hand-side pane.
3. In the Behavior section of the right-hand side pane set the **BehaviorID** property to "RegExpTextBoxSetting", the **EmptyMessage** property to "- Enter an Email address here -", the **SelectionOnFocus** property to "SelectAll". In the Appearance section set the **ErrorMessage** property to "Invalid Email address!", click in the **ValidationExpression** text box and choose the "Internet email address" option from the **Regular Expression Editor** dialog.
4. Check the check box in front of the "ExtendedRegExpTextBox" option in the middle pane of the dialog and click "OK" to add the setting to the manager's settings.

Run the application

1. Press Ctl-F5 to run the application. Note that the empty messages appear for all the text box controls you entered.
2. Tab around the form and enter some values. Note that the entered value gets selected when each control gets focus, based on the **SelectionOnFocus** property. Note that the ranges you specified for the Numeric and Date text boxes are enforced. Note that when you enter a value with more than 2 decimal digits in the numeric text box, the value is not rounded.

Invalid date! 

Invalid Email address! 

Getting and Setting Values

RadInputManager provides you with the ability of setting/getting values of the **TextBox** controls either client-side or server-side.

Getting values Client-Side

To be able to get or set values client-side, first you should know how to get the **TextBox** client-side object. This can be done through the client-side object of the **RadInputManager**. Here is an example:

[ASP .NET] Getting values on the client

```
<telerik:RadCodeBlock ID="RadCodeBlock1" runat="server">
<script type="text/javascript">
function pageLoad()
{
    var inputManager = $find("<%= RadInputManager1.ClientID %>");
    var input = inputManager.get_targetInput("<%= TextBox1.ClientID %>");
}
</script>
</telerik:RadCodeBlock>
<asp:TextBox ID="TextBox1" runat="server">
</asp:TextBox>
<telerik:RadInputManager ID="RadInputManager1" runat="server">
    <telerik:TextBoxSetting BehaviorID="Behavior1">
        <TargetControls>
            <telerik:TargetInput ControlID="TextBox1" />
        </TargetControls>
    </telerik:TextBoxSetting>
</telerik:RadInputManager>
```

For getting and setting values on the client, you can use the methods available in the **RadInput** control client-side API listed below.

Each client-side object has a number of methods for getting the value of the control:

Method	Return Type	Description
get_value(),	NumericTextBoxSetting: number	Gets or sets the value of the TextBox control.
set_value()	All other input controls: string	

In addition to the methods listed above which are present in the client-side object for all the **RadInputManager** target controls, the **TextBox** controls targeted under **DateInputSetting** have the following additional methods:

Method	Return Type	Description
--------	-------------	-------------

get_selectedDate() Date
 set_selectedDate() none

Gets the value of the control as a Date value.
 Sets the value of the control.

[ASP .NET] Getting values on the client

```
<telerik:RadCodeBlock ID="RadCodeBlock1" runat="server">
<script type="text/javascript">
function pageLoad()
{
    var inputManager = $find("<%= RadInputManager1.ClientID %>");
    var input = inputManager.get_targetInput("<%= TextBox1.ClientID %>");
    input.set_value("Value Client Side");
}
</script>
</telerik:RadCodeBlock>
```

Getting values Server-Side

On the Server-side, you can operate with the TextBox value directly through the instance of the TextBox control. Use the **Text** property of the TextBox to set/get its value.

[ASP .NET]

```
<asp:TextBox ID="TextBox1" runat="server">
</asp:TextBox>
<telerik:RadInputManager ID="RadInputManager1" runat="server">
    <telerik:TextBoxSetting BehaviorID="Behavior1">
        <TargetControls>
            <telerik:TargetInput ControlID="TextBox1" />
        </TargetControls>
    </telerik:TextBoxSetting>
</telerik:RadInputManager>
```

[C#] Setting values server-side

```
protected void Page_Load(object sender, EventArgs e)
{
    TextBox1.Text = "Setting Value Server-Side";
}
```

[VB] Setting values server-side

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    TextBox1.Text = "Setting Value Server-Side"
End Sub
```

Validation

You can use ASP.NET validators for validating TextBox controls targeted in the **RadInputManager** settings. In this case validation works the same way as for regular TextBox controls. You simply have to set the ID of the TextBox control as the value of the **ControlToValidate** property of the validator.

Using different Culture

With the **RadInputManager** you can specify different Culture per **RadInputManager** setting. In general this Culture is usually different from the current Page Culture. However, when setting values, the value you assign to the **TextBox** control should correspond to the current Page Culture. You should mind this rule especially when entering dates and floating point numbers.

3.9 Summary

In this chapter you took a tour of the "input" related RadControls and became familiar with how and where they are used. You saw some of the important properties, and noted where they all shared common properties. You created a simple application that used all four types of input control and made use of some of the common properties such as labels and empty messages. You learned to use the server-side API to respond to user input and to create input controls dynamically. You learned to perform common client-side tasks such as enabling and disabling some controls based on the responses to others, restricting input as the user types, and handling parsing errors. You also learned to use the input controls with other controls such as an ASP.NET validator or **RadSpellCheck**.

4 Client-Side API

4.1 Objectives

- Learn the basic techniques for getting RadControls object references in client code.
- Use RadControl properties and methods. Learn the naming convention that will help you out in most RadControls client programming.
- Learn how to use the JavaScript IntelliSense, provided out-of-the-box by RadControls for ASP.NET AJAX Q1 2010 and later.
- Learn how to use RadControl client events, the standard parameter list and naming convention. Learn how to attach and detach events on-the-fly.
- Build an application that displays a bread crumb trail as the mouse hovers over a set of hierarchical tabs. This application incorporates knowledge on how to get object references, how to use client methods and events, and how to build and insert HTML on-the-fly.

4.2 Introduction

RadControls for ASP.NET AJAX brings a rich set of API objects, methods and events to client-side programming that let you achieve complicated tasks with maximum speed and flexibility. It is important to get familiar with client programming early on because every RadControl has a client API that can be used on its own or together with AJAX so that sever and client functionality work smoothly together.

The client API is designed to be consistent between RadControls. Once you learn how to reference a RadControl, call client methods and respond to events, you're on your way to working with the rest of the controls the same way.

4.3 Referencing RadControl Client Objects

There are two helpful short cut methods supplied by the Microsoft AJAX Library, `$find()` and `$get()`, that are used to locate objects on the page:

- **`$find()`**: Provides a shortcut to the **`Sys.Application.findComponent()`** method, which returns the specified Component object. Expect to use this method every time you reference a RadControl on the client. This next example shows `$find()` being used in its simplest form:

[JavaScript] Using `$find()`

```
var menu = $find("RadMenu1");
```

In some cases, "RadMenu1" will be present, but `$find("RadMenu1")` will return null. A safer way to find your RadControl is to use a server tag to output the control ClientID to the `$find()` method. We leave it up to the RadControl to figure out the correct ClientID in case the control is nested within a master page or user control and the ClientID wouldn't be what we expect:

[JavaScript] Using `$find()` with Server Tag

```
var menu = $find("<%= RadMenu1.ClientID %>");
```



ID and ClientID

The **ID** property of a control identifies an ASP.NET server control. The ID is only unique within the current NamingContainer (page, user control, item template).

The **ClientID** property is unique within the entire page. The ClientID will be rendered with the container control, an underscore *and* the control ID. If "RadMenu1" is located directly on the page the two

UI for ASP.NET AJAX

properties would be:

- ID: "RadMenu1"
- ClientID: "RadMenu1"

If "RadMenu1" is located in a user control "WebUserControl1", the properties are:

- ID: "RadMenu1"
- ClientID: "WebUserControl1_RadMenu1"

See the Telerik blog "The Difference between ID, ClientID and UniqueID" by Atanas Korchev for additional exploration of this topic.

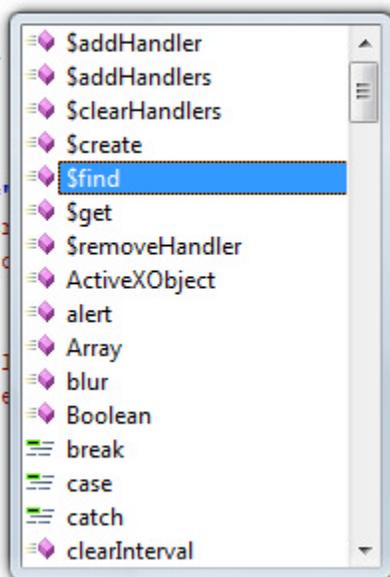
- **\$get(id, parentElement):** This method is just for finding generic HTML elements, not RadControls. \$get() Provides a shortcut to the getElementById() method. "parentElement" is the element to search but is optional. By default, the document is searched.

[JavaScript] Using \$get()

```
var myDiv = $get("myDiv");
```

Fortunately, Visual Studio 2008 has some advanced client-side capabilities including JavaScript debugging, JavaScript IntelliSense and even CSS style intellisense. If you enter a <script> tag to your markup and press Ctrl-Space, JavaScript IntelliSense is invoked and shows available properties and methods:

```
<script type="text/jscript">
  var menu =
</script>
</head>
<body>
  <form id="
  <asp:Sc
  </asp:Sc
  <div>
    <tel
    </tel
  </div>
</form>
</body>
</html>
```



Gotcha! If you can't find any of the "\$" functions, it's likely you don't have a ScriptManager or RadScriptManager on the page. The ScriptManager component brings in the MS AJAX library of functions.



Just for fun, click the Ctrl key to temporarily hide the IntelliSense window so you can see the code below:

```

<script type="text/jscript">
  var menu = |
</script>
</head>
<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManage
    </asp:ScriptManager>
    <div>
      <telerik:RadMenu ID="RadMenu1"
      </telerik:RadMenu>
    </div>
  </form>
</body>
</html>

```



Once you begin typing, IntelliSense provides a hint window with the parameters for the current context (or press Ctrl-Shift-Space to invoke the window). You can see in the screenshot below that an ID is required.

```

<script type="text/jscript">
  var menu = $find(
    Sys.Component $find (String id, parent)
  </script>

```

Finish up by typing the server tag "<%= %>" and reference the RadControl ClientID:

```

<script type="text/jscript">
  var menu = $find("<%=RadMenu1.ClientID%>");
</script>

```

Now you have a reference to your RadControl client object and can use its properties and methods.

For more on the Microsoft AJAX Library, see the [Client Reference \(http://msdn.microsoft.com/en-us/library/bb397536.aspx\)](http://msdn.microsoft.com/en-us/library/bb397536.aspx).

4.4 Using RadControl Client Properties and Methods

Use the online help to list available methods or a JavaScript debugging utility to query the available methods of a client object. You can usually find methods that mirror server side functionality. For example, the JavaScript snippet below shows how to find a menu item by a Text value "Tickets" and perform a method on that item:

[JavaScript] Using Client Object Methods

```

var menu = $find("<%= RadMenu1.ClientID %>");
var item = menu.findItemByText("Tickets");
if (item)
{
  item.open();
}

```

UI for ASP.NET AJAX

```
else
{
    alert("Tickets item not found.");
}
```

Similar to its server-side counterpart, a collection can be iterated and each collection member can have its methods called. In this example, all items of a RadMenu are returned, iterated and output to an alert dialog:

[JavaScript] Collection Methods

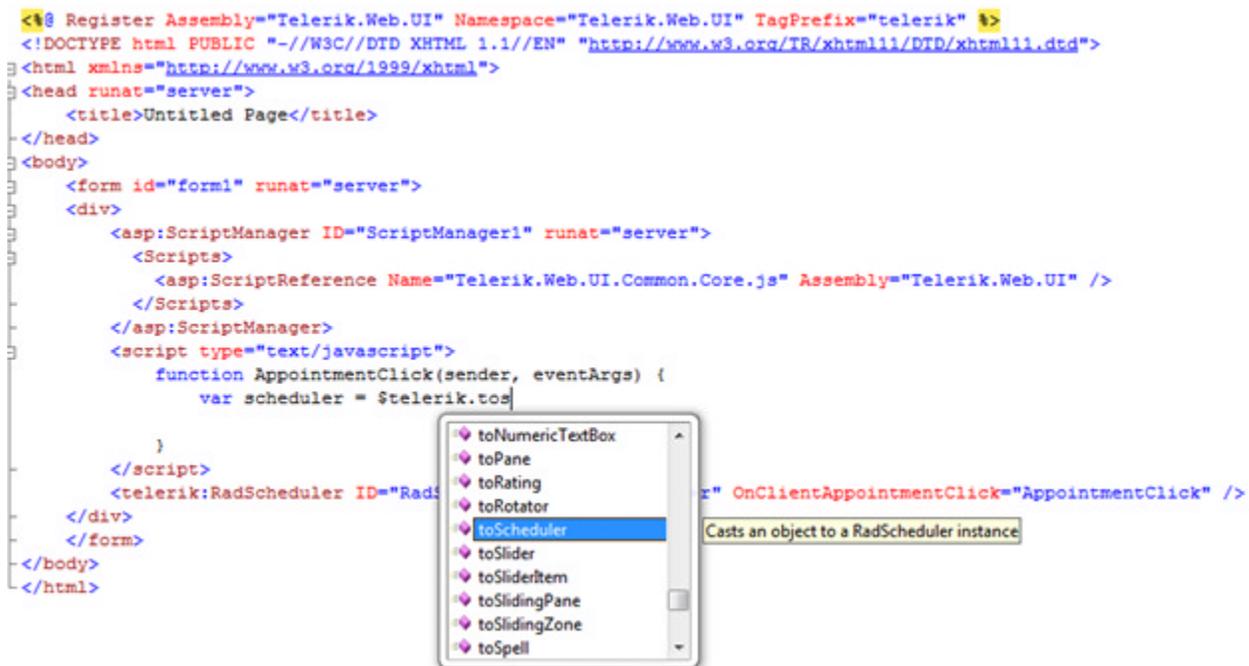
```
var menu = $find("<%= RadMenu1.ClientID %>");
var items = menu.get_items();
for (var i=0; i < items.get_count(); i++)
{
    alert(items.getItem(i).get_text());
}
```

You can replace the *\$find* method with the ones declared in the Telerik's static library (*\$telerik.findGrid* for example) or cast the object returned by the *\$find* method (using *\$telerik.toGrid* for example) to a specific RadControl's client object and then use its methods and properties with the help of the provided JavaScript intellisense (Section 4.5). This new feature was introduced in RadControls for ASP.NET AJAX Q1 2010.

4.5 JavaScript Intellisense

Since the release of RadControls for ASP.NET AJAX Q1 2010 (version 2010.1.309) writing JavaScript code with RadControls for ASP.NET AJAX becomes very easy. No need to bury yourself in the client API documentation, no more annoying *js* errors being generated from non-existing properties/methods or mistyped code slices! Simply register *Telerik.Web.UI.Common.Core.js* using RadScriptManager/MS ScriptManager under VS 2010 or MS ScriptManager under VS 2008 and enjoy the out-of-the-box JavaScript IntelliSense in the markup of your pages.

Visual Studio 2008



```

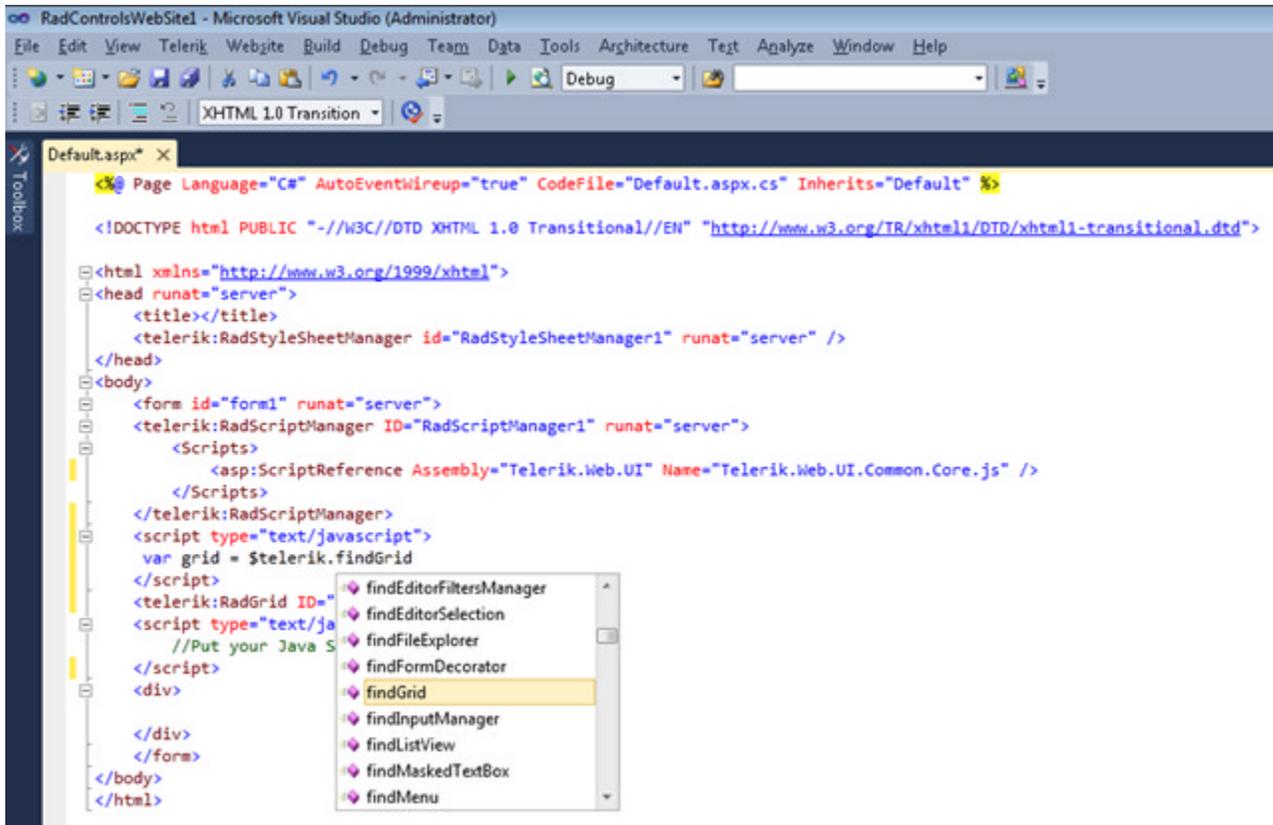
<@ Register Assembly="Telerik.Web.UI" Namespace="Telerik.Web.UI" TagPrefix="telerik" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
<form id="form1" runat="server">
<div>
<asp:ScriptManager ID="ScriptManager1" runat="server">
<Scripts>
<asp:ScriptReference Name="Telerik.Web.UI.Common.Core.js" Assembly="Telerik.Web.UI" />
</Scripts>
</asp:ScriptManager>
<script type="text/javascript">
function AppointmentClick(sender, eventArgs) {
var scheduler = $telerik.toScheduler()
}
</script>
<telerik:RadScheduler ID="RadScheduler1" runat="server" OnClientAppointmentClick="AppointmentClick" />
</div>
</form>
</body>
</html>

```

toScheduler (object)
Casts an object to a RadScheduler instance

An important detail here is that the [Telerik static client library](http://www.telerik.com/help/aspnet-ajax/telerik-static-client-library.html) (<http://www.telerik.com/help/aspnet-ajax/telerik-static-client-library.html>) exposes *to<RadControlName>(object)* and *find<RadControlName>(id, parent)* methods which gives you the ability to cast or find Telerik AJAX control's client object and then the IntelliSense will expose directly its properties, methods and events. Additionally, you will get information about the client methods signature and the type of the arguments passed to or returned from them.

Visual Studio 2010



UI for ASP.NET AJAX

```
<% Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<telerik:RadStyleSheetManager id="RadStyleSheetManager1" runat="server" />
</head>
<body>
<form id="form1" runat="server">
<telerik:RadScriptManager ID="RadScriptManager1" runat="server">
<Scripts>
<asp:ScriptReference Assembly="Telerik.Web.UI" Name="Telerik.Web.UI.Common.Core.js" />
</Scripts>
</telerik:RadScriptManager>
<script type="text/javascript">
var grid = $telerik.findGrid()
</script>
<telerik:RadGrid ID="RadGrid1" runat="server">
<script type="text/javascript">
//Put your JavaScript code here
</script>
</div>
</div>
</form>
</body>
</html>
```

Telerik.Web.UI.RadGrid findGrid(id, parent)
Finds a RadGrid instance
id: A string that contains ID of the RadGrid to find

```
<% Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs" Inherits="Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
<telerik:RadStyleSheetManager id="RadStyleSheetManager1" runat="server" />
</head>
<body>
<form id="form1" runat="server">
<telerik:RadScriptManager ID="RadScriptManager1" runat="server">
<Scripts>
<asp:ScriptReference Assembly="Telerik.Web.UI" Name="Telerik.Web.UI.Common.Core.js" />
</Scripts>
</telerik:RadScriptManager>
<script type="text/javascript">
var grid = $telerik.findGrid("#RadGrid1.ClientID", null);
grid.get_masterTableView().editItem()
</script>
<telerik:RadGrid ID="RadGrid1" runat="server">
<script type="text/javascript">
//Put your JavaScript code here
</script>
</div>
</div>
</form>
</body>
</html>
```

Method which switches the table row passed as an argument or the row corresponding to the index passed as an argument in edit mode. If you set AllowMultiRowEdit to true, you can switch multiple grid items in edit mode with subsequent calls to this method.

4.6 Naming Conventions

The client API follows naming conventions across all RadControls:

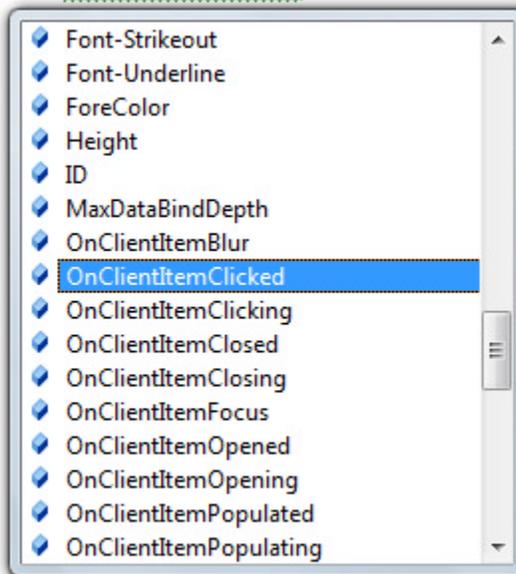
- **Methods** are lower camel-cased. That is, the first character is lower case and the following words making up the method name are title cased. For example `focusNextItem()`, `hide()`, `findControl()`.
- **Properties** are made up of getter and setter methods. The naming consists of the `get/set`, and underscore and lower-camel-cased property name. For example `get_imageUrl()`, `set_imageUrl()`.
- **Internal Methods** are preceded with an underscore. These methods are not intended for public use.
- **Legacy Methods and Properties** may still be present and show upper-camel-casing, e.g. `FocusNextItem()`. Because these are legacy methods and properties, they are deprecated and you cannot count on these methods remaining usable.

4.7 Using Client Events

Each RadControl has a set of client events that you define in the markup or at design-time in the Property window.

If you're working in ASP.NET markup, RadControls work with Visual Studio 2008 IntelliSense to help you find the available client events. When you drop a RadControl on the form, an XML file containing comments for classes properties and methods is automatically added to the bin directory. In the markup, when you type into a RadControl tag or press `Ctrl-Space`, a list of appropriate attributes pops up automatically. As you type, the list will locate on the first letters typed. All RadControl client events are prefixed with "OnClient", so they should be easy to find:

```
<telerik:RadMenu ID="RadMenu1" runat="server" onclientitemcli >
</telerik:RadMenu>
```



To create a client event handler, you enter a JavaScript function name to the "OnClient..." property and create a JavaScript function to match. The parameter list of a RadControl client function will always include "sender", i.e. the initiating object and "args". "Args" contains methods specific to the control and the event. The example below shows the `OnClientItemClicked` event handled by a `itemClicked()` function. In this case "sender" is the RadMenu client object and "args" contains a `get_item()` function. `get_item()`, as you might have guessed returns the menu item that was clicked on. Using the item object returned from `get_item()` you can call the RadMenuItem client methods, i.e. `get_text()`, `get_value()`, `get_level()`.

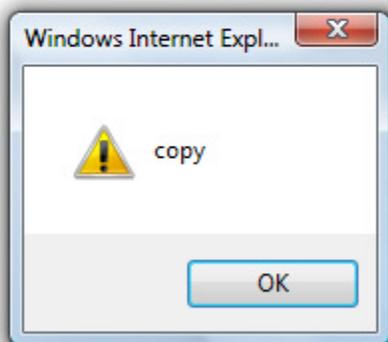
```
<script type="text/jscript">

function itemClicked(sender, args)
{
  if(args.get_item().get_level() == 1)
  {
    alert(args.get_item().get_value());
  }
}

</script>

<form id="form1" runat="server">
  <asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
  <div>
    <telerik:RadMenu ID="RadMenu1" runat="server" Skin="Web20"
      OnClientItemClicked="itemClicked">
      <Items>
        <telerik:RadMenuItem Text="Edit">
          <Items>
            <telerik:RadMenuItem Text="Cut" Value="cut"></telerik:RadMenuItem>
            <telerik:RadMenuItem Text="Copy" Value="copy"></telerik:RadMenuItem>
            <telerik:RadMenuItem Text="Paste" Value="paste"></telerik:RadMenuItem>
          </Items>
        </telerik:RadMenuItem>
      </Items>
    </telerik:RadMenu>
  </div>
</form>
```

In the example, an alert dialog displays the value for the clicked menu item. The itemClicked() client event handler first checks that the item has a "level" of 1, i.e., is a child item, so that clicking the parent "Edit" item will not display the alert.

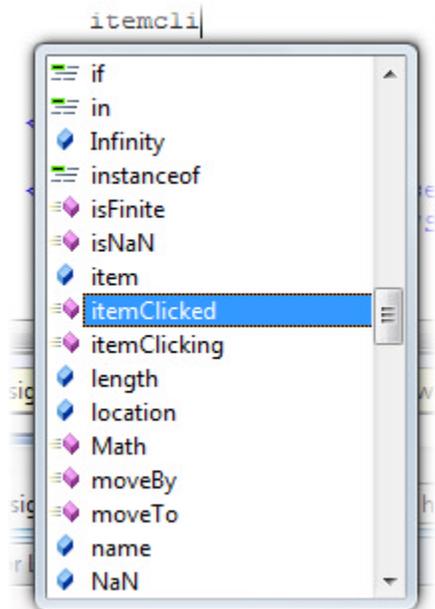



```
<div>
  <telerik:RadMenu ID="RadMenu1" runat="server" Skin="Web20"
    OnClientItemClicked="itemClicked"
    OnClientItemClicking="itemClicking">
    <Items>

      . . .

      <telerik:RadMenuItem Text="Web sites">
        <Items>
          <telerik:RadMenuItem Text="Telerik"
            NavigateUrl="http://www.telerik.com (http://www.telerik.com/)">
          </telerik:RadMenuItem>
          <telerik:RadMenuItem Text="Falafel"
            NavigateUrl="http://www.falafel.com (http://www.falafel.com/)">
          </telerik:RadMenuItem>
          <telerik:RadMenuItem Text="Notes"
            NavigateUrl="#notes">
          </telerik:RadMenuItem>
        </Items>
      </telerik:RadMenuItem>
    </Items>
  </telerik:RadMenu>
  <br /><br /><br /><br /><br /><br />
  <div id="notes">
    Some events can be canceled.</div>
</div>
</form>
```

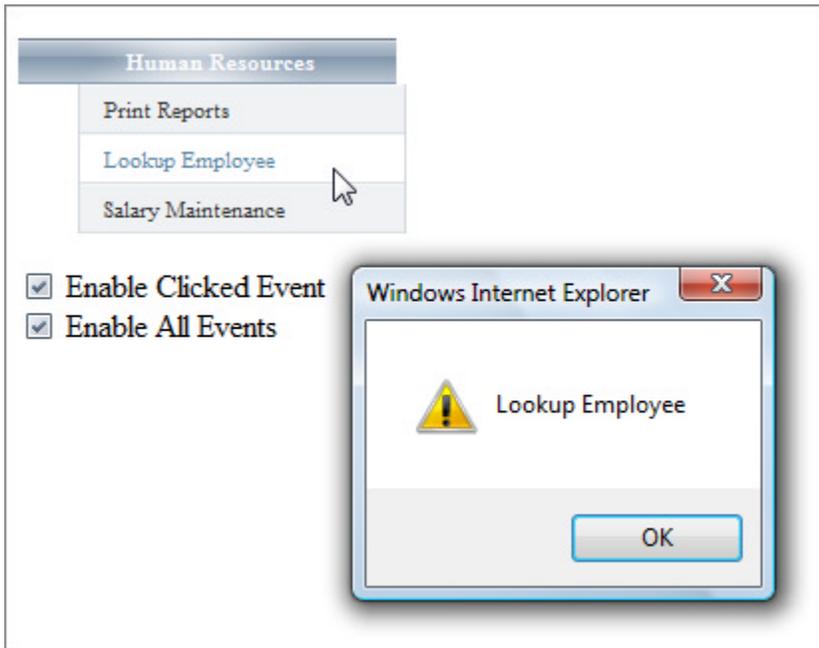
Did you notice the comments in the JavaScript above that start with three slashes? These provide IntelliSense help information whether you add your JavaScript directly to the page or to a separate .js file. Your new functions `itemClicked` and `itemClicking` now show up:



Adding and Removing Events Dynamically

You can also add or remove events on-the-fly. The naming convention here is "add_" + the event name. For example "add_itemClicked()". If you want to temporarily "mute" all events for a RadControl on the client, call

the `disableEvents()` method (or its corresponding `enableEvents()` method to "un-mute"). This next example shows how you can use a check box to toggle the `OnClickItemClicked` event and events as a whole for a `RadMenu`. Both check boxes need to be enabled for the event handler to fire.



When the application first runs, there is no event handling for the menu. When the "Enabled Clicked Event" checkbox is clicked, `checkItemClick()` runs. If the check box is checked, then the `add_itemClicked()` method is called, passing the event handler name "itemClicked". Likewise, if un-checked, the menu's `remove_itemClicked()` method is called, passing the same "itemClicked" event handler name. The same pattern is used for the "Enable All Events" check box.

You can find the project for this example at `\VS Projects\Client API\Events`.

[ASP.NET] Adding and Removing Event Handlers

```
<script type="text/javascript">
function itemClicked(sender, args)
{
    // display the text for the clicked on item
    alert(args.get_item().get_text());
}
function checkItemClick()
{
    // get a reference to the menu
    var menu = $find("<%=RadMenu1.ClientID %>");
    // get a reference to the checkbox
    var checkbox = $get("cbClicked");

    if (checkbox.checked)
    {
        // add the event handler
        menu.add_itemClicked(itemClicked);
    }
    else
    {
        // remove the event handler
        menu.remove_itemClicked(itemClicked);
    }
}
</script>
```

```

    }
}

function checkAllEvents()
{
    // get a reference to the menu
    var menu = $find("<%=RadMenu1.ClientID %>");
    // get a reference to the checkbox
    var checkbox = $get("cbAll");

    if (checkbox.checked)
    {
        // add the event handler
        menu.enableEvents();
    }
    else
    {
        // remove the event handler
        menu.disableEvents();
    }
}
}

</script>
<form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <telerik:RadFormDecorator ID="RadFormDecorator1" runat="server" Skin="WebBlue" />
<div>

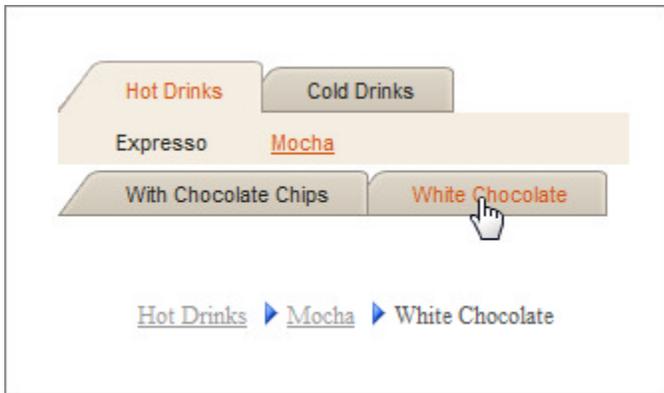
    <telerik:RadMenu ID="RadMenu1" Runat="server" Skin="WebBlue">
        <Items>
            <telerik:RadMenuItem runat="server" Text="Human Resources">
                <Items>
                    <telerik:RadMenuItem runat="server" Text="Print Reports"></telerik:RadMenuItem>
                    <telerik:RadMenuItem runat="server" Text="Lookup Employee"></telerik:RadMenuItem>
                    <telerik:RadMenuItem runat="server" Text="Salary
Maintenance"></telerik:RadMenuItem>
                </Items>
            </telerik:RadMenuItem>
        </Items>
    </telerik:RadMenu>

    <br /><br /><br /><br /><br />
    <input id="cbClicked" type="checkbox" onclick="checkItemClick()" />Enable Clicked Event
    <br />
    <input id="cbAll" type="checkbox" onclick="checkAllEvents()"
        checked="checked" />Enable All Events
</div>
</form>

```

4.8 Client Events Walk Through

This next tutorial will put together some of the client techniques we've described so far. You will use JavaScript to display a "bread crumb" trail while the user moves the mouse over a multi-level tab strip. This technique can be easily adapted to any of the hierarchical navigation controls and could also be coded on the server-side.



1. Create a new web application. Add a ScriptManager to the default page.
2. In the Solution Explorer, add a new folder and name it "Images".
3. From the Visual Studio 2008 installation directory, copy the image "DataContainer_MoveNextHS.png" to the project \Images directory. *This image will contain the rightward pointing arrow that displays between each crumb.*

 Images from Visual Studio 2008 can be found at \Microsoft Visual Studio 9.0\Common7\VS2008ImageLibrary\1033\VS2008ImageLibrary\VS2008ImageLibrary\Actions\pngformat

4. Add a RadTabStrip to the default page. Set the **Skin** property to "Sunset", the **OnClientMouseOut** property to "mouseOut" and the **OnClientMouseOver** property to "mouseOver". We will code the two client event handlers later, after we set up the tab strip items.
5. Copy the ASP.NET markup below to inside your RadTabStrip tags. *This step will populate the tab strip with multiple levels of tabs that can best demonstrate the bread crumbs in action.*

[ASP.NET] Defining the Tabs

```
<Tabs>
  <telerik:RadTab runat="server" Text="Hot Drinks">
    <Tabs>
      <telerik:RadTab runat="server" Text="Expresso">
        </telerik:RadTab>
      <telerik:RadTab runat="server" Text="Mocha">
        <Tabs>
          <telerik:RadTab runat="server" Text="With Chocolate Chips">
            </telerik:RadTab>
          <telerik:RadTab runat="server" Text="White Chocolate">
            </telerik:RadTab>
          </Tabs>
        </telerik:RadTab>
      </Tabs>
    </telerik:RadTab>
  </Tabs>
  <telerik:RadTab runat="server" Text="Cold Drinks" >
    <Tabs>
      <telerik:RadTab runat="server" Text="Frappuccino">
        </telerik:RadTab>
      <telerik:RadTab runat="server" Text="Iced Coffee">
        </telerik:RadTab>
      <telerik:RadTab runat="server" Text="Thai Ice Tea">
        </telerik:RadTab>
      </Tabs>
    </telerik:RadTab>
  </Tabs>
```

```
</telerik:RadTab>  
</Tabs>
```

6. Below the RadTabStrip tab, create a div called "breadCrumbDiv". *The div only needs to have an id so we can locate it. The div will be populated on the fly in client code.*

[ASP.NET] Adding the div that will display the breadcrumb

```
<div id="breadCrumbDiv" ></div>
```

7. In the <head> tag enter the following CSS. *The CSS will style the HTML elements of the breadcrumb trail, which in turn is formed using an HTML un-ordered list . Notice that the list element background automatically places our right-ward pointing arrow graphic "DataContainer_MovenextHS.png" next to each list element.*

[ASP.NET] CSS to Style the Breadcrumbs

```
<style type="text/css">  
#Breadcrumbs  
{  
    position: absolute;  
    top: 135px;  
}  
#Breadcrumbs li  
{  
    color: #999;  
    text-decoration: underline;  
    padding: 0 20px 0 0;  
    float: left;  
    background: transparent url("Images/DataContainer_MoveNextHS.png") no-repeat center  
right;  
    font: 12px "Times New Roman", serif;  
}  
#Breadcrumbs li#LastItem  
{  
    background: none;  
    padding-right: 0;  
    color: #515151;  
    text-decoration: none;  
}  
</style>
```

8. Add a set of <script> tags just inside the <body> tag.
9. Inside the <script> tag add two functions mouseOver(sender, args) and mouseOut(sender, args). Also add a stub for a helper function getPathList(tab). *The getPathList() function will walk starting from the tab under the mouse up to the root node and return an array containing the tab text found along the way:*

[JavaScript] Adding the Client Event Handlers

```
<script type="text/javascript">  
  
function getPathList(tab)  
{  
}  
  
function mouseOver(sender, args)  
{
```

```

}

function mouseOut(sender, args)
{

}
</script>

```

10. Populate `getPathList()` with the following code that a) creates a new Array object called "result", b) iterates until we reach the root item, c) return the result. *The while loop tests for `tab.get_text` to come back null. Notice that the statement doesn't state `tab.get_text()` -- that would fail when we got to the ultimate parent item, the tab strip object itself. Instead we check that the `get_text` function exists. When it doesn't, we're no longer looking at a tab object, but the tab strip. Inside the while loop we use the `push()` method to add the text of each tab item to the array, then get the next parent before looping again.*

[JavaScript] Getting the Path as an Array

```

function getPathList(tab)
{
    // create an array object to return
    result = new Array();
    // loop until the get_text function is null,
    // indicating that we've reached the tab strip object
    while (tab.get_text)
    {
        // save off the text for the tab we're looking at
        result.push(tab.get_text());
        // get the next parent
        tab = tab.get_parent();
    }
    return result;
}

```

11. Populate the `OnClientMouseOut` event handler. *Here we get a reference to the div object and simply clear the contents of the tag.*

[JavaScript] Handling the OnClientMouseOut Event

```

function mouseOut(sender, args)
{
    // get a reference to the div object
    var div = $get("breadCrumbDiv");
    // clear the text
    div.innerHTML = '';
}

```

12. Populate the `OnClientMouseOver` event handler. *Start by getting references to the div that will display the bread crumb trail and the tab that the mouse passed over. Call `getPathList()` and pass the tab reference. Then build the HTML starting with the un-ordered list tag (``) and adding on list item tags for each element in the `pathList` array. Provide a special id "LastItem" just before exiting the loop so that the CSS style for the last item can be applied. Finally, assign the built HTML to the div tag `innerHTML` attribute.*

[JavaScript] Handling the OnClientMouseOver Event

```

function mouseOver(sender, args)
{
    // get a reference to the div that

```

```
// will display the bread crumb trail
var div = $get("breadCrumbDiv");
// get a reference to the tab that the
// user just "moused" over to trigger this event
var tab = args.get_tab();
// call getPathList to get a list of text for
// all tabs starting with the tab passed in
// args up to the root item
var pathList = getPathList(tab);

// declare a variable to contain the breadcrumb html
var crumbText = "<ul id='Breadcrumbs'>";

// iterate the list of tabs in the trail,
// starting with the last in the list and
// backup up to the first.
for (var i = (pathList.length - 1); i >= 0; i--)
{
    // If this isn't the last iteration, add a
    // list item tag.
    if (i != 0)
    {
        crumbText += "<li>";
    }
    // this is the last item, so flag it with the
    // id "LastItem" so the CSS can style it appropriately
    else
    {
        crumbText += "<li id='LastItem'>";
    }

    // add the tab text
    crumbText += pathList[i];
}

// assign the assembled HTML to the div
div.innerHTML = crumbText;
}
```

13. Press Ctrl-F5 to run the application. Open up the tabs to get as much depth as possible and run your mouse over the tabs.

4.9 JSON: Fat-Free Data Interchange

JSON stands for "JavaScript Object Notation" and is a lightweight data-interchange format. JSON is easy to generate and parse but also easily human-readable. JSON has a number of advantages in the JavaScript/client environment:

- JSON can be used as an easy-to-work-with alternative to XML.
- JSON can be de-serialized into objects and the objects serialized back into strings. There are API's that can do these transformations on both the client and server.
- Webservices can return JSON automatically for immediate use within JavaScript.

JSON supports the usual basic type flavors: numbers, strings, booleans, arrays, objects and null.

The quickest way to understand how the JSON syntax works is to look at an example. Below is a sample JSON

object definition called "contact". It has string properties for "firstName" and "lastName". Another property, "address" is an object that has its own properties for "streetAddress", "city", "state" and "postalCode". These address properties are all string except "postalCode" that contains a numeric value. The last property "phoneNumbers" is actually an array of strings.

[JavaScript] JSON Sample

```
var contact = {
  // string property
  "firstName": "John",
  "lastName": "Smith",
  // address property with sub-properties
  "address": {
    "streetAddress": "21 2nd Street",
    "city": "New York",
    "state": "NY",
    // numeric property
    "postalCode": 10021
  },
  // array
  "phoneNumbers": [
    "212 555-1234",
    "646 555-4567"
  ]
};
```

As you can see in the sample above, the JSON object definition appears between curly braces. Each property and value pair are separated by a colon. Arrays are contained within square brackets.

Using JSON Objects

Once the JSON object is defined you can assign and retrieve values using the properties of the object. In this next sample the "contact" object is assigned a new first and last name and the second element of the phoneNumbers array is also replaced with a new value.

[JavaScript] Assigning and Retrieving JSON Properties

```
// change the name and phoneNumbers properties
contact.firstName = "Bob";
contact.lastName = "Jones";
contact.phoneNumbers[1] = "123 555-9999";
alert(contact.firstName + ' ' +
  contact.lastName + ' phone: ' + contact.phoneNumbers[1]);
```

Running this bit of JavaScript fires the alert shown below:



Serializing JSON

UI for ASP.NET AJAX

You can also take a JSON string and transform it into an object. The ASP.NET AJAX Library includes a JavaScriptSerializer object within the Sys.Serialization namespace that you get for free when you include a ScriptManager on the page. If you call the JavaScriptSerializer deserialize() method and pass a JSON string, the method will deserialize the string into a JSON object. Call the serialize() method to transform the a JSON object back to a string.

The sample below shows a JSON string defined for "contact". This is exactly the same as the "contact" object defined in the last example, but surrounded with quotes. A call to deserialize() takes the contact JSON string and transforms it into an object representation. Following that, the contact object is converted back using the serialize() method into its string representation.

[JavaScript] Serialize and Deserialize

```
var contactString = '{"firstName": "John", "lastName": "Smith", ' +  
  '"address": {"streetAddress": "21 2nd Street",' +  
  '"city": "New York","state": "NY", "postalCode": 10021},' +  
  '"phoneNumbers": ["212 555-1234","646 555-4567"]}';  
// deserialize JSON string to an object  
contact =  
  Sys.Serialization.JavaScriptSerializer.deserialize(contactString);  
// serialize the contact JSON into a string  
var contactStrings =  
  Sys.Serialization.JavaScriptSerializer.serialize(contact);
```

Both of these operations can happen on the server too using the JavaScriptSerializer object from the System.Web.Script.Serialization namespace. The example below uses a Contact object (definition not shown here) that is instantiated, populated, serialized and deserialized.

[VB] Serializing and Deserializing in Code-Behind

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)  
  Dim contact As New Contact()  
  contact.FirstName = "Bob"  
  contact.LastName = "Smith"  
  contact.Address.City = "San Francisco"  
  contact.Address.State = "California"  
  contact.Address.StreetAddress = "123 Telerik Ave"  
  contact.Address.PostalCode = 91234  
  contact.PhoneNumbers.Add("123 555-1234")  
  contact.PhoneNumbers.Add("444 555-9876")  
  Dim jss As New JavaScriptSerializer()  
  Dim contactString As String = jss.Serialize(contact)  
  tbServerStatus.Text = contactString  
  Dim contact2 As Contact = jss.Deserialize(Of Contact)(contactString)  
  tbServerStatus.Text += System.Environment.NewLine + System.Environment.NewLine +  
  contact2.FirstName + " " + contact2.LastName  
End Sub
```

[C#] Serializing and Deserializing in Code-Behind

```
protected void Page_Load(object sender, EventArgs e)  
{  
  Contact contact = new Contact();  
  contact.FirstName = "Bob";  
  contact.LastName = "Smith";  
  contact.Address.City = "San Francisco";  
  contact.Address.State = "California";  
  contact.Address.StreetAddress = "123 Telerik Ave";  
  contact.Address.PostalCode = 91234;
```

```
contact.PhoneNumbers.Add("123 555-1234");
contact.PhoneNumbers.Add("444 555-9876");
JavaScriptSerializer jss = new JavaScriptSerializer();
string contactString = jss.Serialize(contact);
tbServerStatus.Text = contactString;
Contact contact2 = jss.Deserialize<Contact>(contactString);
tbServerStatus.Text += System.Environment.NewLine + System.Environment.NewLine +
    contact2.FirstName + " " + contact2.LastName;
}
```

4.10 MS AJAX Library

4.11 Summary

In this chapter you learned the basic techniques used to obtain RadControl object references in client code and how to call methods and use properties of the client objects. You learned the consistent naming convention used throughout the RadControls client API so that you can re-apply that knowledge on new controls. You learned how to implement client side event handlers and how to add and remove event handlers on-the-fly. Finally, you built a web page with a tabbed interface that displays a breadcrumb trail as the mouse hovered each tab.

5 User Interface and Information Controls

5.1 Objectives

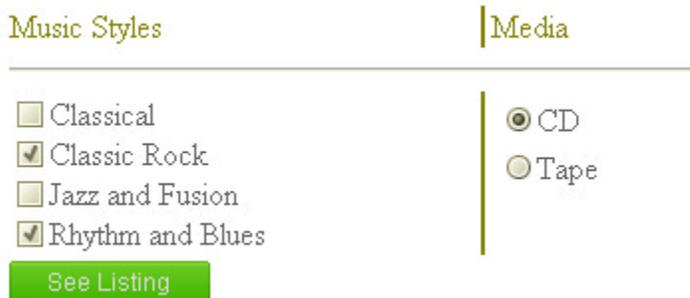
- Examine how RadFormDecorator, RadToolTipManager, and RadToolTip can add polish to your user interface.
- Create a simple application to get confidence in using each of the controls.
- Become familiar with the design time support for working with the user interface and information controls. This support includes Smart Tag, Properties Window, ToolTipTargetControl Collection Editor, and the RadToolTip design surface.
- Explore principal properties and groups of properties where 80% of the functionality is found.
- Learn to supply tool tip content in server-side code.
- Learn how to use the client-side api to work with tool tip properties and control when tool tips appear and disappear.
- Learn how to use RadToolTip to provide tool tips for the areas of an ASP.NET ImageMap.

5.2 Introduction

As you have seen with the RadControls we have examined so far, they all support skinning to give your Web site a consistent look and feel. This adds a level of polish to your application that is simple to achieve. The controls we will examine in this chapter let you extend that skin-based look and feel to standard ASP.NET elements such as buttons or tool tips.

RadFormDecorator

There are no RadControl analogs to the standard ASP.NET Button, CheckBox, RadioButton, or ScrollBar controls. However, when you want to add this functionality to your Web pages, this does not mean that you must go to great lengths in order to make them fit in with the skin you are using. There is a simple way to augment these controls by adding a skinning capability: the **RadFormDecorator** control. When you add RadFormDecorator to your page, you can configure it to apply a skin to any or all of the buttons, check boxes, radio buttons, or scroll bars on the page.



RadToolTipManager

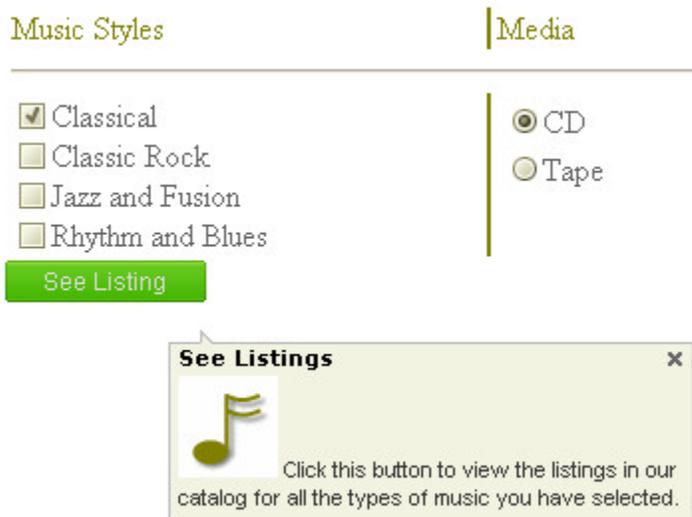
You can use **RadToolTipManager** to apply your preferred skin to all of the tool tips on your page. RadToolTipManager will automatically replace the standard ASP.NET tool tips with customized tool tips that can be as simple or elaborate as you want.



When using RadToolTipManager, you have complete control over the content and behavior of the tool tips on a page. You can add tool tips to any or all of the elements on the page. You can specify when, where, and how those tool tips appear and disappear, add animated effects, and even add your own custom content using an asynchronous server-side callback.

RadToolTip

Where RadToolTipManager associates custom tool tips with multiple elements on the Web page, you can use **RadToolTip** to create a customized tool tip for a single element. It shares many properties with RadToolTipManager, so that you have the same level of control over when, where, and how the tool tip appears and disappears. An advantage to using RadToolTip is that you can add custom content using the Visual Studio designer, rather than in the code-behind.



5.3 Getting Started

In this walk-through you will become familiar with the **RadFormDecorator**, **RadToolTipManager**, and **RadToolTip** controls. These controls will be used to produce the form shown in the following screen shot:



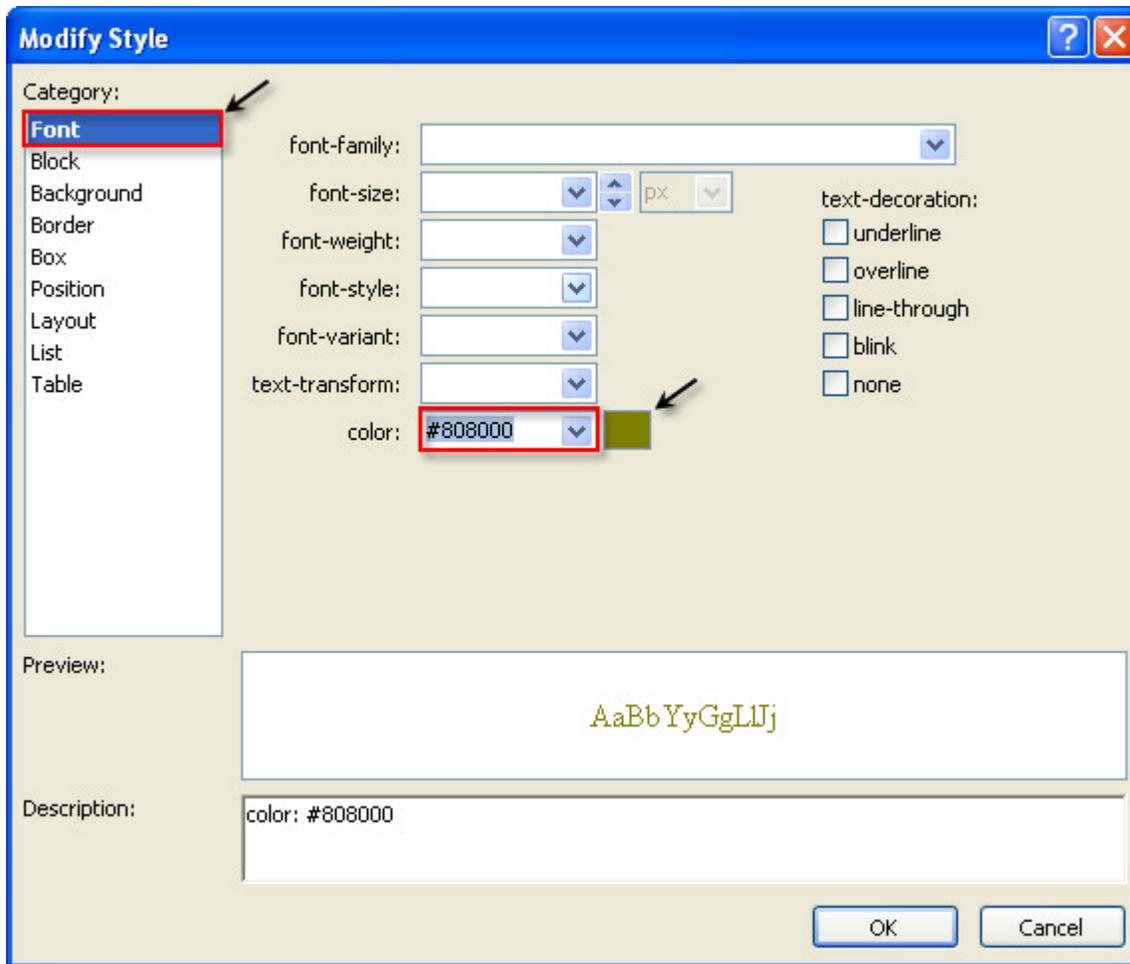
When you are finished, your project should match the one supplied in \VS Projects\UI\GettingStarted.

Set up the project structure

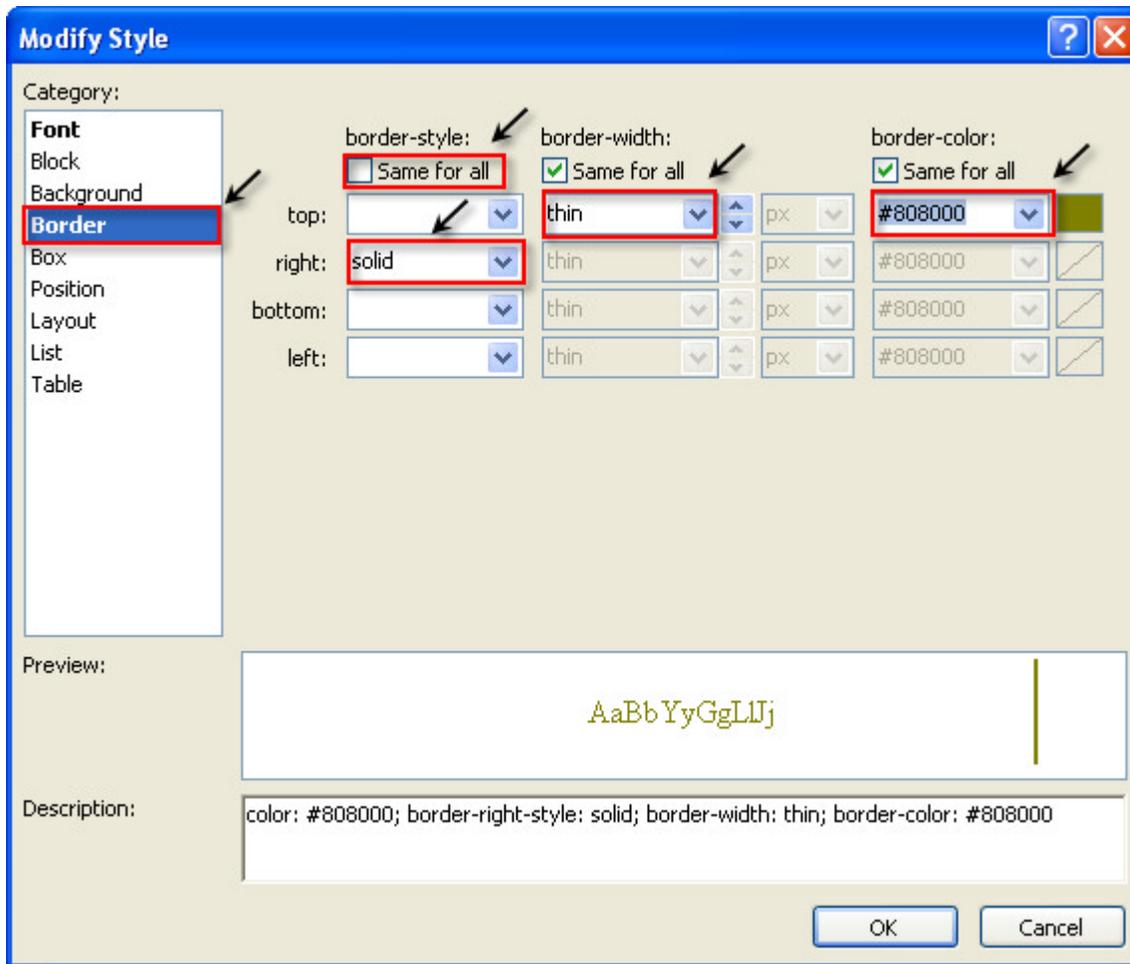
1. Create a new ASP.NET Web Application.
2. In the designer, drag a ScriptManager from the AJAX extensions section of the tool box onto your page.
3. In the solution explorer, create a new \Images folder.
4. Copy the image "music.png" from the \VS Projects\Images folder to your project's \Images folder. This image will appear in a custom tool tip (as shown above).

Build the Web page using standard ASP.NET controls

1. From the HTML section of the Tool Box, drag a **Table** onto your page.
2. Use the Properties Window to assign an ID of "tblOptions" to the table.
3. Select the upper left cell of the table. Then, click the ellipsis button next to the **Style** attribute to display the Modify Style dialog.
4. On the **Font** page of the dialog, set the font color to "Olive".



5. On the **Border** page of the dialog, uncheck the **Same for all** box under border-style and set the right border style to "solid", set the border-width to "thin", and the border-color to "Olive". Then click OK to exit the dialog.



6. Select the cell immediately to the left of the one you just modified, and bring up its Modify Style dialog. In the dialog, set the font color to "Olive" and click OK.
7. If any additional columns appear to the right, select a cell in the column, right click, and select **Delete | Delete Columns** to get rid of them.
8. Select the left-most cell in the second row. Using the Properties Window, set its ColSpan attribute to 2. Then delete the cell to its right.
9. From the HTML section of the Tool Box, drag a **Horizontal Rule** into the cell.
10. Select the left-most cell in the third row, and bring up its Modify Style dialog. Set the Border attributes to match the cell in the upper left corner (a solid right border with "thin" border-width and olive border-color).
11. In the Properties Window for this cell, set the valign attribute to "top" and the Title attribute to "Check all the styles you want to see included in the listing."
12. Select the cell to the right of the cell you just changed, and using the Properties Window, set its valign attribute to "top" and its Title attribute to "Choose the media you want."
13. There should not be any more rows in the table, but if their are, delete them.
14. Back in the upper left cell, type the text "Music Styles". In the upper right cell, type "Media".
15. From the Standard Section of the Tool Box, drag a **CheckBox** control into the upper left cell of the table. Set its **Text** property to "Classical".
16. Add a line break after the check box, and then add another one with the **Text** property of "Classic Rock".
17. Add two more check boxes in the same fashion, and set their **Text** properties to "Jazz and Fusion" and

"Rhythm and Blues".

18. Drag a **RadioButtonList** control from the Tool Box into the upper right cell of the table. Click on the **Edit Items...** link of its Smart Tag to bring up the ListItem Collection Editor. In the editor, add two radio buttons, with **Text** properties set to "CD" and "Tape", respectively. Set the **Selected** property of one of them to true.
19. Drag a **Button** from the toolbox to below the table. Set its **Text** property to "See Listing".
20. Press Ctrl-F5 to run the application and see how it looks without any RadControls. Note that the titles you added to the table cells (<TD> elements) appear as tool tips when you hover the mouse over the cells and that the tool tip for the button is the same as its Text property.



Apply Skins to the UI controls

1. Drag a **RadFormDecorator** control from the Tool Box onto your Web page.
2. Using the Smart Tag, set its **Skin** property to "Hay".
3. Press Ctrl-F5 to run the application again. The controls have all changed to use the skin you specified!



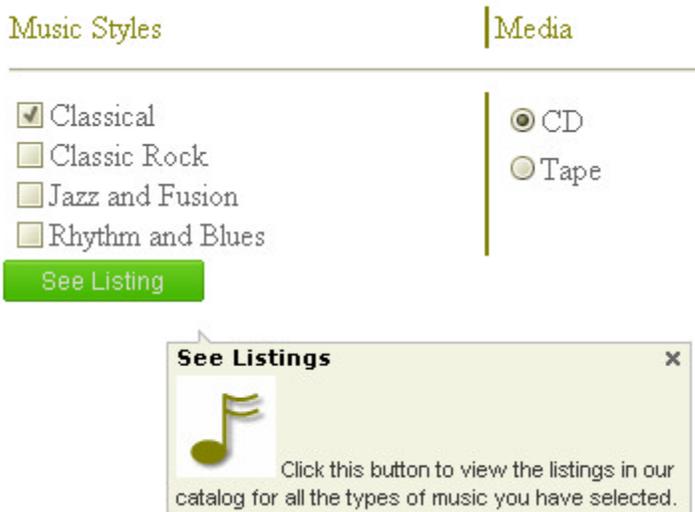
Modify the tool tips using RadToolTipManager

1. Drag a **RadToolTipManager** control from the Tool Box onto your Web page. Using the Smart Tag, set its **Skin** property to "Hay".
2. Using the Properties Window, set the **Animation** property to "FlyIn".
3. Press Ctrl-F5 to see the result of these changes. The tool tips have changed appearance, and appear by "flying in" from the lower edge of the Web page.



Add a custom Tool Tip using RadToolTip

1. Drag a **RadToolTip** control from the Tool Box onto your Web page.
2. Drag an **Image** control from the Tool Box onto the surface of the RadToolTip control.
3. In the Properties Window for the Image control, click the ellipsis button next to the **ImageUrl** property and navigate to the "music.png" file that you added to the Images directory. Then click OK to assign the URL.
4. On the design surface of the RadToolTip control, next to the image, type the text "Click this button to view the listings in our catalog for all the types of music you have selected."
5. In the Properties Window for the RadToolTip control, set the **Skin** property to "Hay", set the **ManualClose** property to "True", set the **TargetControlID** property to "Button1", set the **Title** property to "See Listings", set the **Position** property to "BottomRight", and set the **RelativeTo** property to "Element".
6. When you set the **TargetControlID** property of the RadToolTip control to "Button1", you linked its tool tip to the button. However, the RadToolTipManager is also linking a tool tip with the button. To turn off the RadToolTipManager so that your RadToolTip control's tool tip appears, set the **ToolTipZoneID** property of the **RadToolTipManager** to "tblOptions". This limits the scope of the RadToolTipManager so that it only affects elements inside the table.
7. Press Ctrl-F5 to run the application. Now the tool tips in the table still "fly in", but the button now shows a custom tool tip. The **Position** and **RelativeTo** properties determine the position of that tool tip. Because you set the **ManualClose** property on the RadToolTip control, the button's tool tip does not go away until you click on the close button or display another tool tip.

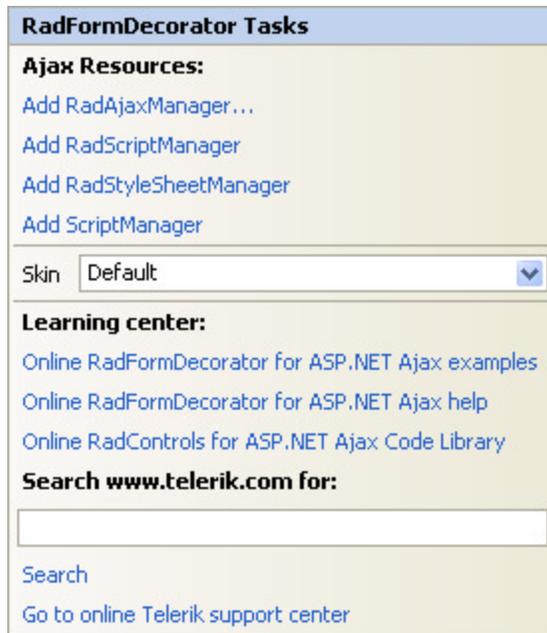


5.4 Designer Interface

In the Visual Studio designer, you can configure RadFormDecorator, RadToolTipManager, and RadToolTip using the Smart Tag and the Properties Window. On RadToolTip, you can use the design window to create custom content for a tool tip.

Smart Tag

The Smart Tag for each of the user interface and information controls is identical (except for the title). It contains only the common elements of RadControls Smart Tags: the Ajax Resources, Skin selection, and Learning center:



Properties Window

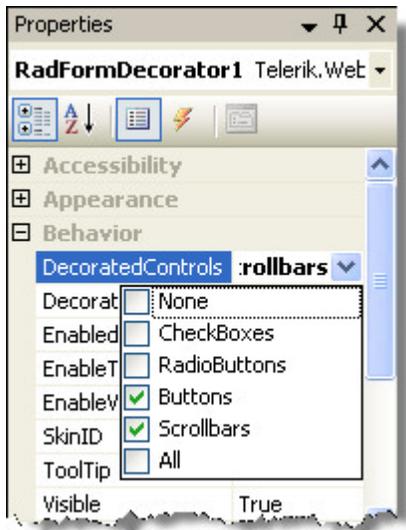
At design time, most of the work you do to configure these controls can be done using the Properties Window. As before, let us look at the most important properties of the controls.

RadFormDecorator

The most important property of RadFormDecorator is the **Skin** property. This property is the reason to use RadFormDecorator, as its entire function is to apply a skin to other ASP.NET controls on the page. You can set the Skin property using either the Smart Tag or the Properties Window.

Two other properties let you specify which controls on the page are assigned the skin you select:

- The **DecoratedControls** property lets you specify the types of controls that RadFormDecorator applies its skin to. By clicking the drop-down arrow in the Properties Window, you can get a list of control types and select the types you want:



Only controls of the selected types have the skin applied. The selections shown above result in the following markup:

[ASP.NET] DecoratedControls

```
<telerik:RadFormDecorator
ID="RadFormDecorator1" Runat="server"
DecoratedControls="Buttons, Scrollbars" />
```

- The **DecorationZoneID** property lets you limit the scope of the RadFormDecorator to apply only to the children of a single element on the page. This is the client-side ID of an HTML element on the page. Only the children of that element are affected by the RadFormDecorator.



Gotcha! Do not set **DecorationZoneID** to the ID of an element whose appearance you want to change. It must be set to the ID of a *parent* element.

RadToolTipManager and RadToolTip

The two tool tip controls share most of the same properties. When assigned to RadToolTipManager, a property affects all of the tool tips it generates, while when assigned to RadToolTip it affects the single generated tool tip.

Specifying the content of the tool tip

The content of a tool tip can come from a variety of sources.

- If you do not specify the content using the properties of RadToolTip or RadToolTipManager, the tool tip displays the text it derives from the HTML element to which it is attached (called the "target" element). If the target element has a ToolTip attribute, that is used. If there is no ToolTip attribute, the Title attribute is used.
- You can override the text derived from the HTML element by assigning a value to the **Text** property of RadToolTip or RadToolTipManager.
- You can override the Text property by supplying custom content for the tool tip. On **RadToolTip**, this can be done using the design surface (described below), while on **RadToolTipManager**, you must use the server-side **AjaxUpdate** event (described in section on server-side programming).

You can add a title area to the tool tip by setting the **Title** property. You can specify whether the content area includes scroll bars by setting the **ContentScrolling** property.

Specifying the position of the tool tip

To specify where a tool tip appears, set the **Position** and **RelativeTo** properties. **RelativeTo** specifies the starting point to use when positioning the tool tip. This can be "Element" (the target element), "Mouse", or "BrowserWindow". **Position** specifies where the tool tip appears relative to that starting point. It can be "TopLeft", "TopCenter", "TopRight", "MiddleLeft", "Center", "MiddleRight", "BottomLeft", "BottomCenter", or "BottomRight". You can further adjust the position by adding an offset using the **OffsetX** and **OffsetY** properties. The **MouseTrailing** property causes the tool tip to follow the mouse when the **RelativeTo** property is set to "Mouse".

Specifying when the tool tip appears

The **ShowEvent** property determines what causes the tool tip to appear. By default, this has the value "OnMouseOver", which causes the tool tip to appear when the mouse hovers over the target element. Other possible values are "OnClick" (when the user left clicks the target element), "OnRightClick" (when the user right clicks the target element), "OnFocus" (when the target element gets focus), and "FromCode" (the tool tip does not appear automatically but must be displayed using client-side or server-side code). The **ShowDelay** property specifies how long (in milliseconds) after the show event occurs that the tool tip appears.

In addition to **ShowEvent**, the **VisibleOnPageLoad** property specifies whether the tool tip is visible when the page is first loaded in the browser.

Specifying when the tool tip disappears

By default, the tool tip disappears after a fixed delay or when the mouse moves off the target element. The **AutoCloseDelay** property specifies how long the tool tip is visible before it disappears when the mouse does not move off the target element. The **HideDelay** property specifies how long the tool tip remains after the mouse moves off the target element.

The **Sticky** property allows the tool tip to remain after the mouse moves off the target element, as long as the mouse moves on to the surface of the tool tip. This is useful for tool tips with custom content such as buttons and input controls. The **ManualClose** property adds a close button to the tool tip, and causes it to remain until the user clicks the close button.

Look-and-feel

Like most RadControls, you can use the **Skin** property to set the general appearance of tool tips. In addition, the **ShowCallout** property lets you specify whether the callout appears. The callout is the triangular notch in the edge of the tool tip that gives it the appearance of a speech bubble. The **Animation** property lets you add animated effects to the way the tool tip appears. The **Modal** property lets you make the tool tip disable the web page while it is showing.

Attaching the tool tip to a target element

So far, we have looked at properties that are present on both **RadToolTip** and **RadToolTipManager**. There are a few properties that are unique to one or the other control which specify how it is attached to a target element.

On **RadToolTip**, the **TargetControlID** property identifies the target element. By default, this is the server-side ID property of the target element. If the target element does not have a server-side ID (for example, if it is not an ASP.NET control or does not have `runat="server"`), you can set **TargetControlID** to the client-side ID of the target element. In that case, you should also set the **IsClientID** property to true.

On **RadToolTipManager**, there are two ways to specify the controls for which it generates tool tips.

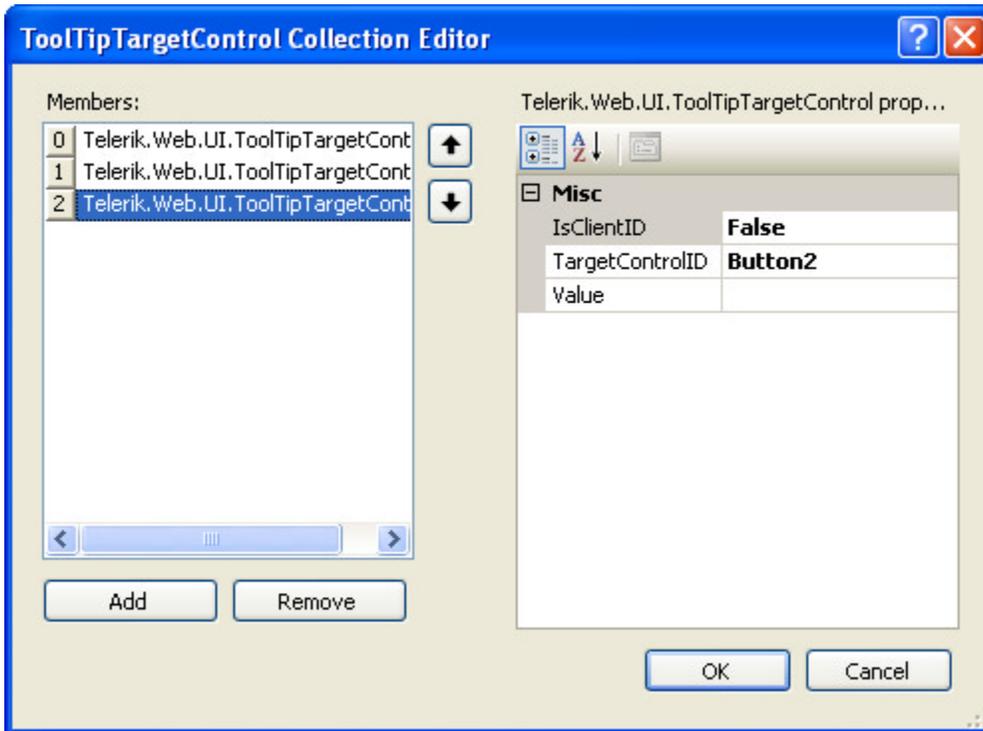
- If the **AutoTooltipify** property is true (the default), the tool tip manager automatically generates tool tips for any control that has a **ToolTip** or **Title** attribute. When attaching tool tips in this manner, you can limit the scope of the tool tip manager by setting the **ToolTipZoneID** property. **ToolTipZoneID** works the same

way as the `DecorationZoneID` property that we saw on `RadFormDecorator`.

- You can specify exactly which controls get tool tips by explicitly adding them to the `TargetControls` property collection.

ToolTipTargetControl Collection Editor

To add HTML elements to the `TargetControls` property collection of `RadToolTipManager`, use the ToolTipTargetControl Collection Editor. You can display this editor by clicking the ellipsis button next to the `TargetControls` property in the Properties Window.



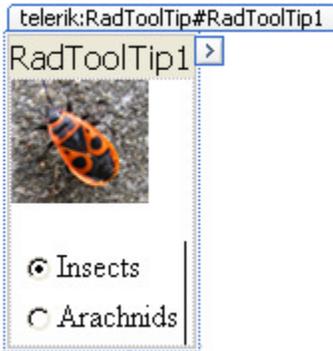
This editor works like most collection editors, with `Add` and `Remove` buttons to add or remove items from the collection and a properties grid on the right to set the properties of the currently selected item. Each item in the collection has a `TargetControlID` property and an `IsClientID` property to let you specify the HTML element for which the tool tip manager should generate a tool tip. These properties work just like the properties on `RadToolTip` that have the same names.

RadToolTip design surface

When you add a `RadToolTip` control to your Web page using the Visual Studio designer, it looks like a little window with the name of the control in the title bar:



The content area of that window is a design surface for providing custom content for the tool tip. You can add any HTML elements to this design surface, including ASP.NET controls. For example, the tool tip shown below has an `IMAGE` element and a `RadioButtonList` control:



When you add content to the design surface, it is automatically added to the RadToolTip control:

[ASP.NET] Custom content on RadToolBar

```
<telerik:RadToolTip ID="RadToolTip1" runat="server" >
  <asp:Image ID="Image1" runat="server"
    ImageUrl="~/Images/redbug.png" />
  <asp:RadioButtonList ID="RadioButtonList1" runat="server">
    <asp:ListItem Selected="True">Insects</asp:ListItem>
    <asp:ListItem>Arachnids</asp:ListItem>
  </asp:RadioButtonList>
</telerik:RadToolTip>
```

When you add custom content to a RadToolTip control, the content is displayed in the tool tip. In this case, the tool tip's Text property is ignored.

5.5 Server-Side Programming

Adding custom content when using RadToolTipManager

We have already seen how to add custom content to RadToolTip using the Visual Studio designer. When using RadToolTipManager, however, there is no design surface to let you visually design the content of the generated tool tips. If you want to provide custom content, you must use the server-side AjaxUpdate event.

 By using RadToolTipManager with an AjaxUpdate event handler, you keep the size of your Web page down because the content of tool tips does not have to be loaded until it is used.

When you provide a handler for the AjaxUpdate event, the tool tip manager automatically generates an asynchronous AJAX callback when it needs to generate a tool tip. The callback is asynchronous so that your Web page does not have to reload every time you bring up a tool tip. If you want to learn more about asynchronous AJAX callbacks, look ahead to the chapter on AjaxPanel, AjaxManager, and AjaxManagerProxy.



Gotcha! The AjaxUpdate event uses an MS AJAX UpdatePanel to handle the asynchronous update. When an UpdatePanel triggers an AJAX update, it causes all UpdatePanels to have their content updated. As a result, if the RadToolTipManager is included in another UpdatePanel, the showing of a tool tip triggers an update of the panel that contains the tool tip manager. This means it is possible that showing a tool tip leads to the situation where the system is deleting the tool tip manager while it is trying to show a tool tip. To prevent this, always set the UpdateMode of any UpdatePanel that contains a RadToolTipManager to **Conditional**.

The following example uses the AjaxUpdate event to supply the content of tool tips generated by RadToolTipManager. The Web page includes five buttons, which are all included in the TargetControls property collection of the RadToolTipManager control. In the code-behind, the AjaxUpdate event handler generates the content of tool tips. The event handler uses the TargetControlID supplied by the event arguments to identify the control whose tool tip needs to be generated. It then generates the controls that make up the content of the tool tip and adds them to the supplied UpdatePanel.



The complete source for this project is in \VS Projects\UI\ServerAjaxUpdate.

[VB] Adding tool tip content in AjaxUpdate

```
Protected Sub RadToolTipManager1_AjaxUpdate(ByVal sender As Object, ByVal e As
Telerik.Web.UI.ToolTipUpdateEventArgs) Handles RadToolTipManager1.AjaxUpdate
    Dim text As String = "Click here to learn more."
    Dim graphic As New Image()
    graphic.ID = "imgExample"
    Select Case e.TargetControlID
        Case "btnInsects"
            graphic.ImageUrl = "~/Images/redbug.png"
            text = btnInsects.ToolTip
            Exit Select
        Case "btnBirds"
            graphic.ImageUrl = "~/Images/blackbird.png"
            text = btnBirds.ToolTip
            Exit Select
        Case "btnMammals"
            graphic.ImageUrl = "~/Images/hedgehog.png"
            text = btnMammals.ToolTip
            Exit Select
        Case "btnReptiles"
            graphic.ImageUrl = "~/Images/lizard.png"
            text = btnReptiles.ToolTip
            Exit Select
        Case "btnAmphibians"
            graphic.ImageUrl = "~/Images/frog.png"
            text = btnAmphibians.ToolTip
            Exit Select
    End Select
    e.UpdatePanel.ContentTemplateContainer.Controls.Add(graphic)
    e.UpdatePanel.ContentTemplateContainer.Controls.Add(New LiteralControl(text))
End Sub
```

[C#] Adding tool tip content in AjaxUpdate

```
protected void RadToolTipManager1_AjaxUpdate(object sender, ToolTipUpdateEventArgs e)
{
    string text = "Click here to learn more.";
    Image graphic = new Image();
    graphic.ID = "imgExample";
    switch (e.TargetControlID)
    {
        case "btnInsects":
            graphic.ImageUrl = "~/Images/redbug.png";
            text = btnInsects.ToolTip;
    }
}
```

```

        break;
    case "btnBirds":
        graphic.ImageUrl = "~/Images/blackbird.png";
        text = btnBirds.ToolTip;
        break;
    case "btnMammals":
        graphic.ImageUrl = "~/Images/hedgehog.png";
        text = btnMammals.ToolTip;
        break;
    case "btnReptiles":
        graphic.ImageUrl = "~/Images/lizard.png";
        text = btnReptiles.ToolTip;
        break;
    case "btnAmphibians":
        graphic.ImageUrl = "~/Images/frog.png";
        text = btnAmphibians.ToolTip;
        break;
    }
    e.UpdatePanel.ContentTemplateContainer.Controls.Add(graphic);
    e.UpdatePanel.ContentTemplateContainer.Controls.Add(new LiteralControl(text));
}

```

Adding dynamic content to RadToolTip

In the previous example, the content of tool tips was added to an UpdatePanel. RadToolTipManager uses an update panel to hold tool tip content because that provides a limited area to update when the AJAX callback occurs. When adding content to **RadToolTip** in the code-behind, there is no need for an UpdatePanel, because there is no asynchronous callback. Instead, you simply add content to the **Controls** collection of the RadToolTip object.

The following example illustrates how to add custom content to RadToolTip in the code-behind. While the properties of the controls it adds are hard-coded in this example, in a real application a similar technique can be used to load content from another source such as a database.

[VB] Adding content in the code-behind

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    If Not IsPostBack Then
        Dim hl As New HyperLink()
        hl.Text = "Learn more about frogs."
        hl.NavigateUrl = "frogs.aspx"
        RadToolTip1.Controls.Add(hl)
        RadToolTip1.Sticky = True
    End If
End Sub

```

[CS] Adding content in the code-behind

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        HyperLink hl = new HyperLink();
        hl.Text = "Learn more about frogs.";
        hl.NavigateUrl = "frogs.aspx";
        RadToolTip1.Controls.Add(hl);
        RadToolTip1.Sticky = true;
    }
}

```

```
}
```

When the application is run, a hyperlink is added to the tool tip that navigates to another page in the project. The Sticky property is also set so that the tool tip does not disappear when the user tries to click on the hyperlink.



The source for this example can be found in \VS Projects\UI\ServerSide.

5.6 Client Side Programming

The last server-side example showed that when using RadToolTip (or RadToolTipManager), you can include controls inside the tool tip that do more than display information. Combining this ability to add controls to the tool tip with a powerful client-side API lets you make use of RadToolTip (or RadToolTipManager) to perform important tasks on your Web page. The following examples illustrate some of the possibilities.

Using a tool tip to assist data entry

One simple task you can perform with RadToolTip is to assist in data entry. The following example uses the tool tip properties to assign the current time to a RadDateTimeInput control.



In this example, the tool tip does not appear automatically. Instead, it appears when the user clicks the assist button on the date input control. When that happens, the current time is assigned to the tool tip's Title property using the `set_title()` method, and the tool tip is displayed by calling its `show()` method.

If the user closes the tool tip using the button it contains, the button calls the tool tip's `get_title()` method to read the current time, and assigns that value to the date input control. The button then calls the tool tip's `hide()` method to close the tool tip.

[ASP.NET] Assisting data entry

```
<script type="text/javascript">
function ShowToolTip(sender, args) {
    var tooltip = $find("<%= RadToolTip1.ClientID %>");
    if (tooltip) {
        var now = new Date();
        var time = now.getHours().toString() + ":";
        var minutes = now.getMinutes();
        if (minutes < 10)
            time = time + "0" + minutes;
        else
            time = time + minutes;
    }
}
```

```

        tooltip.set_title(time);
        tooltip.show();
    }
}
function setTime() {
    var dateInput = $find("<%= RadDateInput1.ClientID %>");
    var tooltip = $find("<%= RadToolTip1.ClientID %>");
    dateInput.set_value(tooltip.get_title());
    tooltip.hide();
}
</script>
<telerik:RadDateInput ID="RadDateInput1" Runat="server"
    DateFormat="h:mm tt" Skin="Sunset"
    Label="Enter Time: " ShowButton="True" >
    <ClientEvents OnButtonClick="ShowToolTip" />
</telerik:RadDateInput>
<telerik:RadToolTip ID="RadToolTip1" runat="server"
    Skin="Sunset" Sticky="True" ShowEvent="FromCode"
    TargetControlID="RadDateInput1">
    <input id="Button1" type="button" value="Set Time" onclick="setTime();" />
</telerik:RadToolTip>
<telerik:RadFormDecorator ID="RadFormDecorator1" Runat="server" Skin="Sunset" />

```

The complete source for this example can be found in `\VS Projects\UI\ClientSide`.

Using a tool tip as a log-in dialog

The last example showed how to use the `show()` and `hide()` methods to display and hide a tool tip. The tool tip was sticky, so it remained visible until the user closed it using the button or moved the mouse off the tool tip. The following example illustrates another approach to controlling when a tool tip appears and disappears. It uses the `OnClientBeforeHide` and `OnClientBeforeShow` client-side events.

This example sets the `Modal` property of `RadToolTip` to true to use it as a modal log-in dialog. The `VisibleOnPageLoad` property is set to true so that the page starts out disabled until the user has logged in using the tool tip. Unlike the previous example, where the tool tip can disappear when the user moves off of it, in this case, you can't let the tool tip go away until the user has logged in. To accomplish this, we use the `OnClientBeforeHide` client side event to cancel the closing of the tool tip.

Because the tool tip is being used as a log-in dialog, the button that closes the tool tip causes a postback: you would not want to execute code that verifies a login on the client, as that would present a huge security problem! In the code-behind on the server, the button's Click handler injects some client-side script onto the page to set a client-side `loggedIn` variable and, in the case of a successful login, close the tool tip.

To further complicate matters, the Web page protected by the tool tip contains a button that causes a postback. After a postback, the `VisibleOnPageLoad` property will cause the tool tip to reappear unless it is suppressed. This is handled by responding to the `OnClientBeforeShow` client-side event. The event handler checks the client-side `loggedIn` flag, and if the user has already logged in, cancels the showing of the tool tip. In order that the `OnClientBeforeShow` event handler can tell whether the user has logged in, the server-side `Page_Load` event handler checks whether the user has logged in (using a Session variable), and injects some client-side script onto the page to set the client-side `loggedIn` flag.



In the ASPX file, a `<script>` block declares the `loggedIn` flag, the `OnClientBeforeShow` and `OnClientBeforeHide` event handlers, and a `HideLoginToolTip` function that hides the login tool tip. This last function will be called from the script that the server-side button handler injects onto the page. Note that the content of the tool tip is contained in an `UpdatePanel`. This causes the "Log In" button to execute inside an asynchronous AJAX callback, so that the page does not need to reload when the user tries to log in.

[ASP.NET] Client-side code and login tool tip declaration

```
<script type="text/javascript">
var loggedIn = false;
// OnClientBeforeHide keeps the login tool tip from closing
// until the user is logged in
function OnClientBeforeHide(sender, args) {
    if (!loggedIn)
        args.set_cancel(true);
}
// OnClientBeforeShow prevents the login tool tip from appearing
// after a postback once the user has logged in
function OnClientBeforeShow(sender, args) {
    if (loggedIn)
        args.set_cancel(true);
}
// HideLoginToolTip closes the login tool tip if it is showing
function HideLoginToolTip() {
    var tooltip = $find("<%=RadToolTip1.ClientID%>");
    if (tooltip)
        tooltip.hide();
}
</script>
<br />
This page cannot be used until you log in.<br />
<asp:Button ID="Button2" runat="server" Text="Post Back" />

<telerik:RadToolTip ID="RadToolTip1" runat="server"
    Skin="Office2007" TargetControlID="form1"
    Position="Center" RelativeTo="BrowserWindow"
    Title="Enter Name and Password"
```

```

Modal="True" ShowDelay="0" VisibleOnPageLoad="True"
ShowCallout="False" ShowEvent="FromCode" Width="250px"
onclientbeforehide="OnClientBeforeHide"
onclientbeforeshow="OnClientBeforeShow">
<asp:UpdatePanel runat="server" ID="UpdatePanel2">
  <ContentTemplate>
    <br />
    <telerik:RadTextBox ID="RadTextBox1" Runat="server"
      Label="Name: " Skin="WebBlue" Width="200px">
    </telerik:RadTextBox>
    <br />
    <telerik:RadTextBox ID="RadTextBox2" Runat="server"
      Skin="WebBlue" Label="Password: " Width="200px" TextMode="Password">
    </telerik:RadTextBox><br />
    <asp:Label ID="lblError" runat="server" ForeColor="Red"
      Text="Invalid Login!" Visible="False">
    </asp:Label>
    <br />
    <br />
    <asp:Button ID="Button1" runat="server" Text="Log In" onclick="Button1_Click" />
  </ContentTemplate>
  <Triggers>
    <asp:AsyncPostBackTrigger ControlID="Button1" EventName="Click" />
  </Triggers>
</asp:UpdatePanel>
</telerik:RadToolTip>
<telerik:RadFormDecorator ID="RadFormDecorator1" Runat="server"
  DecorationZoneID="RadToolTip1" Skin="WebBlue" />

```

On the server-side, the Page_Load event handler injects a client-side script to set the LoggedIn flag.

[VB] Injecting client-side script to set LoggedIn flag

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
  If IsPostBack Then
    'Check the session to see if user has logged in
    Dim LoggedIn As Object = Session("loginFlag")
    If LoggedIn <> Nothing AndAlso DirectCast(LoggedIn, Boolean) Then
      'Set the client side logged in flag
      Dim script As String = "loggedIn = true;"
      ScriptManager.RegisterStartupScript(Me, [GetType](), "script", script, True)
    End If
  End If
End Sub

```

[C#] Injecting client-side script to set LoggedIn flag

```

protected void Page_Load(object sender, EventArgs e)
{
  if (IsPostBack)
  {
    // Check the session to see if the user has logged in
    object LoggedIn = Session["loginFlag"];
    if (LoggedIn != null && (bool)LoggedIn)
    {
      // set the client-side logged in flag
      string script = "loggedIn = true;";
      ScriptManager.RegisterStartupScript(this, GetType(), "script", script, true);
    }
  }
}

```

```
}  
}  
}
```

The button's event handler verifies the user name and password, and if the login is successful, injects a client-side script to close the login tool tip and set the LoggedIn flag. In this example, the event handler only checks that the user name and password are not blank. In a real application, their values would be checked.

[VB] Verifying login details and closing the tool tip

```
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles  
Button1.Click  
    Dim userName As String = RadTextBox1.Text  
    Dim password As String = RadTextBox2.Text  
    ' verify userName and password.  
    If Not userName.Equals(String.Empty) AndAlso Not password.Equals(String.Empty) Then  
        'Use Session to remember that user is logged in  
        Session("loginFlag") = True  
        'Also set client-side logged in flag and hide the tool tip  
        Dim script As String = "alert(""Welcome " + userName + "! You are now logged in.  
"");loggedIn = true;HideLoginToolTip();"   
        ScriptManager.RegisterStartupScript(Me, [GetType](), "script", script, True)  
    Else  
        lblError.Visible = True  
    End If  
End Sub
```

[C#] Verifying login details and closing the tool tip

```
protected void Button1_Click(object sender, EventArgs e)  
{  
    string userName = RadTextBox1.Text;  
    string password = RadTextBox2.Text;  
    // verify userName and password.  
    if (!userName.Equals(string.Empty) && !password.Equals(string.Empty))  
    {  
        // Use session to remember that user is logged in  
        Session["loginFlag"] = true;  
        // Also set client-side logged in flag and hide the tool tip  
        string script = "alert(\"Welcome \" + userName + "! You are now logged in. \");loggedIn =  
true;HideLoginToolTip()";  
        ScriptManager.RegisterStartupScript(this, GetType(), "script", script, true);  
    }  
    else  
    {  
        lblError.Visible = true;  
    }  
}
```

The complete source for this project can be found in \VS Projects\UI\ClientLogin.

5.7 How To

Using RadToolTip with an ImageMap

When using `RadToolTip` (or `RadToolTipManager`) with an `ImageMap`, some extra work is required to attach tool tips to the regions of the image map. This is because `RadToolTip` and `RadToolTipManager` use a control ID to attach to an element, and `ImageMap` does not provide IDs for its hot spots. The easiest way to handle this is to

execute some javascript when the page loads that assigns the id attribute on the hot spots.

The following example illustrates how this is done. A javascript function assigns the id attribute of the hot spots of the ImageMap:

[ASP.NET] JavaScript to assign IDs to hotspots

```
<script type="text/javascript">
  var map = $get("ImageMapImageMap1")[0];
  var areas = map.getElementsByTagName("AREA");

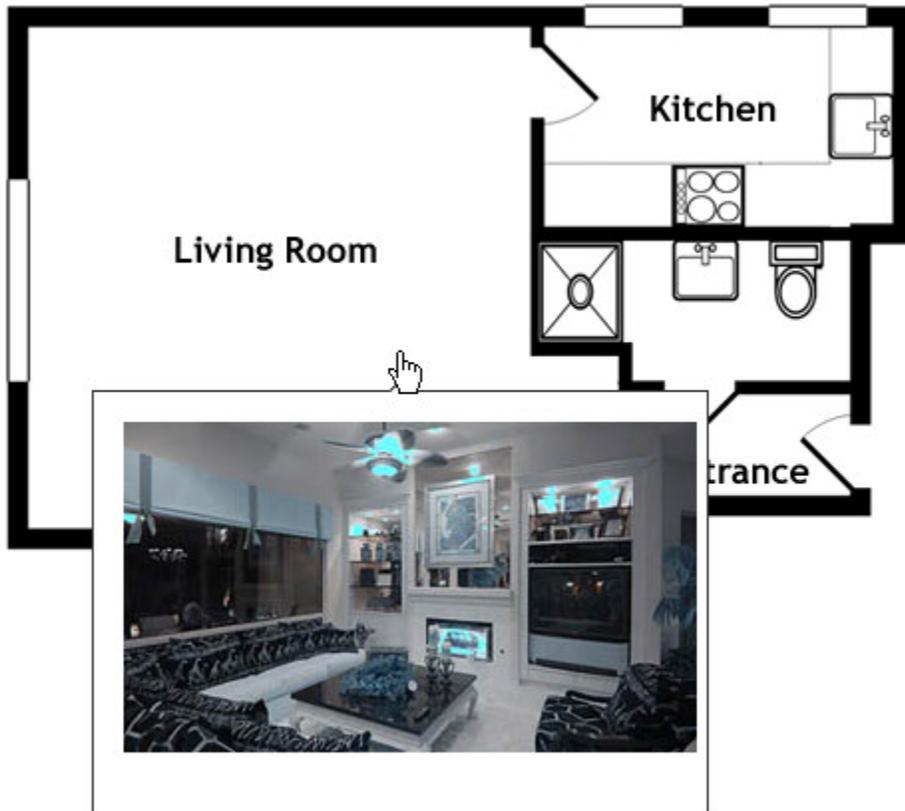
  for (var i = 0; i < areas.length; i++)
  {
    var area = areas[i];
    area.setAttribute("id", "area" + i);
  }
</script>
```

The tool tips can then be attached to the hot spots using the `TargetControlID` property with `IsClient` set to true:

[ASP.NET] Attaching tool tips to hot spots

```
<telerik:RadToolTip Skin="Default" ID="ttLivingRoom" runat="server"
  TargetControlID="area0" IsClientID="true"
  Animation="FlyIn" Position="BottomCenter">
  
</telerik:RadToolTip>
<telerik:RadToolTip ID="ttKitchen" Skin="Default" runat="server"
  TargetControlID="area1" IsClientID="true"
  Animation="FlyIn" Position="TopRight" >
  
</telerik:RadToolTip>
<telerik:RadToolTip ID="ttBathroom" Skin="Default" runat="server"
  TargetControlID="area2" IsClientID="true"
  Animation="FlyIn" Position="MiddleRight">
  
</telerik:RadToolTip>
<telerik:RadToolTip ID="ttEntry" Skin="Default" runat="server"
  TargetControlID="area3" IsClientID="true"
  Animation="FlyIn" Position="BottomRight" >
  
</telerik:RadToolTip>
```

When the mouse moves over a region of the image map, the tool tip flies in:



The source for this project can be found in `\VS Projects\UI\HowToImageMap`.

5.8 Summary

In this chapter you looked at the user interface and information controls **RadFormDecorator**, **RadToolTipManager**, and **RadToolTip**. You created a simple application and saw how these controls can change the look-and-feel of standard ASP.NET controls and tool tips. You became familiar with the design-time support for using these controls and looked at the most important properties. You learned to use the server-side API to supply the content of customized tool tips. You learned to use the client-side API to hide and show tool tips and work with their properties so that they can perform functions on your Web pages. You also learned to add client-side ids to an image map so that it can be used with **RadToolTip**.

6 RadRotator

6.1 Objectives

- Examine how RadRotator can display changing content on your Web page.
- Create a simple application to build confidence in using RadRotator.
- Become familiar with the design time support for working with the rotator. This support includes Smart Tag, Properties Window, and the template design surface.
- Explore principal properties and groups of properties where 80% of the functionality is found.
- Learn to start and stop the rotator using the client-side api.
- Learn to use RadRotator when it is not bound to a data source.
- Coverflow mode
- Carousel mode

6.2 Introduction

RadRotator lets you display data from multiple records using a template. It can scroll through the records either vertically or horizontally, either as a continuous stream or as a slide show.

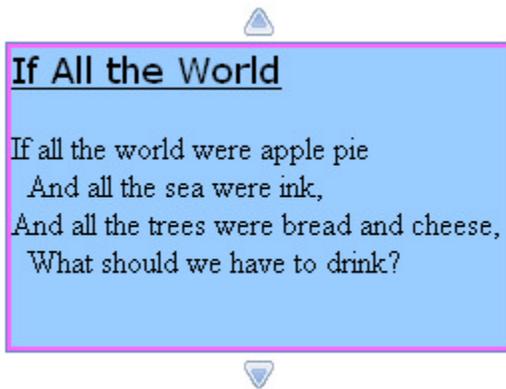


The rotator is highly configurable. The display of each frame is based on a template, so you can include any controls or HTML elements to make up the display. You can also configure the way the rotator cycles through its frames, and what actions cause the frames to cycle.

Typically, the rotator is data bound to fetch records from a data source, although that is not strictly necessary. The items in the template are bound to fields from the data source.

6.3 Getting Started

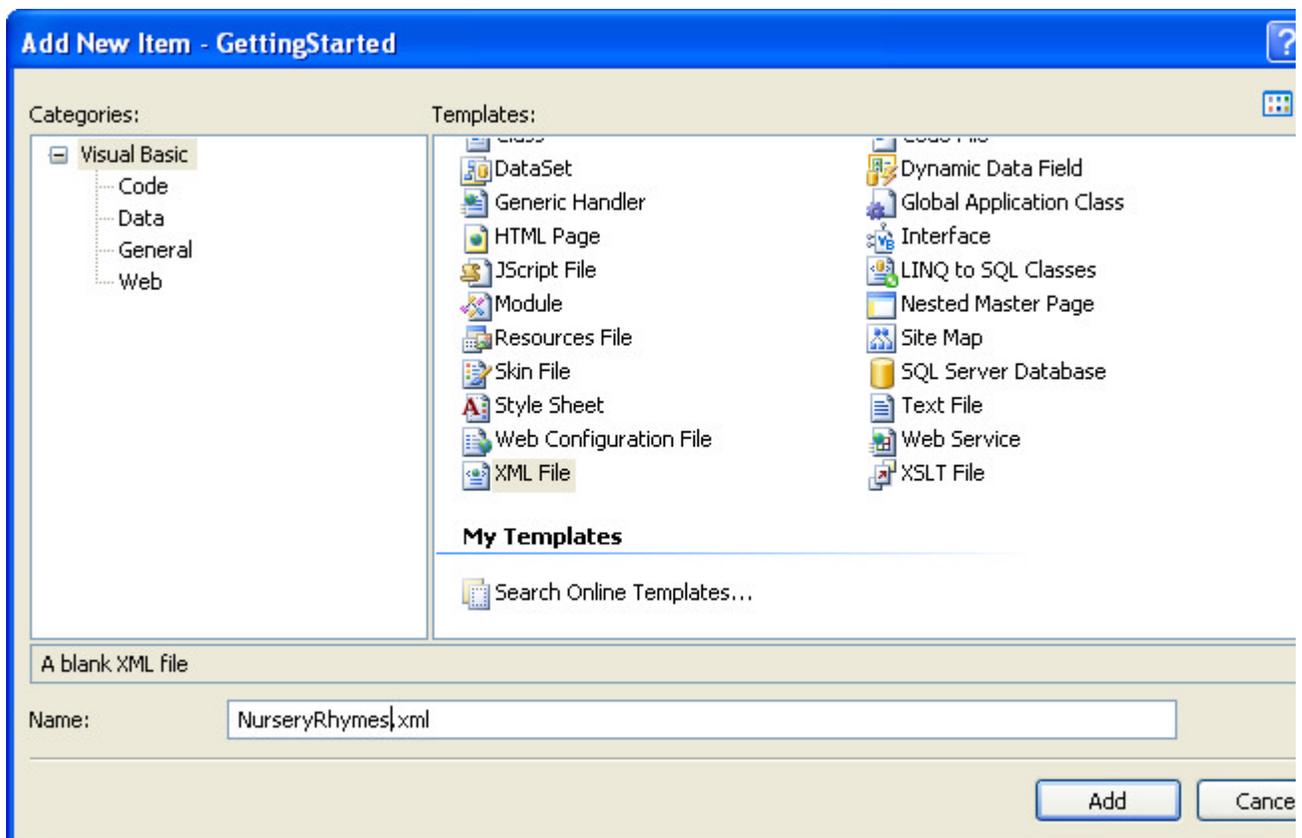
In this walk-through you will become familiar with the **RadRotator** control. You will create a template for rotator frames and bind the items in that template to records from a data source. The resulting Rotator will display a collection of nursery rhymes as a slide show:



When you are finished, your project should match the one supplied in \VS Projects\Rotator\GettingStarted.

Set up the project structure

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. In the solution explorer, right click the project and choose Add|New Item... In the New Items dialog, select XML file, and name the file "NurseryRhymes.xml" and then click OK.



3. Copy the following XML into the new file to create your XML data set and save the file:
[XML] NurseryRhymes.XML

```
<?xml version="1.0" encoding="utf-8" ?>  
<Rhymes>
```

```

<Rhyme>
  <Name>Cock-a-doodle-doo</Name>
  <Line1>Cock-a-doodle-doo</Line1>
  <Line2>My dame has lost her shoe;</Line2>
  <Line3>My master's lost his fiddlestick,</Line3>
  <Line4>And knows not what to do.</Line4>
</Rhyme>
<Rhyme>
  <Name>Little Jack Horner</Name>
  <Line1>Little Jack Horner sat in a corner</Line1>
  <Line2>Eating his Christmas pie.</Line2>
  <Line3>He stuck in his thumb, and pulled out a plumb,</Line3>
  <Line4>And said What a good boy am I!.</Line4>
</Rhyme>
<Rhyme>
  <Name>Peter, Peter, Pumpkin Eater</Name>
  <Line1>Peter, Peter, pumpkin eater,</Line1>
  <Line2>Had a wife and couldn't keep her;</Line2>
  <Line3>He put her in a pumpkin shell,</Line3>
  <Line4>And there he kept her very well.</Line4>
</Rhyme>
<Rhyme>
  <Name>If All the World</Name>
  <Line1>If all the world were apple pie</Line1>
  <Line2>And all the sea were ink,</Line2>
  <Line3>And all the trees were bread and cheese,</Line3>
  <Line4>What should we have to drink?</Line4>
</Rhyme>
<Rhyme>
  <Name>Little Bo-peep</Name>
  <Line1>Little Bo-peep has lost her sheep</Line1>
  <Line2>And can't tell where to find them;</Line2>
  <Line3>Leave them alone and they'll come home</Line3>
  <Line4>Bringing their tails behind them.</Line4>
</Rhyme>
<Rhyme>
  <Name>Little Tommy Tittlemouse</Name>
  <Line1>Little Tommy Tittlemouse</Line1>
  <Line2>Lived in a little house;</Line2>
  <Line3>He caught fishes</Line3>
  <Line4>In other men's ditches.</Line4>
</Rhyme>
<Rhyme>
  <Name>Hickety pickety</Name>
  <Line1>Hickety, pickety, my black hen,</Line1>
  <Line2>She lays eggs for gentlemen.</Line2>
  <Line3>Gentlemen come every day</Line3>
  <Line4>To see what my black hen doth lay.</Line4>
</Rhyme>
</Rhymes>

```

Add the XML data source

1. Drag an `XMLDataSource` component from the Tool Box onto your Web page.
2. In the `XMLDataSource` Smart Tag, click the link labelled "Configure Data Source...".

3. Click the Browse button next to the Data File field and select "NurseryRhymes.xml".
4. Click OK to exit the Configure Data Source dialog without setting a transform file or XPath expression.

Add the RadRotator control

1. Drag a **RadRotator** control from the Tool Box onto your Web page.
2. Using the Smart Tag, set the rotator's **Skin** property to "Web20".
3. Using the Properties Window, set the **DataSourceID** property of the rotator to "XmlDataSource1".
4. Set the **RotatorType** to "SlideShowButtons" and the **ScrollDirection** to "Up, Down".
5. Set the **SlideShowAnimation-Type** to "Fade" and the **SlideShowAnimation-Duration** to "3000".
6. Set the **Height** to "194px", **Width** to "254px", **ItemHeight** to "154px" and **ItemWidth** to "254px". The **ItemHeight** is 40 pixels smaller than the **Height** because the up and down buttons take up 40 pixels. This way, one frame fits exactly in the rotator, so that we will view exactly one rhyme at a time.

Create the item template

1. Drag a **Panel** control from the Tool Box onto the surface of the rotator.
2. Using the Properties Window, set the **BackColor** to a light blue ("#99CCFF"), the **BorderColor** to pink ("#FF66FF"), the **BorderStyle** to "Double" and the **BorderWidth** to "2px".
3. Set the **Height** property to "150px" and the **Width** property to "250px". These values are 4 pixels (the size of the borders) smaller than the **ItemHeight** and **ItemWidth** properties of the rotator. This means that a panel exactly fits into one item frame of the rotator.
4. Drag a **Label** from the Tool Box onto the panel.
5. Expand the **Font** property and set **Name** to "Verdana", **Size** to "Large" and **Underline** to true.
6. Move to the Source window, and set the **Text** property to "< %# XPath("Name") %>". This binds the **Text** property to the "Name" field from the data source.



Gotcha! Be sure to set the **Text** property using single quotes. The string you are using has double quotes around the word "Name", and the quotation marks will not nest. If you use double quotes around the entire property value, you will get an error telling you that the tag is not well formed.

7. The label declaration should now look like the following:

[ASP.NET] Label1 declaration

```
<asp:Label ID="Label1" runat="server"
  Font-Names="Verdana" Font-Size="Large" Font-Underline="True"
  Text='< %# XPath("Name") %>'>
</asp:Label>
```

8. Back in the design window, hit the Enter key twice to insert two breaks after the label, and then drag a second Label onto the panel.
9. Hit the Enter key again and drag a third Label below the others. Repeat this process to add two more labels.
10. Back in the Source window, set the **Text** properties of the four labels you just added to "< %# XPath('Line1') %>", "< %# XPath('Line2') %>", "< %# XPath('Line3') %>", and "< %# XPath('Line4') %>".
11. Add two non-breaking spaces (" ") before the labels that are bound to "Line2" and "Line4". The Item Template should now look like the following:

[ASP.NET] Item template

```
<ItemTemplate>
  <asp:Panel ID="Panel1" runat="server"
```

```

BackColor="#99CCFF" BorderColor="#FF66FF" BorderStyle="Double" BorderWidth="2px"
Height="150px" Width="250px" >
<asp:Label ID="Label1" runat="server"
    Font-Names="Verdana" Font-Size="Large" Font-Underline="True"
    Text='<# XPath("Name") %>'>
</asp:Label>
<br /><br />
<asp:Label ID="Label2" runat="server" Text='<# XPath("Line1") %>' />
<br />
&nbsp;&nbsp;&nbsp;<asp:Label ID="Label3" runat="server" Text='<# XPath("Line2") %>' />
<br />
<asp:Label ID="Label4" runat="server" Text='<# XPath("Line3") %>' />
<br />
&nbsp;&nbsp;&nbsp;<asp:Label ID="Label5" runat="server" Text='<# XPath("Line4") %>' />

</asp:Panel>
</ItemTemplate>

```

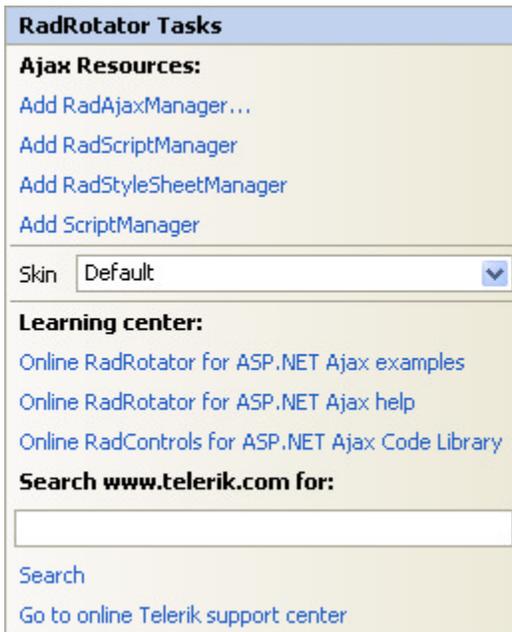
- Press Ctrl-F5 to run the application. The rotator shows a single nursery rhyme. If you click the up and down arrows, you can cycle through the series of rhymes in the XML data set.

6.4 Designer Interface

In the Visual Studio designer, you can configure RadRotator using the Smart Tag and the Properties Window. You can design your item template using the template design surface.

Smart Tag

The RadRotator Smart Tag contains only the common elements of RadControls Smart Tags: the Ajax Resources, Skin selection, and Learning center:



Properties Window

At design time, most of the properties you will want to set to configure the rotator are available in the

Properties Window. Let us look at the most important properties.

Binding the rotator

RadRotator typically gets the information it displays from a data source, where each frame displays data from a single record. You can use the **DataSourceID** property to bind the rotator to a declarative data source, or use the **DataSource** property in the code-behind. RadRotator can be bound to any ASP.NET datasource component, as well as to any object that implements the **IEnumerable** interface (such as **Array** or **ArrayList**). Unlike many other data-bound controls, RadRotator does not have any properties to map specific elements of the rotator to fields from the current record: this is a mapping you must provide in the item template.

Another approach for binding a RadRotator is through a set of images that reside into a single directory. All that needs to be done is setting the **BannersPath** property to the virtual path of the images' directory and the control will be ready for use (of course assuming that the Rotator's and its items' dimensions are set, which is required for all RotatorTypes). The control will handle the creation of the items and the item's html template. This functionality can be used with all RotatorTypes, giving you the ability to choose the right type for your specific scenario.

Specifying how the rotator cycles through its items

The **RotatorType** property specifies how the rotator cycles through its items. You have several options to choose from:

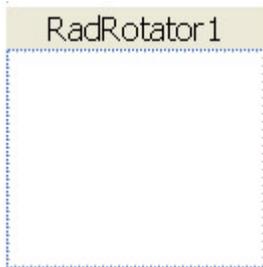
- **AutomaticAdvance** causes the frames to automatically scroll in a single direction. The **ScrollDirection** property specifies the direction ("Up", "Down", "Left", or "Right") that items scroll. The **FrameDuration** property specifies the time, in milliseconds, that the rotator remains still before scrolling to the next set of items. The **ScrollDuration** property specifies how long the rotator spends scrolling before stopping for the next FrameDuration. The **WrapFrames** property specifies whether scrolling starts over at the beginning when it reaches the last frame. The **InitialItemIndex** indicates the item to start on. The default value is 0 (the first item), but you can set it to a higher value to start on a later record. You can also set **InitialItemIndex** to -1 to start just before the first item.
- **Buttons** causes frames to scroll when the user clicks on buttons that appear outside the edges of the rotator. When **RotatorType** is "Buttons", the **ScrollDirection** property indicates both the scroll direction and where the buttons appear. In this case, it can be any or all of the four values, although including both horizontal and vertical scrolling can be a little confusing. To include more than one scroll direction, separate values using commas. (For example: "Up, Down".) As with **AutomaticAdvance**, you can set the **ScrollDuration**, **WrapFrames**, and **InitialItemIndex** properties, but the **FrameDuration** has no effect in this mode.
- **ButtonsOver** behaves just like "Buttons", except that scrolling occurs when the mouse moves over the buttons rather than when they are clicked.
- **SlideShow** is similar to "AutomaticAdvance", except that frames replace each other in a slide show fashion rather than scrolling in a specific direction. When **RotatorType** is "SlideShow", the **FrameDuration** property specifies the time before the current slide changes. Instead of the **ScrollDirection** and **ScrollDuration** properties, you can use the **SlideShowAnimation** property to set the transition effects. The **WrapFrames** and **InitialItemIndex** work the same way with slide shows as they do with scrolling modes.
- **SlideShowButtons** is to "SlideShow" what "Buttons" is to "AutomaticAdvance". That is, the rotator acquires buttons (based on the **ScrollDirection** property) which the user can click to change the current slide.
- **FromCode** leaves the rotator displaying the first frame (specified by **InitialItemIndex**) until you cause a change from a client-side script. We will look at this option more closely in the section on Client-Side Programming.

Managing Layout

In addition to the `ScrollDirection`, four properties control the layout of the rotator and the frames it contains: `Height`, `Width`, `ItemHeight`, and `ItemWidth`. `Height` and `Width` are the dimensions of the rotator, while `ItemHeight` and `ItemWidth` give the dimensions of a single item. If you are trying to fit items neatly into the rotator, it is important to keep in mind that `Height` and `Width` give the dimensions of the entire rotator, including buttons, not just the viewing area.

Template design surface

When you add a `RadRotator` control to your Web page using the Visual Studio designer, it appears with a template design surface showing in the body of the control:



You can add any HTML elements, including ASP.NET controls, to this design surface and they become part of the template that is used to display records from the data source. To display data from the data source, add controls and bind the relevant property to a field from the data source. As with all templates, you can use the ASP.NET expression syntax (`<% %>`) to reference the data item when binding a property of a control in your template. Typically, you use `Container.DataItem` inside the `DataBinder.Eval()` function, although when working with `XmlDataSource`, you can use a simple `XPath()` function call. For examples of using `DataBinder.Eval()` with `Container.DataItem`, revisit the Data Binding chapter (You can also check out the Client-Side Programming example for this chapter). For an example of using `XPath()`, revisit the Getting Started section of this chapter.

6.5 Client-Side Programming

The `RadRotator` client-side api lets you stop and start the cycling through frames. The following example illustrates these capabilities with a rotator that scrolls its items horizontally. To do this, it uses two client-side methods, `startAutoPlay()` and `pause()`.



When the page first loads, the rotator's `RotatorType` property is set to "FromCode", so that it does not move. When the "Play" button executes, it changes the `RotatorType` to "AutomaticAdvance" before calling `startAutoPlay()`. This is so that once the rotator is started, it keeps going. If the `RotatorType` were left as "FromCode", the `startAutoPlay()` method would merely advance the rotator a single frame, and then stop.

[JavaScript] Play button's onclick handler

```
function StartRotator() {
```

```
var rotator = $find("<%= RadRotator1.ClientID %>");
// set rotatorType to automatic advance so that it keeps going
// once started, and then start the rotator.
rotator.set_rotatorType(Telerik.Web.UI.RotatorType.AutomaticAdvance);
rotator.startAutoPlay();
// enable the pause, disable play
var pauseButton = $get("btnPause");
var playButton = $get("btnPlay");
pauseButton.disabled = false;
playButton.disabled = true;
}
```

While the rotator is in "AutomaticAdvance" mode, it responds when the mouse moves over it by temporarily pausing in its scrolling until the mouse moves off.

The handler for the "Pause" button's onclick event sets the **RotatorType** back to "FromCode" before calling the **pause()** method. This is to prevent the automatic pausing and resuming that occurs when the mouse moves over the rotator. If the rotator is left in "AutomaticAdvance" mode, the rotator will automatically pause when the mouse moves over it (which does nothing because the rotator is already paused) and unpause when the mouse moves off of it. By setting the **RotatorType** back to "FromCode", the rotator remains paused when the mouse moves off of it.

[JavaScript] Pause button's onclick handler

```
function PauseRotator() {
    var rotator = $find("<%= RadRotator1.ClientID %>");

    // change the rotator type to FromCode so that it does
    // not resume when the mouse moves over items.
    // Then pause the rotator.
    rotator.set_rotatorType(Telerik.Web.UI.RotatorType.FromCode);
    rotator.pause();

    // disable the pause button, enable play
    var pauseButton = $get("btnPause");
    var playButton = $get("btnPlay");
    pauseButton.disabled = true;
    playButton.disabled = false;
}
```

The complete source for this example can be found in `\\VS Projects\\Rotator\\ClientSide`.

6.6 Client-Side Items Management

RadRotator has three methods that allow the developer to manage the control's items on the client-side:

- **addRotatorItem(radRotatorItemData, index)** - adds a new item to the RadRotator's client-side item collection at the specified index. The HTML markup of the new items is configured in the **Html** property of the **radRotatorItemData** object that is passed to this method. If the **radRotatorItemData** parameter is not specified, an empty item will be inserted. If the **index** parameter is not set, the new item will be added at the end of the collection:

JavaScript

```
radRotatorItemData = {};
radRotatorItemData.Html = "<div>Item's content</div>";
```

- **removeRotatorItem(index)** - removes an item from the RadRotator's client-side item collection. If the

`index` parameter is not set, the last item from the collection will be removed

- `clearItems()` - clears the RadRotator's client-side item collection

Note that the changes made via these methods will affect only the client-side items collection. The listed client-side methods are supported for all RadRotator modes, except **CoverFlow** and **Carousel**.

6.7 Control Specifics

Rotator Items

So far, all of the examples in this chapter have used rotators that were bound to some sort of data source. The rotator has had a single item template, with items bound to fields from the data source. It is possible, however, to use **RadRotator** *without* a data source. Although it is not present in the Properties Window at design time, RadRotator has an **Items** property, which you can use to populate items one by one. Each item has its own template, to which you can add any sort of content.

💡 Because each item in the **Items** property collection has its own template, the items can have drastically different appearances. By contrast, when using a bound rotator, each record from the data source is displayed using a single item template, so every item must look alike. To add items that use a different template to a data-bound rotator, set the **AppendDataBoundItems** property to true. Be aware, however, that you can't interleave data-bound items with items in the **Items** collection.

The following walk-through shows you how to use the **Items** property collection to populate an unbound rotator. It creates a moving banner of fish across the top of the Web page:



The complete source for this example is in `\VS Projects\Rotator\ItemsCollection`.

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. In the solution explorer, add an "Images" folder to your project.
3. Copy the files "fish1.png", "fish2.png", "fish3.png", "fish4.png", and "fish5.png" from the `\VS Projects\Images` folder to your project's new `\Images` folder.
4. Drag a **RadRotator** control from the Tool Box onto your Web page.
5. Using the Properties Window, set the **FrameDuration** property to 1(ms) and the **ScrollDirection** property to "Left". This will result in a continuous stream of items from right to left across the rotator.
6. Set the **Height** property to "40px", the **Width** property to "522px" (9 images will be shown), the **ItemHeight** property to "40px", and the **ItemWidth** property to "58px".
7. Switch to the Source window to add items directly to the markup for your page.
8. Inside the **RadRotator** declaration, type an open bracket ("`<`"). Intellisense will offer you a list of options on what you can add. Choose "Items":
9. Inside the `<Items>` collection, type another open bracket. This time, there is only one option: "telerik:RadRotatorItem". Choose this option to add an item to your rotator. Give it an ID of "item1" and don't forget to add `runat="server"`.
10. Inside your new item, add an `<ItemTemplate>`.
11. Inside the `<ItemTemplate>` node, add an `<asp:Image>`. Set `ID="Image1"` and `runat="server"`.
12. When you type `ImageUrl="` to add the `ImageUrl` attribute, intellisense should pop up a menu of options.

UI for ASP.NET AJAX

Choose "Pick URL..." from the list:



13. Navigate to the Images folder and select "fish1.png".
14. At this point, your RadRotator declaration should look like the following:

[ASP.NET] RadRotator with first item

```
<telerik:RadRotator ID="RadRotator1" runat="server" FrameDuration="1"  
  Width="522px" ItemWidth="58px" Height="40px" ItemHeight="40px" ScrollDirection="Left">  
  <Items>  
    <telerik:RadRotatorItem ID="item1" runat="server" >  
      <ItemTemplate>  
        <asp:Image ID="Image1" runat="server" ImageUrl="~/Images/fish1.png" />  
      </ItemTemplate>  
    </telerik:RadRotatorItem>  
  </Items>  
</telerik:RadRotator>
```

15. Copy the entire declaration for the first item, and paste it as a second item in the **Items** collection. Change the **ID** of the RadRotatorItem to "item2", and on the Image, change the **ID** to "Image2" and the **ImageUrl** to "~/Images/fish2.png".
16. Repeat this process until you have 10 items, giving each item and each image a sequential ID ("item3", "item4", and so on and "Image3", "Image4", and so on), and setting the **ImageUrl** to a randomly selected image chosen from among the 5 images you added to the Images folder.
17. Hit Ctrl-F5 to run the application. A banner at the top of your Web page shows a string of fish all moving continuously to the left.

6.8 Coverflow mode

CoverFlow is an animated, three dimensional graphical user interface that can be used to display and browse your image galleries.



Starting from Q3 2010, Telerik RadRotator supports two additional **RotatorTypes** - **CoverFlow** and **CoverFlowButtons**. Both new modes can be set using **RotatorType** property of the control.

You can enable CoverFlow mode of RadRotator by following the steps below:

1. Set the RotatorType property to CoverFlow.

RadRotator's declaration

```
<telerik:RadRotator RotatorType="CoverFlow" ID="RadRotator1" runat="server"
    Width="748px" ItemWidth="150" ScrollDirection="Left, Right" Height="233px"
    ItemHeight="113"
    ScrollDuration="500" FrameDuration="2000" PauseOnMouseOver="false"
    InitialItemIndex="4">
    <ItemTemplate>
        <asp:Image ID="Image1" runat="server" ImageUrl='<%= Container.DataItem %>'
        AlternateText="<%= VirtualPathUtility.GetFileName(Container.DataItem.ToString()) %>" />
    </ItemTemplate>
</telerik:RadRotator>
```

If the **RotatorType="CoverFlowButtons"** is set, then the items are moved by clicking the RadRotator's default navigation buttons located on the left and right side of the control.

2. You can use the following server code to populate the Rotator with Images:

Populating images in RadRotator - C#

```
string virtualPath = "~/ImagesSource"; //specify Images folder
private void Page_Load(object sender, System.EventArgs e)
{
    if (!IsPostBack)
    {
        RadRotator1.DataSource = GetFilesInFolder(virtualPath); // Set datasource
        RadRotator1.DataBind();
    }
}

// Returns all virtual paths to files located in the given virtual directory
protected List<string> GetFilesInFolder(string folderVirtualPath)
{
    string physicalPathToFolder = Server.MapPath(folderVirtualPath); // Get the physical path
    string[] physicalPathsCollection = System.IO.Directory.GetFiles(physicalPathToFolder);
    Get all child files of the given folder
    List<string> virtualPathsCollection = new List<string>(); // Contains the result
    foreach (String path in physicalPathsCollection)
    {
        // The value of virtualPath will be similar to '~/PathToFolder/Image1.jpg
        string virtualPath = VirtualPathUtility.AppendTrailingSlash(folderVirtualPath) +
        System.IO.Path.GetFileName(path);
        virtualPathsCollection.Add(virtualPath);
    }
    return virtualPathsCollection;
}
```

Populating images in RadRotator - VB.NET

```
Private virtualPath As String = "~/ImagesSource"
Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    If Not IsPostBack Then
        RadRotator1.RotatorType = RotatorType.CoverFlowButtons
        ' Change rotator's type
        RadRotator1.DataSource = GetFilesInFolder(virtualPath)
        ' Set datasource
        RadRotator1.DataBind()
    End If
End Sub

' Returns all virtual paths to files located in the given virtual directory
Protected Function GetFilesInFolder(ByVal folderVirtualPath As String) As List(Of
String)
    Dim physicalPathToFolder As String = Server.MapPath(folderVirtualPath)
    ' Get the physical path
    Dim physicalPathsCollection As String() = System.IO.Directory.GetFiles
(physicalPathToFolder)
    ' Get all child files of the given folder
    Dim virtualPathsCollection As New List(Of String)()
    ' Contains the result
    For Each path As [String] In physicalPathsCollection
        ' The value of virtualPath will be similar to '~/PathToFolder/Image1.jpg
        Dim virtualPath As String = VirtualPathUtility.AppendTrailingSlash
(folderVirtualPath) + System.IO.Path.GetFileName(path)
        virtualPathsCollection.Add(virtualPath)
    Next
    Return virtualPathsCollection
End Function
```

3. More precise configuration of CoverFlow Animation options is available through the following JavaScript code:

CoverFlow Animation options - JavaScript

```
<script type="text/javascript">
    <![CDATA[
        // Animation options - the set_scrollAnimationOptions method takes two arguments - the
        first one is the ClientID of the rotator, which we want to configure and the second one :
        // a hash table with the options we want to apply.
        Telerik.Web.UI.RadRotatorAnimation.set_scrollAnimationOptions("<%= RadRotator1.ClientID
>", // The ClientID of the RadRotator
        {
            minScale: 0.8, // The size scale [0; 1], applied to the items that are not
            selected.
            y0: 60, // The offset in pixels of the center of the selected item from the top
            edge of the rotator.
            xR: -30, // The offset in pixels between the selected items and the first item (
            the left and on the right of the selected item.
            xItemSpacing: 50, // The offset in pixels between the items in the rotator.
            matrix: { m11: 1, m12: 0, m21: -0.1, m22: 1 }, // The 2d transformation matrix,
            applied to the items that appear on the right of the selected item.
            reflectionHeight: 0.5, // The height of the reflection
        }
    ]>
</script>
```

```

        reflectionOpacity: 1 // The opacity, applied to the reflection
    });
    // end of animationOptions
    //]]>
</script>

```

Important: The **CoverFlow** mode of RadRotator works best, when there is only a single image in every rotator item. The reflection feature of the **CoverFlow** mode is supported only when there is an image in every rotator item.

6.9 Carousel mode

Carousel is an animated, graphical user interface used to display and rotate your image galleries.



Starting from Q2 2010, Telerik RadRotator supports two additional **RotatorTypes** - **Carousel** and **CarouselButtons**. Both new modes can be set using **RotatorType** property of the control.

In order to enable Carousel mode of RadRotator follow the steps below:

1. Set the **RotatorType** property to **Carousel**:

RadRotator's declaration

```

<telerik:RadRotator RotatorType="CarouselButtons" ID="RadRotator1" runat="server"
Width="810px" ItemWidth="280"
    Height="350px" ItemHeight="200" ScrollDuration="500" FrameDuration="2000"
PauseOnMouseOver="false">
    <ItemTemplate>
        <asp:Image ID="Image1" runat="server" ImageUrl='<%= Container.DataItem %>'
AlternateText="<%= VirtualPathUtility.GetFileName(Container.DataItem.ToString()) %>" />
    </ItemTemplate>
</telerik:RadRotator>

```

If the **RotatorType="CarouselButtons"** is set, then the items are moved by clicking the RadRotator's default navigation buttons located on the left and right side of the control.

2. You can use the following server code to populate the Rotator with Images:

Populating images in RadRotator - C#

```
string virtualPath = "~/ImagesSource"; //specify images folder
protected void Page_Load(object sender, EventArgs e)
{
    if (!Page.IsPostBack)
    {
        RadRotator1.DataSource = GetFilesInFolder(virtualPath); // Set datasource
        RadRotator1.DataBind();
    }
}
protected List<string> GetFilesInFolder(string folderVirtualPath)
{
    string physicalPathToFolder = Server.MapPath(folderVirtualPath); // Get the physical path
    string filterExpression = "*.gif";
    string[] physicalPathsCollection = System.IO.Directory.GetFiles(physicalPathToFolder,
filterExpression); // Get all child files of the given folder
    List<string> virtualPathsCollection = new List<string>(); // Contains the result
    foreach (String path in physicalPathsCollection)
    {
        // The value of virtualPath will be similar to '~/PathToFolder/Image1.jpg
        string virtualPath = VirtualPathUtility.AppendTrailingSlash(folderVirtualPath) +
System.IO.Path.GetFileName(path);
        virtualPathsCollection.Add(virtualPath);
    }
    return virtualPathsCollection;
}
```

Populating images in RadRotator - VB.NET

```
Dim virtualPath As String = "~/ImagesSource"
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    If Not Page.IsPostBack Then
        RadRotator1.DataSource = GetFilesInFolder(virtualPath)
        ' Set datasource
        RadRotator1.DataBind()
    End If
End Sub

Protected Function GetFilesInFolder(ByVal folderVirtualPath As String) As List(Of
String)
    Dim physicalPathToFolder As String = Server.MapPath(folderVirtualPath)
    ' Get the physical path
    Dim filterExpression As String = "*.gif"

    Dim physicalPathsCollection As String() = System.IO.Directory.GetFiles
(physicalPathToFolder, filterExpression)
    ' Get all child files of the given folder
    Dim virtualPathsCollection As New List(Of String)()
    ' Contains the result
    For Each path As [String] In physicalPathsCollection
        ' The value of virtualPath will be similar to '~/PathToFolder/Image1.jpg
        Dim virtualPath As String = VirtualPathUtility.AppendTrailingSlash
(folderVirtualPath) + System.IO.Path.GetFileName(path)
```

```
        virtualPathsCollection.Add(virtualPath)
    Next
    Return virtualPathsCollection
End Function
```

Important: The **Carousel** mode of RadRotator works best when there is only a single image in every rotator item.

6.10 Summary

In this chapter you looked at the RadRotator control and saw some of the ways it can display a stream of changing content. You saw some of the important properties for configuring the rotator. You created a simple application that displayed data taken from an XML file. You learned to start and stop the rotator using the client-side api. You also learned how to add items explicitly when the rotator is not bound to a data source.

7 Ajax

7.1 Objectives

- Take a tour of the AJAX related controls including RadAjaxManager, RadAjaxPanel, RadAjaxManagerProxy and RadAjaxLoadingPanel.
- Build a simple AJAX-Enabled web application that first uses RadAjaxPanel, then substitutes RadAjaxManager. The application also displays a loading panel during the AJAX request.
- Explore the design time interface for each of the AJAX controls, taking special notice of where the controls are similar. You will learn how to access properties and methods through Smart Tag and Properties Window.
- Programmatically define AJAX settings at run-time on the server, detect which requests are triggered by AJAX and automatically run client-script after a response.
- Create custom AJAX requests to bridge client and server functionality and AJAX-enable controls that lack intrinsic AJAX abilities. Learn important client methods to toggle AJAX functionality, cancel requests and iterate AJAX settings. Also learn how to handle AJAX client events raised at request start and response end.
- Explore design decisions involved with AJAX enabling your application.
- Learn how to make Winforms-like user interfaces using AJAX enabled user controls and how the page lifecycle impacts working with AJAX-enabled user controls.
- See how the RadAjaxManagerProxy control provides visibility to RadAjaxManager settings in complex container-ship scenarios.
- Use RadScriptBlock and RadCodeBlock to handle common script + markup related issues.

7.2 Introduction

The RadAjax controls AJAX-enable your application with little programming or configuration effort on your part. The control set consists of **RadAjaxPanel**, **RadAjaxManager**, **RadAjaxManagerProxy** and **RadAjaxLoadingPanel**.

RadAjaxPanel lets you instantly AJAX-enable an area of a web page simply by dropping controls on the panel. RadAjaxPanel mimics an ASP:UpdatePanel, i.e. any control on the panel that performs a postback automatically uses AJAX updates instead of a postback. The effect from the user's perspective is that only the panel area updates and the full page does not refresh.

Use RadAjaxPanel when...

- You want instant gratification. All controls dropped onto the panel are AJAX-enabled without further configuration.
- You want to start learning about how AJAX-enabled applications behave.
- The page has a simple layout with no complex interactions between controls.
- The controls are placed next to one another on the page.

That AJAX is involved does not guarantee a performant application. Keep in mind that whatever you put into the panel will be sent to the server on the AJAX update. If you put everything in the page into the panel, then the application can perform no better than a standard application. The trick is to cut down the amount of payload between client and server, to make discrete little updates that move the minimum amount of data.

While RadAjaxPanel is an easy development experience, **RadAjaxManager** should be your go-to control when you need to AJAX-enable an application. It can do everything the RadAjaxPanel can and quite a bit more. Use RadAjaxManager when...

- You have complex pages and where only small parts of the page need to be updated at any one time.
- You need fine-tuned control over the updates going to the server via AJAX.

- You need to get every last drop of performance from all aspects of your application.
- Controls to be updated are placed in disparate locations on the page.

RadAjaxManagerProxy is a stand-in at design time when you need to configure RadAjaxManager from within UserControls or Content pages. RadAjaxManager can only be present once in a page, so the proxy is convenient in this situations. Later, when we use RadAjaxManager with a user control, it will be clear how RadAjaxManagerProxy makes the development experience easier.

The **RadAjaxLoadingPanel** control is used to display a "spinnny" graphic in the updating area while the update is performed to provide a little user feedback.

7.3 Getting Started

RadAjaxPanel - Thinking Inside the Box

Let's start with a simple RadAjaxPanel demonstration that has only a button and label so that you can see the AJAX interaction vs. a standard postback.



You can find the complete source for this project at:
 \VS Projects\AJAX\Getting Started

1. Create a new web application and add a ScriptManager component to the page.
2. Add a **RadCalendar** and a standard ASP Label control to the page.
3. Set the **ID** of the Label to "PostBackLabel" and the **Text** property blank ("").
4. Set the **ID** of the calendar to "PostbackCalendar". In the calendar's Smart Tag, check the **Enable AutoPostBack** option and un-check the **Enable Multi-Select** option.
5. In the designer, double-click the calendar to create a "SelectionChanged" event handler and add the code below:

[VB] Handling SelectionChanged on Postback

```
Protected Sub PostbackCalendar_SelectionChanged(ByVal sender As Object, ByVal e As
Telerik.Web.UI.Calendar.SelectedDatesEventArgs)
    PostBackLabel.Text = PostbackCalendar.SelectedDate.ToShortDateString()
End Sub
```

[C#] Handling SelectionChanged on Postback

```
protected void PostbackCalendar_SelectionChanged(
    object sender, Telerik.Web.UI.Calendar.SelectedDatesEventArgs e)
{
    PostBackLabel.Text = PostbackCalendar.SelectedDate.ToShortDateString();
}
```

6. Drop a **RadAjaxPanel** to the form below the calendar and label.
7. Drop another RadCalendar and Label onto the RadAjaxPanel.
8. Set the **ID** of the Label to "AjaxLabel" and the **Text** property blank ("").
9. Set the **ID** of the calendar to "AjaxCalendar". In the calendar's Smart Tag, check the **Enable AutoPostBack** option and un-check the **Enable Multi-Select** option.
10. In the designer, double-click the calendar to create a "SelectionChanged" event handler and add the code below:

[VB] Handling SelectionChanged on Ajax Update

UI for ASP.NET AJAX

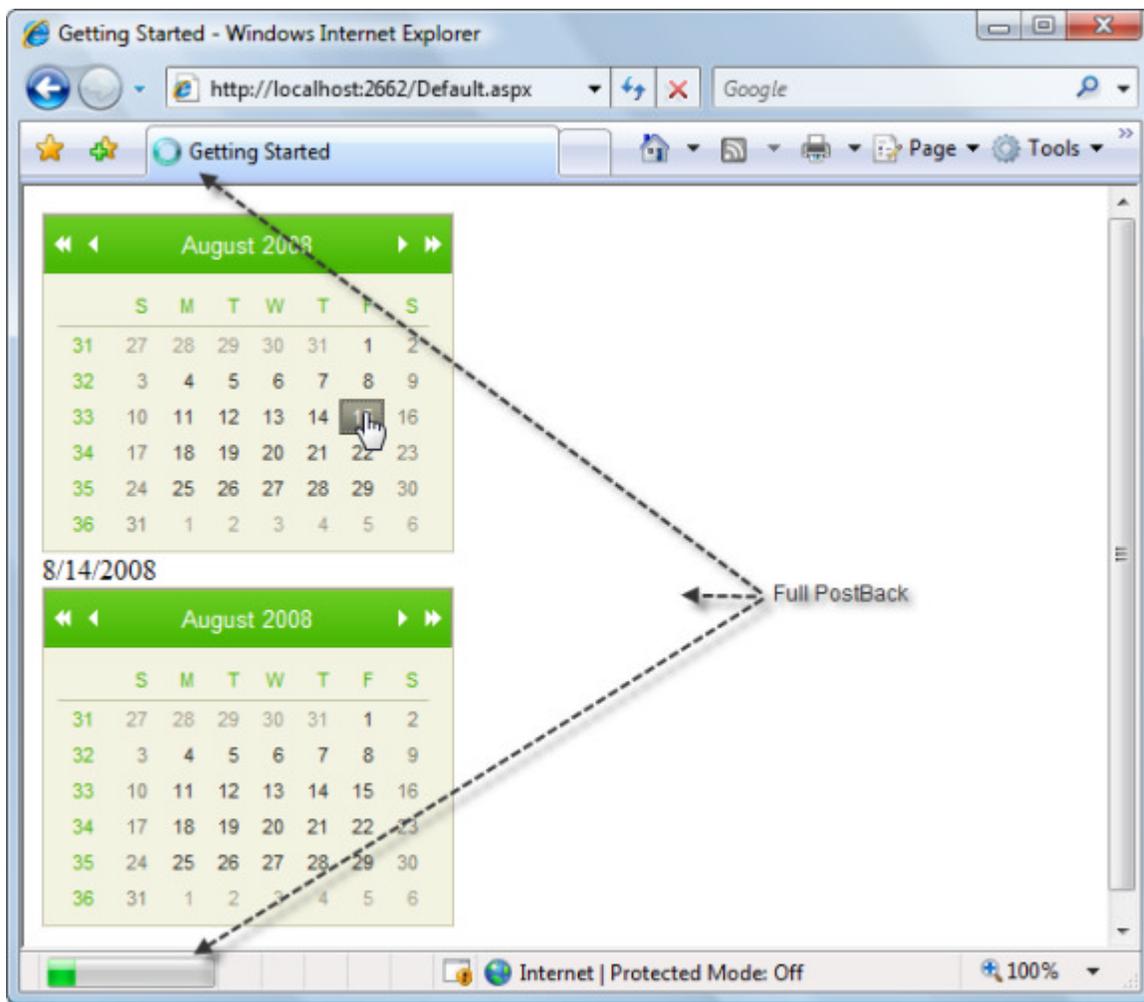
```
Protected Sub AjaxCalendar_SelectionChanged(ByVal sender As Object, ByVal e As  
Telerik.Web.UI.Calendar.SelectedDatesEventArgs)  
    AjaxLabel.Text = AjaxCalendar.SelectedDate.ToShortDateString()  
End Sub
```

[C#] Handling SelectionChanged on Ajax Update

```
protected void AjaxCalendar_SelectionChanged(  
    object sender, Telerik.Web.UI.Calendar.SelectedDatesEventArgs e)  
{  
    AjaxLabel.Text = AjaxCalendar.SelectedDate.ToShortDateString();  
}
```

What has changed between the first set of calendar and label properties and code? Nothing, except that the second calendar and label are housed in a RadAjaxPanel.

- Press Ctrl-F5 to run the application. Click on the "Postback" calendar and observe the indications that a full postback is taking place. In Internet Explorer 7 for example, the "spiny" icon in the page tab displays and animates. The page may appear to blink and the progress bar will display.



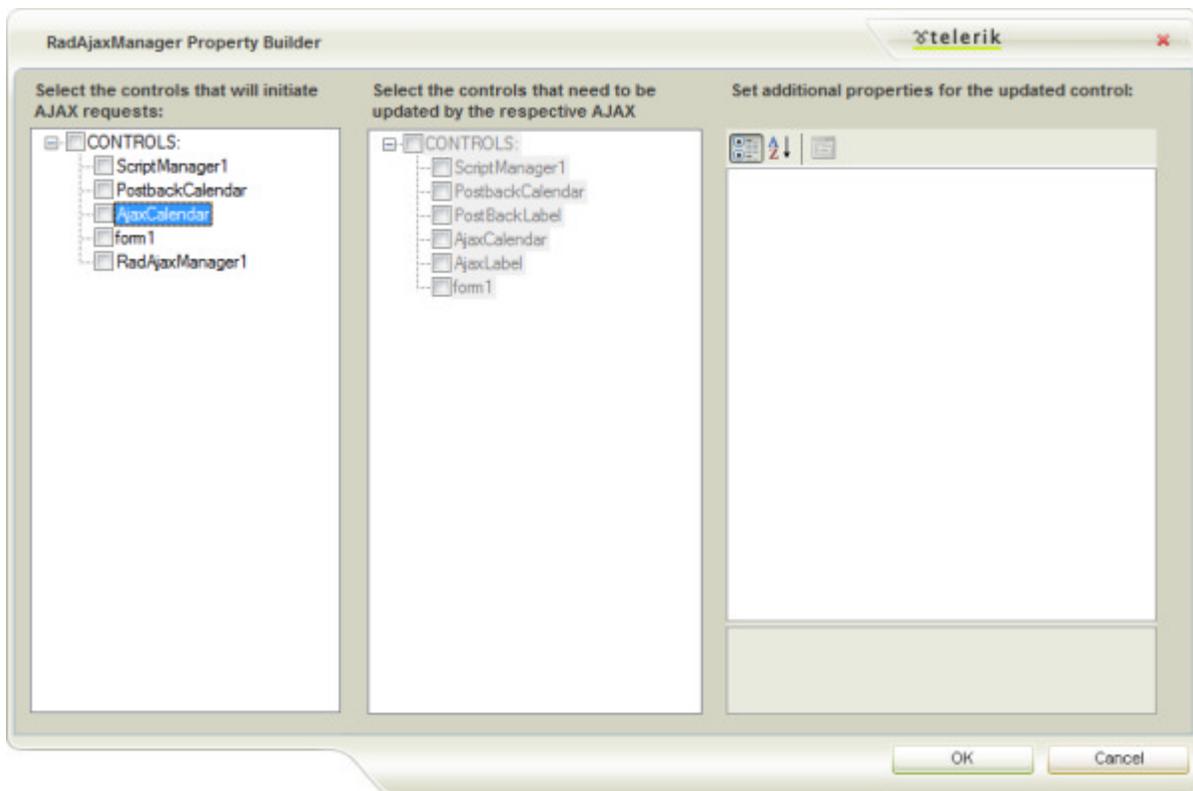
- Now click the "Ajax Update" calendar. The label will change to display the new date and no other part of the page will change: no blinking of the page, no "spiny" in the tab and no loading progress bar.

Having an Out-of-panel Experience with RadAjaxManager

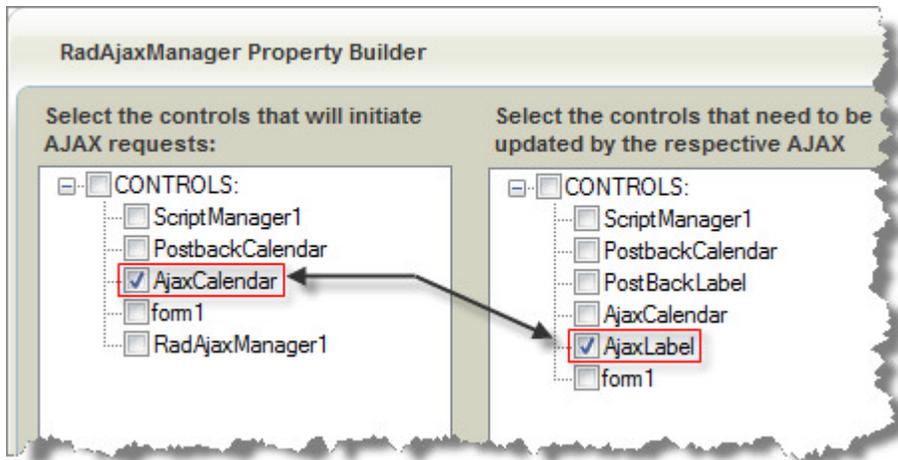
Now try using **RadAjaxManager** instead of RadAjax to see how the development and run-time experience changes.

1. Start with the previous RadAjaxPanel project and navigate to the markup for the page.
2. Remove the RadAjaxPanel from the markup, leaving the calendars and labels in place.
3. Navigate back to the design view for the page.
4. Add a RadAjaxManager component to the web page.
5. In the RadAjaxManager Smart Tag, select **Configure Ajax Manager**.

The dialog shows a treeview list of controls in the first column that can initiate AJAX updates. When one of the "initiator" controls is checked, you can check controls from the next column that can be updated via AJAX. The last column has properties for the currently selected updated control.



6. Check "AjaxCalendar" in the first column and check "AjaxLabel" in the second column. Click **OK** to close the dialog.



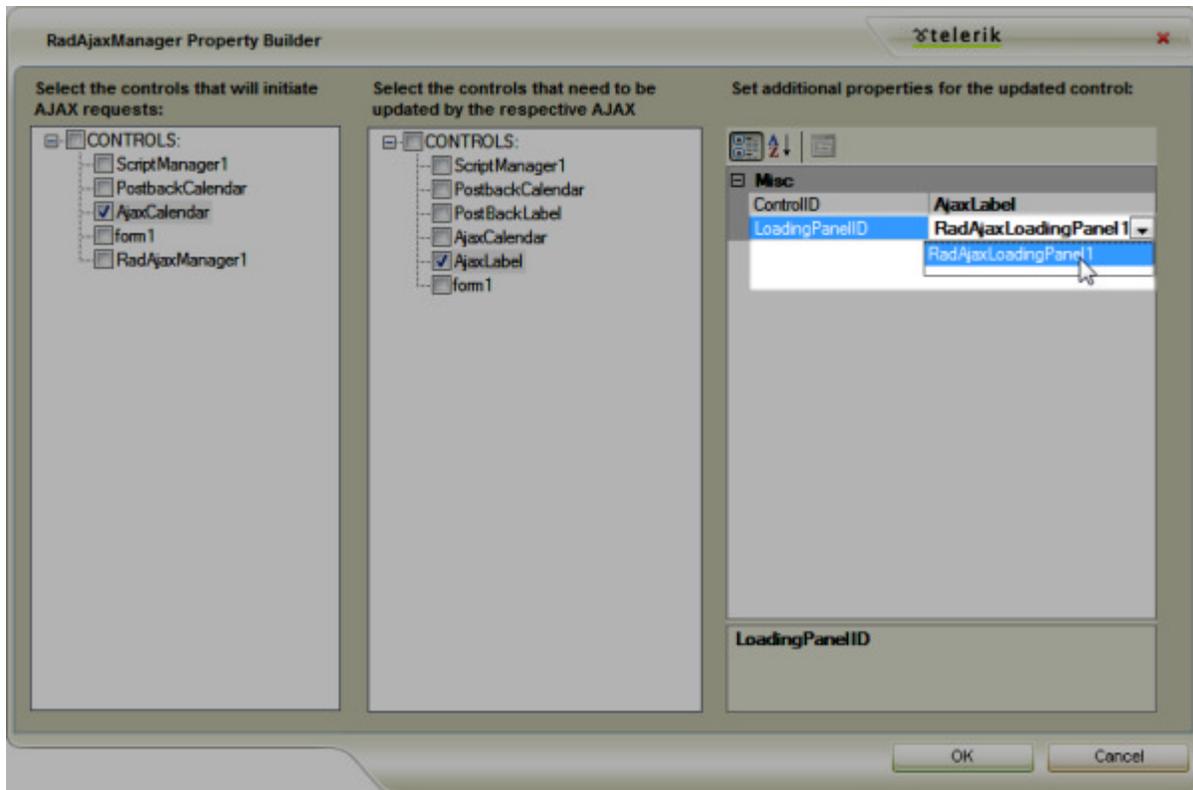
7. Press Ctrl-F5 to run the application. *The behavior is the same as the RadAjaxPanel version. The AJAX update performs very quickly with no screen flicker.*

 While this is a trivial example, complex screens with updated controls in disparate locations can be handled just as easily.

RadAjaxLoadingPanel

The assignment of the label text is so quick the user is not going to notice any appreciable update time. In longer running operations, the user needs a little feedback. The RadAjaxLoadingPanel is a templated container associated with a RadAjaxManager. By default the loading panel has the text "Loading..." and an animated "spiny" image.

1. Add a RadAjaxLoadingPanel to the page.
2. In the RadAjaxManager Smart Tag, select **Configure Ajax Manager**.
3. In the second column (updated controls), select "AjaxLabel". In the third column (updated controls properties) select the loading panel from the **LoadingPanelID** property drop down list.



- To simulate a long-running operation, let us add a "sleep" to the SelectionChanged event handler so you can see the loading panel. Add a call to `System.Threading.Thread.Sleep()` for 200 milliseconds:

[VB] Simulate Long Running Operation

```
Protected Sub AjaxCalendar_SelectionChanged(ByVal sender As Object, ByVal e As Telerik.Web.UI.Calendar.SelectedDatesEventArgs)
    System.Threading.Thread.Sleep(200)
    AjaxLabel.Text = AjaxCalendar.SelectedDate.ToShortDateString()
End Sub
```

[C#] Simulate Long Running Operation

```
protected void AjaxCalendar_SelectionChanged(
    object sender, Telerik.Web.UI.Calendar.SelectedDatesEventArgs e)
{
    System.Threading.Thread.Sleep(200);
    AjaxLabel.Text = AjaxCalendar.SelectedDate.ToShortDateString();
}
```

- Re-run the application. When you click a date, the "spinnny" graphic displays in the same location as the updated control. When the operation finishes, the graphic disappears and the updated control becomes visible again.



Loading...

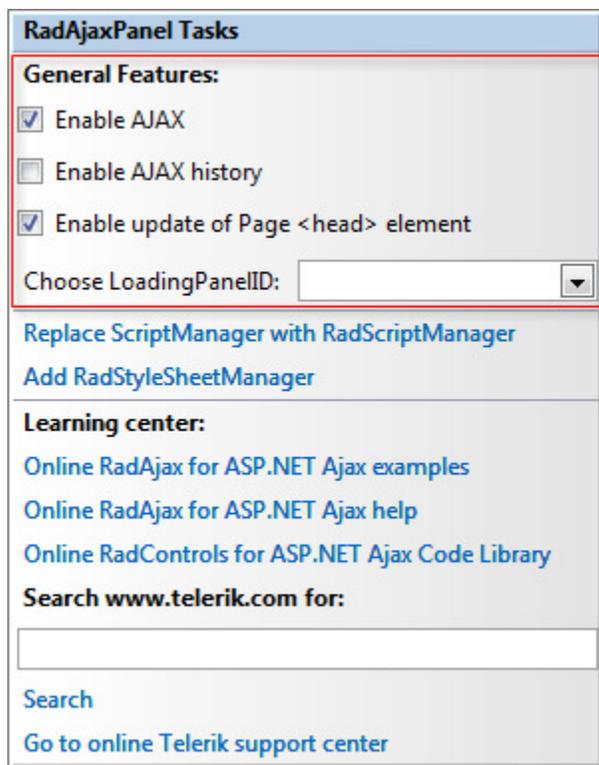


7.4 Designer Interface

Smart Tag

Both RadAjaxPanel and RadAjaxManager have Smart Tag options to control general AJAX behavior and loading panel features (shown in the screenshot below).

- RadAjaxManager has an additional "Configure Ajax Manager" link used to define AJAX settings. This feature was described in the previous "Getting Started" section.



- **Enable AJAX** is on by default, but you can disable this option to verify that your application works with standard post backs.
 - ✍ In most cases your web page should work with standard post backs. A good debugging measure is to simply turn off AJAX and re-test your application without it.
- **Enable AJAX history** is off by default, so the forward and back buttons on the browser are disabled. With

the Enable AJAX History turned on in Internet Explorer, the browser remembers which pages you have been to and the state of any updated controls (assuming ViewState hasn't been disabled) that are updated.

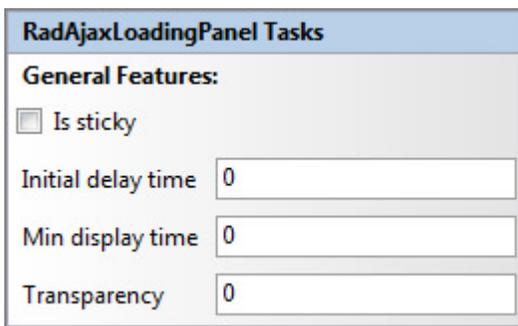
 This feature works only on Internet Explorer. On other browsers the buttons will be disabled even if you check this option.

- **Enable update of Page <head> element** if true allows updates to the page head element so that title and style sheet changes are applied.
- **Choose LoadingPanelID** for RadAjaxPanel lets you select a RadLoadingPanel from a list. On the RadAjaxManager Smart Tag the task is called **Choose DefaultLoadingPanelID** and lets you choose the loading panel that displays for all updated controls where not already defined.

The **RadAjaxManagerProxy** Smart Tag has a single option that refers to the only task this control performs, **Configure Ajax Manager**. RadAjaxManagerProxy doesn't include any of the RadAjaxManager properties or methods except for the **AjaxSettings** property that is populated by this option.

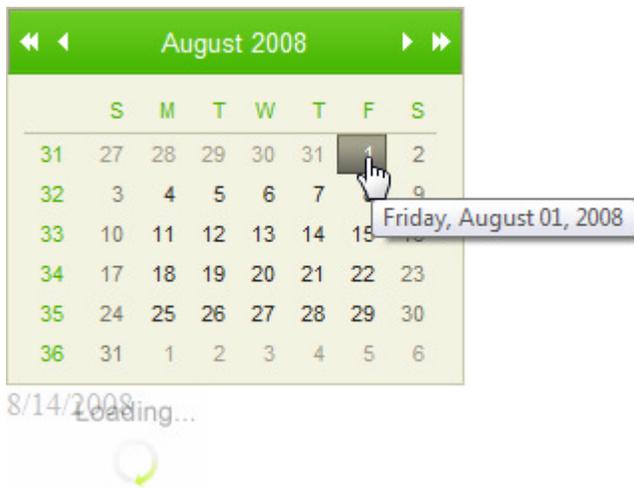


The **RadAjaxLoadingPanel** Smart Tag options configure how the loading panel is positioned, the timing of when it displays and how it displays.



- By default the loading panel displays in the same location as the updated control. If **Is sticky** is checked the loading panel displays where you have it positioned on the page at design-time.
- **Initial delay time** is the number of milliseconds before the loading panel displays. If the request completes before the initial delay time expires, the loading panel doesn't show. In the Getting Started example we had a loading panel with an artificial, "sleep()" induced wait of 200 milliseconds. If you set Initial delay time to "1000", the loading panel never displays.
- **Min display time** is the minimum amount of time (in milliseconds) the loading panel displays even if the update operation completes sooner. This timing setting helps eliminate annoying flicker that can occur if the loading panel has just appeared and the request finishes. In the Getting Started example, if you set Min display time to "1000", the loading panel displays a full second, even though the request takes a little over 200 milliseconds.
 -  Imagine a "heads down" order entry system where a sales person is rapidly keying multiple order lines, one right after the other. Once the sales person enters a part number and a quantity, you might expect the system to lookup the sales price. If the system was running quickly, you wouldn't want to slow the sales person down with "spiny" animation for each entry, but if the database was under load and couldn't locate the product right away you would want an indicator that processing was underway. In this case you could set the Initial delay time to a value that is over the average amount of time it takes to enter a line.
- The loading panel doesn't actually replace the updated control but is displayed right over the top of it. Set a **Transparency** value of 0-100 so that the user can see the updated control underneath loading panel. 0 is opaque and 100 is completely transparent. The loading panel shown below is displaying over a label and

has a transparency of "25".



Properties Window

Most of the principal RadAjaxManager and RadAjaxPanel design-time properties have been touched on so far, but there are a couple of special purpose properties that you can reach at design-time only from the Properties Window:

- **RestoreOriginalRenderDelegate**: If you have configured your applications to run in **medium trust** (<http://msdn.microsoft.com/en-us/library/ms998341.aspx>) you should set this property to **false** to avoid the error "InvalidOperationException: Not enough permissions...".
- **RequestQueueSize**: By design, the ASP.NET AJAX framework cancels the current AJAX request if you try to initiate a second request. Set RequestQueueSize greater than zero to automatically enable RadAjax queueing. You can test this behavior yourself using the Getting Started project. Add a Sleep() of 1000 milliseconds to the calendar SelectionChanged event handler and set the RequestQueueSize to "3". Run the application and click on three dates in a row quickly. Each will execute for a second and all will complete. Set the RequestQueueSize to "0" and you only see the last update.

Queueing may make sense depending on the semantics of your application. For example if you had a list of tasks that could be executed and weren't dependent on one another, say, selecting a series of pre-defined emails to be sent, then queueing could be a good choice. If the application was an entry screen where the user could randomly work on different operations in the UI, then the user might change their mind at any time and suddenly work on something else - here queueing might not be the way to go. The usability requirements determine the route to take.

Requests exceeding RequestQueueSize are discarded.



Gotcha!

"When I use RadAjaxManager, the AJAX-enabled controls are placed on a new line!"

The reason for this behavior is that **RadAjaxManager** dynamically inserts MS AJAX UpdatePanel controls around the updated controls. The default render mode is **Block**. A new RadAjaxManager property **UpdatePanelsRenderMode** can be set to **Inline** so that the layout will not change. The screenshot below shows the effect of the default **Block** layout vs. the new **Inline** render mode:

Block

Inline

The complete source for the project can be found in \VS Projects\Ajax\RenderMode. The project simply defines a button that updates a TextBox with the current time, and a check box that switches between the two render modes. The code to update the date is not strictly necessary to show the change in layout.

Colorized Example Code



[ASP.NET] Render Mode Demo Markup

```

<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<telerik:RadAjaxManager ID="RadAjaxManager1" runat="server">
  <AjaxSettings>
    <telerik:AjaxSetting AjaxControlID="Button1">
      <UpdatedControls>
        <telerik:AjaxUpdatedControl ControlID="TextBox1" />
      </UpdatedControls>
    </telerik:AjaxSetting>
  </AjaxSettings>
</telerik:RadAjaxManager>

<asp:Button ID="Button1" runat="server" OnClick="Button1_Click" Text="Button" />
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
<asp:CheckBox ID="CheckBox1" runat="server" AutoPostBack="True" Checked="True"
  OnCheckedChanged="CheckBox1_CheckedChanged"
  Text="Block" />
    
```

[VB] Changing the Render Mode

```

Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
  TextBox1.Text = DateTime.Now.ToLongTimeString()
End Sub

Protected Sub CheckBox1_CheckedChanged(ByVal sender As Object, ByVal e As
EventArgs)
  RadAjaxManager1.UpdatePanelsRenderMode = IIf((TryCast(sender,
CheckBox)).Checked, UpdatePanelRenderMode.Block, UpdatePanelRenderMode.Inline)
  (TryCast(sender, CheckBox)).Text = IIf((TryCast(sender,
CheckBox)).Checked, "Block", "Inline")
End Sub
    
```

[C#] Changing the Render Mode

```
protected void Button1_Click(object sender, EventArgs e)
{
    TextBox1.Text = DateTime.Now.ToLongTimeString();
}

protected void CheckBox1_CheckedChanged(object sender, EventArgs e)
{
    RadAjaxManager1.UpdatePanelsRenderMode = (sender as CheckBox).Checked ?
        UpdatePanelRenderMode.Block : UpdatePanelRenderMode.Inline;
    (sender as CheckBox).Text = (sender as CheckBox).Checked ?
        "Block" : "Inline";
}
```

7.5 Server-Side Programming

Ajax Settings

You may not know all of the controls that will be AJAX-enabled at design time. You can configure RadAjaxManager settings programmatically using the **AjaxSettings** collection.

You can add a setting using the **AddAjaxSetting()** method that takes an initiating control, an updated control and optionally a loading panel.

The page for this next example starts out with just a Panel:

[ASP.NET] The page with div element

```
<body>
  <form id="form1" runat="server">
    <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server">
      </asp:ScriptManager>
      <telerik:RadAjaxManager ID="RadAjaxManager1" runat="server">
      </telerik:RadAjaxManager>
      <asp:Panel ID="Panel1" runat="server"></asp:Panel>
    </div>
  </form>
</body>
```

In the Page_Load a standard button and label are added to the div, then the AjaxSettings has a single setting added where "btnTime" is the initiator and "lblTime" is the updated control.

☑ Because the control is added dynamically, it has to be re-created along with the AjaxSetting on every page load. The behavior of dynamically added controls will be covered in more detail in the upcoming Page Life Cycle section.

[VB] Adding AJAX Settings

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' create a button and label, add them to the page
    Dim btnTime As New Button()
    btnTime.ID = "btnTime"
```

```

btnTime.Text = "Show Time"
Dim lblTime As New Label()
lblTime.ID = "lblTime"
lblTime.Text = "time"
Panel1.Controls.Add(btnTime)
Panel1.Controls.Add(lblTime)
' add a click event handler
AddHandler btnTime.Click, AddressOf btnTime_Click
' add an ajax setting where the panel is both initiator and label is
' updated control
RadAjaxManager1.AjaxSettings.AddAjaxSetting(btnTime, lblTime)
End Sub
Protected Sub btnTime_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim lblTime As Label = TryCast(FindControl("lblTime"), Label)
    lblTime.Text = DateTime.Now.ToLongTimeString()
End Sub

```

[C#] Adding AJAX Settings

```

protected void Page_Load(object sender, EventArgs e)
{
    // create a button and label, add them to the page
    Button btnTime = new Button();
    btnTime.ID = "btnTime";
    btnTime.Text = "Show Time";
    Label lblTime = new Label();
    lblTime.ID = "lblTime";
    lblTime.Text = "time";
    Panel1.Controls.Add(btnTime);
    Panel1.Controls.Add(lblTime);
    // add a click event handler
    btnTime.Click += new EventHandler(btnTime_Click);
    // add an ajax setting where the button is initiator and label is
    // updated control
    RadAjaxManager1.AjaxSettings.AddAjaxSetting(btnTime, lblTime);
}
protected void btnTime_Click(object sender, EventArgs e)
{
    Label lblTime = FindControl("lblTime") as Label;
    lblTime.Text = DateTime.Now.ToLongTimeString();
}

```



Gotcha!

This is a very important gotcha! The symptoms vary for this gotcha, but in general, if something doesn't show up on the page, disappears from the page in response to user action or bound data doesn't show up, you should double-check your AJAX settings. Make sure that the control that triggers the update and the updated controls are included in the list of settings.

This is more of a designed behavior than a gotcha because we want to send only certain portions of the page to the server. If we forget to include one of those pieces by forgetting a setting, the Ajax Manager will do exactly what we tell it to!

Other Properties

RadAjaxManager and RadAjaxPanel have a properties available at runtime only:

- RadAjaxManager and RadAjaxPanel can both indicate if they are in the middle of an AJAX request using the **IsAjaxRequest** property. For example, if you dropped this line of code into the Getting Started calendar

UI for ASP.NET AJAX

SelectionChanged event handlers, one for the full postback and one for the AJAX-enabled version, the alert dialog would correctly reflect the state of each.

[VB] Detecting AJAX Requests

```
RadAjaxManager1.Alert(RadAjaxManager1.IsAjaxRequest.ToString())
```

[C#] Detecting AJAX Requests

```
RadAjaxManager1.Alert(RadAjaxManager1.IsAjaxRequest.ToString());
```

The screenshot below shows the user clicking on the AJAX-enabled calendar and the alert shows that IsAjaxRequest is true:



8/15/2008



8/7/2008



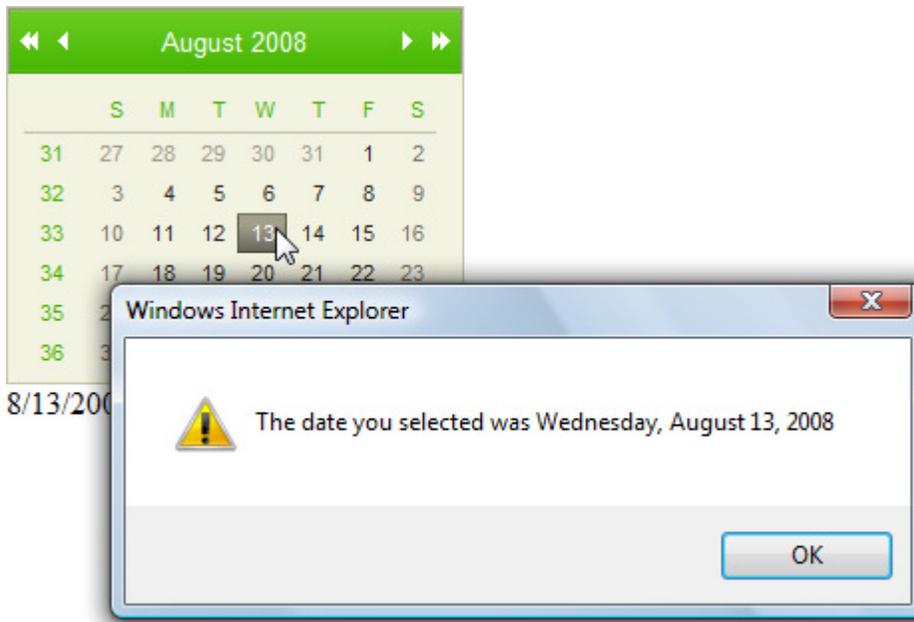
- You can define JavaScript that executes when a response returns to the browser by adding to the ResponseScripts collection:

[VB] Adding a Response Script

```
RadAjaxManager1.ResponseScripts.Add("alert('The date you selected was " + AjaxCalendar.SelectedDate.ToLongDateString() + "')");
```

[C#] Adding a Response Script

```
RadAjaxManager1.ResponseScripts.Add("alert('The date you selected was " + AjaxCalendar.SelectedDate.ToLongDateString() + "')");
```



Accessing RadAjaxManager From Any Page

If you want to access RadAjaxManager from a page that RadAjaxManager is not directly on, e.g. in a Content page or in a WebUserControl for example, you can use the RadAjaxManager.GetCurrent(Page) method.

[VB] Using the GetCurrent() Method

```
Imports Telerik.Web.UI
Namespace RadAjaxManagerMethods
Public Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
        RadAjaxManager.GetCurrent(Me).EnableAJAX = True
    End Sub
End Class
End Namespace
```

[C#] Using the GetCurrent() Method

```
using Telerik.Web.UI;
namespace RadAjaxManagerMethods
{
    public partial class _Default : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            RadAjaxManager.GetCurrent(this).EnableAJAX = true;
        }
    }
}
```

7.6 Client-Side Programming

Creating Custom Ajax Requests

Using the Ajax Manager and Panel you can automatically update controls on the page, but what if you want to

AJAX-enable dynamically created? What if you want to AJAX-enable a control that is not designed with AJAX in mind? RadAjaxManager has a client method called `ajaxRequest()`, a unique method that bridges between client and server. You call `ajaxRequest()` on the client and it triggers an AJAX update that's handled on the server. You can pass any arbitrary arguments as parameters. This very powerful method provides flexibility for any AJAX task you could think up that's not already covered by existing controls and event handlers. You can even add AJAX functionality to controls that are not designed to be AJAX enabled.



For best Winforms-like performance you'll want to dispatch as many tasks as possible to the client. There are some tasks that must be done back at the server, i.e. communication and networking related jobs like database update or calculation intensive jobs where the richer .NET libraries are available. That said, the line between the client and server is progressively blurred with mechanism like `ajaxRequest()`, the advent of client data sources and richer client libraries.

In its simplest form, you have a JavaScript function on the client that calls `ajaxRequest()` with no arguments. Calling the RadAjaxManager client method `ajaxRequest()` on the client causes the RadAjaxManager `OnAjaxRequest` event to fire on the server. Both the client `ajaxRequest()` and the server `OnAjaxRequest` event need to be present to make the conversation between client and server happen.

The screenshot below shows an example with all the pieces needed to communicate from client to server. A standard ASP Button with a client onclick event handler points to "myFunction()". `myFunction()` gets a reference to the ajax manager and calls the `ajaxRequest()` client-side API function. RadAjaxManager also has a property `OnAjaxRequest` which fires in response to the client `ajaxRequest()` client method. The RadAjaxManager `OnAjaxRequest` property points to a handler in the code-behind that performs the actual work that responds to the request. The sequence of events is:

- The user clicks "Button1" and fires the onclick client event.
- "myFunction()" method runs in response to the client onclick event. This method gets a reference to the RadAjaxManager and fires the RadAjaxManager `ajaxRequest()` client API method.
- The `OnAjaxRequest` event fires on the server.
- The `OnAjaxRequest` event handler "RadAjaxManager1_AjaxRequest" runs in response to the event.



Playing with a full deck

For a slightly more involved example, this next demo project shows how to pass a parameter and respond on the server. The project actually injects HTML to the page which in turn calls the JavaScript and again fires the `ajaxRequest()`. The first version of this project displays a playing card on the page. When the user clicks the card, another random card is created and rendered in the page.



Old path: <http://localhost:1908/Images/1.png>

New path: [Images/6.png](#)



You can find the complete source for this project at:
 \VS Projects\AJAX\ClientAPI1

1. Create a new web application and add a ScriptManager component to the page.

UI for ASP.NET AJAX

2. Add a RadAjaxManager control after the ScriptManager on the page.
3. In the Solution Explorer, create a new \Images directory.
4. Copy the 52 image files from the \VS Projects\Images\Playing Cards directory to your project's \images directory. The images are named "1.png" through "52.png".
5. Add an HTML <div> element with id "cardDeckDiv", and runat attribute set to "server". Inside the div, place a standard HTML tag with id "card" and src pointing to "images/1.png". You should include an "onclick" event handler that runs a JavaScript function called "deal();" (we will write code for "deal" momentarily). The markup should look something like the example below.

[ASP.NET] Adding the Image

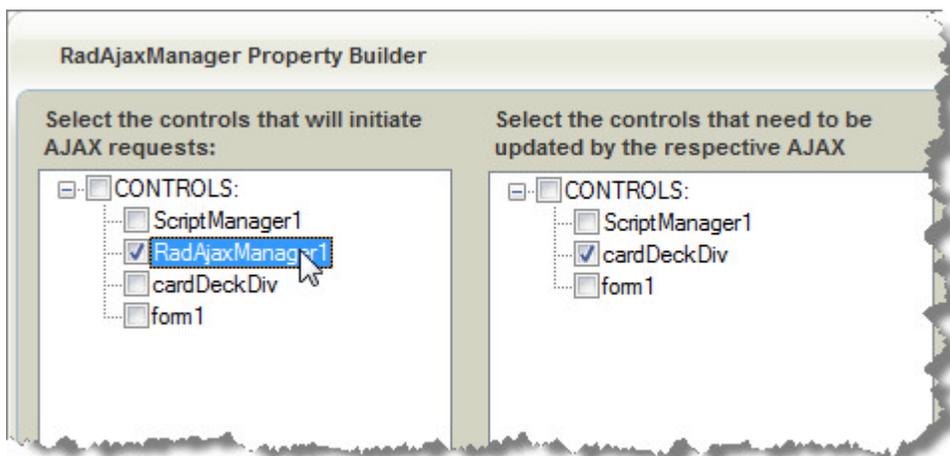
```
<div id="cardDeckDiv" runat="server">  
    
</div>
```

6. Just below the <body> tag add the JavaScript deal() function. The function first gets a reference to the "card" element and gets the source path for it. Then the "deal()" function gets a reference to the RadAjaxManager client object and calls ajaxRequest(), passing the source path.

[JavaScript] Calling ajaxRequest()

```
function deal()  
{  
  // get reference to the card div and extract source path  
  var card = $get("card");  
  var src = card.src;  
  
  // get a reference to the RadAjaxManager client object  
  var ajaxManager = $find("<%=RadAjaxManager1.ClientID %>");  
  // call ajaxRequest and pass the source path  
  ajaxManager.ajaxRequest(src);  
}
```

7. In the designer, use the RadAjaxManager Smart Tag **Configure Ajax Manager** option. In this case, the RadAjaxManager is actually the initiating control here. Select the RadAjaxManager checkbox and select "cardDeckDiv" as the updated control.



If you look in the source view for the page, the markup should look something like the example below:

[ASP.NET] The Full Markup

```

<script type="text/javascript">
function deal()
{
    // get reference to the card div and extract source path
    var card = $get("card");
    var src = card.src;

    // get a reference to the RadAjaxManager client object
    var ajaxManager = $find("<%=RadAjaxManager1.ClientID %>");
    // call ajaxRequest and pass the source path
    ajaxManager.ajaxRequest(src);
}
</script>
<form id="form1" runat="server">
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
<telerik:RadAjaxManager ID="RadAjaxManager1" runat="server"
    OnAjaxRequest="RadAjaxManager1_AjaxRequest">
    <AjaxSettings>
        <telerik:AjaxSetting AjaxControlID="RadAjaxManager1">
            <UpdatedControls>
                <telerik:AjaxUpdatedControl ControlID="cardDeckDiv" />
            </UpdatedControls>
        </telerik:AjaxSetting>
    </AjaxSettings>
</telerik:RadAjaxManager>
<div id="cardDeckDiv" runat="server">
    
</div>
</form>

```

8. In the Properties window select the events icon () and create an event handler for the **OnAjaxRequest** server-side event.
- In the server OnAjaxRequest handler, first use the **System.Random** object to get a number between 1..52.
 - Add references to **System.Web.UI.HtmlControls** and **System.Web.UI.WebControls** in your "Imports" (VB) or "uses" (C#) section of code if they don't already exist there.
 - Create an **HtmlImage** object.
 - Construct the path for the playing card graphic and assign it to the **HtmlImage** **src** property.
 - Make the **ID** property "card" (this is the identifier that the client code will expect to find)
 - Set the "onclick" client event using the **Attributes** collection of the image object.
 - Clear and add the **HtmlImage** to the **Controls** collection of **cardDeckDiv**.
 - Create a **Literal** control and set its text to display the passed in argument and the new path. This literal also gets added to the **cardDeckDiv** **Controls** collection.

The resulting code should look like the example below:

[VB] Handling the Ajax Request

```

Public Partial Class _Default
    Inherits System.Web.UI.Page
    Protected Sub RadAjaxManager1_AjaxRequest(ByVal sender As Object, ByVal e As

```

```
Telerik.Web.UI.AjaxRequestEventArgs)
    ' get a random card number from the "deck" of 52
    Dim random As New Random()
    Dim nextCard As Integer = random.[Next](1, 52)
    ' create the new card
    Dim image As New HtmlImage()
    image.Src = "Images/" + nextCard.ToString() + ".png"
    image.ID = "card"
    image.Attributes("onclick") = "deal()"
    cardDeckDiv.Controls.Clear()
    cardDeckDiv.Controls.Add(image)
    ' display the passed in argument and the new constructed paths
    Dim label As New Literal()
    label.Text = "<br /><br /><b>Old path:</b> " + e.Argument + "<br /><b>New path:</b>"
image.Src
    cardDeckDiv.Controls.Add(label)
End Sub
End Class
```

[C#] Handling the Ajax Request

```
public partial class _Default : System.Web.UI.Page
{
    protected void RadAjaxManager1_AjaxRequest(object sender,
        Telerik.Web.UI.AjaxRequestEventArgs e)
    {
        // get a random card number from the "deck" of 52
        Random random = new Random();
        int nextCard = random.Next(1, 52);
        // create the new card
        HtmlImage image = new HtmlImage();
        image.Src = "Images/" + nextCard.ToString() + ".png";
        image.ID = "card";
        image.Attributes["onclick"] = "deal()";
        cardDeckDiv.Controls.Clear();
        cardDeckDiv.Controls.Add(image);
        // display the passed in argument and the new constructed paths
        Literal label = new Literal();
        label.Text =
            "<br /><br /><b>Old path:</b> " + e.Argument +
            "<br /><b>New path:</b> " + image.Src;
        cardDeckDiv.Controls.Add(label);
    }
}
```

9. Press Ctl-F5 to run the example.

Events

RadAjaxPanel and RadAjaxManager both have two events. **OnRequestStart** fires just before the request is sent to the server. You can examine or alter the arguments sent to the request or you can cancel the request altogether. When the page has been updated by the AJAX request the **OnResponseEnd** event fires. Used together, these two events can be used to log metrics on the performance of each request or to set and restore state. Some state related tasks you might perform are:

- Setting the mouse cursor to a "wait" graphic, then back to its default.
- Disabling or hiding certain controls during a request, then re-enabling or making visible afterward.

- Displaying an animated control to indicate processing is taking place, then displaying an un-animated version of the control when the response returns. This also can typically be done using a RadLoadingPanel.

Both events have the same signature as other events in the client API:

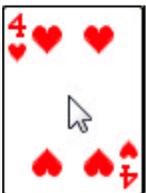
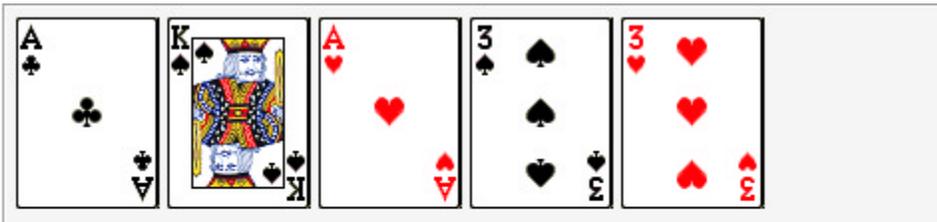
[JavaScript] OnRequestStart, OnResponseEnd Parameters

```
function RequestStart(sender, args)
{
    //..
}
```

In this context "sender" is the RadAjaxManager object. Args has some significant methods to control the current event:

- `get_eventTargetElement()`, `set_eventTargetElement()`: gets or sets the client object that raised the AJAX request.
- `get_eventTarget()`, `set_eventTarget()`: gets or sets the UniqueID of the element that raised the AJAX request.
- `get_enableAjax()`, `set_enableAjax()`: gets or sets if an AJAX request is to be performed at all. So, based on the initiating control or an updated control, you can actually determine if a particular request should instead become a standard postback. **Note:** You'll see how to get the list of updated controls from `AjaxSettings` in the upcoming **Properties** section of this chapter.

In this next sample we'll use both events to set the mouse cursor to reflect a busy state, and we will extend the `ajaxRequest` to pass multiple parameters. The sample uses the previous project as a basis, but adds the clicked card to a second div that contains five `` tags. The effect is that the cards are "dealt" into a row.



You can find the complete source for this project at:
 \VS Projects\AJAX\ClientAPI2.

1. Start with the previous project (or a copy).
2. In the ASP.NET markup, just above "cardDeckDiv" add:
 - A hidden input field tab with id "Index" and value set to "1".
 - Another div with id "discardPileDiv". Make sure it is marked with `runat="server"`. You can modify the style attribute to suit your taste or copy it from the fragment below.
 - Within the "discardPileDiv", add five `` tags with id's "Span1" through "Span5?". The server code is expecting this specific naming convention. Again, make sure it is marked with `runat="server"`.

The *Index* field will be used to track a 'current span' so that as the cards are dealt out, they are displayed in the spans from left to right. After the user deals out five cards to spans 1 through 5, the cycle starts over again at the first span.

The completed markup should look like the example below:

[ASP.NET] Adding Input, New Div and Span Tags

```
<input id="Index" value="1" type="hidden" />
<div id="discardPileDiv" runat="server"
  style="border-style: groove; border-width: 2px; height: 100px;background-color:White
  margin:5px;padding:5px">
  <span id="Span1" runat="server"></span>
  <span id="Span2" runat="server"></span>
  <span id="Span3" runat="server"></span>
  <span id="Span4" runat="server"></span>
  <span id="Span5" runat="server"></span>
</div><br />
```

3. Add "discardPileDiv" to the updated controls in the RadAjaxManager settings. Also, add ClientEvents properties so that OnRequestStart is "RequestStart" and OnResponseEnd is "ResponseEnd". You can do this in the Properties window or just add it directly to the markup as shown below:

[ASP.NET] Adding to the Updated Controls and ClientEvents

```
<telerik:RadAjaxManager ID="RadAjaxManager1" runat="server"
  onajaxrequest="RadAjaxManager1_AjaxRequest">
  <AjaxSettings>
    <telerik:AjaxSetting AjaxControlID="RadAjaxManager1">
      <UpdatedControls>
        <telerik:AjaxUpdatedControl ControlID="discardPileDiv" />
        <telerik:AjaxUpdatedControl ControlID="cardDeckDiv" />
      </UpdatedControls>
    </telerik:AjaxSetting>
  </AjaxSettings>
  <ClientEvents OnRequestStart="RequestStart" OnResponseEnd="ResponseEnd" />
</telerik:RadAjaxManager>
```

4. Add client event handlers to your JavaScript for OnRequestStart and OnResponseEnd events. On the request start, set the cursor to show that the application is busy using the cursor style "wait", then on the response end set the cursor back using the cursor style "default".

[JavaScript] Show 'Busy' Mouse Cursor

```
<script type="text/javascript" >
  //...

  function RequestStart(sender, args)
  {
    document.body.style.cursor = "wait";
  }

  function ResponseEnd(sender, args)
  {
    document.body.style.cursor = "default";
  }
</script>
```

 You can use this same pattern to set and restore different kinds of state on the client: disabling and enabling controls, starting and stopping timers or creating and destroying resources.

5. Augment the deal() function:

- Retrieve a reference to the "card" div element. From the card element, store the "src" property. "src" contains the path of each card image.
- Retrieve the Index hidden field value.
- Populate a new "args" variable that contains the concatenation of the "src" and "Index" joined with a "&" delimiter. *"src" will contain the path to the clicked on image and "Index" will indicate the current span to display the next card in.*

 Instead of passing a single argument to ajaxRequest() directly, you can pass multiple arguments joined by a delimiter as a single string. On the server call the String Split() method to convert the single string to an array for easier use.

- Increment the Index value. You can use the JavaScript parseInt() function to explicitly convert the Index string value to a numeric.
- Pass "args" to the ajaxRequest() method.

[JavaScript] Implementing the deal() function

```
<script type="text/javascript" >
function deal()
{
    // get reference to the card div and extract source path
    var card = $get("card");
    var src = card.src;
    // get the Index hidden field
    var Index = $get("Index");

    // construct arguments to pass to server:
    // use "&" as delimiter. Pass the image path
    // and Index to current span.
    var args = src + "&" + Index.value.toString();
    // increment the index between 1..5
    if (Index.value >= 5)
    {
        Index.value = 1;
    }
    else
    {
        Index.value = parseInt(Index.value) + 1;
    }
    // get ajax manager and kick off ajax request passing arguments
    var ajaxManager = $find("<%=RadAjaxManager1.ClientID %>");
    ajaxManager.ajaxRequest(args);
}

//. . .
</script>
```

6. On the server, add the following namespaces to the "Imports" (VB) or "uses" (C#) section of the code:

[VB] Adding Namespaces

```
' supports generic List
Imports System.Collections.Generic
' supports Control
Imports System.Web.UI
```

[C#] Adding Namespaces

```
using System.Collections.Generic; // supports generic List
using System.Web.UI; // supports Control
```

7. Create a generic List property to contain image paths of cards that have been "dealt". *The generic list of cards will be used to regenerate the card controls on each Page_Load.*

 *Dynamically created controls have to be re-created each page cycle.* Even though we are using AJAX and only partially updating the page, the entire page life-cycle still occurs: the Page_Load still fires. For more detail, see the upcoming Page Life Cycle section.

[VB] Create Generic List Property to Store Card Image Paths

```
' stores a list of card number "dealt" out
Const CardPathsKey As String = "CardPathsKey"
Private Property CardPaths() As List(Of String)
    Get
        Return TryCast(ViewState(CardPathsKey), List(Of String))
    End Get
    Set
        ViewState(CardPathsKey) = value
    End Set
End Property
```

[C#] Create Generic List Property to Store Card Image Paths

```
// stores a list of card number "dealt" out
const string CardPathsKey = "CardPathsKey";
private List<string> CardPaths
{
    get { return ViewState[CardPathsKey] as List<string>; }
    set { ViewState[CardPathsKey] = value; }
}
```

8. Add utility methods to be used later in the Page_Load and OnAjaxRequest event handlers:
 - GetSpan() locates and returns an HtmlGenericControl that represents one of the five tags.
 - AddCard() creates an HTMLImage control, sets the appropriate image path for the card number and adds the HTMLImage control to a given parent.

[VB] Utility Methods

```
' retrieve reference to a given span
Private Function GetSpan(ByVal index As Integer) As HtmlGenericControl
    Return TryCast(FindControl("Span" + index), HtmlGenericControl)
End Function
' create a card image and add it to a parent container
Private Function AddCard(ByVal parent As Control, ByVal CardPath As String) As HtmlImage
    Dim image As HtmlImage = Nothing
    If Not [String].IsNullOrEmpty(CardPath) Then
        image = New HtmlImage()
```

```

        image.Src = CardPath
        parent.Controls.Clear()
        parent.Controls.Add(image)
    End If
    Return image
End Function

[C#] Utility Methods
// retrieve reference to a given span
private HtmlGenericControl GetSpan(int index)
{
    return FindControl("Span" + index) as HtmlGenericControl;
}
// create a card image and add it to a parent container
private HtmlImage AddCard(Control parent, string CardPath)
{
    HtmlImage image = null;
    if (!String.IsNullOrEmpty(CardPath))
    {
        image = new HtmlImage();
        image.Src = CardPath;
        parent.Controls.Clear();
        parent.Controls.Add(image);
    }
    return image;
}

```

9. The Page_Load takes care of creating the generic list of card image paths on the first run of the page. The Page_Load is also tasked with re-creating user controls. The five spans containing the "dealt" cards will "disappear" if not recreated on every page load.

[VB] Handling the Page_Load

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' create the generic list
    If Not IsPostBack Then
        CardPaths = New List(Of String)()
    End If
    ' recreate cards on each postback and display
    ' inside appropriate spans
    Dim index As Integer = 1
    For Each CardPath As String In CardPaths
        AddCard(GetSpan(index), CardPath)
        System.Math.Max(System.Threading.Interlocked.Increment(index), index - 1)
    Next
End Sub

```

[C#] Handling the Page_Load

```

protected void Page_Load(object sender, EventArgs e)
{
    // create the generic list
    if (!IsPostBack)
    {
        CardPaths = new List<string>();
    }
}

```

```
// recreate cards on each postback and display
// inside appropriate spans
int index = 1;
foreach (string CardPath in CardPaths)
{
    AddCard(GetSpan(index), CardPath);
    index++;
}
}
```

10. The final task is to handle the OnAjaxRequest event using the new code shown below.
- Use the String Split() method to convert the AjaxRequestEventArgs.Argument parameter into an array.
 - From the array extract the card image path and the index of the current span to their own variables.
 - Call the AddCard() utility method to add the latest card to the current span and also add the card path to the generic list.
 - Create a new random card and display it in the "cardDeckDiv".

[VB] Handling the OnAjaxRequest Event

```
Protected Sub RadAjaxManager1_AjaxRequest(ByVal sender As Object, ByVal e As
Telerik.Web.UI.AjaxRequestEventArgs)
    ' retrieve arguments sent from client and convert
    ' to array for easier processing
    Dim args As String() = e.Argument.Split("&"C)
    ' store the first argument parameter
    ' (the src property of the clicked-on card)
    Dim CardPath As String = args(0)
    ' store the span index where the next card image will be displayed
    Dim index As Integer = Integer.Parse(args(1))
    ' create and add a card image to the current span:
    ' if the generic list of card image paths isn't populated
    ' yet, add to it, otherwise, re-use the list.
    AddCard(GetSpan(index), CardPath)
    If CardPaths.Count < 5 Then
        CardPaths.Add(CardPath)
    Else
        CardPaths(index - 1) = CardPath
    End If
    ' create the new card
    Dim nextCard As Integer = New Random().[Next](1, 52)
    Dim newCardPath As String = "Images/" + nextCard.ToString() + ".png"
    Dim newCardImage As HtmlImage = AddCard(cardDeckDiv, newCardPath)
    newCardImage.ID = "card"
    newCardImage.Src = newCardPath
    newCardImage.Attributes("onclick") = "deal()"
End Sub
```

[C#] Handling the OnAjaxRequest Event

```
protected void RadAjaxManager1_AjaxRequest(object sender,
Telerik.Web.UI.AjaxRequestEventArgs e)
{
    // retrieve arguments sent from client and convert
    // to array for easier processing
    string[] args = e.Argument.Split('&');
}
```

```

// store the first argument parameter
// (the src property of the clicked-on card)
string CardPath = args[0];
// store the span index where the next card image will be displayed
int index = int.Parse(args[1]);
// create and add a card image to the current span:
// if the generic list of card image paths isn't populated
// yet, add to it, otherwise, re-use the list.
AddCard(GetSpan(index), CardPath);
if (CardPaths.Count < 5)
    CardPaths.Add(CardPath);
else
    CardPaths[index - 1] = CardPath;
// create the new card
int nextCard = new Random().Next(1, 52);
string newCardPath = "Images/" + nextCard.ToString() + ".png";
HtmlImage newCardImage = AddCard(cardDeckDiv, newCardPath);
newCardImage.ID = "card";
newCardImage.Src = newCardPath;
newCardImage.Attributes["onclick"] = "deal()";
}

```

1. Press Ctl-F5 to run the application. Click the card enough times that the first five cards are shown and that additional cards are displayed starting from the left-most span.

Canceling AJAX Requests

What if one of your users goes berserk and starts clicking away like a madman at your UI, attempting to start requests before other requests have completed? The consequences in the previous example are not too serious, but in an application with longer running processes on the server and more complex interrelationships, it's better to control when requests are made to eliminate problems before they have opportunity to occur.

One way to regulate this behavior is by calling `set_cancel()` to abort further processing, including the AJAX request. If you track when you're currently processing an AJAX request, then you can cancel the request from the `OnRequestStart` client event.

1. Taking the previous project as a starting point, navigate to the markup for the default page. In the JavaScript, add a variable to track AJAX requests:

[JavaScript] Adding Variable to Track Ajax Requests

```

<script type="text/javascript" >
    var isAjaxActive;
    //...

```

2. In the `deal()` function, wrap the code there with an `if()` statement so that the code will not execute if another request is already in-progress.

[JavaScript] Wrapping deal() Code

```

function deal()
{
    if (!isAjaxActive)
    {
        //...
    }
}

```

3. In the `RequestStart()` function, if a request is in-process, then simply return `false` to cancel the new

request. If there is no request currently processing, set the `isAjaxActive` flag to true. In the `ResponseEnd()` function, simply set `isAjaxActive` to false, indicating there are no AJAX requests currently processing.

[JavaScript] Handling the `OnRequestStart` and `OnResponseEnd` Client Events

```
function RequestStart(sender, args)
{
    if (isAjaxActive)
    {
        args.set_cancel(true);
    }
    else
    {
        isAjaxActive = true;
        document.body.style.cursor = "wait";
    }
}
function ResponseEnd(sender, args)
{
    isAjaxActive = false;
    document.body.style.cursor = "default";
}
```

Properties

The client API as usual lets you get and set important `RadAjaxManager` properties. For example, you can toggle if AJAX is enabled at all using `get_enableAJAX()/ set_enableAJAX()` methods. You can also get the complete list of `AjaxSettings`.

The example below is added to the `OnRequestStart` client event handler from our previous project, but similar code could be placed in any client event.

- "sender" in this example is a reference to the `RadAjaxManager` object.
- Call `get_ajaxSettings()` to retrieve an array of objects that represents the Ajax Manager's current configuration. Iterate each of the settings and retrieve the initiating control id and yet another array of objects that represent updated controls.
- Iterate the `UpdatedControls` array and collect the control id for each. While iterating these arrays, append to strings that keep track of the initiating control and the updated controls for each initiating control.
- Display an alert that displays the collected information:

[ASP.NET] Retrieving `AjaxSettings` on the Client

```
function RequestStart(sender, args)
{
    var settings = sender.get_ajaxSettings();
    var settingList = '';
    for(setting in settings)
    {
        var initControlID = settings[setting].InitControlID;
        var updatedControls = settings[setting].UpdatedControls;
        var controllist = '';
        for(control in updatedControls)
        {
            controllist += ' ' + updatedControls[control].ControlID;
        }
        settingList += '\nInitiated by: ' + initControlID +
```

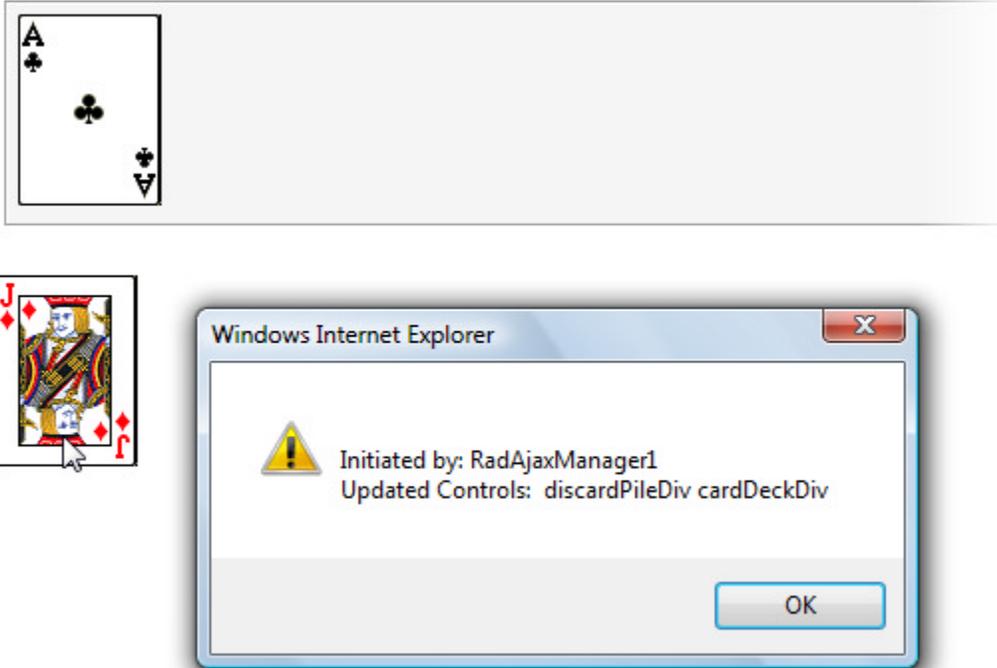
```

        '\nUpdated Controls: ' + controlList;
    }
    alert(settingList);

    //...
}

```

The screenshot below shows the output. In this case, we only have one initiating control, the RadAjaxManager itself, and it updates the two div elements that contain the card images.



7.7 Page vs MasterPage vs UserControl

One of the issues that will impact your design of an AJAX-enabled application is the scope of the AJAX. That is, will the entire page, including navigational controls, be AJAX-enabled, or just items within the page. Many sites use standard postback to navigate between pages while the contents of the page may be heavily "ajaxified". These kinds of sites are relatively straightforward and let you use the built-in capabilities of the navigation controls, so that a RadMenu, for example, can point to other pages using the RadMenuItem.NavigateUrl property. Using the NavigateUrl automatically causes a full postback because you are loading the entire page from scratch.

So how do we AJAX-enable an entire page to replicate that WinForms look-and-feel, but still retain the development advantage of working on one "page" at a time?

What about using ASP.NET 2.0 MasterPage and Content pages? This seems like a clear contender because the visual metaphor is that the MasterPage is a container for place holders where Content pages are injected. Couldn't you put a RadAjaxPanel on the MasterPage surrounding the content place holder to AJAX-enable just the Content page? Nope. When you navigate to a Content page, the MasterPage is merged and again, the entire page is loaded and state is not persisted between Content pages. The MasterPage can be thought of as a template -- it is not the same instance in two different Content pages. Navigating between Content pages in this context works the same as navigating between standard aspx pages and so doesn't allow an opportunity to AJAX-enable the entire page. You can take a look at the "MasterPages" project (in \VS Projects\Ajax\MasterPages) to prove to yourself how Master/Content pages operate.

To AJAX-enable the entire page, including the navigation elements, creating UserControls dynamically is a

workable approach.

7.8 Page Lifecycle

Page Life Cycle Basics

Before we tackle creating pages that swap out UserControls dynamically, you should have a basic understanding of the ASP.NET 2.0 **page lifecycle** so that controls don't seem to appear and disappear like magic. The page lifecycle is a series of stages that the page progresses through when converting your markup and code-behind to produce the final rendered page output. This page lifecycle description will be simplified to focus on stages relating to loading and handling state for controls.

1. **Instantiation:** A class is automatically generated using the ASP.NET markup that declaratively defines your page. This class defines a **control tree**, a hierarchical structure starting with the HTML page as the root and other elements (LiteralControls, WebControls, etc) arranged underneath. **Note:** When you add controls dynamically, you're still adding to this control tree.
2. **Initialization:** Controls are instantiated and their initial properties are set. PreInit and Init events fire for the page and for server controls at this stage. During the Init event, `TrackViewState()` is called so that changes to controls are saved in the "view state".

You may typically work with `ViewState` as a property of the `Page`, but `ViewState` is a protected property of `Control` and is at work in all `WebControls`. Each server control can maintain its own state across postbacks using `ViewState`. `ViewState` rendered on the page is actually a standard HTML hidden type of input tag named "`__VIEWSTATE`" where the value attribute is the encoded state information for the page (see example below).

```
<input id="__VIEWSTATE" type="hidden" value="/wEPDwUJNzgzNDMwNTMzZGR6/Texf1rWyHeYqWXYXaks4wo
```

In the code-behind, view state is represented as a **StateBag** class -- a dictionary with the additional ability of knowing when name/value pairs in the dictionary change. `StateBag` has a **IsItemDirty** boolean property that tracks state changes and allow for efficient restoration of a minimal number of controls. Controls that don't change from their declared definition do not persist `ViewState`.

 In related news, you may see a hidden field `__EVENTVALIDATION` that is used to verify that the controls on the server match the controls rendered on the client. This feature is intended to prevent someone from injecting malicious JavaScript to your page. You may see an error "...Invalid postback or callback argument. Event validation is enabled using <pages enableEventValidation='true'/>..." if a user submits a form before it is completely rendered and the `__EVENTVALIDATION` field is not yet present.

3. **Load View State:** View state information saved in previous incarnations of the page are restored here.
4. **Load:** Page and control Load events are fired.
5. **Save View State:** State information is persisted.



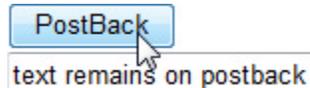
You can find the complete source for this project at:

`\VS Projects\Ajax\PageLifeCycle`. This project simply shows debug output for the page methods and events described above.

Now imagine that you create a control, a standard `TextBox` for example, in reaction to a button click. The `TextBox` is added to the control tree and you see the `TextBox` on the rendered page. Now you click a second button that causes the page to postback and shazaam! the `TextBox` disappears. What happened? Starting at the Instantiation stage, the control tree was created from the markup. The `TextBox` wasn't defined in the markup and the the second button had no code-behind to create the `TextBox`, so on postback, the `TextBox` simply isn't there to display. This brings us to the central truth about using dynamic controls -- dynamically added controls must be recreated on every postback.

Dynamically Added Controls

So if I recreate controls every postback, how do I get changes that the user makes to a control "stick"? If you add controls after view state tracking is on but before values are programmatically added to the control, this behavior comes along for the ride automatically. In fact, even if you add controls at any time up to the actual rendering of the control, ASP.NET plays "catch up" for the control (and any child controls) so that initialization, load view state and load stages still occur. Here's a very brief example that loads a new TextBox on every post back and retains the state of whatever the user enters. The page has a standard Button declared in markup on the page to trigger postbacks, but otherwise the web page is the Visual Studio default. The user can click the postback button multiple times and change the text as well. The text entered or modified in the TextBox remains after the postback.



[VB] Adding Controls in Page_Load

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Me.form1.Controls.Add(New TextBox())
End Sub
```

[C#] Adding Controls in Page_Load

```
protected void Page_Load(object sender, EventArgs e)
{
    this.form1.Controls.Add(new TextBox());
}
```

Just to recap the action so far:

- Control properties are first set to their declared values from the markup. The page has only the single Button control with its declared defaults.
- Each control's `TrackViewState()` method is called during initialization.
- Each control's `LoadViewState()` retrieves state information that was flagged "dirty" in the previous request. The retrieved state information is added back to the control's `StateBags`. `StateBags` are tracking view state at this point, so this state information is flagged "dirty" so that it will be available on the next request. This doesn't have much effect in our "dynamically added controls" scenario because we haven't added the TextBox yet.
- `Page_Load` fires. When the TextBox is added to the form `Controls` array, the control plays "catch up" and so has a chance to load its view state.

 Be aware that even when an action on the page is AJAX-enabled, the entire page lifecycle still executes on the server, albeit with an smaller amount of view state information.

Assigning IDs

How does ASP.NET know what control to update with a given piece of the ViewState? The code above has an omission that happens to work out ok. In the example above ASP.NET automatically assigns the name "ctl02" to the TextBox. In the example below we explicitly assign a random ID to the TextBox for each page load and the result is that state is not retained. ASP.NET sees a different TextBox on each page load:

[VB] Assigning a random ID

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim random As New Random()
    Dim textBox As New TextBox()
    textBox.ID = "myTextBox_" + random.[Next]().ToString()
```

```
Me.form1.Controls.Add(textBox)
End Sub
```

[C#] Assigning a random ID

```
protected void Page_Load(object sender, EventArgs e)
{
    Random random = new Random();
    TextBox textBox = new TextBox();
    textBox.ID = "myTextBox_" + random.Next().ToString();
    this.form1.Controls.Add(textBox);
}
```

If you change the code above and assign the TextBox the same ID value every time, the TextBox will return to its former behavior of automatically persisting user changes.



You can find the complete source for this project at:
\\VS Projects\Ajax\PageLifeCycle2

7.9 Dynamic User Controls for Ajax-Enabling Entire Page

Armed with a basic notion of how the ASP.NET Page Lifecycle works, we can begin to build an interface with multiple user controls that are swapped out based on user selection in a navigation control. First we'll start with a full page refresh version, then add AJAX capability and refine it from there.

Dynamic User Controls With Full Post Back

The first example will have a RadTabStrip on the default page and two UserControl that are loaded at runtime to the default page based on the currently selected tab. Each control will contain a button and a TextBox so you can test how the state is maintained.



You can find the complete source for this project at:
\\VS Projects\Ajax\DynamicControls1

1. Create a new web application and add a ScriptManager component to the default page.
2. Add a RadTabStrip to the default page. Configure two tabs with **Text** property "Page 1" and "Page 2" respectively and **Value** property "WebUserControl1.ascx" and "WebUserControl2.ascx" respectively. Set the SelectedIndex property to "0" so that the first tab is automatically selected. The markup should look something like the example below:

[ASP.NET] RadTabStrip Markup

```
<telerik:RadTabStrip ID="RadTabStrip1" runat="server" SelectedIndex="0">
  <Tabs>
    <telerik:RadTab Text="Page One" Value="WebUserControl1.ascx" Selected="True">
    </telerik:RadTab>
    <telerik:RadTab Text="Page Two" Value="WebUserControl2.ascx">
    </telerik:RadTab>
  </Tabs>
</telerik:RadTabStrip>
```

3. In the Solution Explorer, add two Web User Control items:
 - o Right-click the project
 - o Select **Add | New Item** from the context menu

- Select **Web User Control** in the Add New Item dialog
- Click the **Add** button.

Leave the default names for each: they should be named "WebUserControl1" and "WebUserControl2" to match the tab Value properties.

4. Navigate to the design view for WebUserControl1 and enter the text "Page 1", a standard ASP Button and a standard ASP TextBox.
5. Navigate to the design view for WebUserControl2 and enter the text "Page 2", a standard ASP Button and a standard ASP TextBox.
6. With the RadTabStrip selected in design view, click the Events button (🔗) Properties Window. Double-click the **TabClick** event to create an event handler. Do not enter any code for this event handler. It just needs to be present to trigger a postback but the logic will be housed in the page_load event.
7. In the Page_Load event handler, add the code below.
 - Each tab value has the path to the corresponding user control.
 - The Page_Load event first uses that path to load the control.
 - The same path is used as the control ID.
 - The control is added to the web form's Controls collection.

📌 Note that using the path as an ID directly could backfire if your user control was in a different directory and contained problem "\" and "-" characters. We will address this issue later in this chapter.

[VB] Loading the Selected Control

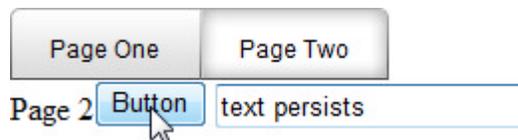
```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim control As Control = Page.LoadControl(RadTabStrip1.SelectedTab.Value)
    control.ID = RadTabStrip1.SelectedTab.Value
    Me.form1.Controls.Add(control)
End Sub
```

[C#] Loading the Selected Control

```
protected void Page_Load(object sender, EventArgs e)
{
    Control control = Page.LoadControl(RadTabStrip1.SelectedTab.Value);
    control.ID = RadTabStrip1.SelectedTab.Value;
    this.form1.Controls.Add(control);
}
```

8. Press Ctl-F5 to run the application.

Notice that as you click the tabs, the corresponding user control displays, and that if you enter text and click the button, the text will also persist.



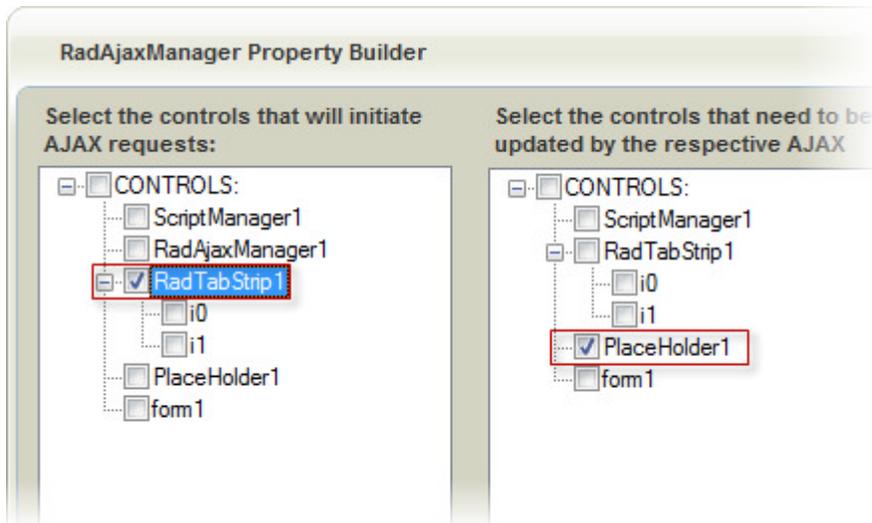
AJAX Enable

This next example simply AJAX-Enables the previous example.



You can find the complete source for this project at:
\\VS Projects\Ajax\DyanmicControls2

1. Starting with the previous project (or a copy), add a standard **PlaceHolder** control below the RadTabStrip. *Instead of using the form we use a PlaceHolder control to provide a little more flexibility and control on where we can place the user controls on the page.*
2. Select **Configure Ajax Manager** from the RadAjaxManager Smart Tag. Select the RadTabStrip check box as the initiating control and the PlaceHolder as the updated control.



3. Change the statement that added the dynamic control to the form's Controls array to use the PlaceHolder.

[VB] Add to PlaceHolder Controls

```
PlaceHolder1.Controls.Add(control)
```

[C#] Add to PlaceHolder Controls

```
PlaceHolder1.Controls.Add(control);
```

4. Press Ctl-F5 to run the application. The functionality should be the same as the previous application, except that now we have the benefits of AJAX performance and no postback flicker.

Initializing User Controls

You may have noticed when working with the previous examples that although state persists between postbacks, state does not persist between tab clicks. You can allow initialization of the user control through a database or other backing store. This next example extends the previous example.



You can find the complete source for this project at:
\\VS Projects\Ajax\DynamicControl3

1. Starting with the previous project (or a copy) create a new class file "IDynamicControl.cs" and add the code to define the IDynamicControl interface.
 - o The interface will have a single method "FirstLoad()" with no parameters.
 - o Each web user control will implement this interface and the page that loads the control will call the

method when the page is first loaded (!IsPostBack) and when the user clicks a tab.

- The method will *not* be fired when the postback is due to activity within the web user control.

[VB] Defining the IDynamicControl Interface

```
Public Interface IDynamicControl
    Sub FirstLoad()
```

2. End Interface

[C#] Defining the IDynamicControl Interface

```
public interface IDynamicControl
{
    void FirstLoad();
}
```

Modify the User Control

1. Navigate to the design-view for WebUserControl1, set the TextBox **AutoPostBack** property to True and create an event handler for the **TextChanged** event.
2. In the WebUserControl1 code-behind, add a property to store text entered by the user. The property stores the value in the Session. Notice that the key to Session is "TextInfoKey1" and will be unique.

[VB] Adding the TextInfo Property

```
Const TextInfoKey As String = "TextInfoKey1"
Private Property TextInfo() As String
    Get
        Return IIf(Session(TextInfoKey) = Nothing, "", Session(TextInfoKey).ToString())
    End Get
    Set
        Session(TextInfoKey) = value
    End Set
End Property
```

[C#] Adding the TextInfo Property

```
const string TextInfoKey = "TextInfoKey1";
private string TextInfo
{
    get { return Session[TextInfoKey] == null ? "" : Session[TextInfoKey].ToString(); }
    set { Session[TextInfoKey] = value; }
}
```

3. In the event handler for the TextChanged event, add the code below to store the text in the TextInfo property:

[VB] Saving to the TextInfo Property

```
Protected Sub TextBox1_TextChanged(ByVal sender As Object, ByVal e As EventArgs)
    Me.TextInfo = (TryCast(sender, TextBox)).Text
End Sub
```

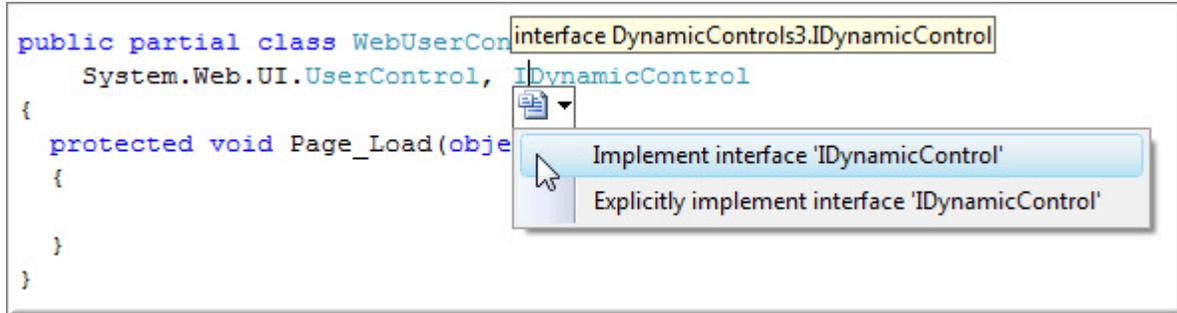
[C#] Saving to the TextInfo Property

```
protected void TextBox1_TextChanged(object sender, EventArgs e)
{
```

UI for ASP.NET AJAX

```
this.TextInfo = (sender as TextBox).Text;
}
```

4. In the code-behind for WebUserController1, add IDynamicControl to the class declaration. Click the indicator line just below "IDynamicControl" and select "Implement interface 'IDynamicControl'" (or right-click IDynamicControl and select Implement Interface from the context menu).



5. In the implementation for IDynamicControl, restore the previously saved TextInfo back to the TextBox:

[C#] Restoring Saved Text On Control's First Load

```
#region IDynamicControl Members
Public Sub FirstLoad()
    Me.TextBox1.Text = Me.TextInfo
End Sub
#End Region
```

[C#] Restoring Saved Text On Control's First Load

```
#region IDynamicControl Members
public void FirstLoad()
{
    this.TextBox1.Text = this.TextInfo;
}
#endregion
```

6. Repeat the steps for the "Modify the User Control" section on WebUserController2. The steps are the same except when you define the "TextInfoKey" constant it should be named "TextInfoKey2" so that it is unique.

Modify The Default Page Code-Behind

1. Create a utility method to convert user control paths to suitable ID names where slashes and tildes are removed.

[VB] Converting Control Paths to IDs

```
Private Function GetControlID(ByVal controlPath As String) As String
    Dim result As String = controlPath.Split("."C)(0)
    Return "uc_" + result.Replace("/", "").Replace("~", "")
End Function
```

[C#] Converting Control Paths to IDs

```
private string GetControlID(string controlPath)
{
    string result = controlPath.Split('.')[0];
    return "uc_" + result.Replace("/", "").Replace("~", "");
}
```

2. Move the control loading logic to its own method. There are several noteworthy changes in this method.
 1. The parameter list includes the path to the user control and a "isFirstLoad" boolean.
 2. The control is loaded to the page.
 3. The Placeholder controls are cleared and the user control is added to the control's collection. Note: Failing to clear the controls collection can lead to collisions of controls with the same id names.
 4. If this is "first load", that is, if the page itself is not a postback or if one of the tabs has just been clicked, provide an opportunity for the user control to initialize itself. You can place a RadAjaxManager.Alert() inside this If statement to get a feel for the circumstances that trigger this method.

[VB] Loading the User Control

```
Private Sub LoadUserControl(ByVal controlPath As String, ByVal isFirstLoad As Boolean)
    Dim control As Control = Page.LoadControl(controlPath)
    control.ID = GetControlID(controlPath)
    Placeholder1.Controls.Clear()
    Placeholder1.Controls.Add(control)
    If isFirstLoad Then
        RadAjaxManager1.Alert("First Load!")
        (TryCast(control, IDynamicControl)).FirstLoad()
    End If
End Sub
```

[C#] Loading the User Control

```
private void LoadUserControl(string controlPath, bool isFirstLoad)
{
    Control control = Page.LoadControl(controlPath);
    control.ID = GetControlID(controlPath);
    Placeholder1.Controls.Clear();
    Placeholder1.Controls.Add(control);
    if (isFirstLoad)
    {
        RadAjaxManager1.Alert("First Load!");
        (control as IDynamicControl).FirstLoad();
    }
}
```

3. Handle the Page_Load and TabClick events. Notice that the TabClick event handler always passes True to LoadUserControl().

[VB] Handle Page_Load and TabClick Events

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    LoadUserControl(RadTabStrip1.SelectedTab.Value, Not IsPostBack)
End Sub
Protected Sub RadTabStrip1_TabClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadTabStripEventArgs)
    LoadUserControl(e.Tab.Value, True)
End Sub
```

[C#] Handle Page_Load and TabClick Events

```
protected void Page_Load(object sender, EventArgs e)
{
    LoadUserControl(RadTabStrip1.SelectedTab.Value, !IsPostBack);
}
```

```
protected void RadTabStrip1_TabClick(object sender,
    Telerik.Web.UI.RadTabStripEventArgs e)
{
    LoadUserControl(e.Tab.Value, true);
}
```

4. Press Ctl-F5 to run the application. Experiment with changing tabs, saving values and clicking the button and see the effect on when the alert pops up.

Handling ViewState Conflicts

One last wrinkle that you may run into is that as your control tree becomes more complex, you may see this error intermittently:

"Failed to load viewstate. The control tree into which viewstate is being loaded must match the control tree that was used to save during the previous request. For example, when adding controls dynamically, the controls added during a post-back must match position of the controls added during initial request"

As the error indicates, this can happen if the expected viewstate doesn't match up with the controls received. So if a control, say the tree view in CategoriesTree, is present in both user controls, but is located in different places on the page the exception will be thrown.

You can shut off viewstate just prior to first loading the control to prevent the comparison between control trees that should not match at all. Then turn it back on so that the control may properly retain viewstate during the first load of the control.

You can reproduce the viewstate error if you take the last example and insert a standard DropDownList control between the button and textbox. Know that the viewstate is evaluated by position, so when we ASP.NET runs into the first control on the page, the button, it's where it should be. Next, it's expecting the text box, but instead it finds a drop down list and so the controls tree does not match and the exception is thrown.

An interesting side note, the internal rules that govern this comparison don't mind if a different control is inserted at the very *beginning* of the control tree. For example if you put in another button before all the other controls on the page, the exception is not thrown. In this case it appears the internal rules recognize that this is a completely new control.

For our purposes, we certainly don't want to play with the page layout just to make the error go away. Here we can shut off the EnableViewState for the user control just before it's added to the place holder's control array and then turn it back on so that it can retain any new changes that occur during our FirstLoad().



You can find the complete source for this project at:
\\VS Projects\Ajax\DynamicControls4

Here's another code example that shows how these pieces go together.

[VB] Handling ViewState

```
Imports System
Imports System.Web.UI
Namespace DynamicControls4
    Public Partial Class _Default
        Inherits System.Web.UI.Page
        #region properties
        ' store the last selected control for reload
        Private Const CurrentControlKey As String = "CurrentControlKey"
        Private Property CurrentControl() As String
            Get
                Return IIf(ViewState(CurrentControlKey) = Nothing, "", ViewState
(CurrentControlKey).ToString())
            End Get
        End Property
    End Class
End Namespace
```

```

End Get
Set
    ViewState(CurrentControlKey) = value
End Set
End Property
#End Region
#region private methods
Private Function LoadUserControl(ByVal parentControl As Control, ByVal newControlPath As
String, ByVal isFirstLoad As Boolean) As Control
    ' Load the control and set its id
    Dim control As Control = Page.LoadControl(newControlPath)
    control.ID = newControlPath
    ' the viewstate control will be out of sync with
    ' the previously loaded control. Temporarily shut off
    ' viewstate if this is the first load of the control
    If isFirstLoad Then
        control.EnableViewState = False
    End If
    ' add to the parent controls collection
    parentControl.Controls.Add(control)
    ' if this is the first load (first time the page is loaded or
    ' a new tab has been clicked) enable the viewstate again. Forgetting to
    ' reenale the viewstate will controls to be loaded only once. Then
    ' call the FirstLoad() method of the web user control for first time
    ' loading tasks.
    If isFirstLoad Then
        control.EnableViewState = True
        (TryCast(control, IDynamicControl)).FirstLoad()
    End If
    Return control
End Function
#End Region
#region page events
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' if this is the first load of the page,
    ' set the CurrentControl to the selected tab value
    If Not IsPostBack Then
        CurrentControl = RadTabStrip1.SelectedTab.Value
    End If
    Dim isNewControl As Boolean = Not CurrentControl.Equals(RadTabStrip1.SelectedTab.Value)
    If isNewControl Then
        CurrentControl = RadTabStrip1.SelectedTab.Value
    Else
        LoadUserControl(Placeholder1, CurrentControl, Not IsPostBack)
        ' new control, so wait for the tabclick to load it
        ' same control, reload it.
    End If
End Sub
Protected Sub RadTabStrip1_TabClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadTabStripEventArgs)
    ' this always is a first load
    LoadUserControl(Placeholder1, CurrentControl, True)
End Sub
#End Region
End Class

```

End Namespace

[C#] Handling ViewState

```
using System;
using System.Web.UI;
namespace DynamicControls4
{
    public partial class _Default : System.Web.UI.Page
    {
        #region properties
        // store the last selected control for reload
        private const string CurrentControlKey = "CurrentControlKey";
        private string CurrentControl
        {
            get
            {
                return ViewState[CurrentControlKey] == null ?
                    "" : ViewState[CurrentControlKey].ToString();
            }
            set
            {
                ViewState[CurrentControlKey] = value;
            }
        }
        #endregion
        #region private methods
        private Control LoadUserControl(Control parentControl,
            string newControlPath, bool isFirstLoad)
        {
            // Load the control and set its id
            Control control = Page.LoadControl(newControlPath);
            control.ID = newControlPath;
            // the viewstate control will be out of sync with
            // the previously loaded control. Temporarily shut off
            // viewstate if this is the first load of the control
            if (isFirstLoad)
            {
                control.EnableViewState = false;
            }
            // add to the parent controls collection
            parentControl.Controls.Add(control);
            // if this is the first load (first time the page is loaded or
            // a new tab has been clicked) enable the viewstate again. Forgetting to
            // reenale the viewstate will controls to be loaded only once. Then
            // call the FirstLoad() method of the web user control for first time
            // loading tasks.
            if (isFirstLoad)
            {
                control.EnableViewState = true;
                (control as IDynamicControl).FirstLoad();
            }
            return control;
        }
        #endregion
        #region page events
```

```

protected void Page_Load(object sender, EventArgs e)
{
    // if this is the first load of the page,
    // set the CurrentControl to the selected tab value
    if (!IsPostBack)
    {
        CurrentControl = RadTabStrip1.SelectedTab.Value;
    }
    bool isNewControl = !CurrentControl.Equals(RadTabStrip1.SelectedTab.Value);
    if (isNewControl)
        // new control, so wait for the tabclick to load it
        CurrentControl = RadTabStrip1.SelectedTab.Value;
    else
        // same control, reload it.
        LoadUserControl(Placeholder1, CurrentControl, !IsPostBack);
}
protected void RadTabStrip1_TabClick(object sender, Telerik.Web.UI.RadTabStripEventArgs
e)
{
    // this always is a first load
    LoadUserControl(Placeholder1, CurrentControl, true);
}
}
#endregion
}
}

```

7.10 Using RadAjaxManagerProxy

RadAjaxManager is only "one per customer" -- you can have only a single RadAjaxManager on the page. In more complex scenarios that involve containers of other controls, e.g. MasterPage/Content Page, how do you get design-time visibility to controls when the RadAjaxManager is perhaps on a MasterPage and the updated controls are on the content page? Likewise, how do you configure settings for controls located in a WebUserControl but the RadAjaxManager is on the page that loads the WebUserControl? You could of course add the settings programmatically (as shown in the Server-Side Programming section of this chapter), but if the controls in the Content Page or WebUserControl are not added dynamically, adding controls programmatically is unnecessary coding work and maintenance overhead.

RadAjaxManagerProxy is a stand-in that lets you configure AJAX settings in the designer and it can be present on as many design surfaces as necessary.

This next example demonstrates loading a single WebUserControl containing a RadCalendar and a CheckListBox that toggles calendar properties. The RadAjaxManager lives on the default.aspx page and the AJAX settings are housed in the WebUserControl inside the RadAjaxManagerProxy tag.



You can find the complete source for this project at:

\VS Projects\Ajax\RadAjaxManagerProxy

1. Create a new web application and add a ScriptManager component to the default page.
2. Add a **RadAjaxManager** and a standard ASP **PlaceHolder** to the default page. Just for fun, the project as a **RadFormDecorator** with **Skin** property set to "Sunset". *Notice at this point that the RadAjaxManager has no configuration settings. All settings will be setup in the user control.*
3. In the code-behind for the default page add the following code load the WebUserControl to the Placeholder on each Page_Load:

[VB] Handle the Page_Load Event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim control As Control = Me.LoadControl("WebUserControl1.ascx")
    control.ID = "uc_WebUserControl1"
    Placeholder1.Controls.Add(control)
End Sub
```

[C#] Handle the Page_Load Event

```
protected void Page_Load(object sender, EventArgs e)
{
    Control control = this.LoadControl("WebUserControl1.ascx");
    control.ID = "uc_WebUserControl1";
    Placeholder1.Controls.Add(control);
}
```

4. In the Solution Explorer add a WebUserControl item to the the project.

1. Add a **RadCalendar** to the user control design surface. Set the RadCalendar **Skin** property to "Sunset".
2. Add a standard ASP **CheckBoxList** to the user control design surface. Add three items with text "ShowColumnHeaders", "ShowOtherMonthDays" and "ShowRowHeaders". Set the **Selected** property of each item to **true**. From the Properties Window Event () tab, double-click the **OnSelectedIndexChanged** event to create an event handler. In the event handler add the code below:

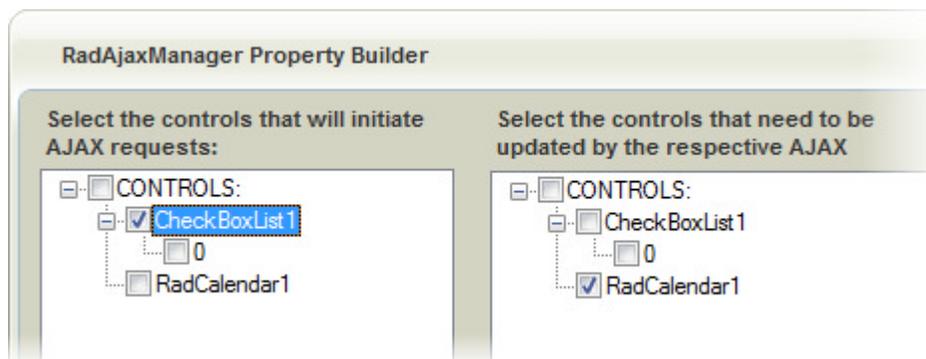
[VB] Handling the OnSelectedIndexChanged Event

```
Protected Sub CheckBoxList1_SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)
    RadCalendar1.ShowColumnHeaders = (TryCast(sender, CheckBoxList)).Items(0).Selected
    RadCalendar1.ShowOtherMonthsDays = (TryCast(sender, CheckBoxList)).Items(1).Selected
    RadCalendar1.ShowRowHeaders = (TryCast(sender, CheckBoxList)).Items(2).Selected
End Sub
```

[C#] Handling the OnSelectedIndexChanged Event

```
protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
{
    RadCalendar1.ShowColumnHeaders = (sender as CheckBoxList).Items[0].Selected;
    RadCalendar1.ShowOtherMonthsDays = (sender as CheckBoxList).Items[1].Selected;
    RadCalendar1.ShowRowHeaders = (sender as CheckBoxList).Items[2].Selected;
}
```

5. On the WebUserControl design surface, open the Smart Tag and select Configure Ajax Manager. When the Property Builder displays, select the CheckBoxList as the initiating control and RadCalendar as the updated control.



6. Press Ctl-F5 to run the application.



- ShowColumnHeaders
- ShowOtherMonthDays
- ShowRowHeaders

7.11 Summary

In this chapter we took a tour of the AJAX related RadControls, paying particular attention to the powerful and flexible RadAjaxManager. You built a simple AJAX-enabled application that first used RadAjaxPanel, then substituted RadAjaxManager to see how the two mechanisms contrast. You also leveraged RadAjaxLoadingPanel to provide better user feedback during AJAX requests.

You learned how to define AJAX settings programmatically at run-time and at design-time using the RadAjaxManager Property Builder dialog to configure settings. Later you used RadAjaxManagerProxy to perform the same settings configuration within a user control.

You built an application that "deals" cards to demonstrate how AJAX requests can be triggered on the client and handled on the server. You coded client-only functions to access common RadAjaxManager properties, e.g. configuration settings, enabling AJAX, canceling requests. You also handled RadAjaxManager client events that let you set and restore state at the beginning and conclusion of AJAX requests.

We looked at design decisions regarding AJAX-enabling applications, took a walk through the ASP.NET page lifecycle and its impact on dynamically created user controls, and finally put this information to use in a Winform-like UI demonstrating dynamic user controls together with AJAX.

You saw how RadAjaxManagerProxy provides visibility to RadAjaxManager settings in complex container-ship scenarios.

Finally, we looked at how RadScriptBlock and RadCodeBlock handle common script + markup related issues.

8 ActiveSkill: Getting Started

8.1 Objectives

- Build the initial framework for a demonstration application that uses many of the RadControls for ASP.NET AJAX in concert.
- Setup the project structure.
- Learn how to setup and use ASP.NET Membership.
- Use RadFormDecorator and RadInput controls in an application.

8.2 Introduction

Learning to use individual RadControls for ASP.NET AJAX is the beginning, not the ending of this tutorial. *Your* use of RadControls will not be in button-and-a-label demos, but in real-world applications that employ multiple controls and technologies in concert including database access, complex user interfaces, user authentication, role assignment and personalization.

This chapter introduces "ActiveSkill", a sample on-line exam application. A user of this application can maintain questions and create exams, as well as take an on-line exam and receive a test score. ActiveSkill has been scoped to be much smaller than a typical business application, but is large enough to involve issues you are likely to face in the trenches.

The ActiveSkill is a MS SQL database that includes tables for Questions, Categories of questions and Exams. Stored procedures are included for most of the CRUD (Create, Read, Update Delete) operations so that the focus stays away from the database mechanics and stays on the use of RadControls for ASP.NET AJAX. ActiveSkill will also use ASP.NET Membership to handle user authentication tasks.

This track of the tutorial will work gradually from simple login and registration pages that use some of the basic controls, to the Administration site that uses RadAjax + server code and finishing up at the user exam taking application that leans towards heavy use of the client API. The entire application will have a custom skin and will use a large portion of the RadControls for ASP.NET AJAX palette.

8.3 Setup ActiveSkill Project Structure

ActiveSkill consists of three projects, a user interface project that contains the web applications for both user and administration pages, a web service used later to supply exam information and a business object project used to define constants, classes and interfaces required by the other two projects.

The first task is to sketch out the general structure of the solution so that later we can fill out the implementation:

1. Create a new ASP.NET Web application. In Visual Studio 2008 you can do this by navigating to **File | New | Project | ASP.NET Web Application**.

Configure the User Interface Project

1. In the Solution Explorer, rename the project to "ActiveSkillUI".
2. Right-click the project and select **Properties**.
3. On the **Application** tab set the Default Namespace to "Telerik.ActiveSkill.UI"
4. On the **Web** tab, in the Servers section, select **Use Local IIS Web Server**. Set the Project Url to `http://<your machine name>/ActiveSkillUI` and click the **Create Virtual Directory** button. *Setting the Url to your machine name will simplify some debugging steps later on.*

Create and Configure the Business Object Project

1. In the Solution Explorer, right-click the solution and select **Add | New Project** from the context menu. Select the **Class Library** project type and name the project "ActiveSkillBO".
2. Right-click the project and select **Properties**.
3. On the **Application** tab set the Default Namespace to "Telerik.ActiveSkill.Common".

Create and Configure the Web Service Project

1. In the Solution Explorer, right-click the solution and select **Add | New Project** from the context menu. Select the **ASP.NET Web Service Application** project type and name the project "ActiveSkillWS".
2. Right-click the project and select **Properties**.
3. On the **Web** tab, in the Servers section, select **Use Local IIS Web Server**. Set the Project Url to `http://<your machine name>/ActiveSkillWS` and click the **Create Virtual Directory** button.

Add User Interface Project Folders

1. In the solution explorer, right-click the ActiveSkillUI project and select **Add | New Folder**. Set the folder name to "Admin". Repeat these steps to create the following folders: "Controls", "Images", "Scripts", "Skins", "Styles" and "User".
2. Rename Default.aspx to "Login.aspx". Right-click "Login.aspx" and select **Set As Start Page** from the context menu.
3. Right-click the ActiveSkillUI project and select **Add | New Item** and choose **Web Form**. Name the web form "Register.aspx".
4. Right-click the "Admin" folder and select **Add | New Item | Web Form**. Name the web form "AdminHome.aspx".
5. Right-click the "User" folder and select **Add | New Item | Web Form**. Name the web form "UserHome.aspx".



You can find the complete source for this project at:

`\VS Projects\ActiveSkill Getting Started\001`

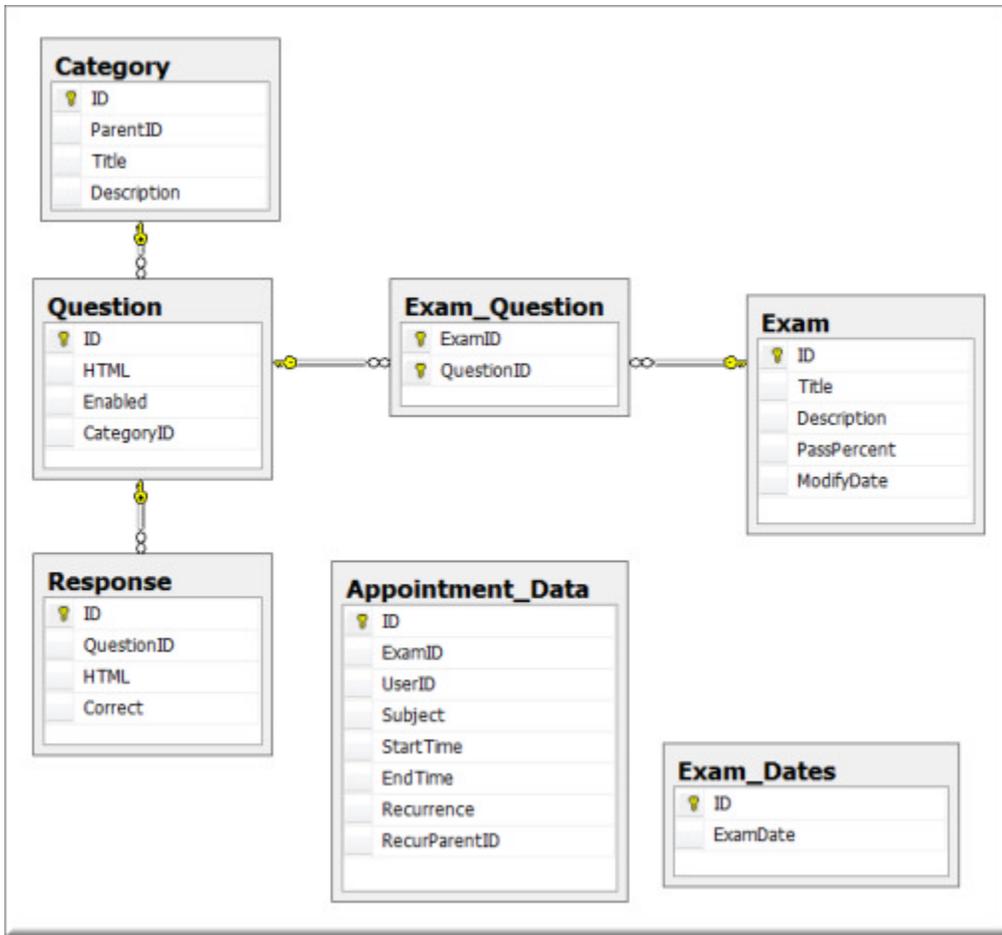
8.4 Setting Up the Database

Introduction to the ActiveSkill Database

The database is simplified and minimal by design. The main part of the database consists of a mere five tables. There are categories of questions, questions that fit in those categories, responses to the questions and Exams that are made up of questions. Exam_Question is a join table that allows the same questions to be used in multiple exams.

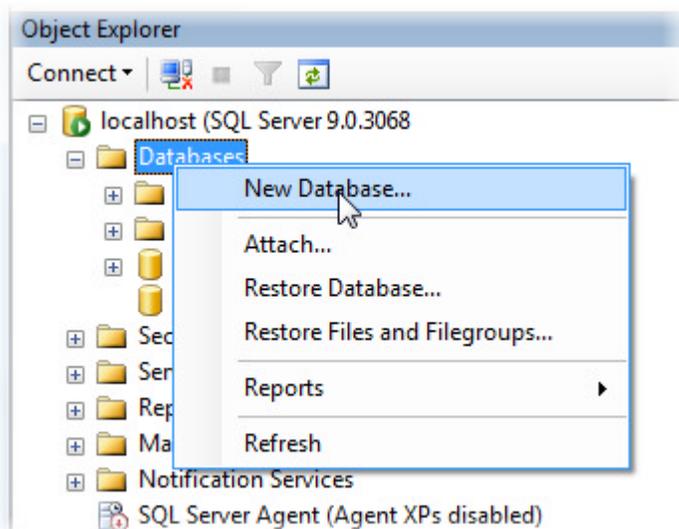
We will cover quite a lot of material just in the maintenance of these tables alone. Notice the Category table ID and ParentID columns; these will help illustrate hierarchical databinding in RadTreeView. The Exam, Question and Response tables will be used in RadGrid to show both master/detail in a single grid and in two related grids.

The other two tables, Appointment_Data and Exam_Dates are used to bind to RadScheduler.



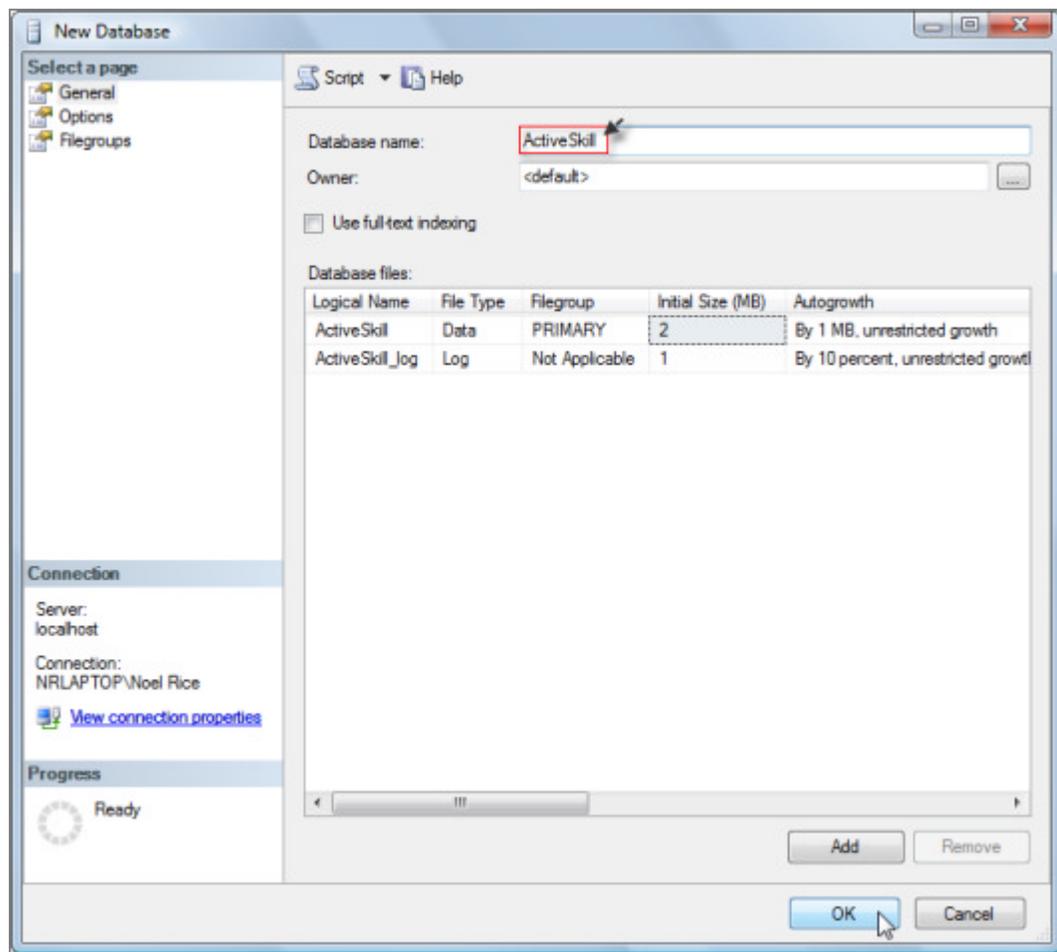
Configuring the ActiveSkill Database

1. In Microsoft SQL Server Management studio, create a new database named "ActiveSkill":
 1. Right-click the Database node of the Object Explorer. Select **New Database**.

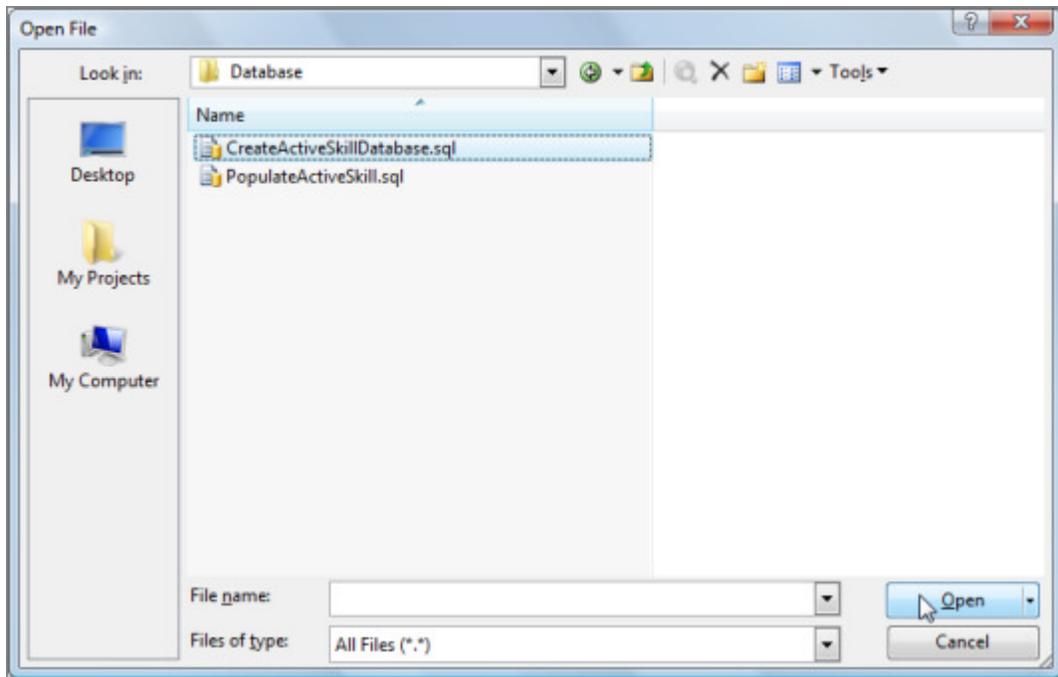


2. Enter "ActiveSkill" as the database name and leave the other settings at their defaults. Click **OK** to

close the **New Database** dialog and create the ActiveSkill database.



2. In Microsoft SQL Server Management studio select the **File** menu, then **Open | File**. Locate the TSQL script file "`\\Database\\CreateActiveSkillDatabase.sql`" and and Click **Open**.



3. Press **F5** to execute the script. *This step will create all the tables and stored procedures required by the application.*
4. In Microsoft SQL Server Management studio select the **File** menu, then **Open | File**. Locate the TSQL script file "\Database\PopulateActiveSkill.sql" and and Click **Open**.
5. Press **F5** to execute the script. *This step will populate the ActiveSkill tables with sample data.*

The database is now ready for the addition of ASP.NET Membership.

8.5 ASP.NET Membership

Introduction to ASP.NET Membership

ASP.NET Membership supplies the infrastructure to manage user accounts on your site and comes with a set of controls for common tasks such as creating new users, logging in, changing passwords, displaying login status and recovering passwords. RadControls for ASP.NET AJAX applications can use the membership system and RadControls can also be used seamlessly *inside of* ASP.NET Membership controls. You can use as much or as little of the membership functionality as your application requires. Here are just a few things you can do with ASP.NET Membership:

- Allow users to create new accounts. The membership system includes behavior to automatically handle familiar situations like requiring the user to verify their email address before their account is activated. ActiveSkill will include an AJAX-enabled registration page.
- Allow users to login to your web application. The membership system can be configured to automatically handle typical login issues: "number of failed attempts", password strength, error messages, etc. The login UI can be completely customized. The ActiveSkill login will include RadControls and will be skinned.
- Create and assign roles. For example, your application could have roles for "admin", "accounting", "browsers", etc., and allow people logged in with those roles to appropriate areas of your web site. ActiveSkill will have two roles, "admin" and "user".
- Work with the ASP.NET Membership API directly. Although the membership system has a lot of functionality that can be used right out of the box, we can also use the API along with the controls to perform any of the membership methods. For example you could list all of the users on the system along with current login

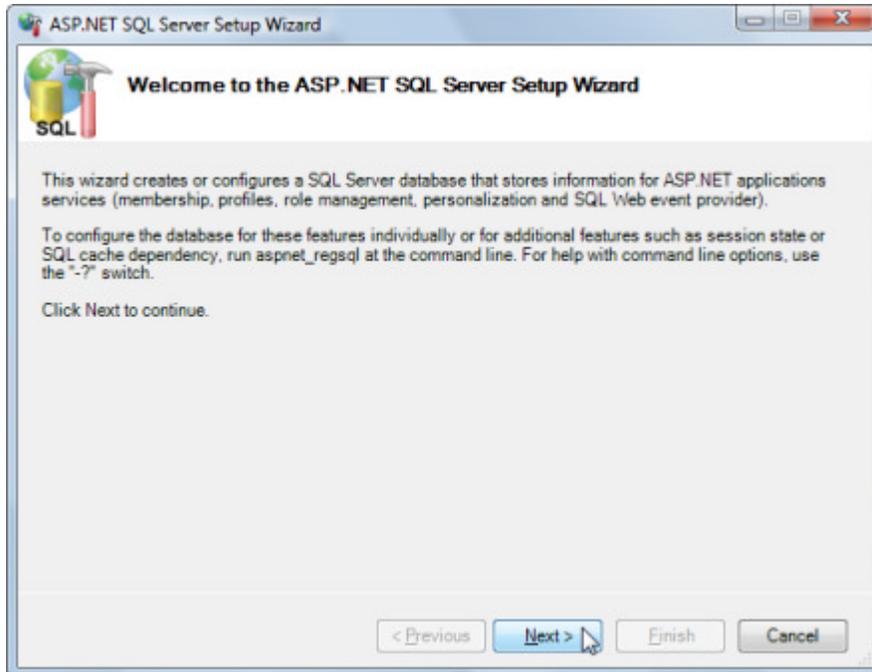
status and display that in a RadGrid. We will use the API in ActiveSkill to create roles and to navigate based on user role.

ASP.NET Membership Database Configuration

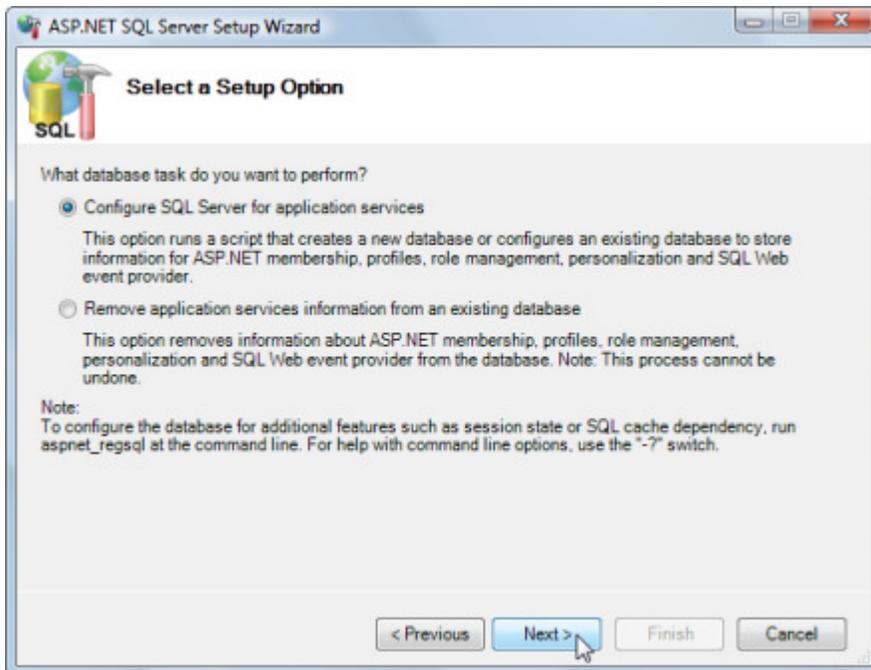
MS SQL includes a ASP.NET SQL Server Registration tool (Aspnet_regsql.exe), located at:

```
[drive:]\%windir%\Microsoft.NET\Framework\v2.0.50727
```

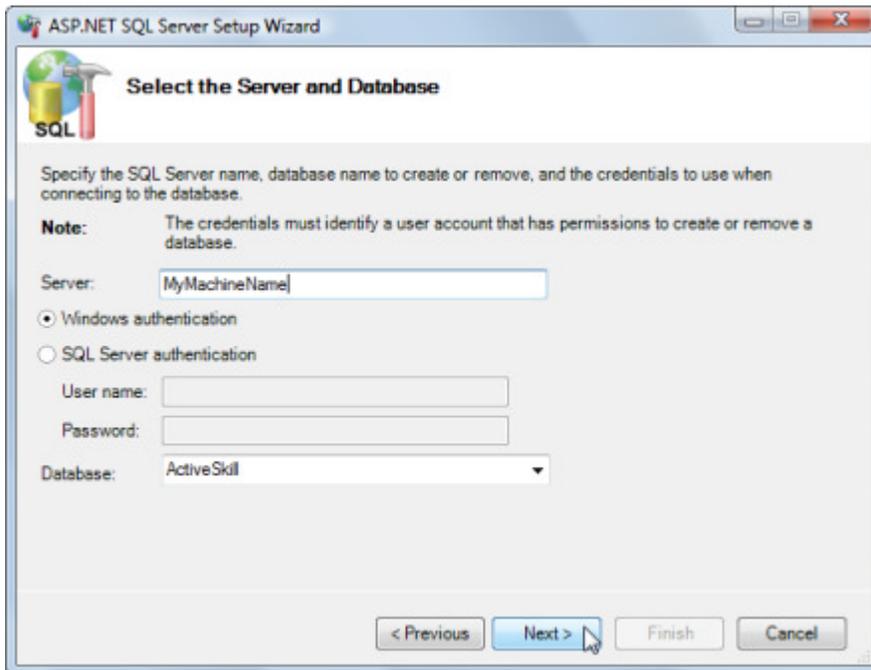
1. Run the command line and navigate to the path listed above. Run Aspnet_regsql.exe without any parameters to display the **ASP.NET SQL Server Setup Wizard**. Click the **Next** button to display the **Select a Setup Option** page.



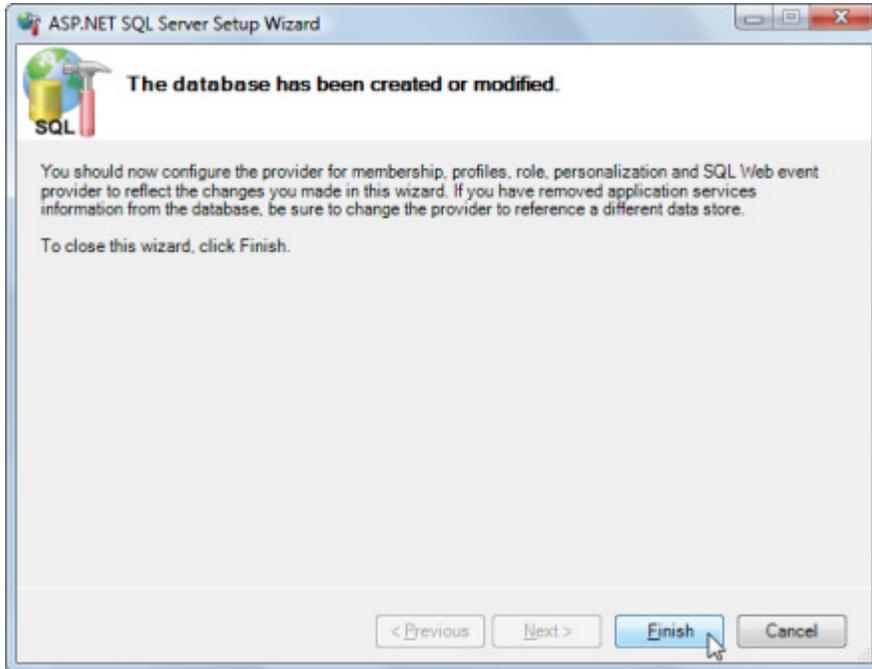
2. Leave the **Configure SQL Server for application services** option selected. Click the **Next** button to display the **Select the Server and Database** page.



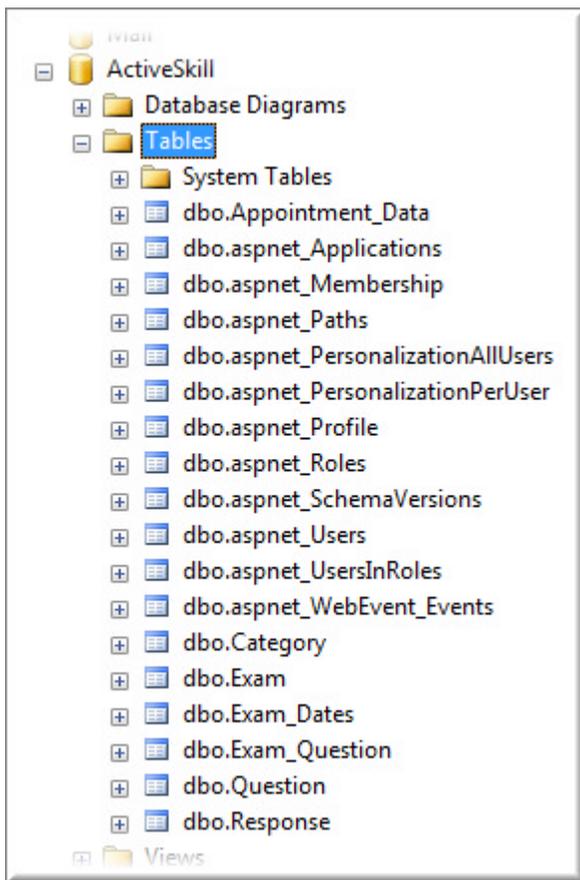
3. Enter the name of your MS SQL server and select the "ActiveSkill" database from the drop down list. Click the **Next** button to display a summary page. Click the **Next** button again to execute the creation of the ASP.NET Membership schema.



4. The completion page of the wizard indicates that the ASP.NET Membership database has been created. Click the **Finish** button.



Now the ActiveSkill database will include the ASP.NET Membership tables and stored procedures:



5. In Microsoft SQL Server Management studio select the **File** menu, then **Open | File**. Locate the TSQL script file "`\\Database\NetworkServicePermissions.sql`" and Click **Open**. *This step will allow ASP.NET permissions to use the database when running on a server.*

✎ For more information on using the ASP.NET SQL Server Registration Tool see <http://msdn.microsoft.com/en-us/library/ms229862>. (<http://msdn.microsoft.com/en-us/library/ms229862.aspx>)

ASP.NET Application Configuration

The following steps will configure the ActiveSkill application to work with ASP.NET Membership.

1. Open the ActiveSkillUI project web.config file.
2. In the <configuration> section of web.config, locate the connection strings element:

```
<connectionStrings/>
```

...and replace it with the connection strings definition below. This defines the "ActiveSkillConnectionString" which will be used through this application. The connection string itself points to the localhost and expects a database called "ActiveSkill" to be present.

[ASP.NET] Defining the ActiveSkill Connection String

```
<!-- RadControls for ASP.NET AJAX Step By Step -->
<connectionStrings>
  <add name="ActiveSkillConnectionString"
        connectionString="Data Source=localhost;Initial Catalog=ActiveSkill;Integrated
Security=True"
        providerName="System.Data.SqlClient"/>
</connectionStrings>
```

3. In the <system.web> tag, add a membership element to the configuration as shown below.

This element specifies that membership information will be stored in the ActiveSkill database.

ASP.NET Membership by default can create an mdb file in the project app_data folder. In our example we want to use a MS SQL database to contain our membership data. Notice the connection string is pointed to our "ActiveSkillConnectionString". Also notice there are a number of settings you can use to specify the exact security profile you want for your web application, e.g. number of retries, required non-alpha characters.

The "applicationName" setting is set to "/ActiveSkill". This is important because all users that get added apply to the "ActiveSkill" application name.

[ASP.NET] Adding the MemberShip Element

```
<membership>
  <providers>
    <clear/>
    <add name="AspNetSqlMembershipProvider"
          type="System.Web.Security.SqlMembershipProvider, System.Web, Version=2.0.0.0,
Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a"
          connectionStringName="ActiveSkillConnectionString"
          enablePasswordRetrieval="false"
          enablePasswordReset="true"
          requiresQuestionAndAnswer="false"
          requiresUniqueEmail="false"
```

```

        passwordFormat="Hashed"
        maxInvalidPasswordAttempts="3"
        minRequiredPasswordLength="7"
        minRequiredNonalphanumericCharacters="1"
        passwordAttemptWindow="10"
        passwordStrengthRegularExpression=""
        applicationName="/ActiveSkill"/>
    </providers>
</membership>

```

- Below the membership element, add a RoleManager element.

This element specifies that Role information are contained in the ActiveSkill database under the "ActiveSkill" application name.

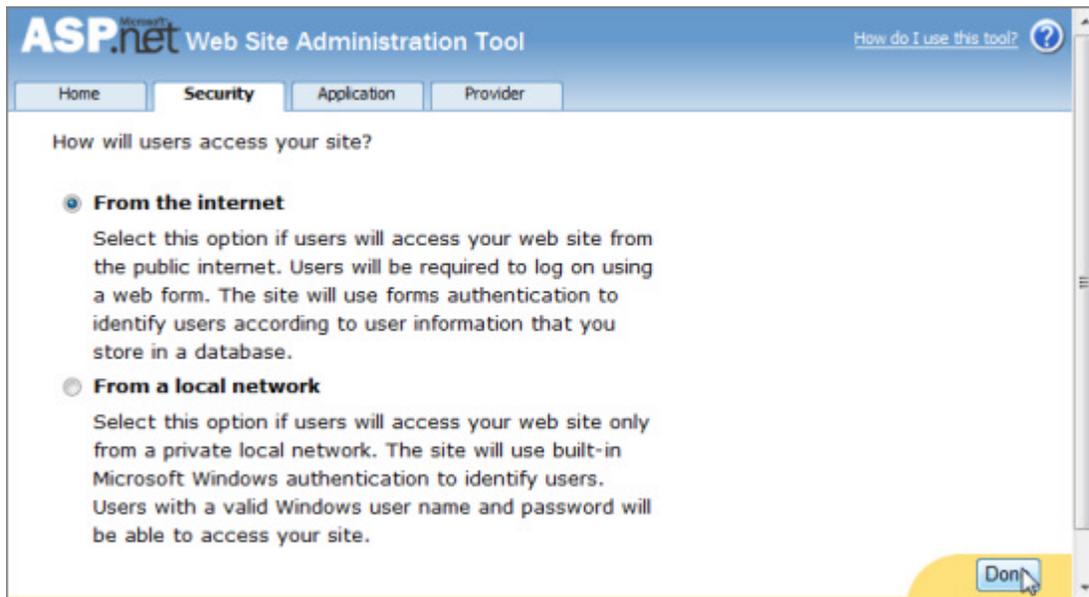
[ASP.NET] Adding the Role Manager Element

```

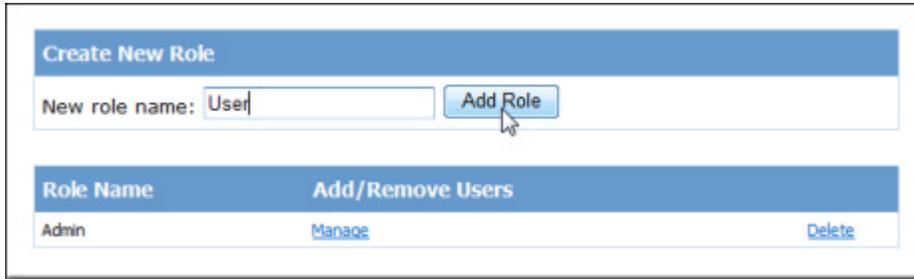
<!-- RadControls for ASP.NET AJAX Step By Step -->
<roleManager enabled="true" defaultProvider="RoleProvider">
  <providers>
    <add connectionStringName="ActiveSkillConnectionString"
        applicationName="/ActiveSkill"
        name="RoleProvider"
        type="System.Web.Security.SqlRoleProvider" />
  </providers>
</roleManager>

```

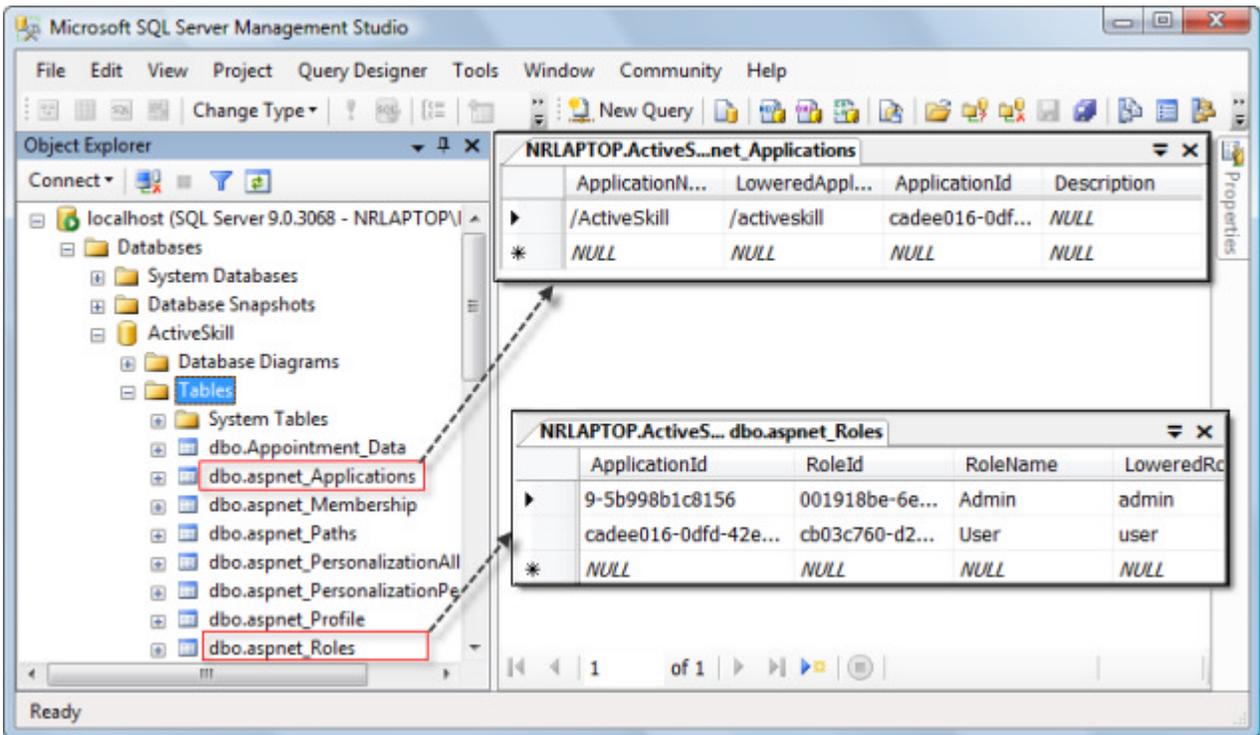
- From the Visual Studio 2008 menu select **Project | ASP.NET Configuration**. *This step will open the Web Site Administration Tool (WSAT).*
- On the Security tab, click the **Select Authentication Type** link. Select the **From the Internet** radio button and click the **Done** button. *This step will change the <authentication> element from mode="Windows" to mode="Forms".*



- Click the **Create or Manage Roles** link. Enter a new role name "Admin" and click the **Add Role** button. Enter another new role name "User" and click the **Add Role** button. Click the **Back** button. *These roles are added to the membership database.*



At this point, if you look at the membership database you will find an entry in the aspnet_Applications table for "/ActiveSkill" and two related records in aspnet_Roles for "Admin" and "User".



8. Create a new admin user:

1. On the Security tab click the **Create User** link.
2. Enter a new user account "Admin", set the Password and Confirm Password to "@password" (this will satisfy the password naming rules we have setup in the config file) and provide an email address.
3. Check the "Admin" role.
4. Click the **Create User** button.
5. Click the **Continue** button.
6. Click the **Back** button.

The new "Admin" user will now be in the membership database.

- In Web.Config in the <system.web> tag, add authorizations for the "admin" and "user" directories. *This step allows members of the "Admin" role to access the \admin folder and the members of the "User" role to access the \user folder.*

[ASP.NET] Adding Access Authorizations

```
<!--RadControls for ASP.NET AJAX Step By Step-->
<location path="admin">
  <system.web>
    <authorization>
      <allow roles="admin"/>
      <deny users="*/>
    </authorization>
  </system.web>
</location>
<!--RadControls for ASP.NET AJAX Step By Step-->
<location path="user">
  <system.web>
    <authorization>
      <allow roles="user"/>
      <deny users="*/>
    </authorization>
  </system.web>
</location>
```



You can find the complete source for this project at:
 \VS Projects\ActiveSkill Getting Started\001\ActiveSkill

8.6 Create the ActiveSkill Login Page

The application is now configured to use ASP.NET membership, so the next step is to create the login page. The login page will use a ASP.NET Membership Login control. The Login control is templated and so can contain other controls, in this case RadTextBox, so long as the control id's expected by Login are maintained. After the user logs in, their role is determined in code and they are redirected to either the admin or user home pages. If the user does not have an account, they can also click a button to land on the Register.aspx page.

1. Go to the design view of the login page.
2. Drop a ScriptManager control on the page. *Even though the page is not AJAX-enabled, the RadTextBox and RadFormDecorator controls require a ScriptManager or RadScriptManager to be present.*
3. Add a RadFormDecorator to the form. Set the Skin property to "Black".
4. From the \Images\ActiveSkill directory, copy the image files "background.jpg" and "bggradient.jpg" to the \images folder of your project.
5. Copy the following <style> tag to the <head> tag:

[CSS] Login Page Styles

```
<style type="text/css" media="screen">
body {
    font-family: ariel;
    font-size: 12px;
    font-color: #376EB1;
    background-image: url('images/bggradient.jpg');
    background-repeat: repeat-x;
    margin: auto;
    text-align: center;
    color: gray;
    vertical-align: middle;
}

#Login1 {
    position: relative;
    top: 330px;
    margin: auto;
    text-align: center;
    left: 0px;
    width: 311px ;
}

#login_position {
    background-image: url('images/background.jpg');
    background-repeat: no-repeat;
    height: 500px;
    width: 375px;
    position: relative;
    top: 40px;
    margin: auto;
    text-align: center;
}
</style>
```

6. Add an ID to the <div> tag that is on the page by default.
7. From the Toolbox drop a Login control to the form. The markup for the page should now look like the example below:

[ASP.NET] The Login Markup

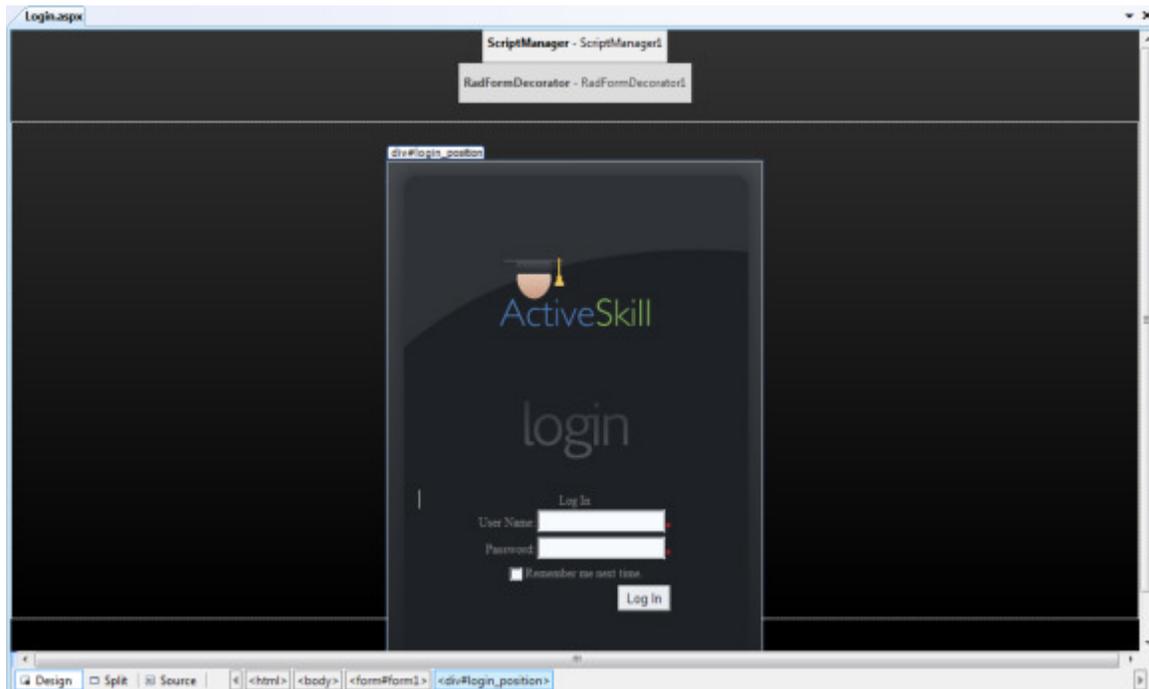
```
<form id="form1" runat="server">
  <asp:ScriptManager ID="ScriptManager1" runat="server">
  </asp:ScriptManager>
  <telerik:RadFormDecorator ID="RadFormDecorator1" runat="server" Skin="Black" />
```

```

<div id="login_position">
    <asp:Login ID="Login1" runat="server">
    </asp:Login>
</div>
</form>

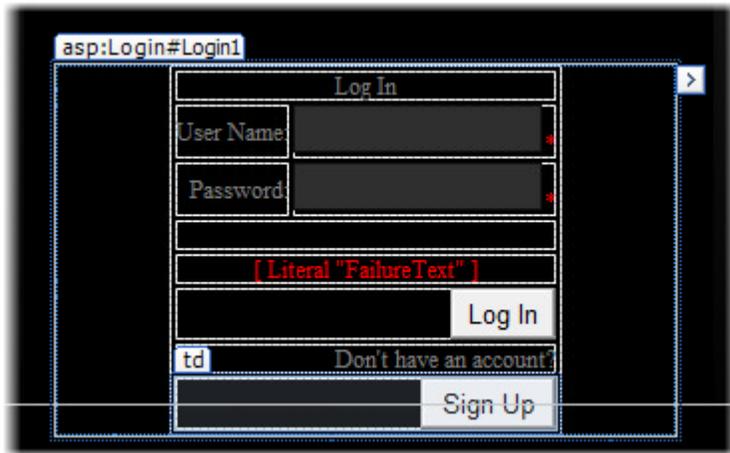
```

The design view of the login page should look something like the example below:



8. From the Login control Smart Tag, Login Tasks, select **Convert to Template**.
9. Select the top-most TextBox control next to "User Name". Notice that the ID property is "UserName". Delete the control.
10. Drag a **RadTextBox** from the ToolBox into the same spot occupied by UserName. Set the ID property to "UserName". Set the **Skin** property to "Black". *The login control is expecting controls with ID's "UserName" and "Password", so the new RadControls we use to replace the existing controls must have those ID's. Forgetting this step generates the error "Login1: LayoutTemplate does not contain an IEditableTextControl with ID UserName for the username."*
11. Locate the TextBox "Password", just below "UserName" and delete it.
12. Drag a **RadTextBox** from the ToolBox into the same spot occupied by Password. Set the ID property to "Password". Set the **Skin** property to "Black".
13. Delete the "Remember me next time" check box. *We will not implement this feature.*
14. Right-click the table cell that holds the "Log In" button and select **Insert | Rows Below** from the context menu. Repeat this step to create a second empty row.
15. In the top empty table cell enter the literal text "Don't have an account?"
16. Drop a standard ASP Button control to the lower empty table cell. Set the ID to "RegisterButton" and the **Text** property to "Sign Up".

The layout should look something like the screenshot below:



17. Select the Login control and in the Property Window Events () button, double-click the LoggedIn event to create an event handler. Add the code below to the event handler. *If the login is successful, the logged in user will be redirected to the admin or user page, depending on their role.*

[VB] Handling the Logged In Event

```
Protected Sub Login1_LoggedIn(ByVal sender As Object, ByVal e As EventArgs)
    If Roles.IsUserInRole(UserNameTextBox.Text, "Admin") Then
        Response.Redirect("~/admin/AdminHome.aspx")
    End If
    If Roles.IsUserInRole(UserNameTextBox.Text, "User") Then
        Response.Redirect("~/user/UserHome.aspx")
    End If
End Sub
```

[C#] Handling the Logged In Event

```
protected void Login1_LoggedIn(object sender, EventArgs e)
{
    if (Roles.IsUserInRole(UserNameTextBox.Text, "Admin"))
        Response.Redirect("~/admin/AdminHome.aspx");
    if (Roles.IsUserInRole(UserNameTextBox.Text, "User"))
        Response.Redirect("~/user/UserHome.aspx");
}
```

18. For the code above to work, you need to add System.Web.Security and Telerik.Web.UI to your "Imports" (VB) or "uses" (C#) section of code.

[VB] Adding References

```
Imports System.Web.Security
Imports Telerik.Web.UI
```

[C#] Adding References

```
using System.Web.Security;
using Telerik.Web.UI;
```

19. The LoggedIn event handler also references a UserNameTextBox property. Add the following property to the code-behind:

[VB] Adding the UserNameTextBox Property

```
#region Properties
Public ReadOnly Property UserNameTextBox() As RadTextBox
    Get
        Return TryCast(Login1.FindControl("UserName"), RadTextBox)
    End Get
End Property
#End Region Properties
```

[C#] Adding the UserNameTextBox Property

```
#region Properties
public RadTextBox UserNameTextBox
{
    get
    {
        return Login1.FindControl("UserName") as RadTextBox;
    }
}
#endregion Properties
```

 You can wrap a call to FindControl() as a property for more readable code, particularly in templated controls.

- Press Ctrl-F5 to run the application. Clicking the Register button should navigate the browser to a blank Register.aspx page. Logging in as "admin" with password "@password" will navigate the browser to the blank "AdminHome.aspx" page.



You can find the complete source for this project at:
 \VS Projects\ActiveSkill Getting Started\002\ActiveSkill

8.7 Create Registration Page

If a new user needs to add an account they are directed to the "register" page where relevant information is gathered, i.e. user name, password, email. This information is automatically stored by the ASP.NET Membership system. That's nice if the information being saved includes all the data you require but what if you have other information you'd like to gather, such as a "Share my exam results" preference, credit card information or some other proprietary data. Wouldn't you need to add a table to your database and use the ASP.NET membership user id as a foreign key? Not really, because ASP.NET membership includes personalization support for custom fields and a CreateUserWizard control that has a number of templates for different pages of the wizard. This allows us the opportunity to add our own page of custom information.

The register page itself will be relatively simple, containing only a user control that wraps the CreateUserWizard control. Within one of the CreateUserWizard "pages" is another user control "BillingControl". BillingControl is its own control to allow this same control to be used elsewhere in the project, for example, in a user preferences page. The BillingControl will be AJAX-enabled using RadAjaxManagerProxy and will use RadTextBox as well as introducing RadMaskedTextBox and RadNumericTextBox.

The general steps to building the registration page are:

- Configure custom fields in web.config.
- Add utility classes to encapsulate session, user and web profile personalization information.
- Working from the inner-most user control, outwards to the page, we will build the BillingControl first.
- Build the CreateUserWizard wrapper user control. This control will consume the BillingControl.
- Implement the Registration page, using the CreateUserWizard wrapper user to supply most of the

functionality.

Configure The Profile

Add the following Profile element to the <system.web> section of web.config. *This step adds a profile with a custom property "ShareMyResults" and a group of properties to contain credit card information. Access to these properties will be encapsulated in utility classes we will define in later steps.*

[ASP.NET] Configuring Profile Properties

```
<!--RadControls for ASP.NET AJAX Step By Step-->
<profile enabled="true">
  <providers>
    <clear/>
    <add name="AspNetSqlProfileProvider"
        type="System.Web.Profile.SqlProfileProvider"
        connectionStringName="ActiveSkillConnectionString"
        applicationName="/ActiveSkill"/>
  </providers>
  <properties>
    <add name="ShareMyResults" type="System.Boolean"/>
    <group name="CreditCard">
      <add name="Type" type="System.Byte"/>
      <add name="Number" type="System.String"/>
      <add name="Name" type="System.String"/>
      <add name="ExpMonth" type="System.String"/>
      <add name="ExpYear" type="System.String"/>
    </group>
  </properties>
</profile>
```

Add Utility Classes

1. In Solution Explorer, navigate to the ActiveSkillBO project, right-click References and select Add Reference. Locate System.Web in the list and click OK to add the System.Web assembly to the references list.
2. Create the WebProfile class. *The class file will contain a CreditCardType enumeration, a CreditCardGroup class and a WebProfile class.*
 1. Right-click the project and select **Add | New Class**. Name the class file WebProfile.cs.
 2. Add references to System.Web.Profile and System.Web.Security to the "Imports" (VB) or "uses" (C#) section of code.

[VB] Adding References

```
Imports System.Web.Profile
Imports System.Web.Security
```

[C#] Adding References

```
using System.Web.Profile;
using System.Web.Security;
```

3. Verify that the namespace for the class is Telerik.ActiveSkill.Common. *This should happen automatically if you set this namespace up as the default for the project during the Setup ActiveSkill Project Structure section earlier.*
4. Add a credit card type enumeration:

[VB] Adding the CreditCardType Enumeration

```
Public Enum CreditCardType
    Unassigned = 0
    Visa = 1
    MasterCard = 2
    Amex = 3
End Enum
```

[C#] Adding the CreditCardType Enumeration

```
public enum CreditCardType
{
    Unassigned = 0,
    Visa = 1,
    MasterCard = 2,
    Amex = 3
};
```

5. Add the CreditCardGroup class. *This class encapsulates the profile group named "CreditCard" that was added to web.config. The class has a property "CreditCardBase" that extracts the group of keys from web.config. The other properties extract the individual keys for easy access.*

[VB] Adding CreditCardGroup

```
''' <summary>
''' This class encapsulates the web.config
''' system.web/profile/properties/group settings
''' for the "Credit Card" group.
''' </summary>
Public Class CreditCardGroup
    Inherits ProfileGroupBase
    Private _profileBase As ProfileBase
    Public Sub New(ByVal profileBase As ProfileBase)
        _profileBase = profileBase
    End Sub
    Const TypeKey As String = "Type"
    Public Property Type() As CreditCardType
        Get
            Return DirectCast(CreditCardGroupBase.GetPropertyvalue(TypeKey), CreditCardType)
        End Get
        Set
            CreditCardGroupBase.SetPropertyvalue(TypeKey, DirectCast(value, Byte))
        End Set
    End Property
    Const NumberKey As String = "Number"
    Public Property Number() As String
        Get
            Return CreditCardGroupBase.GetPropertyvalue(NumberKey).ToString()
        End Get
        Set
            CreditCardGroupBase.SetPropertyvalue(NumberKey, value)
        End Set
    End Property
    Const NameKey As String = "Name"
    Public Property Name() As String
        Get
```

```

    Return CreditCardGroupBase.GetPropertyvalue(NameKey).ToString()
End Get
Set
    CreditCardGroupBase.SetPropertyvalue(NameKey, value)
End Set
End Property
Const ExpMonthKey As String = "ExpMonth"
Public Property ExpMonth() As String
Get
    Return CreditCardGroupBase.GetPropertyvalue(ExpMonthKey).ToString()
End Get
Set
    CreditCardGroupBase.SetPropertyvalue(ExpMonthKey, value)
End Set
End Property
Const ExpYearKey As String = "ExpYear"
Public Property ExpYear() As String
Get
    Return CreditCardGroupBase.GetPropertyvalue(ExpYearKey).ToString()
End Get
Set
    CreditCardGroupBase.SetPropertyvalue(ExpYearKey, value)
End Set
End Property
Const CreditCardGroupKey As String = "CreditCard"
Private ReadOnly Property CreditCardGroupBase() As ProfileGroupBase
Get
    Return _profileBase.GetProfileGroup(CreditCardGroupKey)
End Get
End Property
End Class

```

[C#] Adding CreditCardGroup

```

/// <summary>
/// This class encapsulates the web.config
/// system.web/profile/properties/group settings
/// for the "Credit Card" group.
/// </summary>
public class CreditCardGroup : ProfileGroupBase
{
    private ProfileBase _profileBase;
    public CreditCardGroup(ProfileBase profileBase)
    {
        _profileBase = profileBase;
    }
    const string TypeKey = "Type";
    public CreditCardType Type
    {
        get
        {
            return (CreditCardType)CreditCardGroupBase.GetPropertyvalue(TypeKey);
        }
        set
        {

```

```

        CreditCardGroupBase.SetPropertyValue(TypeKey, (byte) value);
    }
}
const string NumberKey = "Number";
public string Number
{
    get
    {
        return CreditCardGroupBase.GetPropertyValue(NumberKey).ToString();
    }
    set
    {
        CreditCardGroupBase.SetPropertyValue(NumberKey, value);
    }
}
const string NameKey = "Name";
public string Name
{
    get
    {
        return CreditCardGroupBase.GetPropertyValue(NameKey).ToString();
    }
    set
    {
        CreditCardGroupBase.SetPropertyValue(NameKey, value);
    }
}
const string ExpMonthKey = "ExpMonth";
public string ExpMonth
{
    get
    {
        return CreditCardGroupBase.GetPropertyValue(ExpMonthKey).ToString();
    }
    set
    {
        CreditCardGroupBase.SetPropertyValue(ExpMonthKey, value);
    }
}
const string ExpYearKey = "ExpYear";
public string ExpYear
{
    get
    {
        return CreditCardGroupBase.GetPropertyValue(ExpYearKey).ToString();
    }
    set
    {
        CreditCardGroupBase.SetPropertyValue(ExpYearKey, value);
    }
}
const string CreditCardGroupKey = "CreditCard";
private ProfileGroupBase CreditCardGroupBase
{
    get

```

```

    {
        return _profileBase.GetProfileGroup(CreditCardGroupKey);
    }
}
}

```

6. Add the WebProfile class. *This class will contain all the settings for a given profile. This would include the single setting for "ShareMyResults" and also the "CreditCardGroup". When the class is first created, a MembershipUser object is passed to it that ties the logged in user with their profile information.*

[VB] Adding the WebProfile Class

```

''' <summary>
''' This class contains all the settings for a given profile.
''' </summary>
Public Class WebProfile
    Private _profileBase As ProfileBase
    Public Sub New(ByVal user As MembershipUser)
        ' This next line is a key piece that ties together
        ' the logged in user and the profile. Using the
        ' HttpContext current user may be anonymous.
        _profileBase = ProfileBase.Create(user.UserName)
        _creditCardGroup = New CreditCardGroup(_profileBase)
    End Sub
    Private _creditCardGroup As CreditCardGroup
    Public ReadOnly Property CreditCard() As CreditCardGroup
        Get
            Return _creditCardGroup
        End Get
    End Property
    Const ShareMyResultsKey As String = "ShareMyResults"
    Public Property ShareMyResults() As Boolean
        Get
            Return DirectCast(_profileBase.GetPropertyvalue(ShareMyResultsKey), Boolean)
        End Get
        Set
            _profileBase.SetPropertyValue(ShareMyResultsKey, value)
        End Set
    End Property
End Class

```

[C#] Adding the WebProfile Class

```

/// <summary>
/// This class contains all the settings for a given profile.
/// </summary>
public class WebProfile
{
    private ProfileBase _profileBase;
    public WebProfile(MembershipUser user)
    {
        // This next line is a key piece that ties together
        // the logged in user and the profile. Using the
        // HttpContext current user may be anonymous.
        _profileBase = ProfileBase.Create(user.UserName);
        _creditCardGroup = new CreditCardGroup(_profileBase);
    }
}

```

```

    }
    private CreditCardGroup _creditCardGroup;
    public CreditCardGroup CreditCard
    {
        get
        {
            return _creditCardGroup;
        }
    }
    const string ShareMyResultsKey = "ShareMyResults";
    public bool ShareMyResults
    {
        get
        {
            return (bool)_profileBase.GetPropertyValue(ShareMyResultsKey);
        }
        set
        {
            _profileBase.SetPropertyValue(ShareMyResultsKey, value);
        }
    }
}

```

Create the BillingControl User Control UI

1. Right-click the \Controls folder and select Add | New Item and choose **Web User Control**. Name the control "BillingControl.ascx".
2. In the designer, add a RadAjaxManagerProxy control.
3. Below the RadAjaxManager control, add the following table definition to the markup. *This table will contain our controls for "Share my results" and credit card information. The comments indicate where controls will be placed.*

[ASP.NET] Adding the Table Markup

```

<table>
  <tr> <!--Title-->
    <td></td>
    <td></td>
  </tr>
  <tr> <!--Share my results-->
    <td></td>
    <td></td>
  </tr>
  <tr> <!--Credit card type-->
    <td></td>
    <td></td>
  </tr>
  <tr> <!--Credit card number -->
    <td></td>
    <td></td>
  </tr>
  <tr> <!--Credit card name-->
    <td></td>
    <td></td>
  </tr>
  <tr> <!--Expire month and year-->

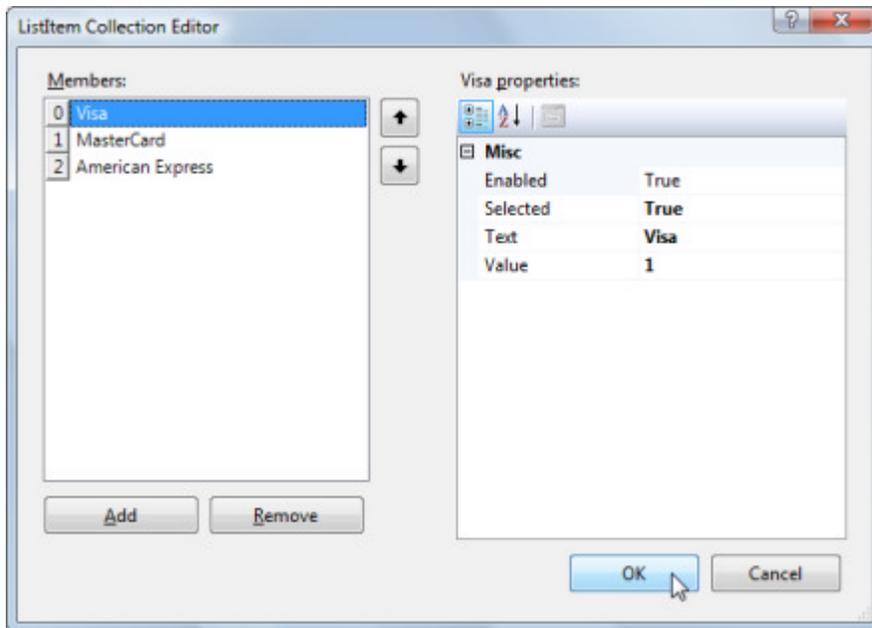
```

```

        <td></td>
        <td></td>
    </tr>
    <tr> <!--Validation summary-->
        <td></td>
        <td></td>
    </tr>
</table>

```

4. Add the "Title" row controls:
 - In the first cell of the "Title" row add a **LoginName** control from the ToolBox "Login" tab.
 - Set the **FormatString** property of the LoginName to "Billing Info for {0}".
5. Add the "Share my results" row controls:
 - In the first cell of the "Share my results" row, add a standard ASP CheckBox control. Set the **ID** property to "cbShareMyResults". Set the **Text** property to "Share My Results".
 - Put the cursor after the checkbox and hit Enter to create a hard break (
).
 - Add a standard ASP Label control. Set the **ID** property to "lblShareMyResults". Set the **Text** property to "".
6. Add the "Credit Card Type" row controls:
 - In the first cell of the "Credit Card Type" row add, a standard ASP RadioButtonList control. Set the **ID** property to "rblCardType". Using the Smart Tag select **Edit Items**. Add three items setting the Text to "Visa", "MasterCard" and "American Express" respectively. Set the Value properties to "1", "2" and "3". Set the first item ("Visa") **Selected** property to True.



7. Add the "Credit Card Number" row controls:
 - Add a **RadMaskedTextBox** to the first cell of the row. Set the **ID** property to "tbCCNumber". Set the **Label** property to "Credit Card Number:", the **Mask** property to "#####-####-####-####" and **Skin** to "Black".
 - Add a standard ASP **RequiredFieldValidator** to the second cell. Set the **ID** property to "valreqCCNumber". Set the **ControlToValidate** property to "tbCCNumber", **ErrorMessage** to "A valid

credit card number is required" and **ValidationGroup** to "BillingGroup".

- Add a second standard ASP **RegularExpressionValidator** to the second cell. Set the **ID** property to "valregexCCNumber". Set the **ControlToValidate** to "tbCCNumber", the **ErrorMessage** to "Enter a valid credit card number" and **ValidationGroup** to "BillingGroup". Set the **ValidationExpression** to "`^((4\d{3})|(5[1-5]\d{2})|(6011))-\d{4}-\d{4}-\d{4}|3[4,7]\d{13})$`".

 The regular expression above is a relatively simple sample that can be used for this demo. You should research your own regular expression based on your security needs.

8. Add the "Credit Card Name" row controls:

- Add a **RadTextBox** to the first cell of the row. Set the **ID** to "tbCCName". Set the **EmptyMessage** property to "Enter Name:" the **Label** to "Credit Card Name:" and **Skin** to "Black".
- Add a standard ASP **RequiredFieldValidator** to the second cell. Set the **ID** property to "valreqCCName". Set the **ControlToValidate** property to "tbCCName", **ErrorMessage** to "Enter the name on the credit card" and **ValidationGroup** to "BillingGroup".

9. Add the "Expire Month and Year" row controls:

- Add a **RadNumericTextBox** to the first cell of the row. Set the **ID** property to "tbExpMonth". Set the **Label** property to "Exp Month:", **MaxValue** to "12", **MinValue** to "1", **ShowSpinButtons** to True, and **Skin** to "Black". In the **NumberFormat** sub-properties set **AllowRounding** to False, **DecimalDigits** to "0".
- Add a second **RadNumericTextBox** to the first cell of the row. Set the **ID** property to "tbExpYear". Set the **Label** property to "Exp Year:", **ShowSpinButtons** to True, and **Skin** to "Black". In the **NumberFormat** sub-properties set **AllowRounding** to False, **DecimalDigits** to "0", **GroupSeparator** to "" and **PositivePattern** to "n".

10. In the last row add a standard ASP **ValidationSummary**. Set the **ValidationGroup** property to "BillingGroup".

The BillingControl User Control Code-Behind

1. Using the Properties Window with the **cbShareMyResults** checkbox selected, click the events button (). Double-click the **CheckChanged** event. In the event handler, add the code below.

[VB] Handling the CheckChanged Event

```
Protected Sub cbShareMyResults_CheckedChanged(ByVal sender As Object, ByVal e As EventArgs)
    lblShareMyResults.Text = IIf(cbShareMyResults.Checked, ExamsVisible, ExamsNotVisible)
End Sub
```

[C#] Handling the CheckChanged Event

```
protected void cbShareMyResults_CheckedChanged(object sender, EventArgs e)
{
    lblShareMyResults.Text = cbShareMyResults.Checked ? ExamsVisible : ExamsNotVisible;
}
```

2. In the **BillingControl** code-behind add the following constants:

[VB] Adding Constants

```
#region constants
Const DigitMask16 As String = "####-####-####-####"
Const DigitMask15 As String = "####-####-####-###"
Const ExamsVisible As String = "Your exam results are publicly available"
Const ExamsNotVisible As String = "Only you can see your exam results"
Const AmexIndex As Integer = 2
```

```
#End Region constants
```

[C#] Adding Constants

```
#region constants
const string DigitMask16 = "####-####-####-####";
const string DigitMask15 = "####-####-####-###";
const string ExamsVisible = "Your exam results are publicly available";
const string ExamsNotVisible = "Only you can see your exam results";
const int AmexIndex = 2;
#endregion constants
```

Implement the Registration Page

8.8 Implement the Registration Page

1. In the <head> element of the Register page add the following styles:

[CSS] Add Styles for Register Page

```
<style type="text/css" media="screen">
body
{
    font-family: ariel;
    font-size: 12px;
    font-color: #376EB1;
    background-image: url('images/bggradient.jpg');
    background-repeat: repeat-x;
    margin: auto;
    text-align: center;
    color: gray;
    vertical-align: middle;
}
#wizard
{
    position: relative;
    top: 300px;
    margin: auto;
    text-align: center;
    left: 10px;
    width: 311px;
}
#login_position
{
    background-image: url('images/background.jpg');
    background-repeat: no-repeat;
    height: 500px;
    width: 375px;
    position: relative;
```

```

    top: 40px;
    margin: auto;
    text-align: center;
}
</style>

```

2. Add a **ScriptManager** to the page.
3. Add a **RadAjaxManager** to the page.
4. Add a **RadToolTipManager** to the page. Set the **Skin** property to "Black".
5. Add a **RadFormDecorator** control to the page. Set the **Skin** property to "Black".
6. Set the **id** attribute for the default **div** on the page to "login_position". Add a second **<div>** tag inside the first with **id** attribute "wizard".

[ASP.NET] Setting Up Div Tags

```

<div id="login_position">
  <div id="wizard">
  </div>
</div>

```

7. Drag the **CreateUserWizardWrapper** control to the page. Verify that the control is placed inside the two **divs**.

The purpose of the <div> tags is to place the login background graphic centered on the page and the inner div to position the wizard control itself.

8. The resulting markup for the **Register.aspx** page should look like the example below:

[ASP.NET] Finished Register.aspx Markup

```

<body>
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <telerik:RadFormDecorator ID="RadFormDecorator1" runat="server" Skin="Black" />
    <div id="login_position">
      <div id="wizard">
        <uc1:CreateUserWizardWrapper ID="CreateUserWizardWrapper1" runat="server" />
      </div>
    </div>
  </form>
</body>

```

9. There is no code for the **Register** page, so press **Ctrl-F5** to run and test the application.
 - Add a new user.
 - Re-run the application and login as your new user.
 - Notice the styled tool tips in the billing information page.

8.9 The CreateUserWizardWrapper Code-Behind

1. In the code-behind for **CreateUserWizardWrapper** add two properties. One to access the "UserName" text box and the second to access **BillingControl**.

*Notice that to access controls on one of the templates that you first go to the object that defines the step, then use the **ContentTemplateContainer FindControl()** method to locate the object.*

[VB] Adding Properties

```
#region Properties
Private ReadOnly Property BillingControl1() As BillingControl
    Get
        Return TryCast(BillingStep.ContentTemplateContainer.FindControl("BillingControl1"),
BillingControl)
    End Get
End Property
#End Region Properties
```

[C#] Adding Properties

```
#region Properties
private BillingControl BillingControl1
{
    get { return BillingStep.ContentTemplateContainer.FindControl("BillingControl1") as
BillingControl; }
}
#endregion Properties
```

2. In the designer, select the CreateUserWizard control and using the Properties Window Events button  double-click the **CreatedUser** event to create an event handler. Add the following code to the event handler.

*Once the user is created by the ASP.NET Membership system, this event fires. The method first retrieves the wizard's **UserName** property, uses the name to create a new **ASUser** object. The **ASUser** is assigned to the **SessionManager User** property. The user name is added to the "User" role.*

[VB] Handling the CreatedUser Event

```
Protected Sub CreateUserWizard1_CreatedUser(ByVal sender As Object, ByVal e As EventArgs)
    Dim userName As String = (TryCast(sender, CreateUserWizard)).UserName
    SessionManager.User = New ASUser(userName)
    Roles.AddUserToRole(userName, "User")
End Sub
```

[C#] Handling the CreatedUser Event

```
protected void CreateUserWizard1_CreatedUser(object sender, EventArgs e)
{
    string userName = (sender as CreateUserWizard).UserName;
    SessionManager.User = new ASUser(userName);
    Roles.AddUserToRole(userName, "User");
}
```

3. Create another event for the CreateUserWizard **FinishButtonClick** event. *This step validates the "BillingGroup" set of validators and if the data is good, the **UpdateUser()** method is called to send the user entry on the billing control to the ASP.NET Membership database.*

[VB] Handling the FinishButtonClick Event

```
Protected Sub CreateUserWizard1_FinishButtonClick(ByVal sender As Object, ByVal e As
WizardNavigationEventArgs)
    Page.Validate("BillingGroup")
    e.Cancel = Not Page.IsValid
    If Page.IsValid Then
```

```

    BillingControl1.UpdateUser()
End If
End Sub

```

[C#] Handling the FinishButtonClick Event

```

protected void CreateUserWizard1_FinishButtonClick(object sender, WizardNavigationEventArgs e)
{
    Page.Validate("BillingGroup");
    e.Cancel = !Page.IsValid;
    if (Page.IsValid)
    {
        BillingControl1.UpdateUser();
    }
}

```

4. Create another event for the CreateUserWizard **ContinueButtonClick** event. *If we reach this event all the validation has been performed; we have a reference to the user and the user has been added to the user role. Here we just navigate directly to the UserHome page.*

[VB] Handling the ContinueButtonClick

```

Protected Sub CreateUserWizard1_ContinueButtonClick(ByVal sender As Object, ByVal e As EventArgs)
    Response.Redirect("~/user\UserHome.aspx")
End Sub

```

[C#] Handling the ContinueButtonClick

```

protected void CreateUserWizard1_ContinueButtonClick(object sender, EventArgs e)
{
    Response.Redirect("~/user\UserHome.aspx");
}

```



If you were allowing more than one kind of role to be created here and wanted to direct the created user to different pages based on role you could use some code similar to this:

```

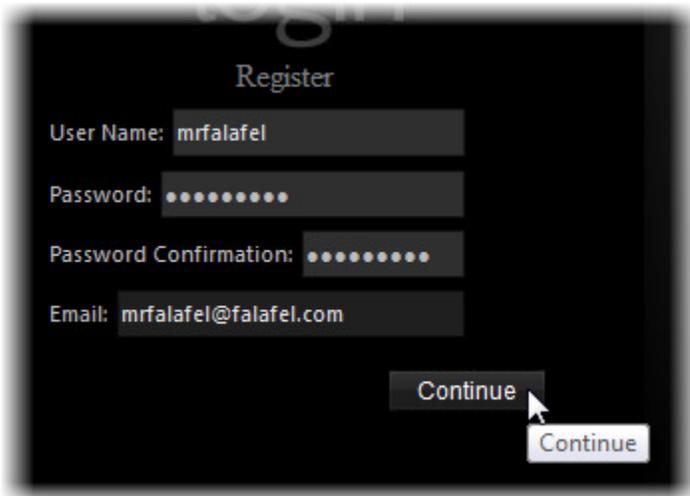
if (Roles.IsUserInRole(SessionManager.User.UserName, "User") { /* redirect to some page */ }

```

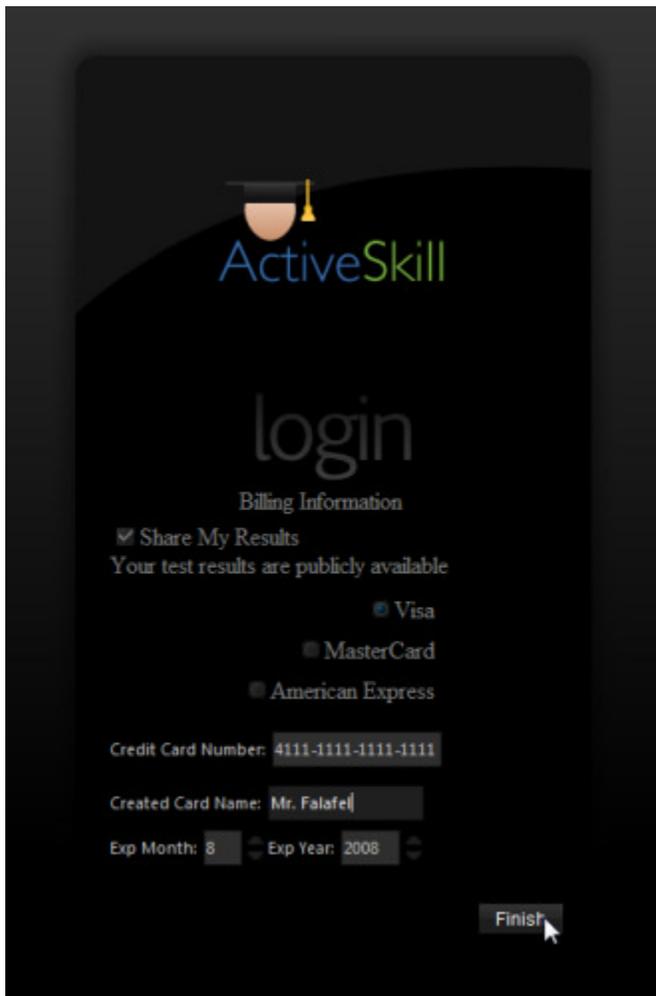
8.10 The CreateUserWizardWrapperUI

ASP.NET Membership includes a CreateUserWizard control from the Login tab of the Toolbox. The control is similar to the Login control in that it can be templated. But this control has multiple templates, one for each page of the wizard. The wizard starts with a "Sign up for your new account" page, has an arbitrary number of pages of your own that you can add on the fly, and then a "Finish" page. Like the Login control, we will substitute our own RadControls to the first page, add a "Billing" page to which we will add our billing user control and leave the Finish page as is.

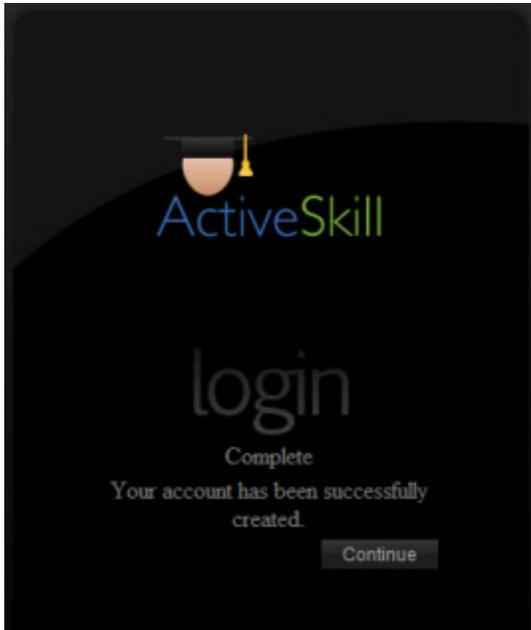
The code-behind is minimal, as most of the detail is packed away in the billing user control. First the user enters their desired user name, enters the password twice and enters an email.



When the user clicks "Continue", the ASP.NET Member user is created, the user is added to the "User" role. The billing page of the wizard shows up prompting for the custom information using BillingControl. The user fills in the information and clicks the "Finish" button. The billing information is validated and the web profile is updated.



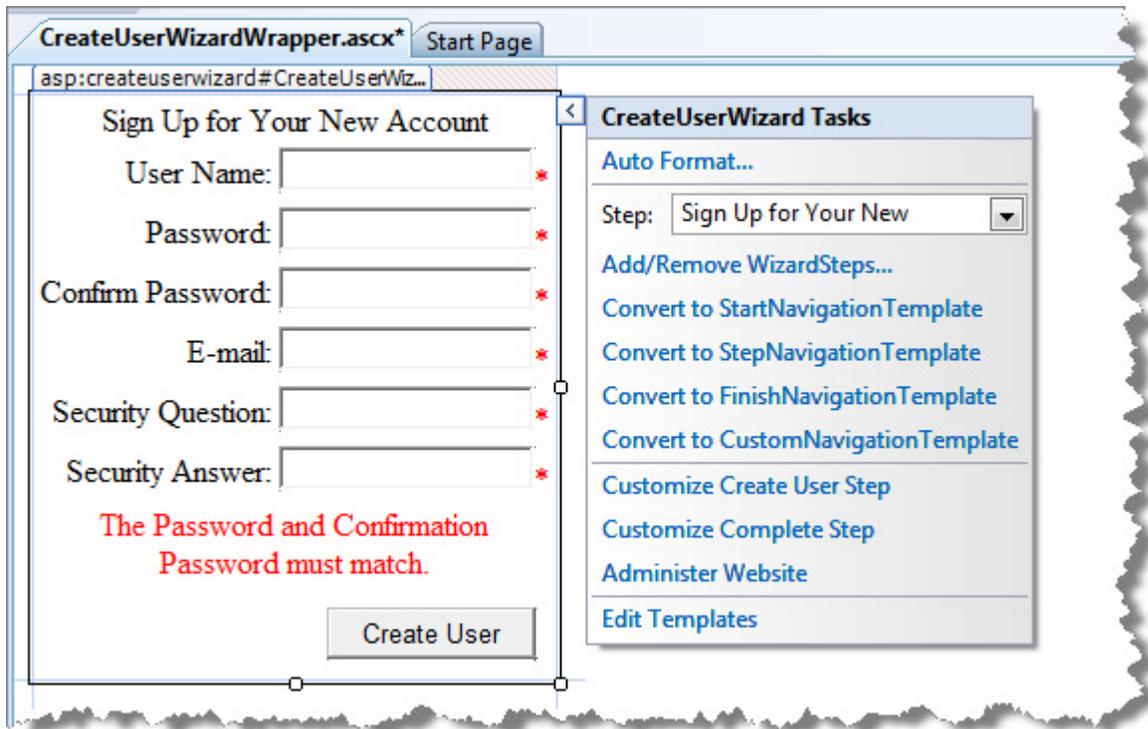
The "Complete" page of the wizard displays. When the user clicks "Continue" they are directed to the user home page.



To begin building the CreateUserWizardWrapper Control:

1. Right-click the \Controls folder and select **Add | New Item** and choose **Web User Control**. Name the control "CreateUserWizardWrapper.ascx".
2. Switch to the Design view of the control. From the Toolbox, Login tab drag a **CreateUserWizard** control and drop it on the design surface.

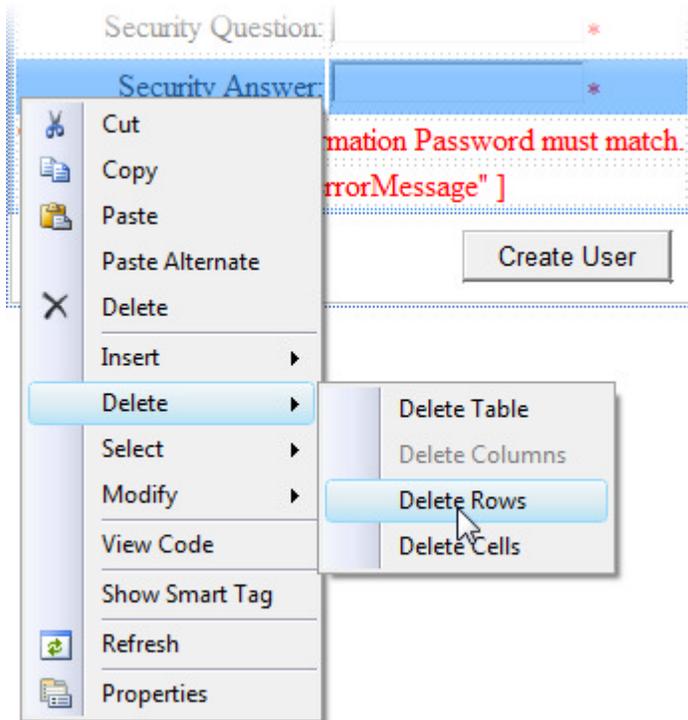
Notice from the CreateUserWizard smart tag that the wizard is made up of "steps" where each step can be converted to a template. The Create user and Complete steps default to a template that can be altered using the "Customize Create User Step" and "Customize Complete Step" links. We will start by customizing the Create User Step to suit our purposes.



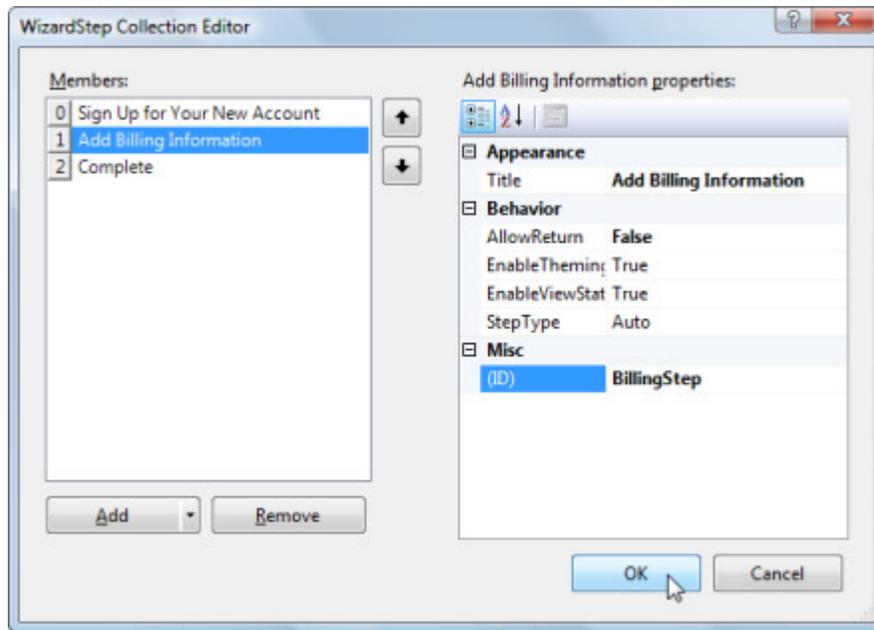
3. In the CreateUserWizard Smart Tag click the Customize Create User Step link. *This change will allow the page to be edited.*
4. Change the default titling "Sign Up for Your New Account" to "Register". You can type this directly in the designer.
5. In the designer, move the mouse to the "Security Question" row of the table, just to the left of the row. The cursor should change to a rightward pointing arrow.



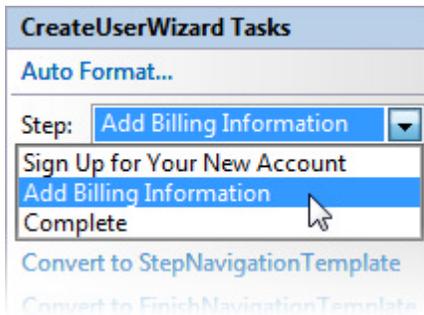
6. Right-click and select Delete | Delete Rows from the context menu. *If you have any difficulty with this you can switch to the source view of the page and delete the row from there.*



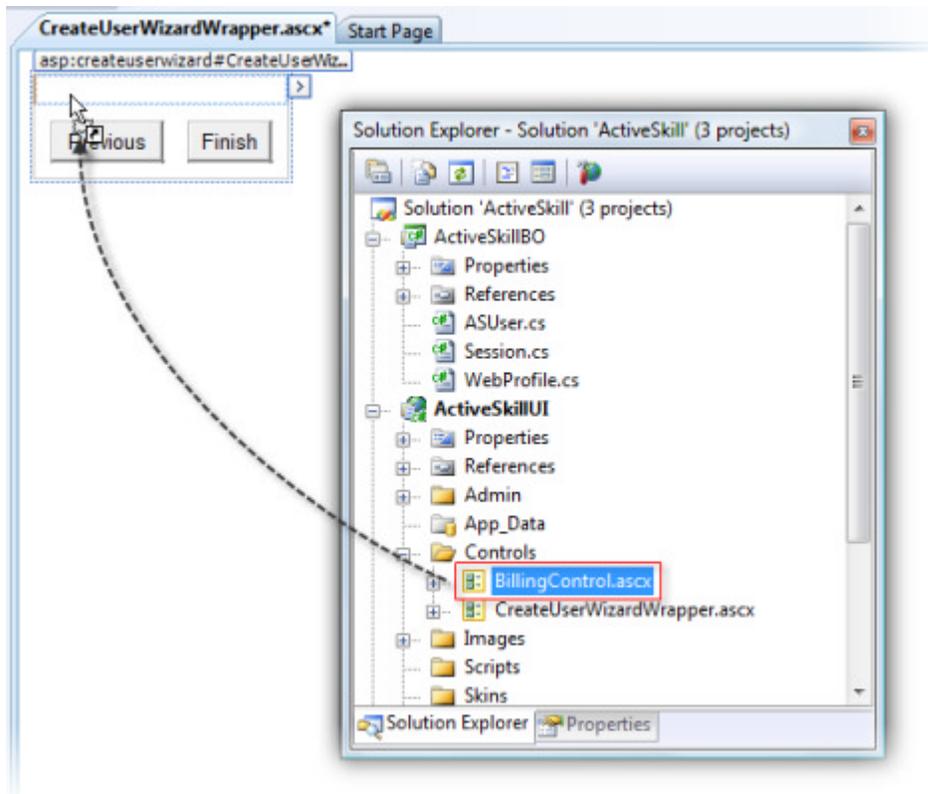
7. Replace the Label and TextBox controls for the "User Name", "Password", "Confirm Password" and "Email" with **RadTextBox** controls.
 1. Be sure to set the RadTextBox ID's to the values of the TextBox controls they are replacing, that is "UserName", "Password", "ConfirmPassword" and "Email".
 2. Set the **Label** property of each RadTextBox to "User Name:", "Password:", "Confirm Password:" and "Email:", respectively.
 3. Set the **Skin** of each control to "Black".
 4. Set the **Width** of each control to "210px".
 5. Set the **TextMode** property for both password controls to "Password".
8. Set the **Align** attribute to "Left" for the table cells containing "User Name:", "Password:", "Confirm Password:" and "Email".
9. From the CreateUserWizard Smart Tag select the **Add/Remove Wizard Steps...** link. *This will display the WizardStep Collection Editor.*
 1. Select the "Sign up for Your New Account" step and set the **ID** to "CreateUserWizardStep1".
 2. Select the "Complete" step and set the **ID** to "CompleteWizardStep1".
 3. Click the Add button to add a wizard step to the list. Use the Up arrow button to position it before the "Complete" step. Set the **Title** to "Add Billing Information", **AllowReturn** to False and the **ID** to "BillingStep".
 4. Click the **OK** button to close the WizardStep Collection Editor.



10. From the CreateUserWizard Smart Tag, drop down the "Step" list and select the new "Add Billing Information" step.



11. Drag BillingControl.ascx from the SolutionExplorer to the upper region of the CreateUserWizard.



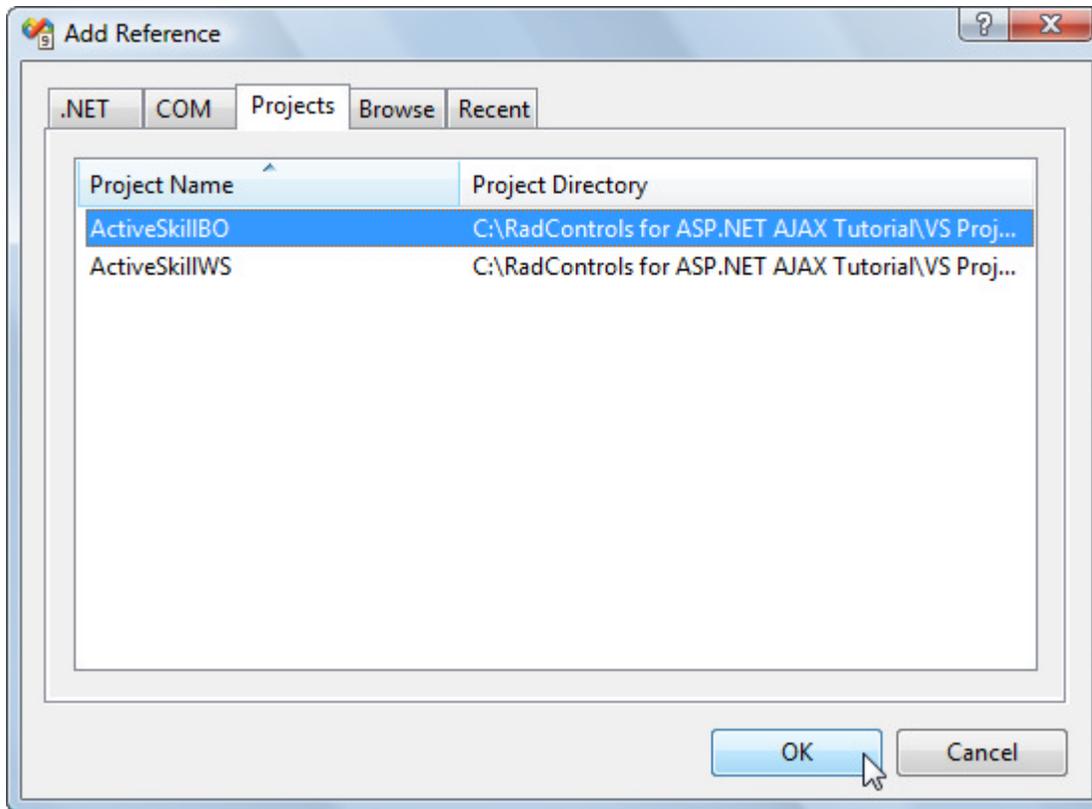
- Using the Smart Tag for the CreateUserWizard, drop the list of steps and select "Sign up for your new accounts".



Gotcha! Make sure you set the step to the first page before you're done. When you run the page it will display the step that is set at design-time.

8.11 Create the Billing Control Code-Behind

- Add a reference to the ActiveSkillBO assembly. *This step will provide access to the Telerik.ActiveSkill.Common namespace.*



2. Add a reference to the Telerik.ActiveSkill.Common namespace. *This will provide access to our utility classes WebProfile, ASUser and SessionManager.*

[VB] Adding References

```
Imports Telerik.ActiveSkill.Common
```

[C#] Adding References

```
using Telerik.ActiveSkill.Common;
```

3. Using the Properties Window with the cbShareMyResults checkbox selected, click the events button () . Double-click the CheckChanged event. In the event handler, add the code below. *If the "Share My Results" check box is checked, the ExamsVisible string constant is displayed, otherwise ExamsNotVisible is shown.*

[VB] Handling the CheckChanged Event

```
Protected Sub cbShareMyResults_CheckedChanged(ByVal sender As Object, ByVal e As EventArgs)  
    lblShareMyResults.Text = IIf(cbShareMyResults.Checked, ExamsVisible, ExamsNotVisible)  
End Sub
```

[C#] Handling the CheckChanged Event

```
protected void cbShareMyResults_CheckedChanged(object sender, EventArgs e)  
{  
    lblShareMyResults.Text = cbShareMyResults.Checked ? ExamsVisible : ExamsNotVisible;  
}
```

4. Using the Properties Window with the cbShareMyResults checkbox selected, click the events button () . Double-click the CheckChanged event. In the event handler, add the code below. *If the "Credit Card Type" radio button list selection changes, the RadTextBox mask changes to reflect the correct number of digits.*

American Express uses 15 digits while Visa and MasterCard use 16 digits.

[VB] Handling the SelectedIndexChanged Event

```
Protected Sub rblCardType_SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)
    tbCCNumber.Mask = IIf(rblCardType.SelectedIndex = AmexIndex, DigitMask15, DigitMask16)
End Sub
```

[C#] Handling the SelectedIndexChanged Event

```
protected void rblCardType_SelectedIndexChanged(object sender, EventArgs e)
{
    tbCCNumber.Mask = rblCardType.SelectedIndex == AmexIndex ? DigitMask15 : DigitMask16;
}
```

- In the BillingControl code-behind add the following constants:

[VB] Adding Constants

```
#region constants
Const DigitMask16 As String = "####-####-####-####"
Const DigitMask15 As String = "####-####-####-###"
Const ExamsVisible As String = "Your exam results are publicly available"
Const ExamsNotVisible As String = "Only you can see your exam results"
Const AmexIndex As Integer = 2
#End Region constants
```

[C#] Adding Constants

```
#region constants
const string DigitMask16 = "####-####-####-####";
const string DigitMask15 = "####-####-####-###";
const string ExamsVisible = "Your exam results are publicly available";
const string ExamsNotVisible = "Only you can see your exam results";
const int AmexIndex = 2;
#endregion constants
```

- Add the utility method UpdateUser(). This method will be called by the CreateUserWizardWrapper control when the user clicks the "Finish" button of the wizard when registering a new account. The user's web profile properties are populated with values scraped from the UI and then the Save() method is called to persist those values to the ASP.NET Membership database.

[VB] Adding the UpdateUser Method

```
Public Sub UpdateUser()
    Dim profile As WebProfile = SessionManager.User.Profile
    profile.ShareMyResults = cbShareMyResults.Checked
    profile.CreditCard.Type = DirectCast(rblCardType.SelectedIndex, CreditCardType) - 1
    profile.CreditCard.Number = tbCCNumber.Text
    profile.CreditCard.Name = tbCCName.Text
    profile.CreditCard.ExpMonth = tbExpMonth.Text
    profile.CreditCard.ExpYear = tbExpYear.Text
    profile.Save()
End Sub
```

[C#] Adding the UpdateUser Method

```
public void UpdateUser()
{
```

```
WebProfile profile = SessionManager.User.Profile;
profile.ShareMyResults = cbShareMyResults.Checked;
profile.CreditCard.Type = (CreditCardType) rblCardType.SelectedIndex - 1;
profile.CreditCard.Number = tbCCNumber.Text;
profile.CreditCard.Name = tbCCName.Text;
profile.CreditCard.ExpMonth = tbExpMonth.Text;
profile.CreditCard.ExpYear = tbExpYear.Text;
profile.Save();
}
```

7. Add the Page_Load event handling code. *Here the ASUser is retrieved from the SessionManager. If the user already exists (the user is already logged in) the profile data is displayed in the UI. If the user is not logged in, we are on the registration page; set the expiration month and year to default values. Note: The application in its current form does not implement a "preferences" page and so only the "else" portion of this code will execute.*

[VB] Handling the Page_Load event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        Dim isUser As Boolean = (SessionManager.User <> Nothing)
        If isUser Then
            Dim profile As WebProfile = SessionManager.User.Profile
            cbShareMyResults.Checked = profile.ShareMyResults
            rblCardType.SelectedIndex = DirectCast(profile.CreditCard.Type, Byte)
            tbCCNumber.Text = profile.CreditCard.Number
            tbCCName.Text = profile.CreditCard.Name
            tbExpMonth.Text = profile.CreditCard.ExpMonth
            tbExpYear.Text = profile.CreditCard.ExpYear
        Else
            ' insert
            tbExpMonth.Text = DateTime.Now.Month.ToString()
            tbExpYear.MinValue = DateTime.Now.Year
            tbExpYear.MaxValue = DateTime.MaxValue.Year
            tbExpYear.Text = DateTime.Now.Year.ToString()
        End If
    End If
End Sub
```

[C#] Handling the Page_Load event

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        bool isUser = (SessionManager.User != null);

        if (isUser)
        {
            WebProfile profile = SessionManager.User.Profile;
            cbShareMyResults.Checked = profile.ShareMyResults;
            rblCardType.SelectedIndex = (byte)profile.CreditCard.Type;
            tbCCNumber.Text = profile.CreditCard.Number;
            tbCCName.Text = profile.CreditCard.Name;
            tbExpMonth.Text = profile.CreditCard.ExpMonth;
            tbExpYear.Text = profile.CreditCard.ExpYear;
        }
    }
}
```

```

else // insert
{
    tbExpMonth.Text = DateTime.Now.Month.ToString();
    tbExpYear.MinValue = DateTime.Now.Year;
    tbExpYear.MaxValue = DateTime.MaxValue.Year;
    tbExpYear.Text = DateTime.Now.Year.ToString();
}
}
}

```

8.12 Create the BillingControl User Control

The BillingControl gathers custom profile information and saves it to the ASP.NET Membership database.

1. Right-click the \Controls folder and select **Add | New Item** and choose **Web User Control**. Name the control "BillingControl.ascx".
2. In the designer, add a RadAjaxManagerProxy control. We will configure the proxy once the controls to be AJAX-enabled are in place.
3. Below the RadAjaxManager control, add the following table definition to the markup. *This table will contain our controls for "Share my results" and credit card information. The comments indicate where controls will be placed.*

[ASP.NET] Adding the Table Markup

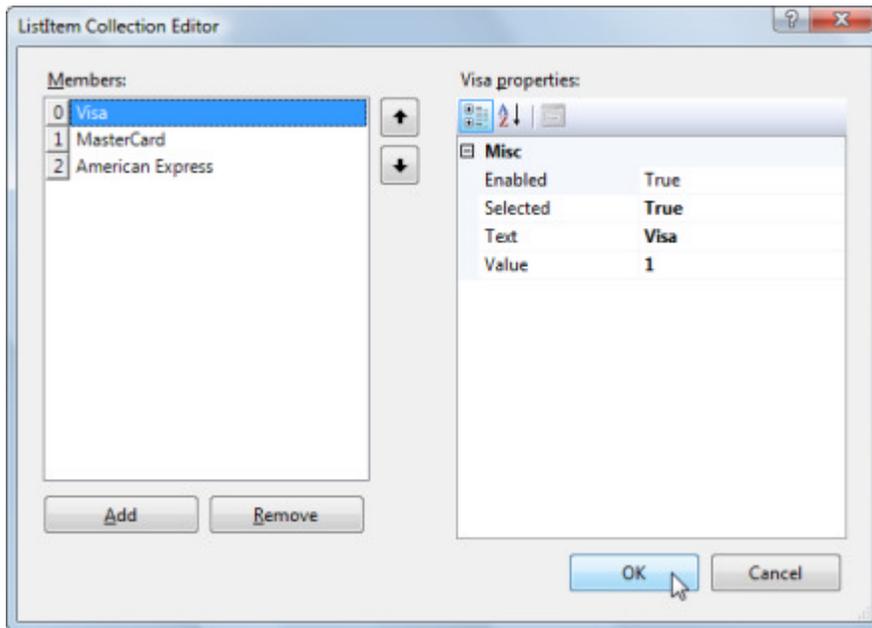
```

<table>
<tr> <!--Title-->
    <td></td>
    <td></td>
</tr>
<tr> <!--Share my results-->
    <td></td>
    <td></td>
</tr>
<tr> <!--Credit card type-->
    <td></td>
    <td></td>
</tr>
<tr> <!--Credit card number -->
    <td></td>
    <td></td>
</tr>
<tr> <!--Credit card name-->
    <td></td>
    <td></td>
</tr>
<tr> <!--Expire month and year-->
    <td></td>
    <td></td>
</tr>
<tr> <!--Validation summary-->
    <td></td>
    <td></td>
</tr>
</table>

```

4. Add the "Title" row controls:

- Enter "Billing Information" as a literal directly into the first cell.
5. Add the "Share my results" row controls:
- In the first cell of the "Share my results" row, add a standard ASP CheckBox control. Set the **ID** property to "cbShareMyResults". Set the **Text** property to "Share My Results".
 - Set the **AutoPostBack** property to **True**.
 - Set the **ToolTip** property to "Check this box to share your results with others".
 - Put the cursor after the checkbox and hit Enter to create a hard break (
).
 - Add a standard ASP Label control. Set the **ID** property to "lblShareMyResults". Set the **Text** property to "".
6. Add the "Credit Card Type" row controls:
- In the first cell of the "Credit Card Type" row, add a standard ASP RadioButtonList control. Set the **ID** property to "rblCardType".
 - Set the **AutoPostBack** property to **True**.
 - Set the **ToolTip** property to "Choose a credit card type".
 - Using the Smart Tag select **Edit Items**. Add three items setting the **Text** to "Visa", "MasterCard" and "American Express" respectively. Set the **Value** properties to "1", "2" and "3". Set the first item ("Visa") **Selected** property to True.
 - Click **OK** to close the collection editor



7. Add the "Credit Card Number" row controls:
- Add a **RadMaskedTextBox** to the first cell of the row. Set the **ID** property to "tbCCNumber". Set the **Label** property to "Credit Card Number:", the **Mask** property to "#####-####-####-####" and **Skin** to "Black". Set the **ToolTip** property to "Enter a valid credit card number".
 - Add a standard ASP **RequiredFieldValidator** to the second cell. Set the **ID** property to "valreqCCNumber". Set the **ControlToValidate** property to "tbCCNumber", **ErrorMessage** to "A valid credit card number is required", **Text** to "*" and **ValidationGroup** to "BillingGroup".
 - Add a second standard ASP **RegularExpressionValidator** to the second cell. Set the **ID** property to

"valregexCCNumber". Set the **ControlToValidate** to "tbCCNumber", the **ErrorMessage** to "Enter a valid credit card number", **Text** to "*" and **ValidationGroup** to "BillingGroup". Set the **ValidationExpression** to `^(4\d{3})|(5[1-5]\d{2})|(6011))-?\d{4}-?\d{4}-?\d{4}|3[4,7]\d{13}$`.

 The regular expression above is a relatively simple sample that can be used for this demo. You should research for your own regular expression based on your security needs.

8. Add the "Credit Card Name" row controls:
 - o Add a **RadTextBox** to the first cell of the row. Set the **ID** to "tbCCName". Set the **EmptyMessage** property to "Enter Name:" the **Label** to "Credit Card Name:" and **Skin** to "Black". Set the **ToolTip** property to "Enter the name on the credit card".
 - o Add a standard ASP **RequiredFieldValidator** to the second cell. Set the **ID** property to "valreqCCName". Set the **ControlToValidate** property to "tbCCName", **ErrorMessage** to "Enter the name on the credit card", **Text** to "*" and **ValidationGroup** to "BillingGroup".
9. Add the "Expire Month and Year" row controls:
 - o Add a **RadNumericTextBox** to the first cell of the row. Set the **ID** property to "tbExpMonth". Set the **Label** property to "Exp Month:", **MaxValue** to "12", **MinValue** to "1", **ShowSpinButtons** to True, and **Skin** to "Black". In the **NumberFormat** sub-properties set **AllowRounding** to False, **DecimalDigits** to "0". Set the **ToolTip** property to "Select the credit card expiration month".
 - o Add a second **RadNumericTextBox** to the first cell of the row. Set the **ID** property to "tbExpYear". Set the **Label** property to "Exp Year:", **ShowSpinButtons** to True, and **Skin** to "Black". In the **NumberFormat** sub-properties set **AllowRounding** to False, **DecimalDigits** to "0", **GroupSeparator** to "" and **PositivePattern** to "n". Set the **ToolTip** property to "Select the credit card expiration month".
10. In the last row add a standard ASP **ValidationSummary**. Set the **ValidationGroup** property to "BillingGroup".
11. Configure AJAX for the page:
 1. From the **RadAjaxManagerProxy** Smart Tag, click the **Configure Ajax Manager** link.
 2. Check "cbShareMyResults" as an initiating control. Check "lblShareMyResults" as the updated control.
 3. Check "rblCardType" as an initiating control. Check "tbCCNumber" as the updated control.



8.13 Add Utility Classes

The utility classes will include:

- A **CreditCardGroup** class to contain profile information for groups custom fields defined in web.config.
- A **WebProfile** class to contain profile information relating to each logged-in user. That information includes the **CreditCardGroup** and other single fields defined in web.config (e.g. "ShareMyResults").
- The **ASUser** class that contains the ASP.NET Membership object for the logged in user and the associated **WebProfile**.
- The **SessionManager** class used to make **ASUser** available anywhere in the application once the user has logged in.

1. In Solution Explorer, navigate to the ActiveSkillBO project, right-click References and select Add Reference. Locate System.Web in the list and click **OK** to add the System.Web assembly to the references list.
2. Create the WebProfile class. *The class file will contain a CreditCardType enumeration, a CreditCardGroup class and a WebProfile class.*
 1. Right-click the project and select **Add | New Class**. Name the class file WebProfile.cs.
 2. Add references to System.Web.Profile and System.Web.Security to the "Imports" (VB) or "uses" (C#) section of code.

[VB] Adding References

```
Imports System.Web.Profile
Imports System.Web.Security
```

[C#] Adding References

```
using System.Web.Profile;
using System.Web.Security;
```

3. Verify that the namespace for the class is Telerik.ActiveSkill.Common. *This should happen automatically if you set this namespace up as the default for the project during the Setup ActiveSkill Project Structure section earlier.*
4. Add a credit card type enumeration:

[VB] Adding the CreditCardType Enumeration

```
Public Enum CreditCardType
    Unassigned = 0
    Visa = 1
    MasterCard = 2
    Amex = 3
End Enum
```

[C#] Adding the CreditCardType Enumeration

```
public enum CreditCardType
{
    Unassigned = 0,
    Visa = 1,
    MasterCard = 2,
    Amex = 3
};
```

5. Add the CreditCardGroup class. *This class encapsulates the profile group named "CreditCard" that was added to web.config. The class has a property "CreditCardBase" that extracts the group of keys from web.config. The other properties extract the individual keys for easy access.*

[VB] Adding CreditCardGroup

```
''' <summary>
''' This class encapsulates the web.config
''' system.web/profile/properties/group settings
''' for the "Credit Card" group.
''' </summary>
Public Class CreditCardGroup
    Inherits ProfileGroupBase
```

```

Private _profileBase As ProfileBase
Public Sub New(ByVal profileBase As ProfileBase)
    _profileBase = profileBase
End Sub
Const TypeKey As String = "Type"
Public Property Type() As CreditCardType
    Get
        Return DirectCast(CreditCardGroupBase.GetPropertyvalue(TypeKey), CreditCardType)
    End Get
    Set
        CreditCardGroupBase.SetPropertyvalue(TypeKey, DirectCast(value, Byte))
    End Set
End Property
Const NumberKey As String = "Number"
Public Property Number() As String
    Get
        Return CreditCardGroupBase.GetPropertyvalue(NumberKey).ToString()
    End Get
    Set
        CreditCardGroupBase.SetPropertyvalue(NumberKey, value)
    End Set
End Property
Const NameKey As String = "Name"
Public Property Name() As String
    Get
        Return CreditCardGroupBase.GetPropertyvalue(NameKey).ToString()
    End Get
    Set
        CreditCardGroupBase.SetPropertyvalue(NameKey, value)
    End Set
End Property
Const ExpMonthKey As String = "ExpMonth"
Public Property ExpMonth() As String
    Get
        Return CreditCardGroupBase.GetPropertyvalue(ExpMonthKey).ToString()
    End Get
    Set
        CreditCardGroupBase.SetPropertyvalue(ExpMonthKey, value)
    End Set
End Property
Const ExpYearKey As String = "ExpYear"
Public Property ExpYear() As String
    Get
        Return CreditCardGroupBase.GetPropertyvalue(ExpYearKey).ToString()
    End Get
    Set
        CreditCardGroupBase.SetPropertyvalue(ExpYearKey, value)
    End Set
End Property
Const CreditCardGroupKey As String = "CreditCard"
Private ReadOnly Property CreditCardGroupBase() As ProfileGroupBase
    Get
        Return _profileBase.GetProfileGroup(CreditCardGroupKey)
    End Get
End Property

```

End Class

[C#] Adding CreditCardGroup

```
/// <summary>
/// This class encapsulates the web.config
/// system.web/profile/properties/group settings
/// for the "Credit Card" group.
/// </summary>
public class CreditCardGroup : ProfileGroupBase
{
    private ProfileBase _profileBase;
    public CreditCardGroup(ProfileBase profileBase)
    {
        _profileBase = profileBase;
    }
    const string TypeKey = "Type";
    public CreditCardType Type
    {
        get
        {
            return (CreditCardType)CreditCardGroupBase.GetPropertyvalue(TypeKey);
        }
        set
        {
            CreditCardGroupBase.SetPropertyvalue(TypeKey, (byte) value);
        }
    }
    const string NumberKey = "Number";
    public string Number
    {
        get
        {
            return CreditCardGroupBase.GetPropertyvalue(NumberKey).ToString();
        }
        set
        {
            CreditCardGroupBase.SetPropertyvalue(NumberKey, value);
        }
    }
    const string NameKey = "Name";
    public string Name
    {
        get
        {
            return CreditCardGroupBase.GetPropertyvalue(NameKey).ToString();
        }
        set
        {
            CreditCardGroupBase.SetPropertyvalue(NameKey, value);
        }
    }
    const string ExpMonthKey = "ExpMonth";
    public string ExpMonth
    {

```

```

    get
    {
        return CreditCardGroupBase.GetPropertyvalue(ExpMonthKey).ToString();
    }
    set
    {
        CreditCardGroupBase.SetPropertyvalue(ExpMonthKey, value);
    }
}
const string ExpYearKey = "ExpYear";
public string ExpYear
{
    get
    {
        return CreditCardGroupBase.GetPropertyvalue(ExpYearKey).ToString();
    }
    set
    {
        CreditCardGroupBase.SetPropertyvalue(ExpYearKey, value);
    }
}
const string CreditCardGroupKey = "CreditCard";
private ProfileGroupBase CreditCardGroupBase
{
    get
    {
        return _profileBase.GetProfileGroup(CreditCardGroupKey);
    }
}
}
}

```

6. Add the WebProfile class. This class will contain all the settings for a given profile. This would include the single setting for "ShareMyResults" and also the "CreditCardGroup". When the class is first created, a MembershipUser object is passed to it that ties the logged in user with their profile information. This class also persists the profile information when the user creates a new profile.

[VB] Adding the WebProfile Class

```

''' <summary>
''' This class contains all the settings for a given profile.
''' </summary>
Public Class WebProfile
    Private _profileBase As ProfileBase
    Public Sub New(ByVal user As MembershipUser)
        ' This next line is a key piece that ties together
        ' the logged in user and the profile. Using the
        ' HttpContext current user may be anonymous.
        _profileBase = ProfileBase.Create(user.UserName)
        _creditCardGroup = New CreditCardGroup(_profileBase)
    End Sub
    Private _creditCardGroup As CreditCardGroup
    Public ReadOnly Property CreditCard() As CreditCardGroup
        Get
            Return _creditCardGroup
        End Get
    End Property
    Const ShareMyResultsKey As String = "ShareMyResults"

```

```
Public Property ShareMyResults() As Boolean
    Get
        Return DirectCast(_profileBase.GetPropertyValue(ShareMyResultsKey), Boolean)
    End Get
    Set
        _profileBase.SetPropertyValue(ShareMyResultsKey, value)
    End Set
End Property
Public Sub Save()
    _profileBase.Save()
End Sub

End Class
```

[C#] Adding the WebProfile Class

```
/// <summary>
/// This class contains all the settings for a given profile.
/// </summary>
public class WebProfile
{
    private ProfileBase _profileBase;
    public WebProfile(MembershipUser user)
    {
        // This next line is a key piece that ties together
        // the logged in user and the profile. Using the
        // HttpContext current user may be anonymous.
        _profileBase = ProfileBase.Create(user.UserName);
        _creditCardGroup = new CreditCardGroup(_profileBase);
    }
    private CreditCardGroup _creditCardGroup;
    public CreditCardGroup CreditCard
    {
        get
        {
            return _creditCardGroup;
        }
    }
    const string ShareMyResultsKey = "ShareMyResults";
    public bool ShareMyResults
    {
        get
        {
            return (bool)_profileBase.GetPropertyValue(ShareMyResultsKey);
        }
        set
        {
            _profileBase.SetPropertyValue(ShareMyResultsKey, value);
        }
    }
    public void Save()
    {
        _profileBase.Save();
    }
}
```

3. Create the ASUser class. *The class file will encapsulate the ASP.NET Membership user information.*

1. Right-click the project and select **Add | New Class**. Name the class file ASUser.cs.
2. Add a reference to the System.Web.Profile namespace to the "Imports" (VB) or "uses" (C#) section of code.

[VB] Adding References

```
Imports System.Web.Security
```

[C#] Adding References

```
using System.Web.Profile;
```

3. Verify that the namespace for the class is Telerik.ActiveSkill.Common. *This should happen automatically if you set this namespace up as the default for the project during the Setup ActiveSkill Project Structure section earlier.*
4. Add the class implementation shown below. *The class includes properties for the ASP.NET membership, the web profile that in turn includes custom information about the logged in user. The class also provides easy access to the user ID and name.*

[VB] Implementing the ASUser Class

```
Public Class ASUser
    Private _membershipUser As MembershipUser
    Private _profile As WebProfile
    Public Sub New(ByVal userName As String)
        _membershipUser = Membership.GetUser(userName)
        _profile = New WebProfile(_membershipUser)
    End Sub
    Public ReadOnly Property MembershipUser() As MembershipUser
        Get
            Return _membershipUser
        End Get
    End Property
    Public ReadOnly Property UserName() As String
        Get
            Return _membershipUser.UserName
        End Get
    End Property
    Public ReadOnly Property UserID() As Guid
        Get
            Return DirectCast(_membershipUser.ProviderUserKey, Guid)
        End Get
    End Property
    Public ReadOnly Property Profile() As WebProfile
        Get
            Return _profile
        End Get
    End Property
End Class
```

[C#] Implementing the ASUser Class

```
public class ASUser
{
    private MembershipUser _membershipUser;
    private WebProfile _profile;
```

```
public ASUser(string userName)
{
    _membershipUser = Membership.GetUser(userName);
    _profile = new WebProfile(_membershipUser);
}
public MembershipUser MembershipUser
{
    get { return _membershipUser; }
}
public string UserName
{
    get { return _membershipUser.UserName; }
}
public Guid UserID
{
    get { return (Guid)_membershipUser.ProviderUserKey; }
}
public WebProfile Profile
{
    get { return _profile; }
}
}
```

4. Create the ASUser class. *The class file will encapsulate the ASP.NET Membership user information.*

1. Right-click the project and select **Add | New Class**. Name the class file ASUser.cs.
2. Add a reference to the System.Web.SessionState namespace to the "Imports" (VB) or "uses" (C#) section of code.

[VB] Adding References

```
Imports System.Web.SessionState
```

[C#] Adding References

```
using System.Web.SessionState;
```

3. Verify that the namespace for the class is Telerik.ActiveSkill.Common. *This should happen automatically if you set this namespace up as the default for the project during the Setup ActiveSkill Project Structure section earlier.*
4. Add the class SessionContext implementation shown below. *The class provides a convenient wrapper for the session in the HttpContext.*

[VB] Implementing the SessionContext Class

```
Public Class SessionContext
    Protected Shared ReadOnly Property Session() As HttpSessionState
    Get
        Return IIf(HttpContext.Current <> Nothing,HttpContext.Current.Session,Nothing)
    End Get
    End Property
End Class
```

[C#] Implementing the SessionContext Class

```
public class SessionContext
{
    protected static HttpSessionState Session
```

```

    {
        get
        {
            return HttpContext.Current != null ? HttpContext.Current.Session : null;
        }
    }
}

```

5. Add the SessionManager class implementation shown below. *This class only contains a single property for "User" so that it can be accessed throughout the application once the user logs onto the system.*

[VB] Implementing the SessionManager Class

```

Public Class SessionManager
    Inherits SessionContext
    Private Const _CurrentUserKey As String = "CurrentUserKey"
    Public Shared Property User() As ASUser
    Get
        Return DirectCast(Session(_CurrentUserKey), ASUser)
    End Get
    Set
        Session(_CurrentUserKey) = value
    End Set
End Property
End Class

```

[C#] Implementing the SessionManager Class

```

public class SessionManager: SessionContext
{
    private const string _CurrentUserKey = "CurrentUserKey";
    public static ASUser User
    {
        get
        {
            return (ASUser)Session[_CurrentUserKey];
        }
        set
        {
            Session[_CurrentUserKey] = value;
        }
    }
}

```

8.14 Configure the Profile

Add the following Profile element to the <system.web> section of web.config. *This step adds a profile with a custom property "ShareMyResults" and a group of properties to contain credit card information. Access to these properties will be encapsulated in utility classes we will define in later steps.*

[ASP.NET] Configuring Profile Properties

```

<!--RadControls for ASP.NET AJAX Step By Step-->
<profile enabled="true">
    <providers>
        <clear/>
        <add name="AspNetSqlProfileProvider"
            type="System.Web.Profile.SqlProfileProvider"

```

```
        connectionStringName="ActiveSkillConnectionString"
        applicationName="/ActiveSkill"/>
</providers>
<properties>
  <add name="ShareMyResults" type="System.Boolean"/>
  <group name="CreditCard">
    <add name="Type" type="System.Byte"/>
    <add name="Number" type="System.String"/>
    <add name="Name" type="System.String"/>
    <add name="ExpMonth" type="System.String"/>
    <add name="ExpYear" type="System.String"/>
  </group>
</properties>
</profile>
```

8.15 Summary

In this chapter you built the initial framework for a demonstration application that uses many of the RadControls for ASP.NET AJAX. You setup the project structure, learned how to setup and use ASP.NET Membership and finally used RadFormDecorator and RadInput controls.

9 Screen "Real Estate" Management

9.1 Objectives

- Examine how the "real estate" management controls can help you manage the content areas of your Web pages.
- Create a simple application to build confidence in using the controls.
- Become familiar with the design time support for working with the "real estate" management controls. This support includes Smart Tag, Properties Window, and some collection editors.
- Explore principal properties and groups of properties where 80% of the functionality is found.
- Learn how to perform common server-side tasks on the RadDock control.
- Learn how to use the client-side API to perform common tasks.
- Learn how to use these controls for more complicated tasks, such as creating dialogs and tool boxes or populating a portal page.

9.2 Introduction

The controls we will examine in this chapter are designed to help you manage the layout (or "real estate") of your Web pages. All of them define regions of the Web page where you can add the content you want to display. Some of these regions can move around the screen, others can be minimized or hidden away. By using these "real estate" controls, you can organize your Web pages and add flexibility that lets your users configure the layout in an individualized way.

RadWindow

RadWindow implements a pop-up window that displays content from an external source (another URL). You can use this control for a pop-up dialog or tool box, or simply as a secondary window for displaying additional content. Pop-up windows can be modal (disabling the rest of the page) or nonmodal (allowing the user to interact with the rest of the page while the pop-up is showing).

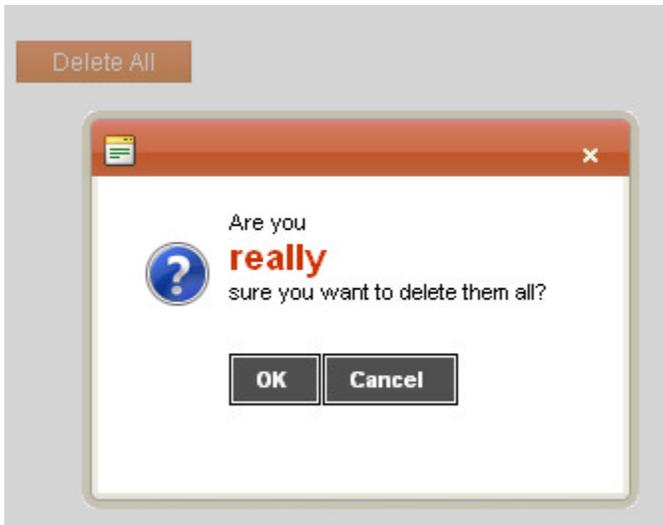


You have full control over what causes windows to appear, where they appear, and what size they start at. By specifying the icons that appear in the title bar, you can let users minimize, maximize, resize, move, pin, and close RadWindow controls with no coding on your part. If you don't want to allow users these capabilities, you can remove any or all of the controls from the title bar, or even hide the title bar entirely.

Unlike ordinary browser pop-up windows, RadWindow objects are not suppressed by the Windows XP SP2 pop-up blocker mechanism. Also unlike browser windows, you can minimize RadWindow pop-ups into minimization zones that you add to the parent window.

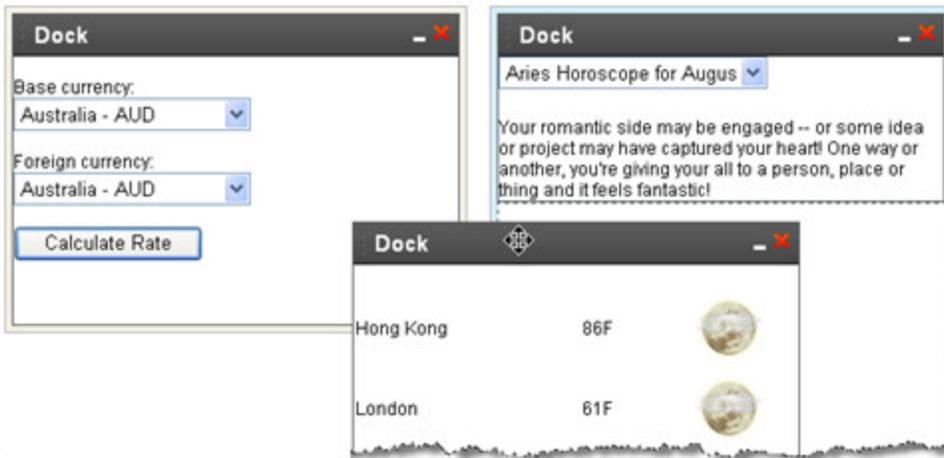
RadWindowManager

If your Web application uses multiple pop-up Windows, you can organize them using RadWindowManager. By using RadWindowManager, your application also has access to "rad" versions of the alert, confirm, and prompt pop-ups, so that you can control the appearance of these useful dialogs instead of relying on the built-in browser versions. Unlike the built-in browser versions, which are limited to displaying simple text, you can even add HTML content to the "rad" pop-ups:



RadDock and RadDockZone

RadDock appears similar to RadWindow, in that each RadDock control represents a movable "window" that contains content and that the user can drag around the Web page. Unlike RadWindow, however, RadDock displays content that is loaded with your Web page rather than from an external URL. RadDock windows can be docked into special zones, implemented by RadDockZone, in the way most portal sites let users configure the layout of controls.



As with RadWindow, you can control the command icons that appear in the RadDock title bar. RadDock windows have built-in commands for pinning and unpinning, expanding and collapsing, or closing the window, plus the ability to add your own custom commands. You can hide the RadDock title bar, replacing it with a simple grip for dragging and dropping:



RadDock windows can be configured so that they must be docked, must be floating, or so that they can move freely between the two states. You can also limit individual RadDock windows so that they can only be docked in certain zones.

RadSplitter

RadSplitter also creates separate regions for displaying content to users. Unlike RadWindow and RadDock, however, the content regions that RadSplitter uses are not pop-up windows. Instead, they are resizable frames, called panes, that divide up a region of the Web page. The splitter can be configured to lay out its panes either horizontally or vertically. By adding split bars between the panes, you can enable the user to resize panes in the browser. Alternately, you can leave out the split bars, to create a static layout of separate panes on your Web page. In a splitter that contains split bars, individual panes can be "locked", so that they are not resizable along with the other panes of the splitter.

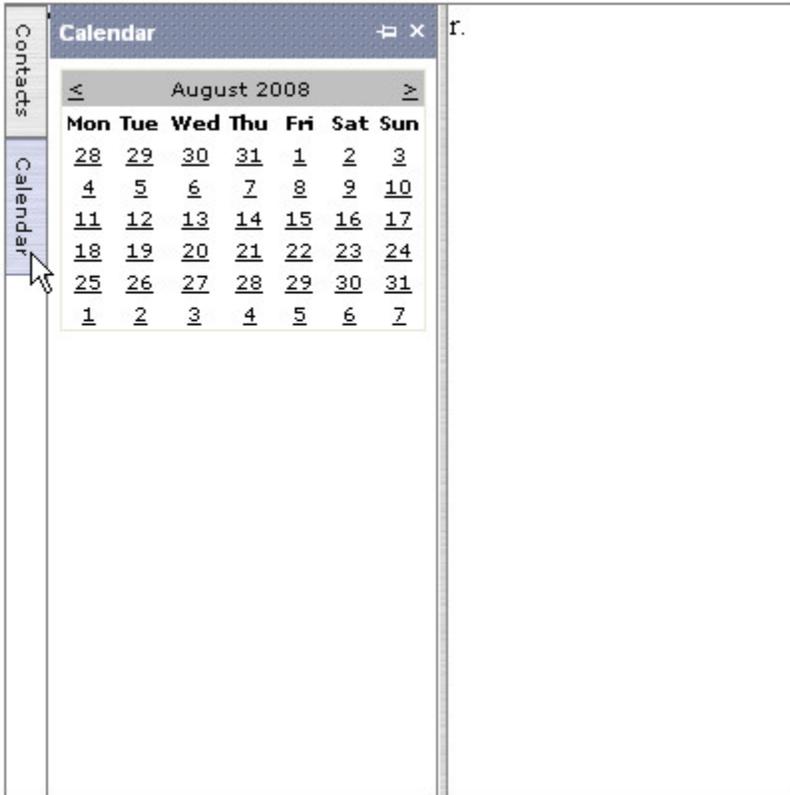
Panes can display content from an external URL, like RadWindow, or content that is loaded with the Web page, like RadDock. The screenshot below shows a splitter that displays a radio button list that is loaded with the Web page in the left pane, and content from an external Web site in the right pane:



Panes that load their content with the Web page can hold any HTML elements, even another splitter. By nesting splitters with alternating horizontal and vertical orientations, you can create arbitrarily complex layouts.

RadSlidingZone

RadSlidingZone is a specialized control for optimizing layout that can only be placed directly inside the pane of a splitter. RadSlidingZone implements a set of tabs that can be used to slide out additional panes, called sliding panes, similar to the way Visual Studio lets you slide out panels such as the Properties Window or Solution Explorer. Like the sliding panels in Visual Studio, the sliding panes of a RadSlidingZone control can be docked in place by the user. By defining sliding panes in a sliding zone container, you can initially hide content that your users do not need to see all the time.



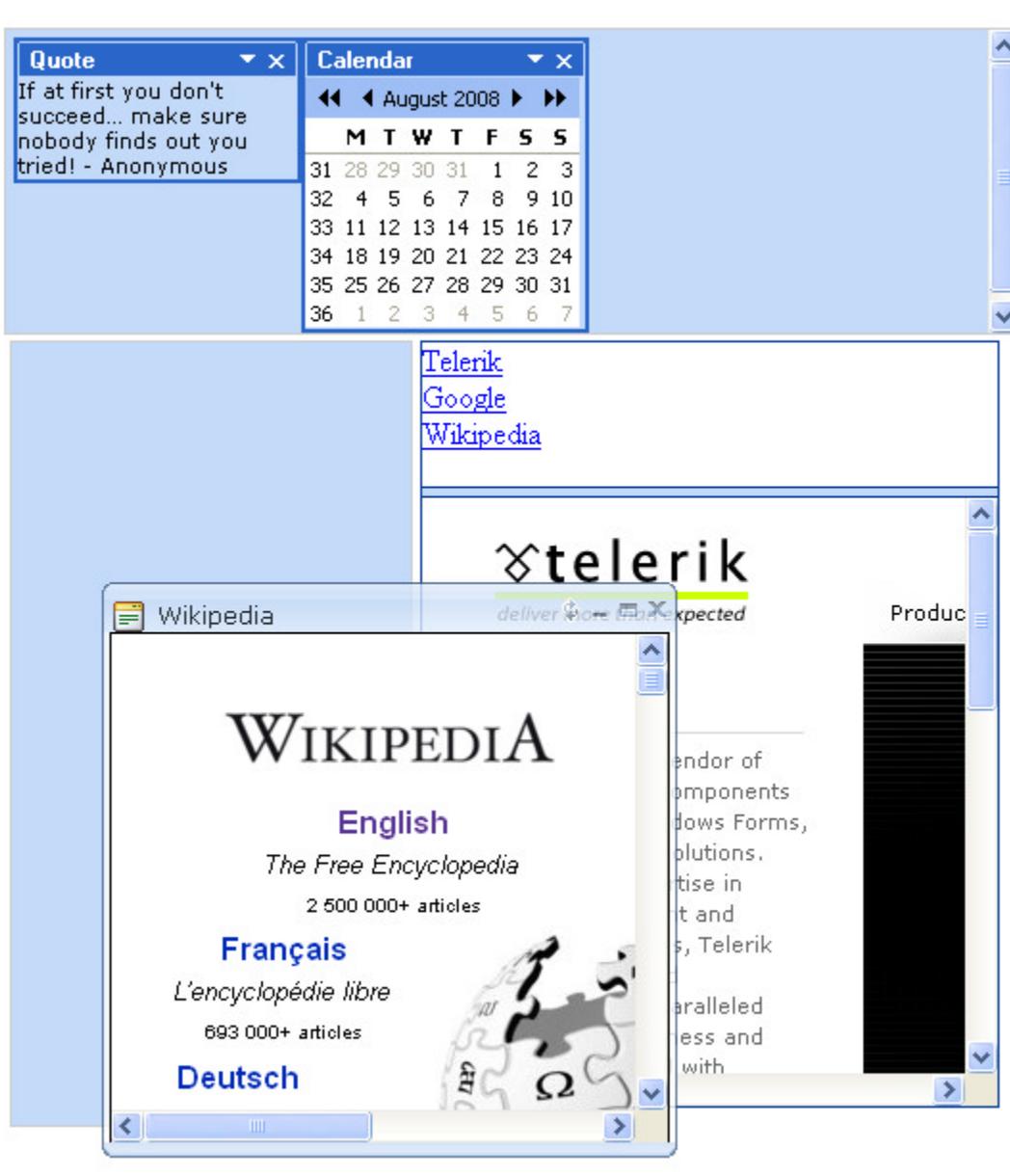
You can configure the orientation of the sliding zone and whether sliding panes expand when the user moves the mouse over their tabs or whether the user must click on a tab to expand it. Individual tabs can be configured to display text, an icon, or both. Sliding panes can be fixed in size, or resizable in the direction that they expand. You can also suppress the ability of the user to dock individual sliding panes.

9.3 Getting Started

In this walk-through you will become familiar with the following "real estate" controls:

- RadDockZone and RadDock
- RadSplitter, RadPane, and RadSplitBar
- RadWindowManager and RadWindow

These controls will be used to generate the layout shown in the following screen shot:



☑ We will look at RadSlidingZone and RadSlidingPane later, in the Control Specifics section.



You can find the complete source for this project at:
 \VS Projects\RealEstate\GettingStarted

Set up the project structure

1. Create a new ASP.NET Web Application.
2. Drag a ScriptManager from the Tool Box onto the Web page.

Add the RadDock controls

1. Drag a RadDockZone control onto your Web page. Using the Smart Tag, set the Skin property to "Outlook".

- Using the Properties Window, set the **Height** property to "150px", the **Width** property to "100%" and the **Orientation** property to "Horizontal".
- Drag a **RadDock** control from the Tool Box onto the surface of the RadDockZone. Use the Smart Tag to make sure that the **Skin** property is set to "Outlook".
- Using the Properties Window, set the **Width** property to "150px", the **Title** property to "Quote" and the **Text** property to "If at first you don't succeed... make sure nobody finds out you tried! - Anonymous".
- Drag a second RadDock control onto the surface of the RadDockZone. On this one, set the **Skin** property to "Outlook", the **Width** to "150px", the **Title** to "Calendar", and the **DockMode** to "Docked". That last property (DockMode) stipulates that this RadDock must always be docked in a dock zone. The last RadDock control left the value of DockMode as "Default", meaning that RadDock control can be dragged out of the docking zones to act as a free-standing window.
- Drag a **RadCalendar** control from the Tool Box onto the surface of the second RadDock control. Set its Skin property to "Outlook".
- Switch to the Source window, and make sure that the RadCalendar control is surrounded by `<ContentTemplate>` and `</ContentTemplate>` tags. It is a known issue about RadDock that it sometimes loses these tags in the designer. The second RadDock control should now look like the following:

[ASP.NET] Second RadDock control

```
<telerik:RadDock ID="RadDock2" Runat="server"
Skin="Outlook" Width="235px"
Title="Calendar" DockMode="Docked">
<ContentTemplate>
  <telerik:RadCalendar ID="RadCalendar1" Runat="server"
font-names="Arial, Verdana, Tahoma" forecolor="Black"
Skin="Outlook" style="border-color:#ecec">
  </telerik:RadCalendar>
</ContentTemplate>
</telerik:RadDock>
```

 The `<ContentTemplate>` tag signals that the calendar is part of a template. You do not need to worry about templates right now, as this one is pretty simple, but if you want to learn more about them, look ahead at the chapter on Templates.

- Drag a **Table** from the **HTML** section of the tool box to below the first RadDockZone.
- Select the upper-left cell in the table, right click, and choose Delete|Delete Rows to delete the first row. Repeat this process until the table has only one row.
- Select the upper-left cell in the Table, right click, and choose Delete|Delete Cell to delete that cell. The table should now have two columns. (If it has more, continue deleting until there are only two columns.)
- Select the lefthand cell of the table and use the Properties Window to set its **Width** attribute to "200px".
- Drag a second **RadDockZone** from the Tool Box into the lefthand cell of the table. Set its **Skin** property to "Outlook", **Height** to "400px", **Width** to "200px" and **FitDocks** to false. The last property (FitDocks) ensures that when RadDock controls are docked in this zone, they do not get resized to the width of the dock zone, but instead keep their original width.

Add the RadSplitter controls

- Drag a **RadSplitter** control from the Tool Box onto the right-hand cell of the table.
- Using the Smart Tag, set its **Skin** property to "Outlook".
- Using the Properties Window, set the **Orientation** property to "Horizontal" and the **Width** to "100%"
- Drag a **RadPane** control from the Tool Box onto the surface of the RadSplitter. Set its **Height** property to "75px".
- Drag a **LinkButton** from the Tool Box onto the surface of the RadPane control. Set its **Text** property to

"Telerik".

6. Hit the **Enter** key to add a line break after the link button, and add a second **LinkButton** below the first. Set this one's **Text** property to "Google".
7. Hit the **Enter** key again and add a third **LinkButton**. Set its **Text** property to "Wikipedia".
8. Drag a **RadSplitBar** control from the Tool Box onto the surface of the **RadSplitter**, below the **RadPane** you just filled. This can be a little tricky, so feel free to use the Source view to move it into place if you need to.
9. Drag a second **RadPane** control onto the **RadSplitter**. Once again, this can be a bit tricky, so check in the Source view to make sure that it landed in the right place. At this point, your **RadSplitter** declaration should look like the following:

[ASP.NET] RadSplitter declaration

```
<telerik:RadSplitter ID="RadSplitter1" Runat="server"
  Orientation="Horizontal" Skin="Outlook" Width="100%">
  <telerik:RadPane ID="RadPane1" Runat="server" Height="75px">
    <asp:LinkButton ID="LinkButton1" runat="server">Telerik</asp:LinkButton><br />
    <asp:LinkButton ID="LinkButton2" runat="server">Google</asp:LinkButton><br />
    <asp:LinkButton ID="LinkButton3" runat="server">Wikipedia</asp:LinkButton>
  </telerik:RadPane>
  <telerik:RadSplitBar ID="RadSplitBar1" Runat="server" />
  <telerik:RadPane ID="RadPane2" Runat="server">
  </telerik:RadPane>
</telerik:RadSplitter>
```

10. Using the Properties Window, set the **Skin** property of the second **RadPane** control to "Outlook" and the **ContentUrl** property to "http://www.telerik.com".



Gotcha! When assigning the **ContentUrl**, be sure to type the entire URL including the "http://" and avoid the "Resource not found" error.

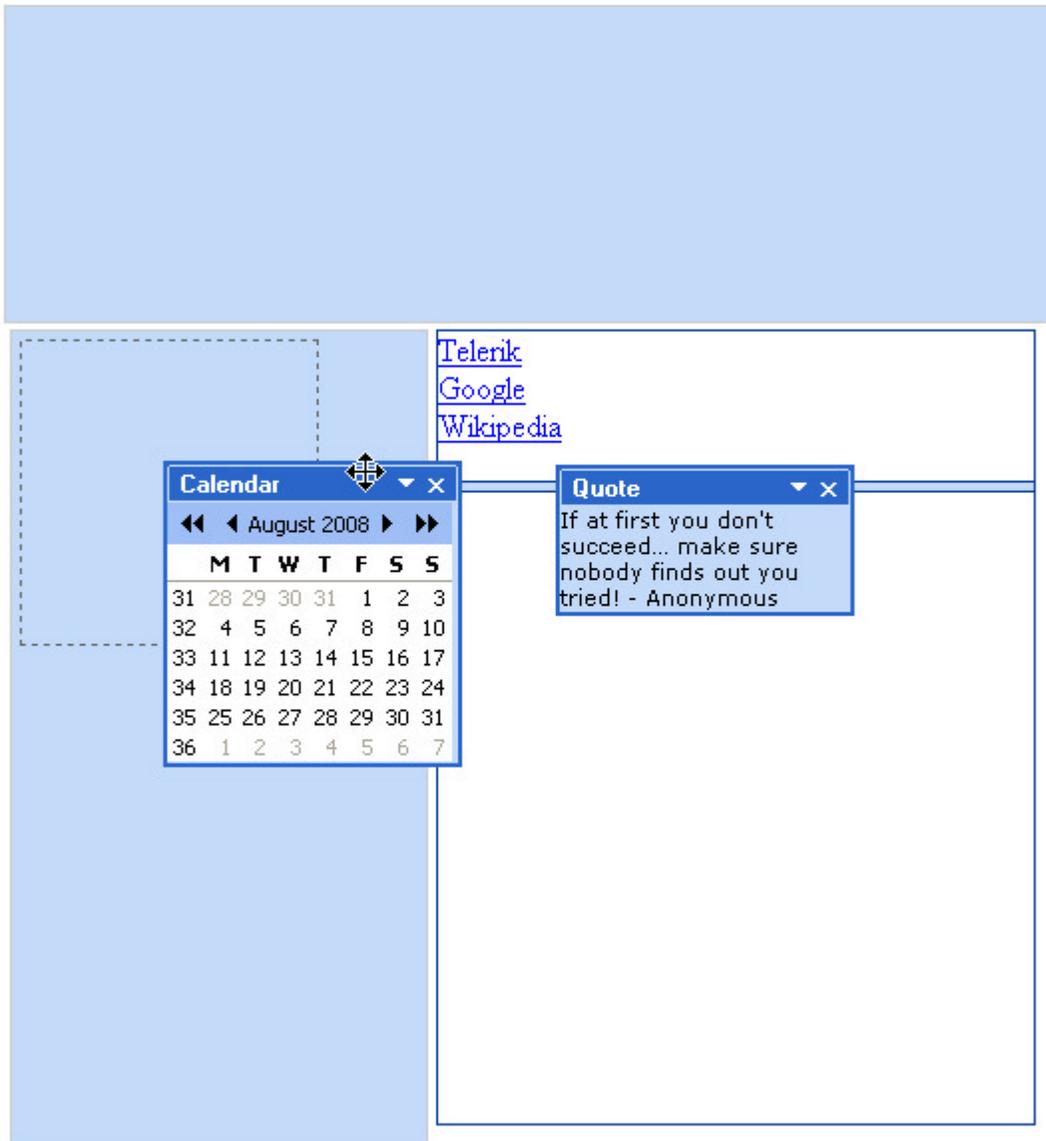
Add the RadWindowManager and RadWindow controls

1. Drag a **RadWindowManager** control from the Tool Box and place it at the bottom of the Web page.
2. Using the Properties Window, set the **Skin** property to "Outlook".
3. Click on the Configuration Manager in the Smart Tag to bring up the **RadWindow** collection editor.
4. In the Collection Editor, click the **Add** button to add a **RadWindow** to the **RadWindowManager**'s **Windows** collection.
5. Assign the following properties to the new window you just added:
 - Set **Animation** to "Fade".
 - Set **Behaviors** to "Resize, Minimize, Close, Maximize, Move, Reload".
 - Set **OpenerElementID** to "LinkButton1".
 - Set **Title** to "Telerik".
 - Set **VisibleStatusBar** to `false`.
 - Set **NavigateUrl** to "http://www.telerik.com". As before, be sure to include the entire URL.
6. Click the **Add** button again to add a second **RadWindow** to the collection. Set the following properties:
 - Set **Animation** to "FlyIn".
 - Set **Behaviors** to "Resize, Minimize, Close, Maximize, Move, Reload".

- Set **OpenerElementID** to "LinkButton2".
 - Set **Title** to "Google".
 - Set **VisibleStatusBar** to false.
 - Set **NavigateUrl** to "http://www.google.com".
7. Click the Add button again to add a third RadWindow to the collection. Set the following properties:
- Set **Animation** to "Resize".
 - Set **Behaviors** to "Resize, Minimize, Close, Maximize, Move, Reload".
 - Set **OpenerElementID** to "LinkButton3".
 - Set **Title** to "Wikipedia".
 - Set **VisibleStatusBar** to false.
 - Set **NavigateUrl** to "http://www.wikipedia.org".
8. Click Ok to exit the collection editor.

Run the application

1. Press Ctrl-F5 to run the application. The two dock zones appear at the top and left of the browser window. The top dock zone contains the two RadDock windows. Between the two dock zones, the splitter sits with two panes: an upper pane with three link buttons, and a lower pane that displays the Telerik Web site.
2. Experiment with dragging the RadDock windows. You can drag the Quote window anywhere, but the Calendar window snaps back to its original position unless you drop it in a dock zone.



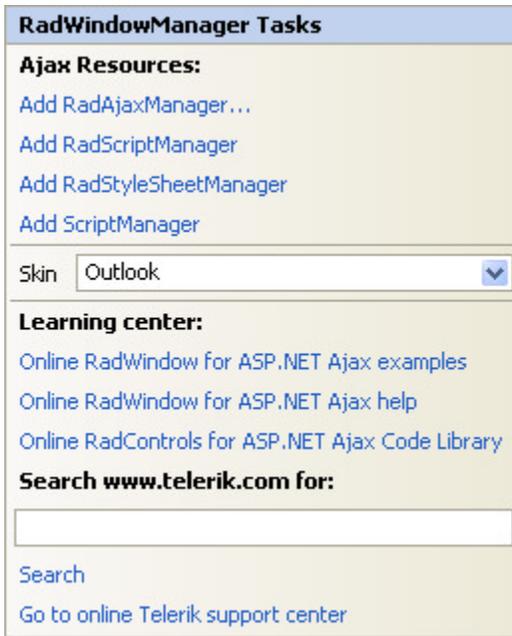
3. Experiment with the splitter by dragging on the split bar.
4. Click on the three link buttons to display the RadWindow controls. Note the different animation effects as they appear.

9.4 Designer Interface

In the Visual Studio designer, you can configure the "real estate" management controls using the Smart Tag and the Properties Window. In addition, you can use collection editors with the RadDockZone, RadDock, and RadWindowManager controls to set property collections.

Smart Tag

The Smart Tag for each of the "real estate" management controls contains only the common elements of RadControls Smart Tags: the Ajax Resources, Skin selection, and Learning center:



The **Skin** property for these controls takes a bit of explanation. As you saw in the project you built in the Getting Started section, working with the "real estate" management controls involves nesting controls inside controls: you nested RadDock controls inside RadDockZone, RadWindow controls inside RadWindowManager, and RadPane and RadSplitBar controls inside RadSplitter. In all cases, the controls have a Skin property, even when (as in the case of RadWindowManager) they have no visual aspect on the Web page. For most of these controls, setting the Skin property of the parent control changes the default skin for all of its children. Thus, when you set the Skin property for RadWindowManager, the skin was inherited by all the child RadWindow controls, and when you set the Skin property for RadSplitter, the skin was inherited by the child RadPane and RadSplitBar controls. (This did not occur for RadDockZone and RadDock, but in the next section we will encounter another control, RadDockLayout, which *does* set the default skin for child RadDockZone and RadDock controls).

📌 Even if the skin is inherited from a parent control, an individual child control can override that default by setting its own **Skin** property.

Properties Window

At design time, most of the work you do to configure these controls can be done using the Properties Window. As before, let us look at the most important properties of the controls.

RadWindowManager

The most important property of RadWindowManager is the **Windows** property. This holds the set of child windows the manager controls. In the Properties Window, you can use the Windows property to bring up the RadWindow Collection Editor to add and configure each window in this collection.

📌 In addition to the **RadWindow** controls in the Windows property collection, you can use RadWindowManager to generate additional RadWindow controls from client-side code.

RadWindow

When RadWindow controls are added as children of RadWindowManager (using the **Windows** property collection), you can use either the Properties Window or the RadWindow Collection Editor to set their properties. If you add RadWindow controls to a page without using RadWindowManager, you can use the Properties Window to configure them.

💡 It is not necessary to use RadWindowManager with your RadWindow controls unless you want to use certain parts of the client-side api. It does, however, provide a convenient place to keep all of your RadWindow

controls at design time.

Because RadWindow must load its content from an external source, the most important property is the **NavigateUrl** property. This property specifies the URL where the window gets its contents. Its value can be another page in the same project (e.g. "MyPage.aspx"), or a reference to another server (e.g. "http://www.google.com"). When using a reference to another server, you must always include the entire URL, including the "http://".

By default RadWindow controls surround their content with a title bar at the top and a status bar at the bottom. You can hide either of these using the **VisibleTitleBar** and **VisibleStatusBar** properties. The title bar displays the window title and a set of command buttons. You can specify the window title by setting the **Title** property, or leave the window to pick up its title from the title attribute of its content. The command buttons that appear are controlled by the **Behaviors** property. Possible buttons include "Close", "Pin", "Minimize", "Maximize" and "Reload". If you are adding more than one type of button, separate values with commas. Behaviors can also include some values that do not result in buttons: "Move" and "Resize". When these values are included, the user can move or resize the window with the mouse. A related property, **InitialBehaviors**, can be set to a subset of Behaviors, made up from the "Pin", "Minimize", and "Maximize" options to specify the initial state of the window.

Two properties let you specify when the window should appear if you are not using the client-side api to display it: **VisibleOnPageLoad** lets you have the window pop up when the page loads, and **OpenerElementID** lets you specify a control on the page that causes the window to open when the user clicks it.

You can control where on the page the window appears using the **Top**, **Left**, and **OffsetElementID** properties.

The **Modal** property lets you use the window as a modal dialog. When Modal is true, the parent page is disabled while the window is showing and it is opened centered on the screen.

Since Q1 2011, RadWindow has WAI-ARIA support - it can be enabled by setting **EnableAriaSupport="true"**.

RadWindow also supports **MaxWidth**, **MaxHeight**, **MinWidth** and **MinHeight** properties - if they are set, when you resize it they will be respected

Note 1: There are default minimum width and height, determined by the icons in titlebar, etc.; if we set less size, the visual appearance of the RadWindow is spoiled. That is why, if you set MinWidth or MinHeight less than them, the default absolute minimums will be set

Note 2: The AutoSize functionality respects the MaxWidth, MaxHeight, MinWidth and MinHeight properties.

Note 3: If you set Width(Height) bigger than the MaxWidth(MaxHeight) or less than MinWidth(MinHeight) you have set, the Width and Height properties will be modified in order to respect the set limits.

Note 4: The client-side width and height setters respect the set limits.

RadDockZone

The most important property on RadDockZone is the **Orientation** property. It specifies how the dock zone lays out its docked controls.

- When Orientation is "Vertical" (the default), docked controls are added one below the other in a single column. If this takes up more space than the value specified by its **Height** property, the dock zone acquires a scroll bar. If Height is not set, the dock zone expands to fit its docked controls. The **FitDocks** property specifies whether docked controls are resized when docked so that they fit exactly in the **Width** of the dock zone.
- When Orientation is "Horizontal", docked controls are added in rows. When a row is filled (the control would exceed the value of the dock zone's **Width** property), a new row is started. As with a vertical dock zone, the dock zone can sprout a scroll bar if the **Height** property is exceeded, or grow to fit if the Height property is not set. (The FitDocks property has no effect on a horizontal dock zone).

When the **Width** or **Height** property is not set, the dock zone expands and contracts to accommodate its docked controls. You can place a limit on how small the dock zone gets in this case by setting the **MinWidth** and **MinHeight** properties.

While the default appearance of the dock zone is controlled by its skin, you can augment this by setting the **CssClass** property. If you want to change the appearance of the dock zone when it can accept a dragged RadDock object, you can use the **HighlightedCssClass** property.

The **Docks** property collection lists the RadDock controls currently docked in the dock zone. In the properties window, you can click the ellipsis button for this control to bring up the RadDock Collection Editor. You can use the collection editor to add RadDock controls at design time, or to change the properties of the RadDock controls that start out docked in the zone.

RadDock

RadDock controls can display any HTML content, including other ASP.NET controls. The simplest type of content you can specify is a string of text. To populate a RadDock control with text, simply set the **Text** property. To add more complex elements, you can't use the Properties Window; instead, use the Visual Studio designer to create a **ContentTemplate**. A simple example of this process was shown in the Getting Started project.

 You can also add content to a RadDock control in server-side code. An example of this is shown in the section on Server-Side Programming.

You can use the **Title** property to specify the title that appears in the RadDock title bar. By default, this title bar is where the user clicks and drags to move the RadDock control around the Web page. If you want to hide the title bar, you can change the **DockHandle** property from "TitleBar" to "Grip", and the title bar is changed to a small grip area at the top of the control. (You can remove even that small grip area by changing the **DockHandle** property to "None", but then the user has no way to drag the control unless you provide it using client-side code).

In addition to the title string, the title bar also displays a set of command buttons. There are three basic built-in command buttons: a Close button that is always present until the user clicks on it to hide the RadDock control, an Expand/Collapse button that lets the user minimize the dock window so that it hides its content or restore it so that it displays its content, and a Pin/Unpin button that appears when the dock window is not docked. You can use the **DefaultCommands** property to specify which of these built-in commands you want to have appear on the title bar. You can choose a single command, list two commands separated by a comma, or set **DefaultCommands** to "All" or "None". You can augment this list by adding your own custom commands using the **Commands** property. When adding custom commands, you can implement their behavior on the client-side by assigning a function to the **OnClientCommand** event handler, or you can implement them on the server-side by setting the **AutoPostBack** property of the command and supplying a handler for the server side **Command** event. When you set the **Commands** property, the **DefaultCommands** property is ignored, so be sure to add any built-in commands you want to the **Commands** collection as well.

There are two properties that affect where the user can drag a RadDock control:

- **DockMode** specifies whether the dock window must always be docked in one of the dock zones ("Docked"), whether it can never be docked but must always float ("Floating") or whether it can move freely between the two states ("Default").
- **ForbiddenZones** is a comma-separated list of IDs for all the dock zones where the dock window cannot be docked. By using forbidden zones, you can arrange your Web page with different functional docking areas.

RadSplitter

The most important property on RadSplitter is **Orientation**. Orientation can be "Horizontal", in which case the splitter lays out its panes in a single column, or "Vertical", in which case the panes are laid out in a single row. That is, Orientation refers to the direction of the split bars between panes, rather than the direction in which panes are laid out. If the splitter contains split bars between panes, the split bars resize the **Height** of panes in a horizontal splitter, and the **Width** of panes in a vertical splitter.

The **ResizeMode** property lets you configure how panes resize when the user drags on a split bar. **ResizeMode** can be "AdjacentPane", in which case only the panes adjacent to a split bar are resized when the bar is

dragged, "EndPane", in which case only the pane immediately before the split bar and the rightmost (or bottom) pane are affected, or "Proportional", in which case the pane immediately before the split bar is resized and all panes that follow the split bar divide up the remaining space according to their current proportions.

The **ResizeWithBrowserWindow** and **ResizeWithParentPane** properties let you control whether the size of the splitter (and hence the size of the panes it contains) is changed when its container is resized.

ResizeWithBrowserWindow is for splitters that sit on the Web page, while **ResizeWithParentPane** affects splitters that are nested in a pane on another splitter.

RadPane

RadPane represents one of the panes laid out by a splitter. It can hold any HTML content that you add in the designer (including another splitter), or it can display external content (like **RadWindow**) if you set the **ContentUrl** property.

When specifying the dimensions of a pane, you only need to specify the size in one direction, because the pane matches the size of the splitter in the other direction. Thus, in a horizontal splitter, you only set the **Height** of a pane, while in a vertical splitter, you only set its **Width**. Because users can resize the panes of a splitter, you can place limits on how much a pane can be resized by setting the **MinHeight** and **MaxHeight** or **MinWidth** and **MaxWidth** properties. You can prevent the splitter from resizing a pane by setting the **Locked** property to true.

When a pane is resized so that it is too small to display all of its content, it can either crop its display, or display scroll bars. The **Scrolling** property lets you specify which option is used. Scrolling can be "None" (always crop), "X" (display horizontal scroll bars but crop vertically), "Y" (display vertical scroll bars but crop horizontally), or "Both" (display both vertical and horizontal scroll bars as necessary).

In addition to resizing panes, split bars also include the ability to collapse and restore adjacent panes. You can use the **Collapsed** property of a pane to specify that it starts out in the collapsed state when the page first loads.

RadPane has built-in ability to show a loading sign while a content page set through the **ContentUrl** property is being loaded in it. To turn on this functionality you should set **ShowContentDuringLoad** to **false** (the default value is true).

RadSplitBar

RadSplitBar has two important properties that determine how it can influence adjacent panes:

- **CollapseMode** specifies whether the split bar has the ability to collapse and restore adjacent panes. It can be "None", "Forward" (it can only collapse and restore the next pane), "Backward" (it can only collapse and restore the previous pane), or "Both" (it can collapse and restore both the next and previous panes). When you set **CollapseMode** to let the split bar collapse and restore adjacent panes, it acquires one or two collapse buttons. When the adjacent pane is collapsed, the collapse button changes to a restore button.
- **ResizeStep** lets you configure the split bar so that it only resizes adjacent panes in fixed increments. **ResizeStep** is the size, in pixels, of one increment.

Collection Editors

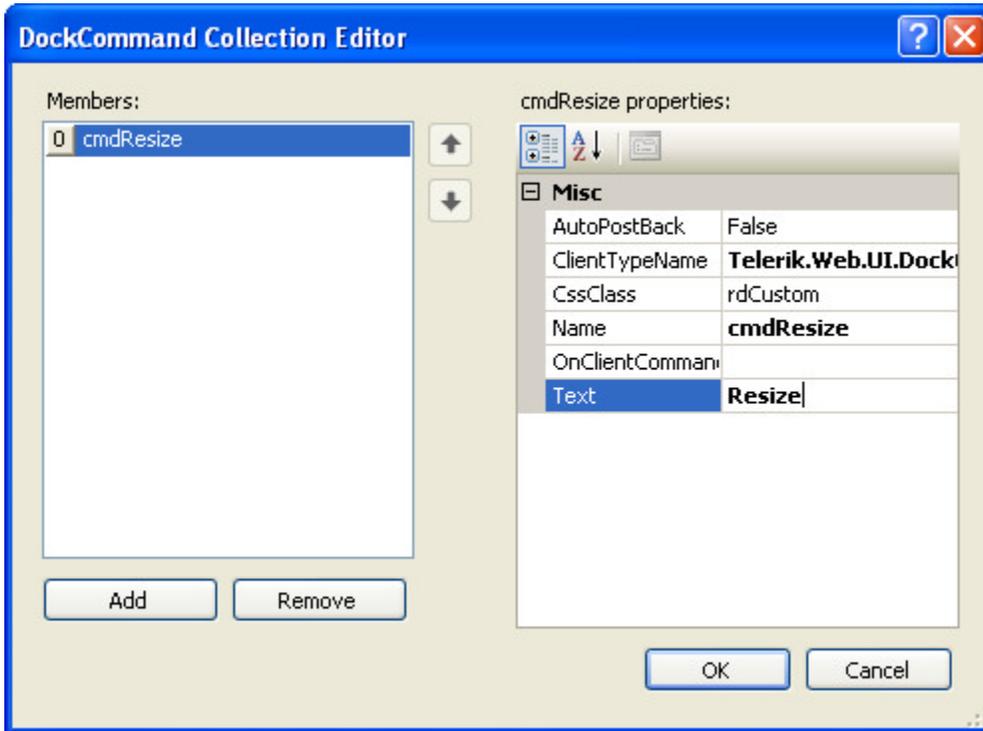
All collection editors work essentially the same way, with an Add button to add items to the collection and a Remove button to remove the selected item, up and down arrow buttons to rearrange items, and a properties pane to set the properties of the currently selected item. You display the collection editor by clicking the ellipsis button in the Properties Window next to the property whose value is a collection.

Some of the "real estate" management controls have collection properties that let you use a Collection Editor:

- On **RadWindowManager**, the **Windows** property brings up the **RadWindow Collection Editor** where you can add and remove windows and set their properties. You can also use the **CofigurationManager** from the **SmartTag**.
- On **RadDockZone**, the **Docks** property brings up the **RadDock Collection Editor**, where you can edit the

RadDock controls that appear docked in the zone when the page loads.

- On RadDock, the **Commands** property brings up the **DockCommand Collection Editor**, where you can add custom commands.



We have already looked at the main properties of **RadWindow** and **RadDock**, but it is worthwhile, at this point, to look at the properties of the commands you can add using the **DockCommand Collection Editor**.

When adding commands to the Commands collection, use the **ClientTypeName** property to specify the type of a command. Remember that when you set the value of **Commands**, the **DefaultCommands** property is ignored, so add in any built-in commands you want to include. You can use any of the following types:

- **Telerik.Web.UI.DockPinUnpinCommand**: the built-in pin/unpin command.
- **Telerik.Web.UI.DockExpandCollapseCommand**: the built-in expand/collapse command.
- **Telerik.Web.UI.DockCloseCommand**: the built-in close command.
- **Telerik.Web.UI.DockCommand**: the default class for custom commands with one state (like the built-in close command).
- **Telerik.Web.UI.DockToggleCommand**: the default class for custom commands with two states (like the expand and collapse states of the built-in expand/collapse command).

For custom commands, you will want to set some other properties besides the **ClientTypeName**:

If you want to implement the behavior of your custom command in server-side code, set the **AutoPostBack** property to true and provide a handler for the server-side **Command** event. The Command event handler is passed the value of the **Name** property in its arguments so that you can identify which command generated the postback if you have multiple custom commands.

If you want to implement the behavior of your custom command in client-side code, leave the **AutoPostBack** property set to false and set the **OnClientCommand** property to the name of a client-side function that is called when the user clicks on the command icon.

The **Text** property specifies the text of the tool tip that appears when the mouse hovers over the command icon and the **CssClass** lets you change the appearance of the button from the built-in icon supplied by the skin.

When adding a custom toggle command, there are a few more properties you will probably want to set that are not available using the DockCommand Collection Editor. To set these properties, switch to the Source window, and edit the HTML markup for the commands. These properties are the **AlternateText** property (the text of the tool tip for the alternate state of the command), the **AlternateCssClass** property (to set the appearance of the button for the alternate state), and the **State** property (to specify whether the command starts in its primary or alternate state).

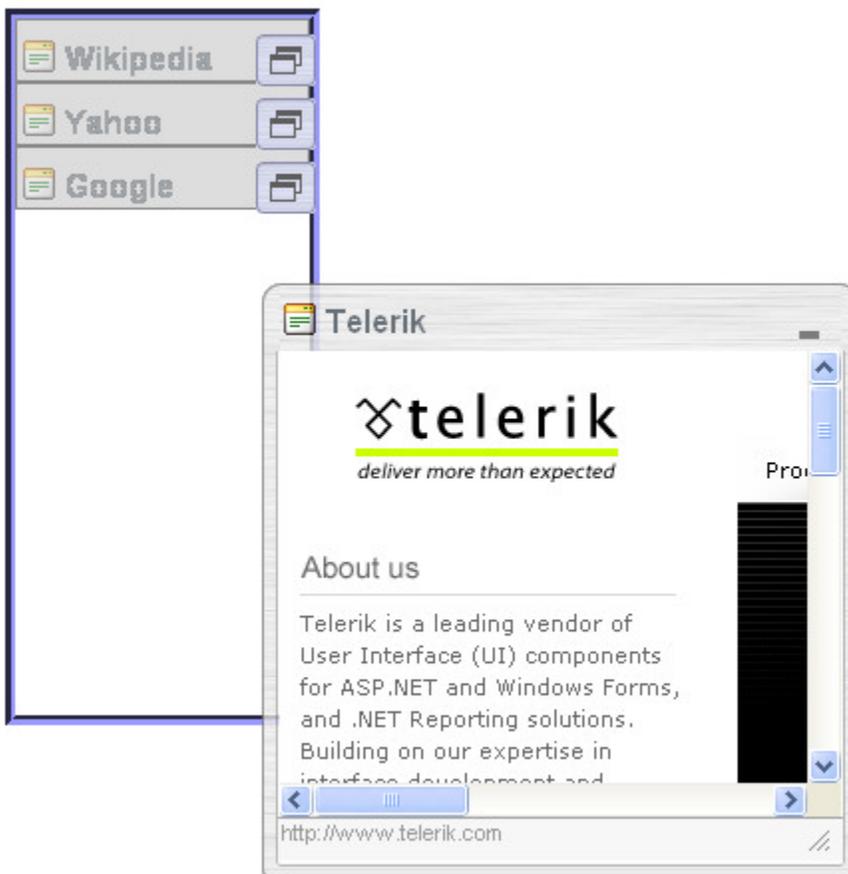
9.5 Control Specifics

Minimize Zones

As you have seen, unlike the pop-up windows in most Web applications, RadWindow-based pop-up windows can be minimized as long as the **Behaviors** property includes "Minimize". By default, when the user clicks on the minimize icon on the window's title bar, the window is minimized in its current location. If your Web page includes many pop-up windows, this can get messy and unmanageable. To help with this issue, you can create "minimize zones", which hold all the minimized windows. When the windows are restored from the minimized state, they return to their previous position and size. By using minimize zones, you can enable your users to organize their pop-up windows on the Web page using a familiar metaphor: the task bar that is available for desktop applications.

Any control that can contain child controls, such as a panel or <div> can act as a minimize zone. All you need do is set the **MinimizeZoneID** property of the **RadWindow** control or, if you are using one, the **RadWindowManager** control, to the ID of the element that you want to have act as a minimize zone.

The following walk-through uses an ASP Panel control as a minimize zone.





You can find the complete source for this project at:
\\VS Projects\RealEstate\MinimizeZones

1. Create a new ASP.NET Web Application.
2. Drag a ScriptManager from the Tool Box onto the Web page.
3. Drag a Panel from the Standard section of the Tool Box onto the Web page. Set its **Height** property to "350px" and its **Width** property to "200px".
4. Give the Panel a border by setting the **BorderStyle** property to "Groove", the **BorderWidth** property to "5px" and the **BorderColor** property to "#9999FF".
5. Drag a RadWindowManager onto the Web Page below the Panel.
 - Set the **Behaviors** property to "Resize, Minimize, Move".
 - Set the **OffsetElementID** property to "Panel1".
 - Set the **VisibleOnPageLoad** property to true.
 - Set the **MinimizeZoneID** property to "Panel1". This sets the panel as the minimize zone for all the windows in the window manager's **Windows** property collection.
6. Click the ellipsis button next to the **Windows** property to bring up the **RadWindow Collection Editor**.
7. In the Collection editor, add four windows to the **Windows** property collection.
 - On the first window, set its **Left** property to "210px", **Top** property to "10px", **Title** property to "Telerik", and **NavigateUrl** property to "http://www.telerik.com".
 - On the second window, set its **Left** property to "230px", **Top** property to "30px", **Title** property to "Google", and **NavigateUrl** property to "http://www.google.com".
 - On the third window, set its **Left** property to "250px", **Top** property to "50px", **Title** property to "Yahoo", and **NavigateUrl** property to "http://www.yahoo.com".
 - On the fourth window, set its **Left** property to "270px", **Top** property to "70px", **Title** property to "Wikipedia", and **NavigateUrl** property to "http://www.wikipedia.org".
8. Press Ctrl-F5 to run the application. When the application starts up, the four windows appear cascading to the right of the panel. You can move them around the Web page or resize them. When you click the minimize button in the title bar, they move to the minimize zone you created. When you restore the windows that are in the minimize zone, they return to their last position and size.

Sliding zones

Sliding zones comprise another desktop metaphor that you can bring to your Web applications: the sliding panels in applications like Visual Studio that hold content hidden away until it is needed. You can add Sliding zones to the panes of a splitter by placing a **RadSlidingZone** control into the content area of a **RadPane** control.

✍ **RadSlidingZone** is restricted so that it can *only* be placed inside a **RadPane** control. You can't use **RadSlidingZone** anywhere else.

The **RadSlidingZone** acts as a parent to one or more **RadSlidingPane** controls. Each sliding pane implements a sliding panel that appears either in its "closed" state, as a tab in the sliding zone, or in an expanded state to display its content. Expanded sliding panes can optionally be made resizable, with a resize grip on the outer edge, and can be dockable, with a pin button in the title bar to let the user lock them in an expanded position.

Before moving on to a walk-through that lets you create an application which includes a sliding zone, let's look at some of the important properties of these controls.

RadSlidingZone

The properties of `RadSlidingZone` let you configure the way the panes slide out of the sliding zone. Probably the most important property is `SlideDirection`, which determines the direction that sliding panes expand from their tabs. In the pane of a vertical splitter, setting `SlideDirection` to "Right" places the sliding zone at the left of the pane with sliding panes expanding to the right, while setting `SlideDirection` to "Left" places the sliding zone at the right of the pane with sliding panes expanding to the left. Similarly, in the pane of a horizontal splitter, you can set `SlideDirection` to "Bottom" or "Top" to put the zone at the top with panes that expand downward or at the bottom with panes that expand upward.

In addition to `SlideDirection`, you can set the `SlideDuration`, to specify how long the sliding panes take to expand from their tabs. The `ClickToOpen` property specifies whether sliding panes expand when the user clicks the tab with the mouse, or whether they expand when the mouse simply moves over the tab of a sliding pane.

By default, when the Web page loads, all the sliding panes in a sliding zone are hidden, showing only their tabs. You can start with a single pane expanded or docked by setting the `ExpandedPanelId` or `DockedPanelId` property.

RadSlidingPane

The dimensions of a sliding pane (`Height` and `Width` properties) refer to its dimensions when it is in the expanded state. Because sliding panes are part of a splitter control, only one of the two properties is meaningful: the one that indicates how far from the tab the sliding pane expands. You can control whether users are able to resize the sliding pane from the original width or height by setting the `EnableResize` property. When `EnableResize` is true, a resize grip appears on the outer edge to let the user resize the pane. For resizable sliding panes, the `MinHeight` and `MaxHeight` or `MinWidth` and `MaxWidth` properties let you place limits on how much the user can resize the sliding pane.

The `Title` property lets you label the sliding pane. The title appears in the title bar of the sliding pane, and, by default, labels the tab in the sliding zone when the sliding pane is hidden. In addition to the title, you can set the `IconUrl` property to supply an image for labelling the tab in the sliding zone. The `TabView` property controls whether the tab in the sliding zone shows the title, the icon, or both.

The `EnableDock` property specifies whether the sliding pane can be "docked" (locked into place in its expanded state). When `EnableDock` is true, a pin icon appears in the title bar to let the user dock the sliding pane. While the sliding pane is docked, the parent `RadPane` of the slider is automatically resized so that the sliding pane no longer obscures any of the content of other panes.

Sliding Zone walk-through

The following walk-through creates a horizontal splitter with a sliding zone that expands to the right.



You can find the complete source for this project at:
\\VS Projects\RealEstate\SlidingZones

1. Create a new ASP.NET Web Application.
2. Drag a ScriptManager from the Tool Box onto the Web page.
3. Using the Solution Explorer, add an **Images** folder to your project. Add the files "Calendar.gif" and "Colors.gif", which can be found in \\VS Projects\Images, to the Images folder of your project.
4. Drag a **RadSplitter** from the Tool Box onto the Web page. Set its **Height** property to "300px", its **Width** property to "75%", and its **Skin** property to "Sunset".
5. Drag a **RadPane** control from the Tool Box onto the surface of the splitter. Set its **Width** property to "25px".
6. Before adding the sliding zone to the pane you just added, add a second RadPane control to the splitter below the first RadPane. In the content area of the second RadPane, type "This is the main pane of the splitter." Note that in this application, we are not using any split bars between the panes.
7. Drag a **RadSlidingZone** onto the first RadPane in the splitter (the one without any text).
8. Drag a **RadSlidingPane** from the Tool Box onto the RadSlidingZone.
 - o Set its **Width** property to "226px".
 - o Set its **Title** property to "Calendar".
 - o Set its **ImageUrl** property to "-\Images\Calendar.gif".
9. Drag a **RadCalendar** control from the Tool Box onto the RadSlidingPane and set its **Skin** property to "Sunset".
10. Add a second **RadSlidingPane** to the RadSlidingZone, beneath the first one.
 - o Set its **Title** property to "Colors".
 - o Set its **ImageUrl** property to "-\Images\colors.gif".
11. Drag a **RadColorPicker** control from the Tool Box onto the second RadSlidingPane and set its **Skin** property to "Sunset".

- Press Ctrl-F5 to run the application. When the application starts up, the main pane takes up most of the splitter, with the sliding zone a small region on the left that displays two tabs. If you move the mouse over either of the tabs, it expands to show its sliding pane. Note how the sliding pane covers a portion of the main pane in the slider. Try docking a sliding pane. Note how the main pane resizes, so that none of it is hidden by the sliding pane any more. With one sliding pane docked, move the mouse over the other tab and see how you can expand the second sliding pane without the docked pane closing.

9.6 Server-Side Programming

Adding content to a RadDock control

You have already seen how to fill a RadDock control in the designer by assigning the **Text** property or creating a **ContentTemplate**. It is also possible to provide content by assigning a value to the **ContentTemplate** property in the code-behind, but that involves a familiarity with templates that we will not cover until later. However, there is another way to add HTML elements to the RadDock control in server-side code without having to deal with templates: using the **ContentContainer** property.

The following walk-through shows how to use the **ContentContainer** property to populate the two RadDock controls shown in the following screenshot:



You can find the complete source for this project at:
 \VS Projects\RealEstate\ServerSide

- Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
- Drag a **RadDockLayout** control from the Tool Box onto your Web page and set its **Skin** property to "Sunset".

We have not yet talked about the **RadDockLayout** control, and, as you saw in the Getting Started project, it is not always necessary to include one in a project that uses **RadDockZone** and **RadDock** controls. The purpose of the **RadDockLayout** control is to manage the state of the dock zones and dock windows. Without a **RadDockLayout** control, any changes that the user makes to the position or state of dock windows are lost as soon as the application performs a postback. **RadDockLayout** "remembers" the state of **RadDock** windows and restores it after a postback.



Gotcha! **RadDockLayout** only "remembers" the state of **RadDock** and **RadDockZone** controls that are created inside it. If you add other **RadDockZone** and **RadDock** controls to your Web page outside the **RadDockLayout**, they will not retain their state after a postback.

3. Drag a RadDockZone control from the Tool Box onto the surface of the RadDockLayout control. Change its **Height** to "600px" and its **Width** to "100px".
4. Drag a RadDock control from the Tool Box onto the surface of the RadDockZone. Set its **Height** to "100px" and its **Width** to "200px". Set the **Title** property to "Details".
5. Drag a second RadDock control onto the RadDockZone below the first. Set its **Height** to "100px" and its **Width** to "100px".
6. Switch to the code-behind, where we will be adding the content of the two RadDock controls. Before you add code to the Page_Load event handler to populate the RadDock controls, add the following helper method to add some space to a ControlCollection:

[VB] AddSpacer

```
Private Sub AddSpacer(ByVal container As ControlCollection)
    ' add a LiteralControl for a line break
    Dim spacer As New LiteralControl("<br/>")
    container.Add(spacer)
    ' add a non-breaking space as well
    spacer = New LiteralControl(" ")
    container.Add(spacer)
End Sub
```

[C#] AddSpacer

```
private void AddSpacer(ControlCollection container)
{
    // add a LiteralControl for a line break
    LiteralControl spacer = new LiteralControl("<br/>");
    container.Add(spacer);
    // add a non-breaking space as well
    spacer = new LiteralControl("&nbsp;");
    container.Add(spacer);
}
```

7. Add the following code to the Page_Load event handler to populate the two RadDock controls. Don't forget to add an Imports or using statement for Telerik.Web.UI!

[VB] Populating RadDock controls on Page_Load

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
    ' add some space to the content container
    AddSpacer(RadDock1.ContentContainer.Controls)
    ' create a text box
    Dim tb As New RadTextBox()
    tb.ID = "tbName"
    tb.Label = "Name: "
    tb.Width = New Unit("190px")
    ' add the text box to the content container
    RadDock1.ContentContainer.Controls.Add(tb)
    ' add more space
    AddSpacer(RadDock1.ContentContainer.Controls)
    ' add a second text box
    tb = New RadTextBox()
    tb.ID = "tbAddress"
    tb.Label = "Address: "
    tb.Width = New Unit("190px")
    RadDock1.ContentContainer.Controls.Add(tb)
    ' Add a spacer and button to the second RadDock control
    AddSpacer(RadDock2.ContentContainer.Controls)
    Dim postbackButton As New Button()
```

```

    postbackButton.ID = "btnPostback"
    postbackButton.Text = "Postback"
    RadDock2.ContentContainer.Controls.Add(postbackButton)
End Sub

```

[C#] Populating RadDock controls on Page_Load

```

protected void Page_Load(object sender, EventArgs e)
{
    // add some space to the content container
    AddSpacer(RadDock1.ContentContainer.Controls);
    // create a text box
    RadTextBox tb = new RadTextBox();
    tb.ID = "tbName";
    tb.Label = "Name: ";
    tb.Width = new Unit("190px");
    // add the text box to the content container
    RadDock1.ContentContainer.Controls.Add(tb);
    // add more space
    AddSpacer(RadDock1.ContentContainer.Controls);
    // add a second text box
    tb = new RadTextBox();
    tb.ID = "tbAddress";
    tb.Label = "Address: ";
    tb.Width = new Unit("190px");
    RadDock1.ContentContainer.Controls.Add(tb);
    // Add a spacer and button to the second RadDock control
    AddSpacer(RadDock2.ContentContainer.Controls);
    Button postbackButton = new Button();
    postbackButton.ID = "btnPostback";
    postbackButton.Text = "Postback";
    RadDock2.ContentContainer.Controls.Add(postbackButton);
}

```



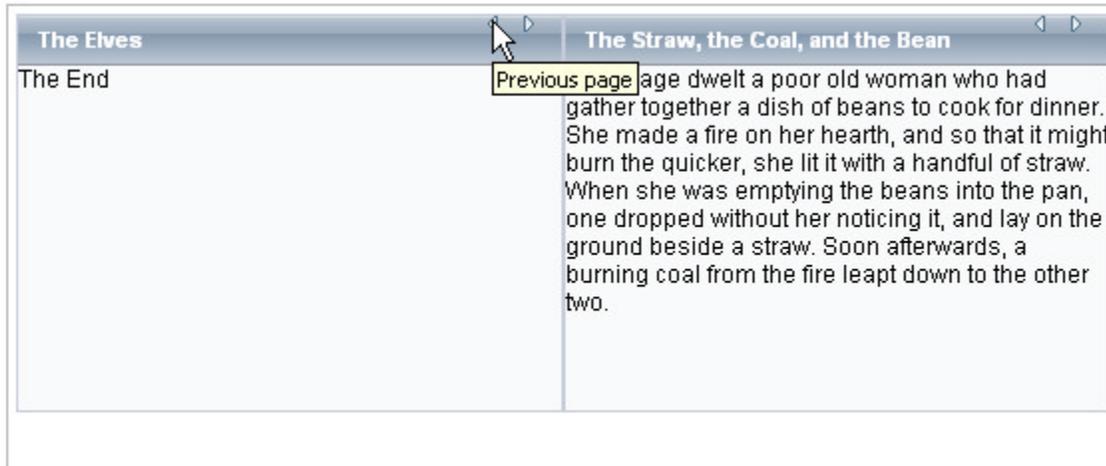
Gotcha! Be sure to add the controls even on a postback, as shown above. Controls added to the ContentContainer in the code-behind are not persisted in the view state.

- Press Ctrl-F5 to run the application. Drag the RadDock controls off the dock zone, or rearrange them. Add some text to the text boxes. Then click the Postback button you added. Notice that not only do the RadDock controls retain their new positions, but the text in the text boxes is retained as well.
 - To see why you use RadDockLayout, try removing that control from your application, running it again, and pressing the Postback button.

Implementing Custom Commands

When you add custom commands to the title bar of a RadDock control, you must implement the code that executes when the user clicks on the new custom command. You can do this in either client-side code or server-side code.

The following example shows how to add custom commands that execute on the server. The RadDock controls display a series of "pages" from a Grimm's fairy tale. Custom commands on the title bar move to the next or previous page:



You can find the complete source for this project at:
\\VS Projects\RealEstate\ServerCommand

This project adds "next page" and "previous page" custom commands to the RadDock controls from the code-behind. This is accomplished by a helper function called AddButtons:

[VB] Populating the Commands collection

```
Private Sub AddButtons(ByVal dock As RadDock)
    ' Create a command for the "Next" button
    Dim cmd As New DockCommand()
    ' Assign a name to identify it in event handlers
    cmd.Name = "cmdNextPage"
    ' Set AutoPostBack so it raises a Command event
    cmd.AutoPostBack = True
    ' Set CssClass to specify the appearance
    cmd.CssClass = "NextButton"
    ' Set Text to provide a tool tip
    cmd.Text = "Next page"
    ' Add the command to the Commands collection
    dock.Commands.Add(cmd)
    ' Repeat the process for the "Previous" button
    cmd = New DockCommand()
    cmd.AutoPostBack = True
    cmd.CssClass = "PrevButton"
    cmd.Name = "cmdPreviousPage"
    cmd.Text = "Previous page"
    dock.Commands.Add(cmd)
End Sub
```

[C#] Populating the Commands collection

```
private void AddButtons(RadDock dock)
{
    // Create a command for the "Next" button
    DockCommand cmd = new DockCommand();
    // Assign a name to identify it in event handlers
    cmd.Name = "cmdNextPage";
    // Set AutoPostBack so it raises a Command event
```

```

cmd.AutoPostBack = true;
// Set CssClass to specify the appearance
cmd.CssClass = "NextButton";
// Set Text to provide a tool tip
cmd.Text = "Next page";
// Add the command to the Commands collection
dock.Commands.Add(cmd);
// Repeat the process for the "Previous" button
cmd = new DockCommand();
cmd.AutoPostBack = true;
cmd.CssClass = "PrevButton";
cmd.Name = "cmdPreviousPage";
cmd.Text = "Previous page";
dock.Commands.Add(cmd);
}

```

Note that the code above assigned a `CssClass` of "NextButton" or "PrevButton" to the commands it added. This CSS class is defined in the `<head>` section of the `aspx` file:

[ASP.NET] CSS classes for custom buttons

```

<head id="Head1" runat="server">
  <title>Server-side custom command</title>
  <style type="text/css">
    .NextButton
    {
      width:18px;
      background:url(Images/arrowRight.gif) no-repeat !important;
    }
    .PrevButton
    {
      width:18px;
      background:url(Images/arrowLeft.gif) no-repeat !important;
    }
  </style>
</head>

```

When the page first loads, the `Page_Load` event handler initializes the `RadDock` controls to display the first page of a story that is stored in a string array. It also saves the current page number for each story using session variables. Finally, the `Page_Load` event handler calls the `AddButtons` helper function to add the custom commands:

[VB] Initializing the RadDock controls on Page_Load

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
  If Not IsPostBack Then
    ' when page first loads, initialize the text to the first page
    RadDock1.Text = TheElves(0)
    RadDock2.Text = StrawCoalBean(0)
    ' Store the current page number in the Session
    Session("RadDock1Page") = 0
    Session("RadDock2Page") = 0
  End If
  ' Always add the command buttons
  AddButtons(RadDock1)
  AddButtons(RadDock2)
End Sub

```

[CS] Initializing the RadDock controls on Page_Load

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // when page first loads, initialize the text to the first page
        RadDock1.Text = TheElves[0];
        RadDock2.Text = StrawCoalBean[0];
        // Store the current page number in the Session
        Session["RadDock1Page"] = 0;
        Session["RadDock2Page"] = 0;
    }
    // Always add the command buttons
    AddButtons(RadDock1);
    AddButtons(RadDock2);
}
```

Note that although the RadDock control only needs to be initialized when the page first loads, the command buttons must be added on every page load, just like we had to add content to the ContentContainer every time in the previous example.

Because the **AutoPostBack** property was set on the custom commands, they cause a server-side **Command** event to fire in response to user clicks. The Command event handler checks the name of the command, and based on the name, changes the current page by updating the text of the RadDock control and the session variable that stores the current page:

[VB] Implementing the command

```
Protected Sub ChangePage(ByVal sender As Object, ByVal e As DockCommandEventArgs) Handles
RadDock1.Command, RadDock2.Command
    ' sender is the RadDock control
    Dim dock As RadDock = DirectCast(sender, RadDock)
    ' set the story based on the sender
    Dim story As String()
    If dock.ID = "RadDock1" Then
        story = TheElves
    Else
        story = StrawCoalBean
    End If
    ' retrieve the current page from the Session
    Dim curPage As Integer = DirectCast(Session(dock.ID + "Page"), Integer)
    ' Check the command name to determine which command fired
    If e.Command.Name = "cmdPreviousPage" Then
        ' only move to previous page if we are past the first page
        If curPage > 0 Then
            ' update curPage to the Previous page
            curPage -= 1
            ' set the text
            dock.Text = story(curPage)
            ' update the session with the new page
            Session(dock.ID + "Page") = curPage
        End If
    ElseIf e.Command.Name = "cmdNextPage" Then
        ' update curPage to the next page
        curPage += 1
        ' check if the next page exists and if so, set the text
        If curPage < story.Length Then
            dock.Text = story(curPage)
        End If
    End If
End Sub
```

```

        ' if we passed the end, set text to "The End"
        dock.Text = "The End"
        ' don't keep increasing curPage once we reach the end
        curPage = story.Length
    End If
    ' update the session with the new page
    Session(dock.ID + "Page") = curPage
End If
End Sub

```

[CS] Implementing the command

```

protected void ChangePage(object sender, DockCommandEventArgs e)
{
    // sender is the RadDock control
    RadDock dock = (RadDock)sender;
    // set the story based on the sender
    string[] story;
    if (dock.ID == "RadDock1")
        story = TheElves;
    else
        story = StrawCoalBean;
    // retrieve the current page from the Session
    int curPage = (int)Session[dock.ID + "Page"];
    // Check the command name to determine which command fired
    if (e.Command.Name == "cmdPreviousPage")
    {
        // only move to previous page if we are past the first page
        if (curPage > 0)
        {
            // update curPage to the Previous page
            curPage -= 1;
            // set the text
            dock.Text = story[curPage];
            // update the session with the new page
            Session[dock.ID + "Page"] = curPage;
        }
    }
    else if (e.Command.Name == "cmdNextPage")
    {
        // update curPage to the next page
        curPage += 1;
        // check if the next page exists and if so, set the text
        if (curPage < story.Length)
            dock.Text = story[curPage];
        else
        {
            // if we passed the end, set text to "The End"
            dock.Text = "The End";
            // don't keep increasing curPage once we reach the end
            curPage = story.Length;
        }
        // update the session with the new page
        Session[dock.ID + "Page"] = curPage;
    }
}
}

```

Before leaving this example, a mention should be made about using AJAX when implementing the server-side Command event. You will probably want to AJAX-enable your Web page in some way so that you can avoid the disruption of a page re-load when the Command event handler is called. However, if you are updating the content of a RadDock control, this can be tricky:

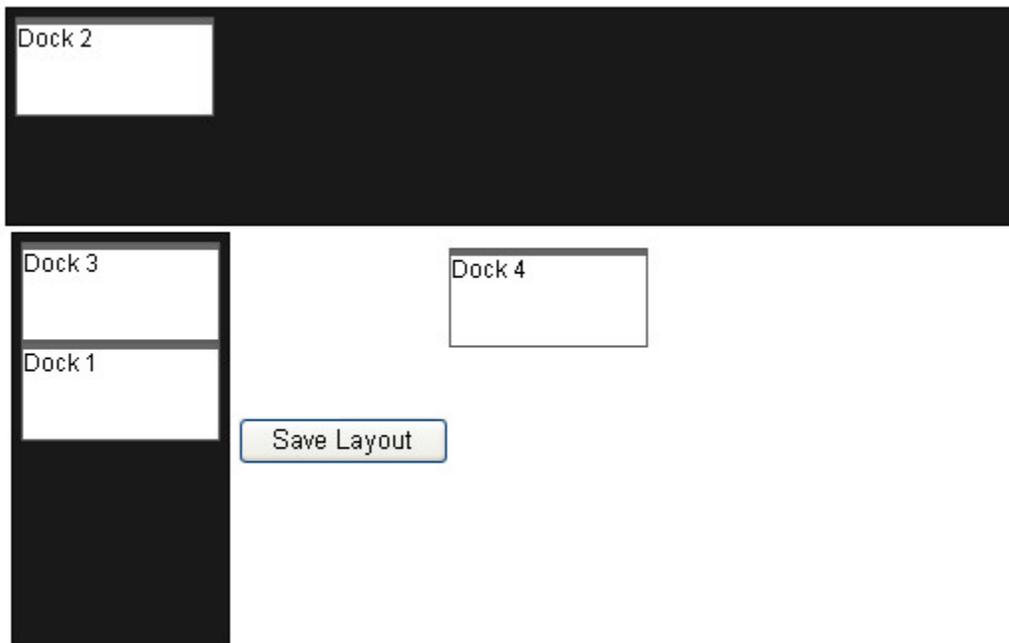
You can't put an UpdatePanel or RadAjaxPanel around the RadDock control, because that causes problems with RadDockZone, which will not accept anything other than a RadDock control as a child. You also can't assign the RadDock control as the UpdatedControl of a RadAjaxManager, because it inserts a similar element behind the scenes.

The best solution is to put an UpdatePanel or RadAjaxPanel inside the RadDock's ContentTemplate, or assign a control inside the ContentTemplate of the RadDock control as the UpdatedControl of an AJAX manager. That approach was not possible in this example, because this project uses the Text property rather than a content template. This example therefore assigns the entire RadDockLayout as the UpdatedControl. This choice, however, has its own problems. If the RadDock control is floating, a conflict arises between the RadDockLayout and the AJAX manager, where they both try to re-create the RadDock control after a postback. To avoid this conflict, this example sets the **DockMode** of the RadDock controls to "Docked", so that they can never be floating.

Preserving Dock Layout

In the last two projects, we introduced the **RadDockLayout** control, and saw how it acts to preserve the layout of RadDock controls on the page after a postback. What if you want to preserve this layout beyond the current session? It turns out that RadDockLayout exposes some server-side methods and events that let you save the current layout and restore it the next time the user visits your Web site.

The following example uses the methods and events of the RadDockLayout control to save the configuration of RadDock controls on the page in a cookie and restore that configuration when the user returns to the Web page.



You can find the complete source for this project at:
\\VS Projects\RealEstate\ServerDockLayout

The RadDockLayout control exposes two key events: **SaveDockLayout** and **LoadDockLayout**. By saving the current layout in the SaveDockLayout event handler, and restoring it in the LoadDockLayout event handler, you can ensure that the layout does not change, even after the session ends.

To help you save the current layout, RadDockLayout exposes the **GetRegisteredDocksState** method. This method returns a **List** of DockState objects, each of which represents the state of a RadDock control. By converting each of these to a string, the **SaveDockLayout** handler can store the current configuration in a cookie:

[VB] Saving the current layout

```
Protected Sub RadDockLayout1_SaveDockLayout(ByVal sender As Object, ByVal e As
DockLayoutEventArgs) Handles RadDockLayout1.SaveDockLayout
    ' Check whether there is already a cookie for saving the layout
    Dim dockState As HttpCookie = Page.Response.Cookies.[Get]("DockLayouts")
    If dockState Is Nothing Then
        ' cookie does not exist, create it and add it to the response
        dockState = New HttpCookie("DockLayouts")
        Page.Response.Cookies.Add(dockState)
    End If
    ' get the current layout from the RadDockLayout control
    Dim stateList As List(Of DockState) = (DirectCast(sender,
RadDockLayout)).GetRegisteredDocksState()
    ' convert the stateList into a string that can be added to the cookie
    Dim serializedList As New StringBuilder()
    Dim i As Integer = 0
    While i < stateList.Count
        serializedList.Append(stateList(i).ToString())
        serializedList.Append("|")
        System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
    End While
    ' Add the serialized stateList to the cookie
    dockState.Value = serializedList.ToString()
    ' Provide an expiration date for the cookie
    dockState.Expires = DateTime.Today.AddDays(1)
End Sub
```

[C#] Saving the current layout

```
protected void RadDockLayout1_SaveDockLayout(object sender, DockLayoutEventArgs e)
{
    // Check whether there is already a cookie for saving the layout
    HttpCookie dockState = Page.Response.Cookies.Get("DockLayouts");
    if (dockState == null)
    {
        // cookie does not exist, create it and add it to the response
        dockState = new HttpCookie("DockLayouts");
        Page.Response.Cookies.Add(dockState);
    }
    // get the current layout from the RadDockLayout control
    List<DockState> stateList = ((RadDockLayout)sender).GetRegisteredDocksState();
    // convert the stateList into a string that can be added to the cookie
    StringBuilder serializedList = new StringBuilder();
    for (int i = 0; i < stateList.Count; i++)
    {
        serializedList.Append(stateList[i].ToString());
        serializedList.Append("|");
    }
}
```

UI for ASP.NET AJAX

```
// Add the serialized stateList to the cookie
dockState.Value = serializedList.ToString();
// Provide an expiration date for the cookie
dockState.Expires = DateTime.Today.AddDays(1);
}
```

✎ The event handler shown above uses a `StringBuilder` to build the string for the cookie. To get the reference to this type to compile, you must add an `Imports` or `using` statement for `System.Text` in addition to the one we routinely add for `Telerik.Web.UI`.

When restoring the saved dock layout, the `LoadDockLayout` event handler must first convert the string back into a set of `DockState` objects. To convert the string for a state into a `DockState` object, you can use the `DockState.Deserialize` method. Once you have a `DockState` object, the event handler performs two tasks to restore the dock state:

- Provide the dock zones with information about the positioning of `RadDock` controls that they contain. This is done by setting the `Positions` and `Indices` properties of the event arguments. Both of these properties are indexed by the unique name of the `RadDock` control for each state. `Positions` holds the ID of the dock zone that contains the `RadDock` control, and `Indices` holds the index of that control's position within the zone.
- Apply the properties to the `RadDock` controls. This is done by calling the `RadDock.ApplyState` method. If all the `RadDock` controls are contained inside dock zones, and none of the `RadDock` properties can change with the layout, you can omit this task. The current example performs this task because if a `RadDock` control is free-floating, its position must be restored.

[VB] Restoring the layout

```
Protected Sub RadDockLayout1_LoadDockLayout(ByVal sender As Object, ByVal e As
DockLayoutEventArgs) Handles RadDockLayout1.LoadDockLayout
    ' Check whether there is a cookie with a saved layout
    Dim dockState As HttpCookie = Page.Request.Cookies.[Get]("DockLayouts")
    If Not dockState Is Nothing Then
        ' get the serialized state list from the cookie
        Dim serializedList As String = dockState.Value
        If serializedList <> Nothing Then
            ' break the serialized list into individual strings
            Dim states As String() = serializedList.Split("|"c)
            Dim i As Integer = 0
            While i < states.Length
                ' deserialize each state, and use it to assign
                ' the position and index of each state to the event arguments
                Dim state As DockState = Telerik.Web.UI.DockState.Deserialize(states(i))
                e.Positions(state.UniqueName) = state.DockZoneID
                e.Indices(state.UniqueName) = state.Index
                ' apply the state to the RadDock control
                Dim dock As RadDock = DirectCast(FindControl(state.UniqueName), RadDock)
                If Not dock Is Nothing Then
                    dock.ApplyState(state)
                End If
                System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
            End While
        End If
    End If
End Sub
```

[C#] Restoring the layout

```
protected void RadDockLayout1_LoadDockLayout(object sender, DockLayoutEventArgs e)
{
```

```

// Check whether there is a cookie with a saved layout
HttpCookie dockState = Page.Request.Cookies.Get("DockLayouts");
if (dockState != null)
{
    // get the serialized state list from the cookie
    string serializedList = dockState.Value;
    if (serializedList != null)
    {
        // break the serialized list into individual strings
        string[] states = serializedList.Split('|');
        for (int i = 0; i < states.Length; i++)
        {
            // deserialize each state, and use it to assign
            // the position and index of each state to the event arguments
            DockState state = DockState.Deserialize(states[i]);
            e.Positions[state.UniqueName] = state.DockZoneID;
            e.Indices[state.UniqueName] = state.Index;
            // apply the state to the RadDock control
            RadDock dock = (RadDock)FindControl(state.UniqueName);
            if (dock != null)
                dock.ApplyState(state);
        }
    }
}
}
}

```

Once you have supplied a handler for the **SaveDockLayout** event, the state is saved every time the page does a postback and restored every time the page loads. You must therefore add a control that causes a postback so that the layout gets saved in the first place. This example uses a **Button** control. The button causes a postback, but has no Click handler, because once the postback occurs, the RadDockLayout events handle saving and loading the layout.



Gotcha!

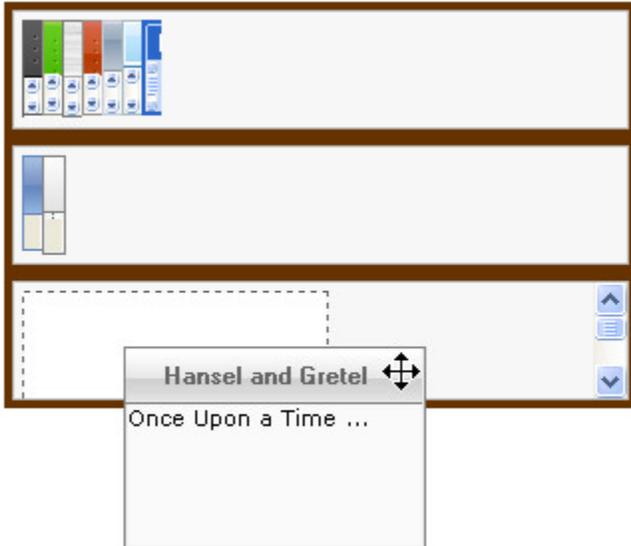
If you want to use an Ajax asynchronous callback for the postback that saves the layout, you must update the entire **RadDockLayout** control after the callback. If you are doing this, you must set the **DockMode** property of all RadDock controls to "Docked" to prevent the types of conflicts discussed in the previous example.

9.7 Client-Side Programming

Responding to layout changes

The screen "real estate" management controls all support a wealth of client-side events so that you can respond when significant changes occur. In most cases, there is a pair of events, one before the change occurs, which lets you cancel the event, and another after the event has occurred. (e.g. The **OnClientBeforeResize** and **OnClientResized** events of RadSplitter or RadPane). The following example illustrates one possible use of these client-side events.

In the previous section, you saw how to use custom commands on a RadDock control to "turn the pages" of a RadDock that acts like a book, supplying content that is fetched from the server. The following example shows how to use the client-side **OnClientDockPositionChanged** event to reinforce the book metaphor, creating a bookcase out of panels and dock zones, and resizing the "books" when they are shelved or unshelved.



The client-side **OnClientDockPositionChanged** event occurs whenever the user stops a drag operation by dropping the RadDock control in a new position. The event handler checks the parent dock zone ID by calling the **get_dockZoneID()** method. If this is an empty string, the control was dropped in a floating position; otherwise, it was dropped on a dock zone. When the RadDock control is dropped in a floating position, the event handler "opens" the book by enlarging it. When the control is dropped on a dock zone, the book is "shelved" by resizing it to the dimensions of a book spine.

[JavaScript] Resizing docked controls in OnClientDockPositionChanged

```
function PositionChanged(dock) {
    // Check to zone ID to see if it landed in a dock zone
    if (dock.get_dockZoneID() == "")
        // if not, take it off the shelf
        TakeOffShelf(dock);
    else
        // if so, put it on the shelf
        PutOnShelf(dock);
}
// PutOnShelf sets the dimensions of a shelved book
function PutOnShelf(dock) {
    dock.set_width(20);
    dock.set_height(49);
}
// TakeOffShelf sets the dimensions of an open book
function TakeOffShelf(dock) {
    dock.set_width(150);
    dock.set_height(100);
}
```



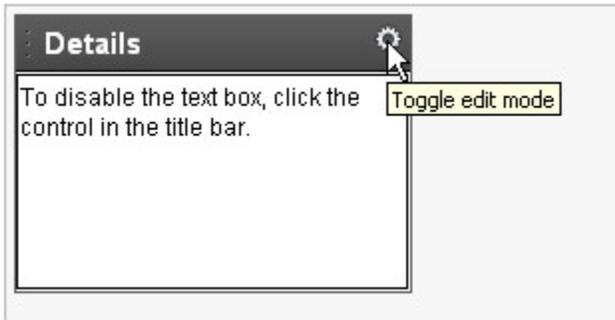
You can find the complete source for this project at:
\\VS Projects\RealEstate\ClientSide

Implementing RadDock client-side custom commands

You have already seen how to implement RadDock custom commands in the code-behind using the server-side

Command event. If you want to avoid the postback required for the Command event, you can implement custom commands on the client by using the client-side **OnClientCommand** event.

The following example uses the client-side OnClientCommand event to enable or disable a text box in the content area of a RadDock control:



You can find the complete source for this project at:
 \VS Projects\RealEstate\ClientCommand

The **OnClientCommand** handler uses the RadTextBox client-side methods to check the enabled state of the text box and change it. The handler is hooked up to the command object by setting the "OnClientCommand" property of the command object in the <Commands> collection of the RadDock control:

[ASP.NET] Client-side commands

```
<script type="text/javascript">
function ChangeEnable(dock, args) {
    // get the text box
    var edit = $find("<%= RadTextBox1.ClientID %>");
    // toggle the enabled state of the text box
    if (edit.get_enabled())
        edit.disable();
    else
        edit.enable();
}
</script>

<telerik:RadDockZone ID="RadDockZone1" Runat="server"
    Height="150px" Width="300px" FitDocks="false">
</telerik:RadDockZone>
<telerik:RadDock ID="RadDock1" Runat="server"
    Width="200px" Title="Details">
    <ContentTemplate>
        <telerik:RadTextBox ID="RadTextBox1" Runat="server"
            EmptyMessage="(No Details)" TextMode="MultiLine"
            Height="100px" Width="180px" Rows="3" >
        </telerik:RadTextBox>
    </ContentTemplate>
    <Commands>
        <telerik:DockCommand
            Text="Toggle edit mode"
            OnClientCommand="ChangeEnable" />
    </Commands>
</telerik:RadDock>
```

Manipulating RadWindow controls

The RadWindow client-side methods give you almost complete control over the pop-up window. You can show or hide the window, change its properties, minimize it, maximize it, move it, pin it, and so on. The following example illustrates a few of these methods.



In this example, the RadWindow control is visible when the page first loads, but does not have its **NavigateUrl** property set, so its content area is empty. In addition to the window, the page contains two buttons and two links (<a> elements).

When the user clicks on the first button, the onclick handler assigns the content of the window using its **setUrl()** method. Before setting the URL, the onclick handler first checks whether the window is open, and if not, it sets the size and title of the window, calls the **center()** method to position it, and then uses the window's **show()** method to open it.

💡 If there is a RadWindowManager present on the Web page, you can also display a window by calling the **radopen(URL, windowID)** function.

The links do not use the **setUrl()** method of the window. Instead, they simply set their **target** attribute to the ID of the window. Because RadWindow controls are automatically added to the frames collection of the browser, this works to assign the window as the target of the link, setting its content to the URL specified by the link's **href** attribute.

The first link uses only the **target** and **href** attributes. If the window is open, the URL is displayed in the window. If not, the link does not work.

The second link behaves like the first link, except that it also has an **onclick** handler that calls the window's **show()** method to display the window. Because of the onclick handler, this link always works.

Finally, the second button calls the window's **show()** method to display the window if it is not showing.

[ASP.NET] Manipulating RadWindow

```
<script type="text/javascript">
function SetWindowUrl() {
    // get a reference to the window
    var window1 = $find("<%= RadWindow1.ClientID %>");
    // check if it is closed
```

```

if (window1.isClosed()) {
    // assign a new width and height
    window1.set_width(400);
    window1.set_height(400);
    // set the title
    window1.set_title("Reopened window");
    // display the window
    window1.show();
    // center it on the page
    window1.center();
}
// assign the URL
window1.setUrl("http://www.wikipedia.org (http://www.wikipedia.org/)");
}
function OpenWindow() {
    // get a reference to the window and open it
    var window1 = $find("<%= RadWindow1.ClientID %>");
    window1.show();
}
</script>
<button id="setUrl" onclick="SetWindowUrl();" >Wikipedia</button><br />
<a href="http://www.google.com (http://www.google.com/)" target="RadWindow1"
>Google</a><br />
<a href="http://www.yahoo.com (http://www.yahoo.com/)" target="RadWindow1"
onclick="OpenWindow();">Yahoo</a><br />
<button id="openWindow" onclick="OpenWindow();">Open Window</button>
<telerik:RadWindow
OffsetElementId="setUrl"
Runat="server"
Width="200px"
Height="200px"
Top = "100px"
Id="RadWindow1"
style="display:none;"
Behaviors="Default"
InitialBehaviors="None"
VisibleOnPageLoad="true">
</telerik:RadWindow>

```



You can find the complete source for this project at:

\\VS Projects\\RealEstate\\ClientRadWindow

If you run this example and click on the first button or either of the links, you will see that they all do pretty much the same thing: They set the URL of the window so that it displays the associated URL. If you close the window before clicking on the first button or one of the links, you will see a difference. The button changes the window properties and opens it to display its URL. The first link does not seem to work when you click on it with the window closed. The second link opens the window to display its link. If you close the window again, click on the first link, and then click the "Open Window" button, you can see that the first link actually did change the URL of the window, it simply had no visible effect because the window was closed.

Using the alert, confirm, and prompt dialogs

If you have a **RadWindowManager** on the Web page, you can replace the browser-supplied alert, confirm, and prompt dialogs with more flexible versions that are based on **RadWindow**. The replacement dialogs can be displayed using the `radalert()`, `radconfirm()`, and `radprompt()` functions.

UI for ASP.NET AJAX

When calling the `radalert()`, `radconfirm()` or `radprompt()` function, you do not need to get a reference to the `RadWindowManager` control, the window manager simply needs to be present on the page. Calling `radalert()`, therefore, can be as simple and straightforward as calling the browser-supplied `alert()` function:

[JavaScript] Calling `radalert()`

```
radalert("Assignment cancelled.");
```

The resulting alert gets its appearance from the `Skin` property of the window manager. Unlike the browser-implemented `alert()` function, with `radalert()`, you can display a formatted message by supplying a string of HTML rather than simple text as the argument to the function. You can also use additional arguments to specify the width and height of the dialog, and to supply a title:

[JavaScript] Calling `radalert()` with additional arguments

```
radalert("Assignment cancelled.", 160, 75, selectedValue);
```

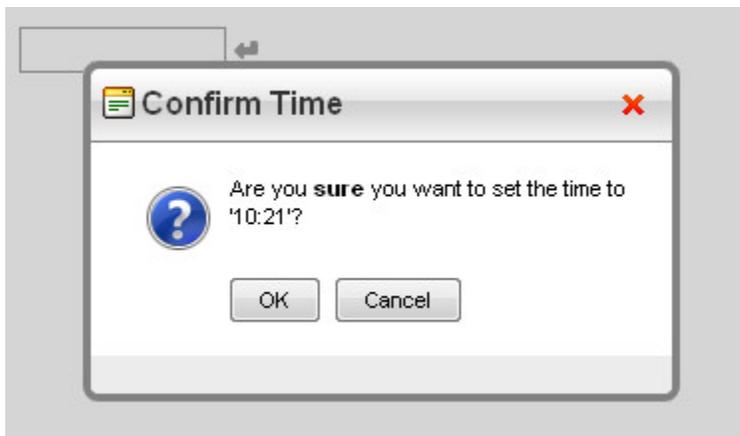
Where `callbackFn` is the function that will be called when the dialog is closed. It receives an argument that indicates whether it was closed via the OK button (`true`) or via the red [X] button (`null`):

[JavaScript] Handling the result when the `RadAlert` is closed

```
function callbackFn (arg){  
    alert(arg);  
}
```

The `radconfirm()` and `radprompt()` functions work slightly differently from the browser-supplied `confirm()` and `prompt()` functions. Instead of working with a return value from the function, the `radconfirm()` and `radprompt()` functions take a callback function as a second argument. The user's response is passed to the callback function as an argument. As with `radalert()`, you can supply a string of HTML instead of plain text to display a formatted prompt in these dialogs, and optionally supply additional arguments to specify width, height, and title of the dialog.

The following example illustrates the use of the `radalert()`, `radconfirm()` and `radprompt()` dialogs. The three dialogs are used with a `RadDateInput` control to prompt for a value, confirm it, and assign the value to the date input control. If the user cancels from the prompt or confirm dialog, an alert dialog is displayed.



You can find the complete source for this project at:
\\VS Projects\RealEstate\ClientDialogs

[ASP.NET] Using `radprompt()`, `radconfirm()`, and `radalert()`

```
<script type="text/javascript">
```

```

var selectedValue = "";
// callback function for the confirm dialog
function assignConfirmedValue(value) {
    // value is true if the user clicked OK
    if (value) {
        // get the date input and assign the supplied value
        var di = $find("<%= RadDateInput1.ClientID %>");
        di.set_value(selectedValue);
    }
    else
        // the user canceled, display an alert
        radalert("Assignment cancelled.", 150, 75, selectedValue);
}
// callback function for the prompt dialog
function confirmValue(value) {
    // save the value the user supplied in 'selectedValue'
    selectedValue = value;
    // if the user hits Cancel, value is empty
    if (value == "")
        radalert("Assignment cancelled.");
    else {
        // display the confirm dialog for the supplied value
        // note the use of rich text in the message
        var msg = "Are you <b>sure</b> you want to set the time to '" + value + "'?";
        radconfirm(msg, assignConfirmedValue, 300, 100, null, "Confirm Time");
    }
}
// OnButtonClick event handler
function PromptForValue(object, args) {
    // get the current time
    var now = new Date();
    // format the current time into a default value
    var curTime = now.getHours().toString() + ":";
    if (now.getMinutes() < 10)
        curTime = curTime + "0";
    curTime = curTime + now.getMinutes().toString();
    // display the prompt dialog, supplying the default value
    radprompt("Enter the time:", confirmValue, 200, 100, null, "Current Time", curTime);
}
</script>
<telerik:RadDateInput ID="RadDateInput1" Runat="server" ShowButton="True"
    DateFormat="h:mm tt">
<ClientEvents OnButtonClick="PromptForValue" />
</telerik:RadDateInput>
<telerik:RadWindowManager ID="RadWindowManager1" runat="server" Skin="Telerik">
</telerik:RadWindowManager>

```

When the user clicks the button on the date input control, the OnButtonClick handler calls **radprompt()** to display a prompt dialog. When the user exits the prompt dialog, the callback function either calls **radalert()** if the user canceled, or **radconfirm()** to confirm the supplied value. When the user exits the confirm dialog, the callback function either assigns the supplied value to the date input control or displays an alert to indicate that the assignment was cancelled.

Since Q1 2011 the predefined dialogs (**radalert**, **radconfirm** and **radprompt**) can be now called by using the new server methods **RadAlert**, **RadConfirm** and **RadPrompt** of the **RadWindowManager**. Note, that the callback function is a client side javascript function in all cases, it does not matter if you show the dialog from the

server or from the client.

Here is an example:

[ASP.NET]Default.aspx

```
<telerik:RadWindowManager ID="RadWindowManager1" runat="server" EnableShadow="true">
</telerik:RadWindowManager>
<asp:Button ID="btnAlert" Width="150" runat="server" OnCommand="Btn_OnCommand"
Text="radalert from server"
CommandArgument="radalert" />
    <br style="clear: both" />
    <br style="clear: both" />
<asp:Button ID="btnConfirm" Width="150" runat="server" OnCommand="Btn_OnCommand"
Text="radconfirm from server" CommandArgument="radconfirm" /><br style="clear: both" />
    <br style="clear: both" />
<asp:Button ID="btnPrompt" Width="150" runat="server" OnCommand="Btn_OnCommand"
Text="radprompt from server"
CommandArgument="radprompt" />
<script type="text/javascript">
function alertCallBackFn(arg) {
    radalert("<strong>radalert</strong> returned the following result: <h3 style='color:
#ff0000;'>" + arg + "</h3>", null, null, "Result");
}
function confirmCallBackFn(arg) {
    radalert("<strong>radconfirm</strong> returned the following result: <h3 style='color:
#ff0000;'>" + arg + "</h3>", null, null, "Result");
}
function promptCallBackFn(arg) {
    radalert("After 7.5 million years, <strong>Deep Thought</strong> answers:<h3 style='color:
#ff0000;'>" + arg + "</h3>", null, null, "Deep Thought");
}
</script>
```

Default.aspx.cs

```
protected void Btn_OnCommand(Object sender, CommandEventArgs e)
{
    switch (e.CommandArgument.ToString())
    {
        case "radalert":
            RadWindowManager1.RadAlert("RadAlert is called from the server", 330, 100, "Server
RadAlert", "alertCallBackFn");
            break;
        case "radconfirm":
            RadWindowManager1.RadConfirm("Server radconfirm: Are you sure?", "confirmCallBackFn", 330,
100, null, "Server RadConfirm");
            break;
        case "radprompt":
            RadWindowManager1.RadPrompt("Server RadPrompt: What is the answer of Life, Universe and
Everything?", "promptCallBackFn", 330, 160, null, "Server RadPrompt", "42");
            break;
    }
}
```

Printing the contents of a RadPane control

The client-side api for RadPane includes a `print()` method that lets you print the content of the pane as long as that content comes from the same domain as your Web page. This capability is demonstrated in the following example.

The Web page contains a splitter with two panes. One has content that is internal (included as part of the Web page). The other has content that is loaded from an external source using the `ContentUrl` property. Both panes display a mixture of text and ASP.NET controls.

Internal Content

This is some internal content that is declared as part of the RadPane control. When printing internal content, you can supply one or more style sheet (CSS file) that is used to format the content. That way, the printed pane can use the skins assigned to controls or format elements differently from the page in the browser.

August 2008						
M	T	W	T	F	S	S
31	28	29	30	31	1	2
32	4	5	6	7	8	9
33	11	12	13	14	15	16
34	18	19	20	21	22	23
35	25	26	27	28	29	30
36	1	2	3	4	5	6

External Content

This is some external content that comes from the same domain as the page with the splitter. To print external content, it must come from the same domain so that there are no problems with browser security restrictions.

August 2008						
Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	31	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

Print Left Pane
Print Right Pane

When the user clicks on one of the two print buttons on the bottom of the page, the `onclick` handler obtains a reference to the splitter and calls its `getPaneById()` method to get a reference to the pane that should be printed. When calling the pane's `print()` method, the `onclick` handler passes in an array of style sheets. Each element in the array is the path to a CSS file. (In this example, the array elements are simple file names because the CSS files are sitting in the project folder.) The first style sheet ("printStyles.css") changes the default properties of text on the page. The second style sheet ("Calendar.Office2007.css") is a copy of the Calendar style sheet for the Office2007 skin. If you click the buttons to print the splitter panes, you will see that the style sheets are applied to the internal pane, turning the text blue, but not to the external pane,

where the text remains its original color.

✎ We will look at style sheets for skins in the next chapter. For now, it is enough to understand that by including the "Calendar.Office2007.css" style sheet, the printed calendar can reflect the "Office2007" skin that was assigned to the RadCalendar control in the pane. You can find the "Calendar.Office2007.css" file in the "Skins\Office2007" folder inside the folder where you installed RadControls for ASPNET AJAX.

[JavaScript] onclick handler to print panes

```
function PrintPane(paneID) {  
    // get a reference to the splitter  
    var splitter = $find('<%= RadSplitter1.ClientID%>');  
    // use the getPaneById method to get a reference to the pane  
    var pane = splitter.getPaneById(paneID);  
    if (!pane) return;  
    // call the pane's print method, passing in the style sheets  
    // note that the style sheets are only used with internal content  
    var arrExtStylsheetFiles = ['printStyles.css', 'Calendar.Office2007.css' ];  
    pane.Print(arrExtStylsheetFiles);  
}
```



You can find the complete source for this project at:
VS Projects\RealEstate\ClientPrinting

9.8 How To

Using RadWindow as a floating tool window

You can use RadWindow pop-ups as floating tool windows that let the user make changes to the main Web page. The following example illustrates how this is done by using a RadWindow that contains a color picker to change the color of a panel on the main Web page.



The main Web page contains three elements:

- a <div> element that the pop-up tool window can influence.
- a <button> element to act as an opener element for the pop-up tool window.
- a **RadWindow** control to hold the color picker. The **NavigateUrl** property of the RadWindow control is set to a Web Form that contains the color picker.

In addition to these three elements, the main Web page includes a JavaScript function to assign a color to the <div> element.

[JavaScript] Function to change the <div> color

```
// SetPanelColor sets the background color of the
// <div> element to the specified color
function SetPanelColor(newColor) {
    var colorpanel = $get("ColorDiv");
    colorpanel.style.backgroundColor = newColor;
}
```

The project also contains a Web Form that supplies the content of the pop-up on the main page. This Web Form is very simple; it contains a single **RadColorPicker** control with an **OnClientColorChange** event handler. The event handler gets a reference to the **RadWindow** control that is hosting the Web Form, uses that to get a reference to the parent window, and uses the reference to the parent window to call the "SetPanelColor" function defined on its Web page.

[JavaScript] OnClientColorChange handler

```
// GetRadWindow returns a reference to the RadWindow
// wrapper for this page
function GetRadWindow() {
    var oWindow = null;
    if (window.radWindow)
        oWindow = window.radWindow;
    else if (window.frameElement.radWindow)
        oWindow = window.frameElement.radWindow;
    return oWindow;
}
// SelectColor handles the OnClientColorChange event
function SelectColor(sender, args) {
    // get the selected color
    var newColor = sender.get_selectedColor();
    if (newColor == null)
        newColor = "#ffffff";
    // get the RadWindow wrapper for this page
    var thisWindow = GetRadWindow();
    if (thisWindow)
        // use its BrowserWindow property to get the parent window
        // and call the SetPanelColor function on the browser window
        thisWindow.BrowserWindow.SetPanelColor(newColor);
}
```

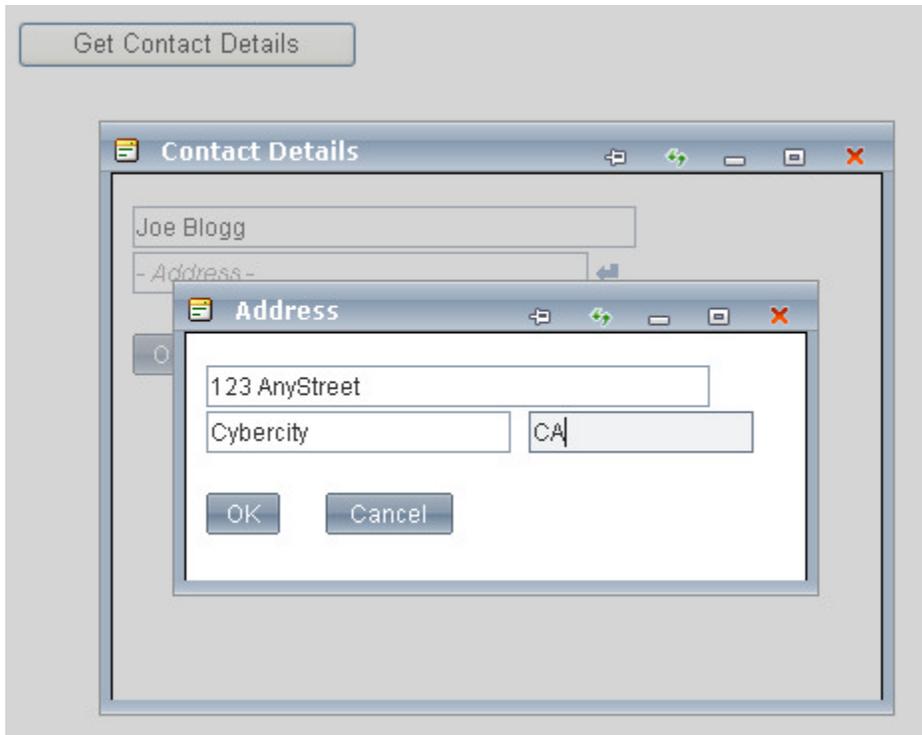


You can find the complete source for this project at:
 \VS Projects\RealEstate\HowToFloatingToolWindow

Using RadWindow as a modal dialog

Another use for **RadWindow** controls is to act as modal dialogs. You have already seen some such dialogs: the pop-up dialogs that **RadWindowManager** creates for the **radalert()**, **radconfirm()**, and **radprompt()** functions. The following example illustrates how you can create your own dialogs.

There are two ways to pass arguments back from a **RadWindow** control used as a dialog. This example implements two dialogs, where each one illustrates one of the techniques. The main form has a button to bring up the first dialog. That dialog returns values using a callback function. The first dialog contains a text box with a button to bring up the second dialog. The second dialog uses the properties of the **RadWindow** object to return values.



The main Web page

The main Web page is very simple. It contains only a button to bring up the "Contact Details" dialog, and a **RadWindow** control to hold that dialog. The **RadWindow** control has its **OpenerElementId** property set to the button, which is all that is needed to display the dialog. The **Modal** property of the window is true, so that the dialog is modal, and the **NavigateUrl** property is set to the aspx file that implements the "Contact Details" dialog. Because the "Contact Details" dialog returns its results using a callback, the **RadWindow** control assigns a handler to the **OnClientClose** client-side event.

[JavaScript] OnClientClose handler

```
function ProcessContactDetails(sender, args) {  
    alert("Contact Dialog returned " + args + "");  
}
```

The Contact Details dialog

The Contact Details dialog is quite a bit more complicated, because it must interact with both the main Web page, to which it is a child dialog, and the "Address" dialog, to which it is a parent. Let us first look at how it communicates with the main Web page.

Any code to communicate with the main Web page must do so using the **RadWindow** object that hosts the dialog. The first JavaScript function contained in this Web page is therefore one that provides a reference to that **RadWindow** object. This is a function you have seen before (in the floating tool window example).

[JavaScript] Getting a reference to the RadWindow wrapper

```
// GetRadWindow returns a reference to the RadWindow  
// wrapper for this page  
function GetRadWindow() {  
    var oWindow = null;  
    if (window.radWindow)  
        oWindow = window.radWindow;  
    else if (window.frameElement)
```

```

    if (window.frameElement.radWindow)
        oWindow = window.frameElement.radWindow;
    return oWindow;
}

```

The dialog contains two text boxes, one for name and one for address. When the user clicks the OK button, the button's `onclick` handler retrieves a reference to the window wrapper, assembles a return value from the text boxes, and passes it to the window's `close()` method. Passing an argument to the `close()` method causes the RadWindow wrapper to execute its `OnClientClose` handler function, which was supplied in the main Web page.

[JavaScript] Triggering the OnClientClose handler from the OK button

```

// HandleOk is the handler for the OK button
function HandleOk() {
    // get the RadWindow wrapper
    var currentWindow = GetRadWindow();
    if (currentWindow) {
        // combine the name and address into a string
        // and pass it as an argument to the callback
        var name = $find("<%= RadTextBox1.ClientID %>");
        var address = $find("<%= RadTextBox2.ClientID %>");
        currentWindow.close(name.get_value() + "/" + address.get_value());
    }
}

```

The `onclick` handler for the cancel button is even simpler. It only needs to get a reference to the window wrapper and close it:

[JavaScript] Closing the window from the cancel button

```

// HandleCancel is the handler for the Cancel button
function HandleCancel() {
    var currentWindow = GetRadWindow();
    if (currentWindow) {
        // close the dialog
        currentWindow.close();
    }
}

```

That is it for communicating with the main Web page. Now let's turn our attention to bringing up and communicating with the "Address" dialog. This is done using a **RadWindow** control on the Contact Details form.

The Address dialog is displayed when the user clicks the button of the "Address" text box. This is done by calling the `show()` method of the RadWindow control.

[JavaScript] Displaying the Address dialog from the OnButtonClick handler

```

// ShowAddressDialog is the OnButtonClick handler for the address text box
function ShowAddressDialog() {
    // just display the dialog, the rest is handled by the window
    var dialog = $find("<%= RadWindow1.ClientID %>");
    dialog.show();
}

```

When the main page displays the Contact Details dialog, it does not pass in any arguments. When bringing up the Address dialog, however, the Contact Details dialog uses the current value of the Address text box to pass in some arguments. The arguments are assigned in an `OnClientShow` event handler.

The Address dialog expects an arguments object with three fields: "street", "city", and "state", so the Contact Details dialog must parse the address into its parts, create an arguments object, and assign the three fields to

it. All of this is then assigned to the **argument** property of the RadWindow that hosts the Address dialog.

[JavaScript] Passing arguments to the dialog in the OnClientShow handler

```
// ParseAddress is a helper function
// It parses the value in the address text box
// and assigns the fields of the arguments object
// which will be used to initialize the address dialog
function ParseAddress(address, args) {
    // initialize variables
    var street = "";
    var city = "";
    var state = "";
    var curPart = "street";
    // go through the address string, splitting it up at commas
    // curPart keeps track of which section we are on
    for (var i = 0; i < address.length; i++) {
        if (address[i] != ",") {
            if (curPart == "street")
                street = street + address[i];
            else if (curPart == "city")
                city = city + address[i];
            else
                state = state + address[i];
        }
        else if (curPart == "street")
            curPart = "city";
        else if (curPart == "city")
            curPart = "state";
    }
    // assign the variables to the args object
    args.street = street.trim();
    args.city = city.trim();
    args.state = state.trim();
}
// InitializeDialog is the OnClientShow handler of the address dialog window
function InitializeDialog(sender, args) {
    // create a new object to pass arguments to the dialog
    var args = new Object();
    args.street = "";
    args.city = "";
    args.state = "";
    var addr = $find("<%= RadTextBox2.ClientID %>");
    // call ParseAddress to assign the current address to the args object
    ParseAddress(addr.get_value(), args);
    // assign the args object as the argument of the dialog window
    sender.argument = args;
}
```

The Address dialog passes back return values using the same **argument** property that was used to pass in the arguments. When the Address dialog closes, its **OnClientClose** event fires, and an event handler reads the return values, assembles them into a single string, and assigns it to the address text box.

[JavaScript] Assigning return values in the OnClientClose handler

```
// AssignAddress is the OnClientClose handler for the address dialog window
// It is called when the dialog closes
function AssignAddress(sender, args) {
    // check if the dialog closed with an argument set
```

```

if (sender.argument != null) {
    // if so, combine the fields of the argument
    // into an address string
    var newAddress = sender.argument.street;
    if (newAddress.length > 0)
        newAddress = newAddress + ", ";
    newAddress = newAddress + sender.argument.city;
    if (sender.argument.city.length > 0)
        newAddress = newAddress + ", ";
    newAddress = newAddress + sender.argument.state;
    // get a reference to the address text box and assign the value
    var addr = $find("<%= RadTextBox2.ClientID %>");
    addr.set_value(newAddress.trim());
}
}

```

The Address dialog

When the address dialog first appears, it initializes itself based on the arguments passed in the argument property of the RadWindow wrapper. This is handled in a **pageLoad** event function which is fired once the entire page has loaded.

[ASP.NET] Initializing the dialog

```

<body>
    <form id="form1" runat="server">
        <asp:ScriptManager ID="ScriptManager1" runat="server">
        </asp:ScriptManager>
        <script type="text/javascript">
            // GetRadWindow returns a reference to the RadWindow
            // wrapper for this page
            function GetRadWindow()
            {
                var oWindow = null;
                if (window.radWindow)
                    oWindow = window.radWindow;
                else if (window.frameElement)
                    if (window.frameElement.radWindow)
                        oWindow = window.frameElement.radWindow;
                return oWindow;
            }
            // Initialize initializes the text boxes
            function pageLoad()
            {
                // get a reference to the RadWindow wrapper
                var currentWindow = GetRadWindow();
                // read the argument of the wrapper and use it to
                // initialize the text boxes
                if (currentWindow)
                {
                    var args = currentWindow.argument;
                    if (args)
                    {
                        var street = $find("<%= RadTextBox1.ClientID %>");
                        var city = $find("<%= RadTextBox2.ClientID %>");
                        var state = $find("<%= RadTextBox3.ClientID %>");
                        street.set_value(args.street);
                    }
                }
            }
        </script>
    </form>

```

```
        city.set_value(args.city);
        state.set_value(args.state);
    }
}
...

```

Note the use of our old friend, the **GetRadWindow()** method.

The handler for the cancel button is pretty similar to the one in the Contact Details dialog. The only difference is that this time, the handler must set the argument property to null. That is because this dialog uses the argument property to pass back return values, and setting argument to null signals that the user cancelled.

[JavaScript] The cancel button onclick handler

```
// HandleCancel is the handler for the cancel button
function HandleCancel() {
    // get a reference to the window wrapper
    var currentWindow = GetRadWindow();
    if (currentWindow) {
        // make sure the window argument is null and close
        currentWindow.argument = null;
        currentWindow.close();
    }
}

```

The handler for the OK button assigns return values to the **argument** property rather than passing them as a parameter to the **close()** method. Because the **close()** method is called without arguments, no callback event occurs, and the parent window must retrieve the arguments in response to the **OnClientClose** event.

[JavaScript] Assigning argument in the OK button onclick handler

```
// HandleOk is the handler for the OK button
function HandleOk() {
    // get a reference to the window wrapper
    var currentWindow = GetRadWindow();
    if (currentWindow) {
        // read the values entered into the dialog
        var street = $find("<%= RadTextBox1.ClientID %>");
        var city = $find("<%= RadTextBox2.ClientID %>");
        var state = $find("<%= RadTextBox3.ClientID %>");
        // create an object to hold the arguments and assign values
        var args = new Object();
        args.street = street.get_value();
        args.city = city.get_value();
        args.state = state.get_value();
        // assign the arguments object to the window and close it
        currentWindow.argument = args;
        currentWindow.close();
    }
}

```



You can find the complete source for this project at:
\\VS Projects\RealEstate\HowToModalDialog

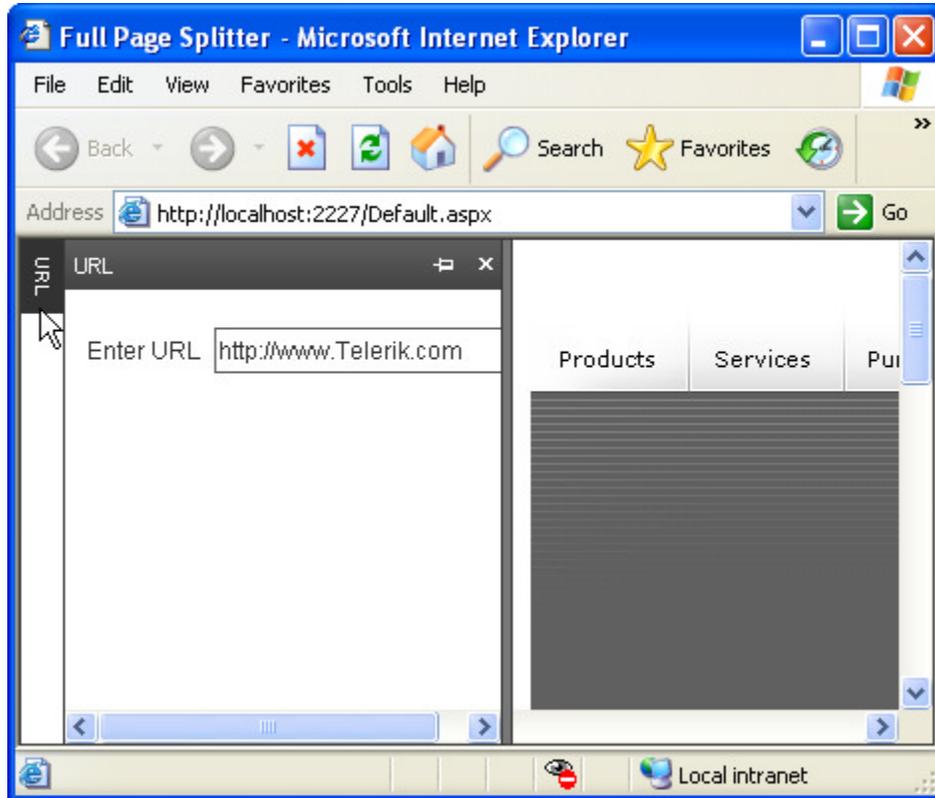
Filling the Web page with a splitter

To create the feel of a desktop application, you may want to have a splitter control fill the entire Web page. To do this, you must obviously set the **Width** and **Height** properties of the splitter to "100%". However, if you just set those properties and run your application, you may encounter some unexpected behavior. For example, you may find the splitter with a height of 0!

To achieve the expected behavior, you need to make some changes to the tags that Visual Studio automatically adds to the aspx file:

1. On the `<html>` tag, set the **style** attribute to give the document a height of 100%:
`style="height:100%"`.
2. On the `<body>` tag, you must again set the **style** attribute to include a height of 100%. In addition, the style attribute must reset the margin. The default margin of the body is 5 pixels, and you will want to change this to 0. Optionally, you may also want to remove the page scroll bars for a more desktop-like feel. Putting these changes together gives the tag:
`<body style="height:100%;margin:0px" scroll="no">`
3. Any elements that contain the splitter must also have their height set to 100% and their margin set to 0. This includes the `<form>` tag:
`<form id="form1" runat="server" style="height:100%;margin:0px">`

Once you have added these changes, the splitter can fill the entire Web page.



✎ Not all of the changes listed above are required for all browsers, but by including them all, you can achieve the same effect regardless of the browser viewing your page.

The full-page splitter shown above was generated using the following markup:

[ASP.NET] Full-page splitter

```
<html xmlns="http://www.w3.org/1999/xhtml" style="height:100%" >
<head id="Head1" runat="server">
```

```
<title>Full Page Splitter</title>
</head>
<body style="height:100%;margin:0px" scroll="no">
  <form id="form1" runat="server" style="height:100%;margin:0px">
    <asp:ScriptManager ID="ScriptManager1" runat="server">
    </asp:ScriptManager>
    <script type="text/javascript">
      function ChangeUrl(sender, args) {
        var pane = $find("<%= RadPane2.ClientID %>");
        pane.set_contentUrl(args.get_newValue());
      }
    </script>
    <div style="height:100%;margin:0px">
      <telerik:RadSplitter ID="RadSplitter1" Runat="server"
        Height="100%" Width="100%">
        <telerik:RadPane ID="RadPane1" Runat="server" Width="25px">
          <telerik:RadSlidingZone ID="RadSlidingZone1" Runat="server">
            <telerik:RadSlidingPane ID="RadSlidingPane1" Runat="server"
              Title="URL" Width="240px">
              <br />&nbsp;
              <telerik:RadTextBox ID="RadTextBox1" Runat="server"
                Label="Enter URL " Text="http://www.telerik.com (http://www.telerik.com)/"
                Width="220px">
                <ClientEvents OnValueChanged="ChangeUrl" />
              </telerik:RadTextBox>
            </telerik:RadSlidingPane>
          </telerik:RadSlidingZone>
        </telerik:RadPane>
        <telerik:RadPane ID="RadPane2" Runat="server"
          ContentUrl="http://www.telerik.com (http://www.telerik.com)"/>
        </telerik:RadPane>
      </telerik:RadSplitter>
    </div>
  </form>
</body>
</html>
```

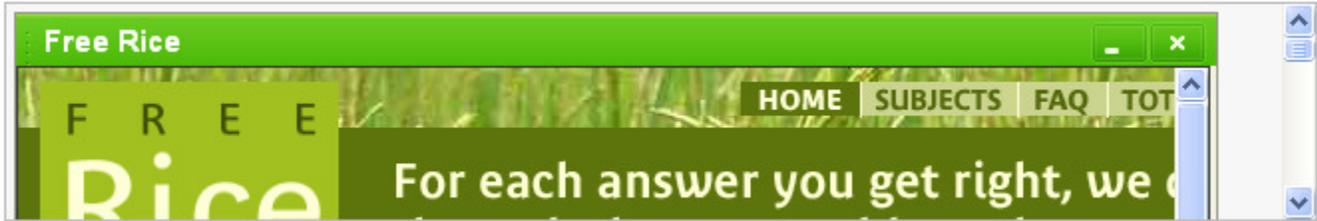


You can find the complete source for this project at:
\\VS Projects\RealEstate\HowToFullPageSplitter

Displaying external content in a RadDock control

One of the differences between RadDock controls and RadWindow controls is that RadDock controls use content that is loaded with the Web page, while RadWindow uses external content. However, sometimes you may want to display content from an external source in a dockable window. You can accomplish this by putting a simple iframe inside the content template of a RadDock control.

This is illustrated in the following example.



Because there is no property to suppress the scroll bars of a RadDock control, you need to play a bit with the size of the iframe and the size of the RadDock control, until the iframe fits exactly in the RadDock client area. Thus, the only scroll bars that appear are the scroll bars of the page that has been loaded inside the iframe, which appear as necessary based on the external content.

[ASP.NET] Using an iframe to add external content to a RadDock control

```
<telerik:RadDockZone ID="RadDockZone1" runat="server" Height="100px" Width="650px"
  Orientation="Horizontal">
  <telerik:RadDock ID="RadDock1" runat="server" Height="595px" Width="570px" Title="Free
  Rice">
    <ContentTemplate>
      <iframe src="http://www.freerice.com (http://www.freerice.com/)" style="width:
  550px; height: 550px;"></iframe>
    </ContentTemplate>
  </telerik:RadDock>
</telerik:RadDockZone>
```



You can find the complete source for this project at:
 \VS Projects\RealEstate\HowToExternalDockContent

Dynamically creating RadDock controls

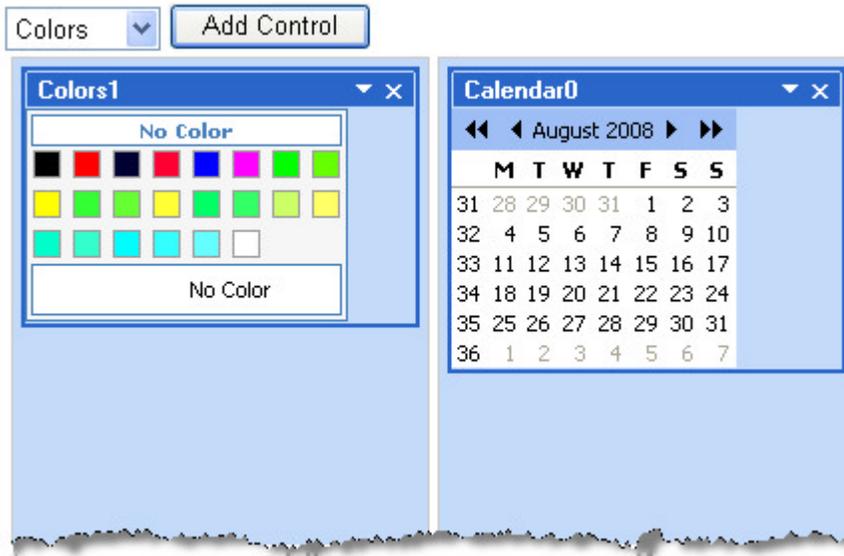
A common use of RadDock controls is to add controls to a portal page. To accomplish this, the Web application needs to dynamically create the new RadDock controls that the user adds to the page. You have already seen many of the techniques that are required to accomplish this: In the section on Server-Side Programming, you saw how to add controls to the **ContentContainer** of a RadDock control and how to use the **LoadDockLayout** and **SaveDockLayout** events to save and restore the dock layout. There are a few additional considerations to take into account when the RadDock controls are dynamically created. These are illustrated in the following example.



You can find the complete source for this project at:
 \VS Projects\RealEstate\HowToDynamicDocks

The Web page

The Web page in this example includes a drop-down list so that the user can select the type of control to add, a button for adding controls, and a **RadDockLayout** control with two dock zones where the dynamically created dock windows can be placed:



The most important thing to note about these controls is that the **EnableViewState** property of the **RadDockZone** control where dynamic docks are added is set to false. Because the contents of this dock zone change in the code-behind, the controls collection of the dock zone after a postback does not match the collection when the postback starts. This will lead to view state errors unless you disable the view state of the dock zone.

Maintaining the dock layout

Recall how the **RadDockLayout** control manages the current layout of the dock windows and dock zones using a List of **DockState** objects, which you can retrieve by calling its **GetRegisteredDocksState** method. Because the application changes this layout every time a new dock is added, the application needs to save and access this list every time there is a postback. To make that easier, a private property is added in the code-behind which stores the list in a **Session** variable.

[VB] CurrentDockStates property

```
' Private property for storing the current dock states
Private Property CurrentDockStates() As List(Of DockState)
    Get
        Dim _currentDockStates As List(Of DockState) = DirectCast(Session("CurrentDockStates"),
List(Of DockState))
        If [Object].Equals(_currentDockStates, Nothing) Then
            _currentDockStates = New List(Of DockState)()
            Session("CurrentDockStates") = _currentDockStates
        End If
        Return _currentDockStates
    End Get
    Set(ByVal value As List(Of DockState))
        Session("CurrentDockStates") = value
    End Set
End Property
```

[C#] CurrentDockStates property

```
// Private property for storing the current dock states
private List<DockState> CurrentDockStates
{
    get
    {
```

```

List<DockState> _currentDockStates = (List<DockState>)Session["CurrentDockStates"];
if (Object.Equals(_currentDockStates, null))
{
    _currentDockStates = new List<DockState>();
    Session["CurrentDockStates"] = _currentDockStates;
}
return _currentDockStates;
}
set
{
    Session["CurrentDockStates"] = value;
}
}

```

This variable is maintained using the **LoadDockLayout** and **SaveDockLayout** events of the RadDockLayout control. This is very similar to the "preserving dock layout" example, except that this time, the LoadDockLayout event handler does not apply the dock states to the RadDock controls themselves. That is handled separately, in a Page_Init event handler.

[VB] LoadDockLayout and SaveDockLayout

```

' SaveDockLayout updates the CurrentDockStates variable when necessary
Protected Sub RadDockLayout1_SaveDockLayout(ByVal sender As Object, ByVal e As
DockLayoutEventArgs) Handles RadDockLayout1.SaveDockLayout
    CurrentDockStates = RadDockLayout1.GetRegisteredDocksState()
End Sub
' LoadDockLayout tells the dock layout control
' where the docks sit in their dock zones
Protected Sub RadDockLayout1_LoadDockLayout(ByVal sender As Object, ByVal e As
DockLayoutEventArgs) Handles RadDockLayout1.LoadDockLayout
    ' use the layout that was saved in the session
    For Each state As DockState In CurrentDockStates
        ' set the event arguments to the saved values
        e.Positions(state.UniqueName) = state.DockZoneID
        e.Indices(state.UniqueName) = state.Index
    Next
End Sub

```

[C#] LoadDockLayout and SaveDockLayout

```

// SaveDockLayout updates the CurrentDockStates variable when necessary
protected void RadDockLayout1_SaveDockLayout(object sender, DockLayoutEventArgs e)
{
    CurrentDockStates = RadDockLayout1.GetRegisteredDocksState();
}
// LoadDockLayout tells the dock layout control
// where the docks sit in their dock zones
protected void RadDockLayout1_LoadDockLayout(object sender, DockLayoutEventArgs e)
{
    // use the layout that was saved in the session
    foreach (DockState state in CurrentDockStates)
    {
        // set the event arguments to the saved values
        e.Positions[state.UniqueName] = state.DockZoneID;
        e.Indices[state.UniqueName] = state.Index;
    }
}

```

Creating and adding dock windows in the button's Click handler

The button's Click event handler creates and initializes a new RadDock control. Using the **ContentContainer**, it adds the contents of the RadDock control based on the current selection in the drop-down list. In this example, the content is built up dynamically in code, but you could also load a custom control (ascx) or even (by using the trick with a splitter shown above) display content from another URL. Once its content has been added, the new RadDock control is added to the dock zone.

[VB] Creating and adding a dock window in the button's Click handler

```
'Button to add a new dock
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
Button1.Click
    Dim dock As RadDock = CreateRadDock()
    ' Store the type of content in the tag
    dock.Tag = DropDownList1.SelectedItem.Text
    ' Use the type of content to create a title
    dock.Title = dock.Tag + CurrentDockStates.Count.ToString()
    ' add the content to the RadDock
    LoadContent(dock)
    ' Add the RadDock to the first dock zone
    RadDockZone1.Controls.Add(dock)
End Sub
' Create and initialize a new RadDock control
Private Function CreateRadDock() As RadDock
    ' get the current count of RadDock controls
    Dim docksCount As Integer = CurrentDockStates.Count
    ' Create the RadDock control
    Dim dock As New RadDock()
    ' use the skin of the dock layout
    dock.Skin = RadDockLayout1.Skin
    ' use the count to give it an ID
    dock.ID = String.Format("RadDock{0}", docksCount)
    ' use a guid to set the UniqueName
    dock.UniqueName = Guid.NewGuid().ToString()
    ' set the width to the width of the dock zones
    dock.Width = Unit.Pixel(200)
    ' make sure it must be docked
    dock.DockMode = DockMode.Docked
    Return dock
End Function
' add content based on the tag value
Private Sub LoadContent(ByVal dock As RadDock)
    Select Case dock.Tag
        Case "Calendar"
            Dim calendar As New RadCalendar()
            calendar.Skin = RadDockLayout1.Skin
            calendar.ID = dock.ID + "calendar"
            'Add the calendar to the ContentContainer
            dock.ContentContainer.Controls.Add(calendar)
        Exit Select
        Case "Colors"
            Dim colors As New RadColorPicker()
            colors.Skin = RadDockLayout1.Skin
            colors.ID = dock.ID + "colors"
            colors.Preset = ColorPreset.ReallyWebSafe
```

```

    'Add the color picker to the ContentContainer
    dock.ContentContainer.Controls.Add(colors)
Exit Select
Case "Text"
    Dim text As New RadTextBox()
    text.Skin = RadDockLayout1.Skin
    text.ID = dock.ID + "text"
    'Add the text box to the ContentContainer
    dock.ContentContainer.Controls.Add(text)
Exit Select
End Select
End Sub

```

[CS] Creating and adding a dock window in the button's Click handler

```

// Button to add a new dock
protected void Button1_Click(object sender, EventArgs e)
{
    RadDock dock = CreateRadDock();
    // Store the type of content in the tag
    dock.Tag = DropDownList1.SelectedItem.Text;
    // Use the type of content to create a title
    dock.Title = dock.Tag + CurrentDockStates.Count.ToString();
    // add the content to the RadDock
    LoadContent(dock);
    // Add the RadDock to the first dock zone
    RadDockZone1.Controls.Add(dock);
}
// Create and initialize a new RadDock control
private RadDock CreateRadDock()
{
    // get the current count of RadDock controls
    int docksCount = CurrentDockStates.Count;
    // Create the RadDock control
    RadDock dock = new RadDock();
    // use the skin of the dock layout
    dock.Skin = RadDockLayout1.Skin;
    // use the count to give it an ID
    dock.ID = string.Format("RadDock{0}", docksCount);
    // use a guid to set the UniqueName
    dock.UniqueName = Guid.NewGuid().ToString();
    // set the width to the width of the dock zones
    dock.Width = Unit.Pixel(200);
    // make sure it must be docked
    dock.DockMode = DockMode.Docked;
    return dock;
}
// add content based on the tag value
private void LoadContent(RadDock dock)
{
    switch (dock.Tag)
    {
        case "Calendar":
            RadCalendar calendar = new RadCalendar();
            calendar.Skin = RadDockLayout1.Skin;
            calendar.ID = dock.ID + "calendar";

```

```
// add calendar to the ContentContainer
dock.ContentContainer.Controls.Add(calendar);
break;
case "Colors":
    RadColorPicker colors = new RadColorPicker();
    colors.Skin = RadDockLayout1.Skin;
    colors.ID = dock.ID + "colors";
    colors.Preset = ColorPreset.ReallyWebSafe;
    // add color picker to the ContentContainer
    dock.ContentContainer.Controls.Add(colors);
    break;
case "Text":
    RadTextBox text = new RadTextBox();
    text.Skin = RadDockLayout1.Skin;
    text.ID = dock.ID + "text";
    // add text box to the ContentContainer
    dock.ContentContainer.Controls.Add(text);
    break;
}
}
```

There are a few important things to notice in the code above. The first is the use of the **Tag** property. Tag is a string property of RadDock with no pre-defined use. It is a convenient place to store application-specific information. In our case, we store the type of control the RadDock contains. If you recall from the section of Server-Side Programming, controls added directly to the **ContentContainer**, as we are doing here, do not persist in the view state. They must be re-created every time the page loads. By saving the type of control in the **Tag** property, which is saved in the dock state, the application can know what types of controls to re-create on the next postback.

The second thing to note is that the code to add content to the new RadDock control is handled by a separate method that reads the **Tag** property and uses that to generate the content. By structuring the code in this way, we can re-use the method that initially creates content when we must re-create that content on the next postback.

The third thing to note is that when the CreateRadDock method initializes a RadDock control, it sets the **UniqueName** property. RadDockLayout uses the UniqueName property to identify each of the RadDock controls on the page. When this property is not explicitly set, it is the same as the ID property. However, by using a GUID for UniqueName, the application can more robustly ensure that the dock windows are correctly identified.

Finally, note that the CreateRadDock method sets the **DockMode** property to "Docked". This example uses a RadAjaxManager to handle the button click in an asynchronous postback that updates the dock layout. As we have seen before, when updating a dock layout from an AJAX callback, all RadDock controls must be docked.

Re-creating controls on a postback

As was mentioned previously, content that is added to the **ContentContainer** must be re-created on every postback, because it does not persist in the view state. There is a similar problem with the dynamically created RadDock controls themselves: because they are added to a dock zone in the code-behind, they also do not persist in the view state. As a result, the application must add a **Page_Init** event handler to regenerate all of the dock windows, including their content. The properties of the dock windows can be obtained from the dock states, which are available from our private **CurrentDockStates** property.

[VB] Re-creating the dock windows

```
' the saved dock state must be restored on page init
Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Init
    Dim i As Integer = 0
    While i < CurrentDockStates.Count
        ' re-create the radDock control
```

```

Dim dock As New RadDock()
dock.ID = String.Format("RadDock{0}", i)
' restore properties from Current Dock States
dock.ApplyState(CurrentDockStates(i))
' add it to the dock layout
' the dock layout can restore its position
RadDockLayout1.Controls.Add(dock)
' restore the content
LoadContent(dock)
System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
End While
End Sub

```

[C#] Re-creating the dock windows

```

// the saved dock state must be restored on page init
protected void Page_Init(object sender, EventArgs e)
{
    for (int i = 0; i < CurrentDockStates.Count; i++)
    {
        // re-create the radDock control
        RadDock dock = new RadDock();
        dock.ID = string.Format("RadDock{0}", i);
        // restore properties from the Current Dock States
        dock.ApplyState(CurrentDockStates[i]);
        // add it to the dock layout
        // the dock layout can restore its position
        RadDockLayout1.Controls.Add(dock);
        // restore the content
        LoadContent(dock);
    }
}

```



Gotcha! Be sure to use the **Page_Init** event to re-create dock windows. This event occurs at the proper time so that the **CurrentDockStates** property holds the controls you need to re-create. Using another event, such as **Page_Load** can result in dock windows that do not reflect the user's most recent changes to the layout.

9.9 Summary

In this chapter you looked at the "real estate" management controls, learning how they can help organize the content on your Web pages into flexible content areas that can be moved, resized, or hidden. You saw many of the important properties. You created a simple application that used dock zones and dock windows, a splitter, and some pop-up windows managed by a window manager. You also created simple applications to become familiar with minimize zones and sliding zones.

You learned to use the server-side API with the RadDock family of controls, adding content in the code-behind, implementing custom commands, and preserving dock layout in a cookie. You learned to perform common client-side tasks such as responding to layout changes, implementing custom commands, manipulating windows, printing the panes of a splitter, and using the customizable alert, confirm, and prompt dialogs.

Finally, you learned techniques that are important to some of the more common applications that use the "real estate" management controls, including implementing tool windows and modal dialogs, creating a desktop-like window by filling the entire Web page with a splitter, and creating dockable windows dynamically.

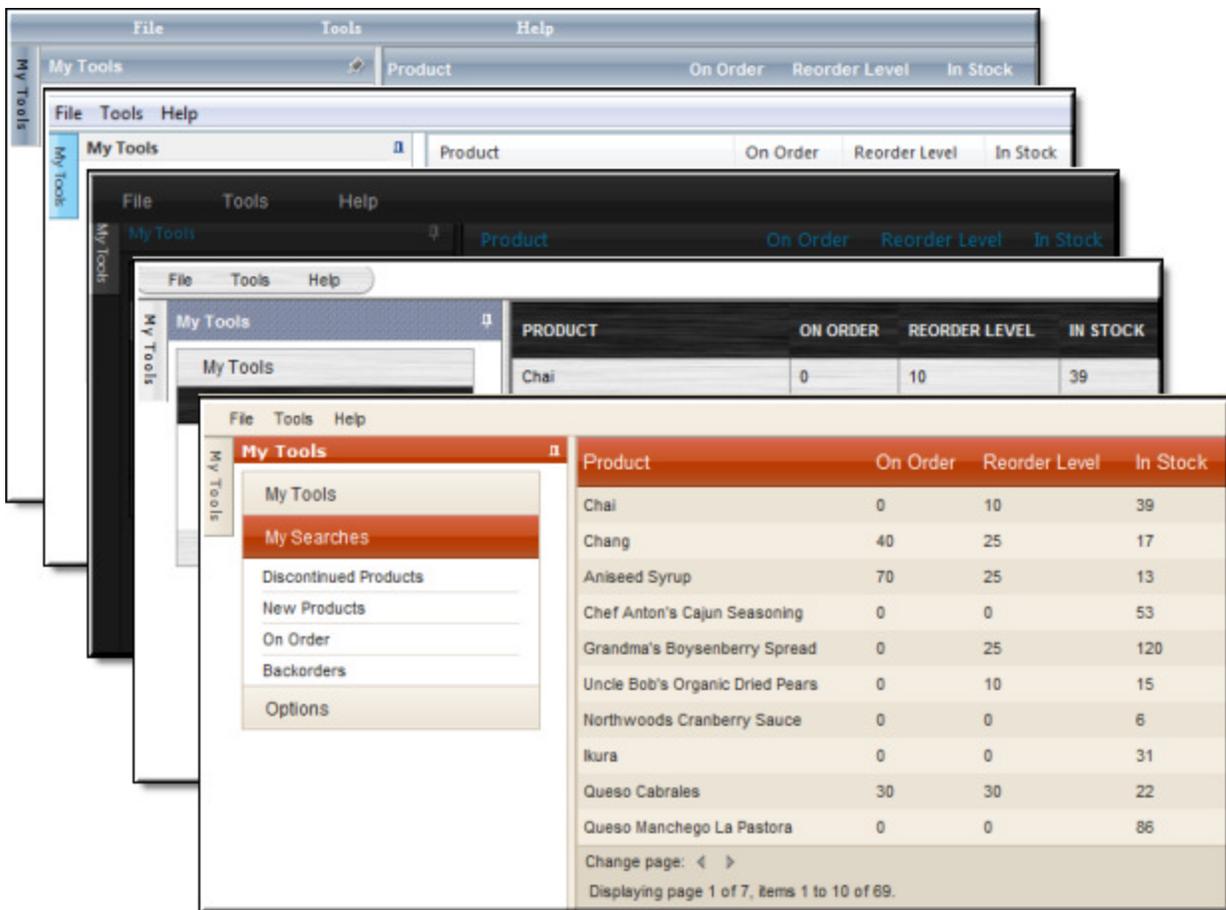
10 Skinning

10.1 Objectives

- Learn how to use the built-in skins to provide a consistent style to your controls and overall application
- Learn how skins use standard CSS files and how the styles interact with the rendered controls.
- Explore different techniques for registering and assigning skins.
- Alter an existing skin and create a custom skin.

10.2 Introduction

Skins let you style your entire application with a consistent look and feel. This example (screenshot below) shows the same application in multiple different skins. The code-behind and markup is completely identical between screenshots, so you can see that skinning truly separates appearance from functionality.



RadControls for ASP.NET AJAX use skins to control overall look-and-feel. A skin is a set of images and a CSS stylesheet that can be applied to control elements (items, images, etc.) to define their look and feel. To apply a skin to a RadControl, set its `Skin` property, either using the properties pane or the Smart Tag. Each control is installed with a number of preset skins that coordinate with the look of other RadControls using the same skin:

- Black
- Default

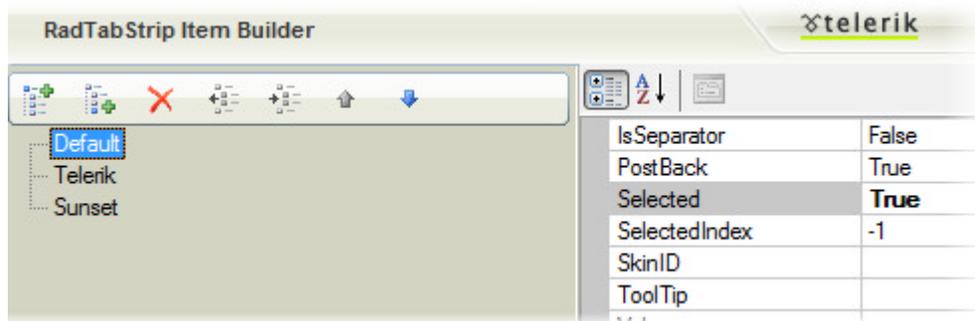
- Forest
- Hay
- Office2007
- Outlook
- Simple
- Sitefinity
- Sunset
- Telerik
- Vista
- Web20
- WebBlue
- Windows7

You can also alter existing skins or create your own skin from scratch. Set the Skin individually for each control in the designer, declaratively or in code. Or you can set the skin for the entire application in the web.config or using code you can dynamically change the appearance of all controls in the application at runtime.

10.3 Getting Started

This walk-through demonstrates assigning skins in code.

1. In a new web application, add a ScriptManager to the default page.
2. Add a **RadFormDecorator** to the page.
3. Add a **RadTabStrip** to the page. Set the **AutoPostBack** property to **True**.
4. Use the Smart Tag and click the **Build RadTabStrip...** option.
 - Add three tabs with **Text** "Default", "Telerik" and "Sunset".
 - Make sure that the first tab "Default" has the **Selected** property set to **True**.



5. Add some standard ASP Button, RadioButton and CheckBox controls on the page. These will be skinned automatically by the RadFormDecorator.



6. Double-click the RadTabStrip to create a **TabClick** event handler. Change the event handler to use the code shown below. *The code simply assigns the tab text, which happens to contain skin names, to the RadFormDecorator and RadTabStrip Skin properties.*

[VB] Handling the TabClick Event

```
Protected Sub RadTabStrip1_TabClick(ByVal sender As Object, ByVal e As Telerik.Web.UI.RadTabStripEventArgs)
    RadFormDecorator1.Skin = e.Tab.Text
    RadTabStrip1.Skin = e.Tab.Text
End Sub
```

[C#] Handling the TabClick Event

```
protected void RadTabStrip1_TabClick(object sender, Telerik.Web.UI.RadTabStripEventArgs e)
{
    RadFormDecorator1.Skin = e.Tab.Text;
    RadTabStrip1.Skin = e.Tab.Text;
}
```

7. Press **Ctrl-F5** to run the application. Try clicking the tabs to see how the three skins are applied.



10.4 Registering and Assigning Skins

Registering Skins

Skins use Cascading Style Sheets (CSS) to define a control's visual appearance. This CSS file needs to be registered in the page for the skin to take effect. By default, each RadControl has an **EnableEmbeddedSkins** property that is set to **True**. When you set the Skin property to a built-in skin name the CSS file will be automatically registered. If you have a custom skin, set **EnableEmbeddedSkins** to **False** and register the CSS manually. Later in this chapter we will get into the specifics of creating your own custom skin starting from an existing skin as a base.

You can drag the stylesheet from the Solution Explorer or type it directly to the markup. Both routes add a `<link>` tag inside the `<head>`. Depending on where in the solution folder explorer you have the CSS stored the reference can look like the example below:

[ASP.NET] Registering the CSS Manually

```
<head runat="server">
    . . .
    <link href="../../Skins/TabStrip.MySkin.css" rel="stylesheet" type="text/css" />
</head>
```

You could also programmatically add the link on the server. The example code below creates an `HtmlLink` and sets the properties to that matching a file "stylesheet1.css" residing in the root of the project and adds it to the page header controls collection. Note: `HtmlLink` requires a reference to the `System.Web.UI.HtmlControls` assembly.

[VB] Adding Style Sheet Programmatically

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' create and populate an HtmlLink so that it
    ' points to the file stylesheet1.css located
    ' in the root of the web application.
    Dim link As New HtmlLink()
    link.Href = "stylesheet1.css"
    link.Attributes.Add("type", "text/css")
    link.Attributes.Add("rel", "stylesheet")
    Page.Header.Controls.Add(link)
End Sub
```

[C#] Adding Style Sheet Programmatically

```
protected void Page_Load(object sender, EventArgs e)
{
    // create and populate an HtmlLink so that it
    // points to the file stylesheet1.css located
    // in the root of the web application.
    HtmlLink link = new HtmlLink();
    link.Href = "stylesheet1.css";
    link.Attributes.Add("type", "text/css");
    link.Attributes.Add("rel", "stylesheet");
    Page.Header.Controls.Add(link);
}
```

Assigning Skins

As we saw earlier, you can assign the `Skin` property declaratively, in the designer or in code. If you want to register the skin globally (setting the entire application to use a particular embedded skin), you can add application settings to `web.config`. The example `<appSettings>` element adds a "Telerik.Skin" key with value "Hay". This sets all the `RadControls` for the application that uses this configuration to display the "Hay" skin.

[ASP.NET] Adding Application Settings

```
<appSettings>
<!-- Sets the skin for all RadControls to Hay -->
<add key="Telerik.Skin" value="Hay"/>
</appSettings>
```

If you want to create a custom skin, you can also set the `EnableEmbeddedSkins` property for every `RadControl` in the application at one time. The example below registers a custom skin for every `RadControl` in the application:

[ASP.NET] Disable Embedded Skins and Add Custom Skin

```
<appSettings>
<add key="Telerik.Skin" value="MySkin" />
<add key="Telerik.EnableEmbeddedSkins" value="false" />
</appSettings>
```

You can also mix-and-match settings so that one skin applies to all `RadControls` except a control that has a specific skin applied. The example below assigns the skin "Hay" globally except for `RadMenu` which is assigned the "Sunset" skin.

[ASP.NET] Assigning Global and Specific Skins

```
<appSettings>
<add key="Telerik.Skin" value="Hay" />
<add key="Telerik.Menu.Skin" value="Sunset" />
</appSettings>
```

There is a specific priority to what skin is used when skins are assigned at different levels:

- The settings applied at page level have top priority.
- The settings applied for a particular control in the web.config file are next in priority.
- The settings applied for all RadControls globally in the web.config file are last in priority.

Assigning Skins Programmatically at Run Time

Somewhere back in the inheritance chain of each RadControl you will find ISkinnableControl that defines how the control interacts with skins. The interface definition is shown below. You can see that it defines the **Skin** property itself used to get and set the skin name. The **EnableEmbeddedSkins** property is also defined here. There is also a **GetEmbeddedSkinNames()** method that returns a list of skin names as a generic list of string. We can use this last method to populate a combo box so that we can select a skin name at run time.

[VB] ISkinnableControl Interface Definition

```
Public Interface ISkinnableControl
    Inherits IControl
    Property AjaxCssRegistrations() As String
    Property EnableAjaxSkinRendering() As Boolean
    Property EnableEmbeddedBaseStylesheet() As Boolean
    Property EnableEmbeddedSkins() As Boolean
    ReadOnly Property IsSkinSet() As Boolean
    Property Skin() As String
    Function GetEmbeddedSkinNames() As List(Of String)
End Interface
```

[C#] ISkinnableControl Interface Definition

```
public interface ISkinnableControl : IControl
{
    string AjaxCssRegistrations { get; set; }
    bool EnableAjaxSkinRendering { get; set; }
    bool EnableEmbeddedBaseStylesheet { get; set; }
    bool EnableEmbeddedSkins { get; set; }
    bool IsSkinSet { get; }
    string Skin { get; set; }
    List<string> GetEmbeddedSkinNames();
}
```

Skins can be assigned to all RadControls on the page at runtime. For that purpose you can use the **RadSkinManager** control by setting its **Skin** property. By default the **Skin** property of **RadSkinManager** is an empty string. If the property is set to a different value and **Enabled** is set to true for the control, the manager will apply automatically the specified skin to all RadControls on the form. If you set the **ShowChooser** property to true the manager will display run-time **RadComboBox** as a part of its smart tag, populated with all embedded skins, where you can pick a skin. You can use the **RadSkinManager** to assign specific Skin to a particular control by adding a setting to its **TargetControls** collection. However in the cases you want **RadSkinManager** Skin to be applied on any other RadControl on the page, the control **Skin** property should not be set explicitly. We will create an interface with a sampling of controls and use the control to apply the chosen style at runtime.

1. Create a new web application and add a **ScriptManager** to the default page.

- Drop a **RadSkinManager** on the form. Set the **ShowChooser** property to **True**.
- Add the following markup. It defines a **RadSplitter** layout and has comments embedded that we'll use to add additional controls.

[ASP.NET] Defining the RadSplitter Layout

```
<telerik:RadSplitter ID="RadSplitter1" runat="server" Orientation="Horizontal" Width="100%" Height="100%">
  <telerik:RadPane ID="menuPane" Runat="server" Height="23px" Scrolling="None">

    <!--add menu here-->

  </telerik:RadPane>
  <telerik:RadPane ID="RadPane4" runat="server" Scrolling="None">
    <telerik:RadSplitter ID="RadSplitter2" runat="server" Orientation="Vertical" Width="100%" Height="100%">
      <telerik:RadPane ID="RadPane2" runat="server" Scrolling="None">
        <telerik:RadSlidingZone ID="RadSlidingZone1" runat="server"
ExpandedPaneId="RadSlidingPane1">
          <telerik:RadSlidingPane ID="RadSlidingPane1" runat="server" Title="My Tools"
Scrolling="None">

            <!--add panel bar here -->

          </telerik:RadSlidingPane>
        </telerik:RadSlidingZone>
      </telerik:RadPane>
      <telerik:RadSplitBar ID="RadSplitBar1" runat="server" />
      <telerik:RadPane ID="RadPane1" runat="server" Scrolling="None">

        <!--add grid here-->

      </telerik:RadPane>
    </telerik:RadSplitter>
  </telerik:RadPane>
</telerik:RadSplitter>
```



As the layout becomes more complex, especially with **RadSplitter** controls, the design environment can become less useful. As an application reaches this complexity you may wish to work directly in the ASP.NET markup.

- Add a **RadMenu**, **RadPanelBar** and **RadGrid** to the markup. **Note:** we haven't covered the **RadGrid** yet but the purpose of the controls here is simply as a canvas to demonstrate skinning. The next steps will just involve copying in the ASP.NET markup:
 - Add **RadMenu** markup to the markup for the page where indicated by the comment. The markup simply defines a few items and sub-items.

[ASP.NET] The RadMenu Definition

```
<telerik:RadMenu ID="RadMenu1" runat="server" Width="100%" >
  <Items>
    <telerik:RadMenuItem runat="server" Text="File">
      <Items>
        <telerik:RadMenuItem runat="server" Text="Products">
        </telerik:RadMenuItem>
        <telerik:RadMenuItem runat="server" Text="Customers">

```

```

        </telerik:RadMenuItem>
        <telerik:RadMenuItem runat="server" Text="Orders">
        </telerik:RadMenuItem>
    </Items>
</telerik:RadMenuItem>
<telerik:RadMenuItem runat="server" Text="Tools">
</telerik:RadMenuItem>
<telerik:RadMenuItem runat="server" Text="Help">
</telerik:RadMenuItem>
</Items>
</telerik:RadMenu>

```

- Add RadPanelBar markup to the markup for the page where indicated by the comment. Again, this just sets up a couple levels of panel items so we can see how that looks when styled.

[ASP.NET] The RadPanelBar Definition

```

<telerik:RadPanelBar ID="RadPanelBar1" runat="server" Width="200px">
  <CollapseAnimation Duration="100" Type="None" />
  <Items>
    <telerik:RadPanelItem runat="server" Text="My Tools">
    </telerik:RadPanelItem>
    <telerik:RadPanelItem runat="server" Text="My Searches">
    <Items>
      <telerik:RadPanelItem runat="server" Text="Discontinued Products"
    ></telerik:RadPanelItem>
      <telerik:RadPanelItem runat="server" Text="New Products" ></telerik:RadPanelItem>
      <telerik:RadPanelItem runat="server" Text="On Order" ></telerik:RadPanelItem>
      <telerik:RadPanelItem runat="server" Text="Backorders" ></telerik:RadPanelItem>
    </Items>
    </telerik:RadPanelItem>
    <telerik:RadPanelItem runat="server" Text="Options">
    </telerik:RadPanelItem>
  </Items>
  <ExpandAnimation Duration="100" Type="None" />
</telerik:RadPanelBar>

```

- Add RadGrid markup to the markup for the page where indicated by the comment. We will add some minimal server code-behind to display a little data in the grid.

[ASP.NET] The RadGrid Definition

```

<telerik:RadGrid ID="RadGrid1" runat="server" GridLines="None" >
  <MasterTableView AutoGenerateColumns="true" ></MasterTableView>
</telerik:RadGrid>

```

5. Handle the **Page_Load** event to get a list of skins. It is used as datasource for the RadGrid. RadGrid implements **ISkinnableControl** and so has a **GetEmbeddedSkinNames()** method. (More on RadGrid and databinding in later chapters).

[VB] Handling the Page_Load Event

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  If Not IsPostBack Then
    Dim skins As List(Of String) = RadGrid1.GetEmbeddedSkinNames()
    RadGrid1.DataSource = skins
  End If

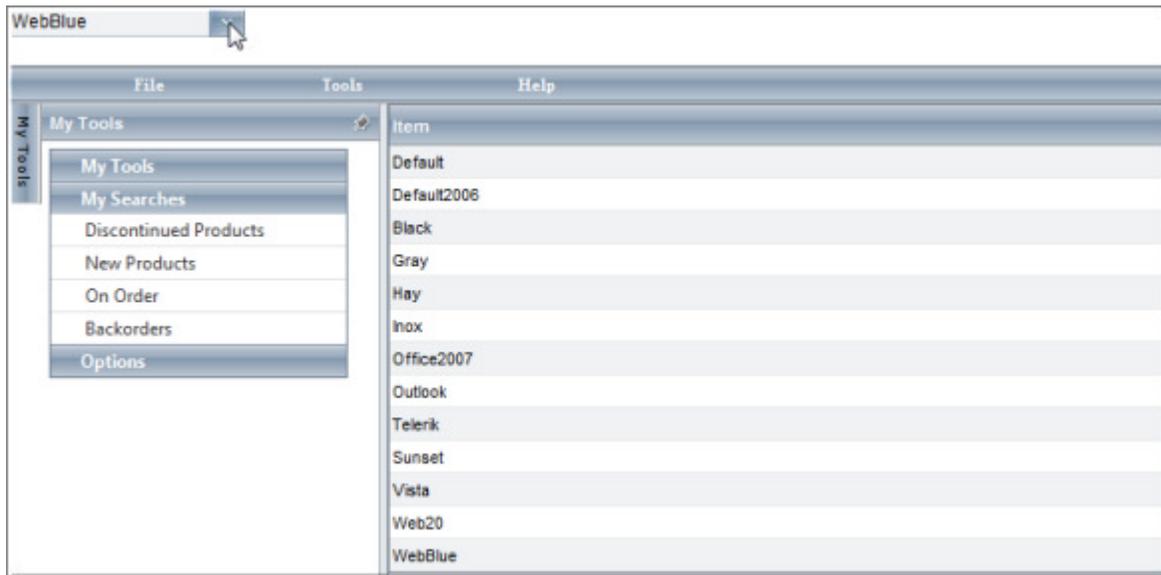
```

End Sub

[C#] Handling the Page_Load Event

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        List<string> skins = RadGrid1.GetEmbeddedSkinNames();
        RadGrid1.DataSource = skins;
    }
}
```

- Press **Ctrl-F5** to run the application. Select different skins and see the effect to the UI.



You can find the complete source for this project at:
 \VS Projects\Skinning\AssigningSkins

10.5 Understanding the Skin CSS File

Styles for RadControls are defined using Cascading Style Sheet (CSS) syntax. Each style consists of a selector that identifies an HTML element to be styled, and property/value pairs that describe each of the style specifics, e.g. color, padding, margins, etc. For example, the ".rcbHovered" style will have a light green background and white text.

[CSS] Example Style

```
.rcbHovered
{
    background:LightGreen;
    color:#fff;
}
```

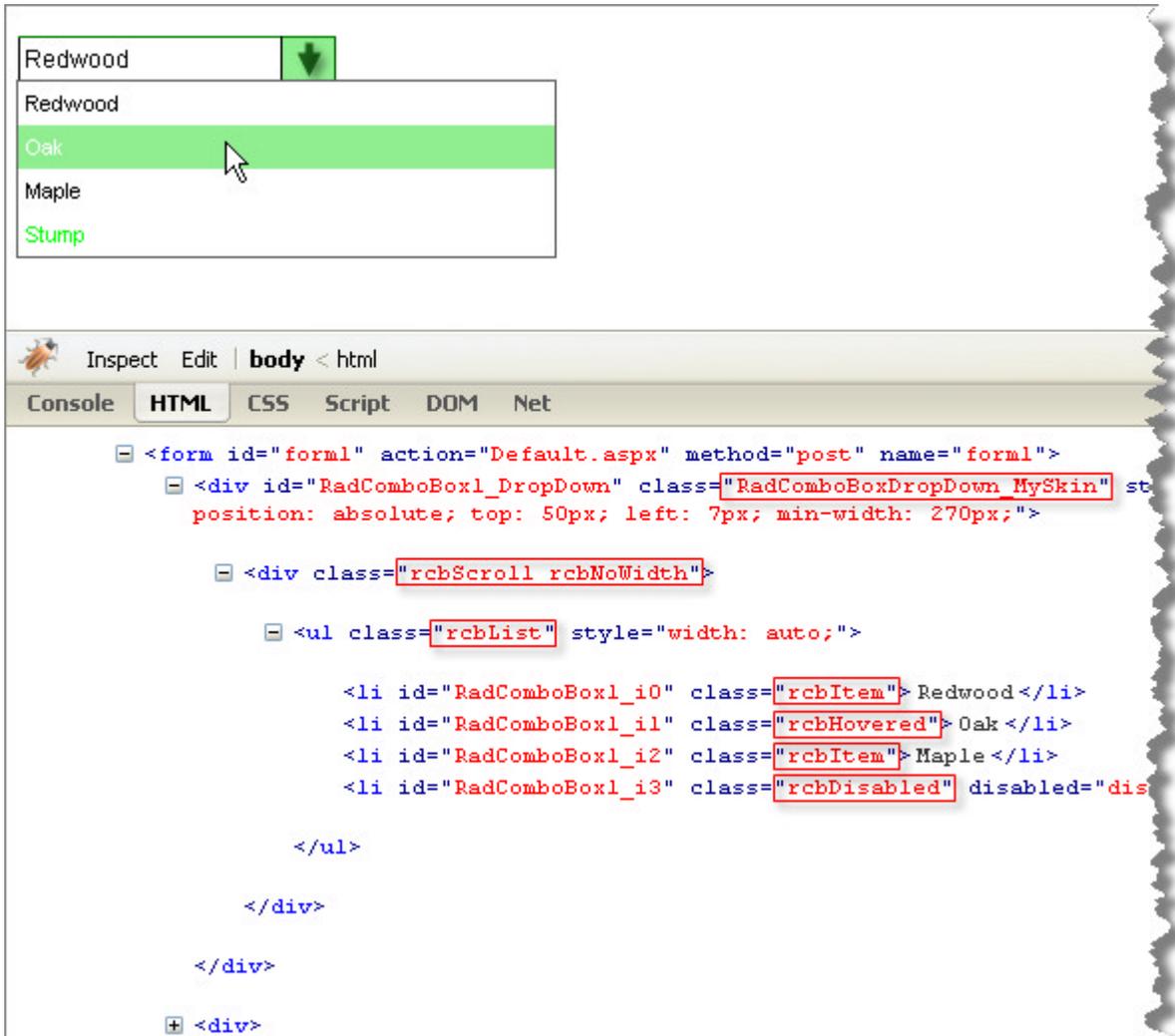
You can see custom styles applied to a RadComboBox in the screenshot below.

Each style maps to a "class" attribute in an HTML tag. For example, the RadComboBoxDropDown_MySkin is a general style that applies to the entire <div> tag that represents the control. Class attributes change as the

UI for ASP.NET AJAX

property values for the control change.

The image below shows a snapshot of what the rendered HTML looks like as the mouse passes over the "Oak" entry in the combo box. Notice the list of `` list item tags and that the class attribute for the second item is "rcbHovered".



When you need to change an existing skin, you can use various tools (to be discussed in a later chapter) to visualize the HTML and styles. From there you can discover specific styles that affect the aspects of appearance that you want to tweak. You can also find more information in the online help about specific RadControl CSS skin file selectors.

Skin Files

The directory where RadControls for ASP.NET AJAX are installed contains a `\Skins` folder. Typically the structure will be:

```
\Program Files\Telerik\RadControls for ASP.NET AJAX <version>\Skins
```

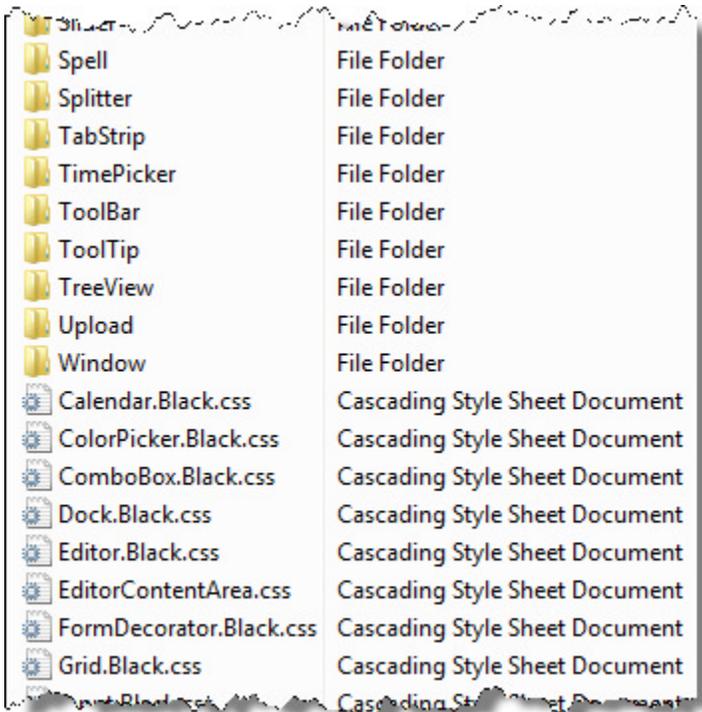
Here's a snapshot of the `\Skins` directory:

Name	Type
Black	File Folder
Default	File Folder
Default2006	File Folder
Gray	File Folder
Hay	File Folder
Img	File Folder
Inox	File Folder
Office2007	File Folder
Outlook	File Folder
Sunset	File Folder
Telerik	File Folder
Vista	File Folder
Web20	File Folder
WebBlue	File Folder
Chart.css	Cascading Style Sheet Document
ColorPicker.css	Cascading Style Sheet Document
Dock.css	Cascading Style Sheet Document
Editor.css	Cascading Style Sheet Document
FormDecorator.css	Cascading Style Sheet Document
Menu.css	Cascading Style Sheet Document
PanelBar.css	Cascading Style Sheet Document
Rotator.css	Cascading Style Sheet Document
RotatorButtons.gif	GIF Image
Scheduler.css	Cascading Style Sheet Document

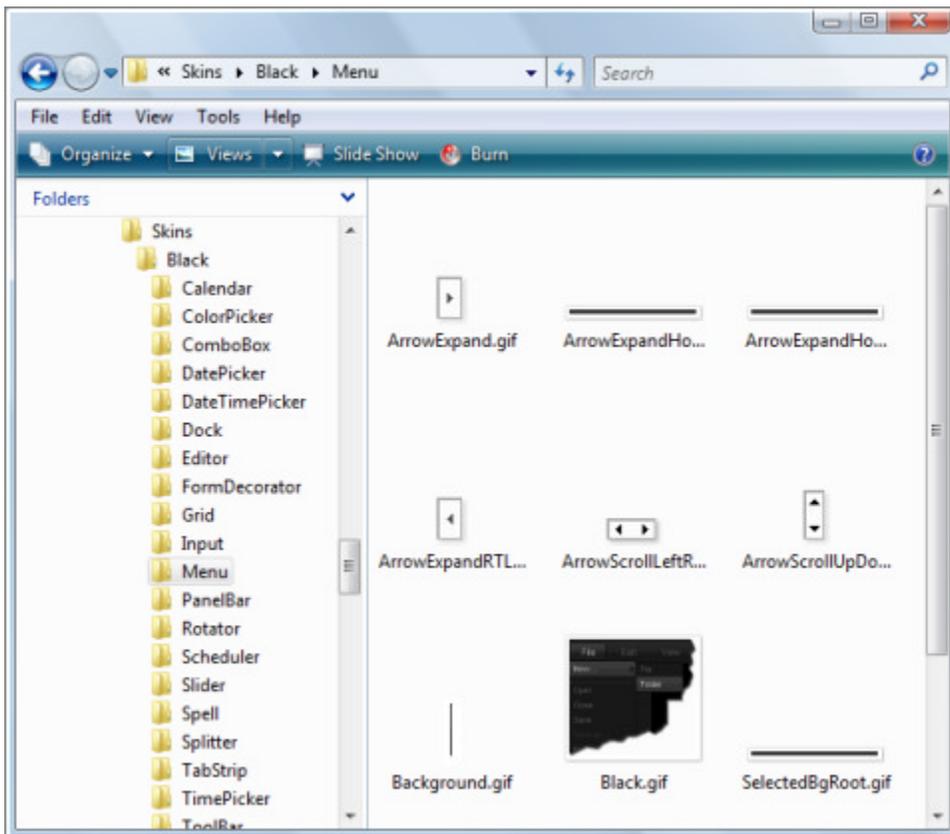
Within the \Skins directory you will find a series of CSS files, one for each control, e.g. Editor.css, ColorPicker.css, etc. These are the **base style sheets** and govern the basic appearance and layout of the control that are not skin specific. For instance, a RadMenu has a certain general layout with menu elements having a particular relationship to one another that makes it look like a menu and not a combo box for instance. In the base style sheet you won't typically find colors and images. These are left to the skin specific style sheets.

At the same level in the \Skins directory alongside the base style sheets you find a series of folders named after the skins they define. Inside one of these skin-named folders is another set of CSS files with naming convention <control name>.<skin name>.css.

UI for ASP.NET AJAX



The folders contain a set of images that are referenced by the the stylesheets:



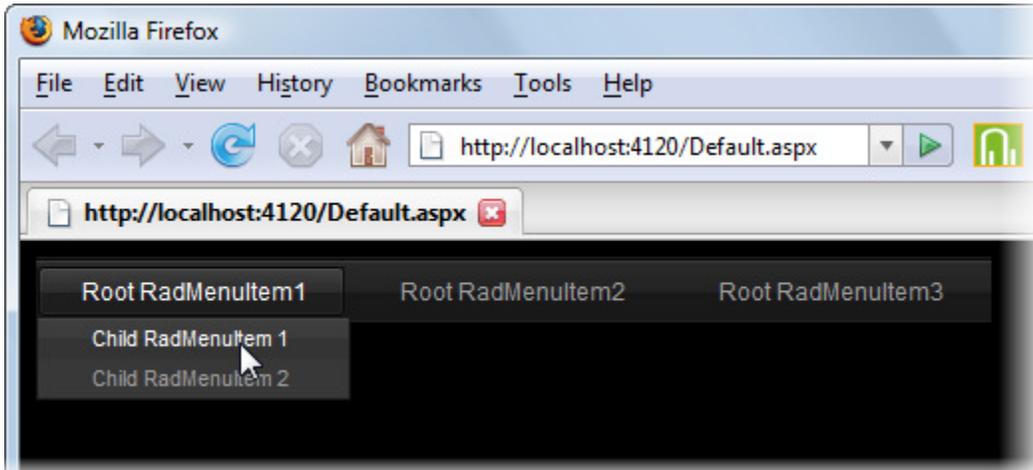
The images and style sheets can be used as starting points for your own custom skins or as reference material when creating skins from scratch.

💡 You can also get the original PhotoShop files for skin images at: <http://www.telerik.com/skins>.

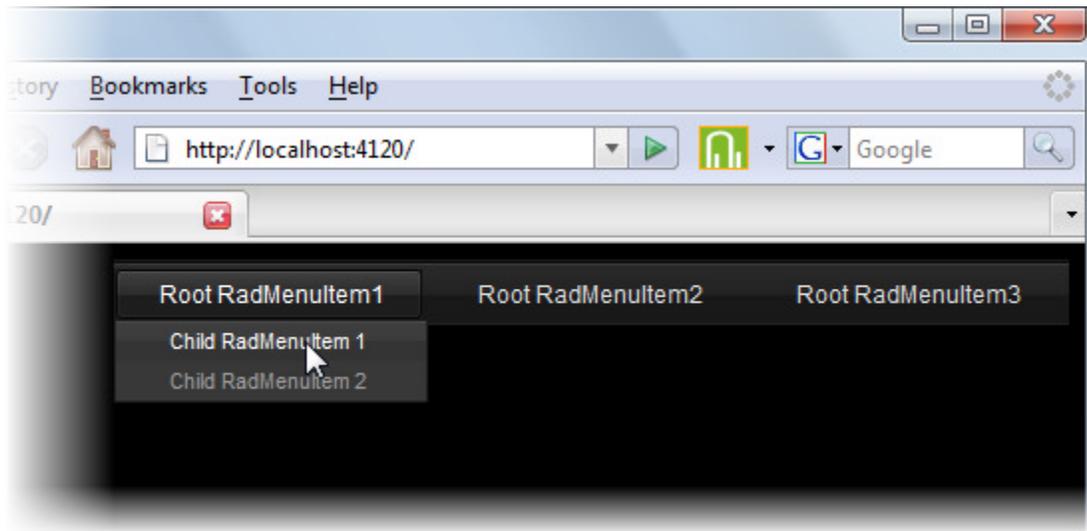
10.6 Creating a Custom Skin

Alter an Existing Skin

The "Black" skin for RadMenu floats the menu to the left and the un-selected top level menu items are relatively dim (see screenshot below).

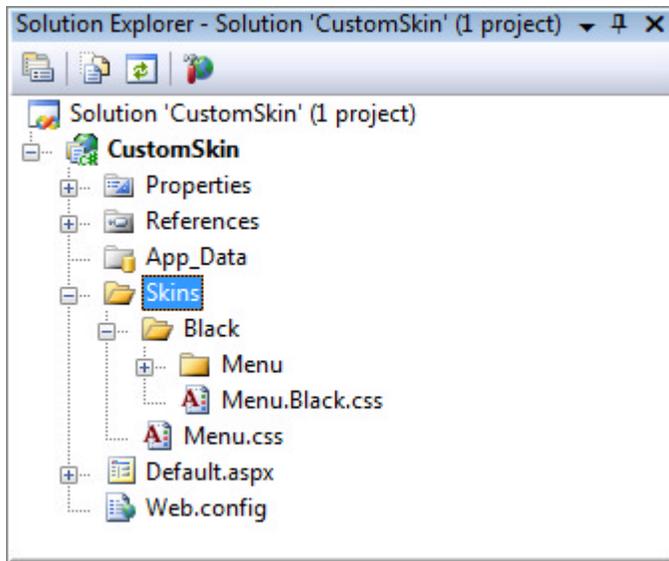


We can change one of the built-in skins to float the menu to the right and brighten up the top level font a few shades (see screenshot below).



The position of the menu is defined in the base style sheet, i.e. Menu.css and the color of the top level menu items is defined in Menu.Black.css.

1. Create a new web application, add a **ScriptManager** and a **RadMenu** to the page. Use the RadMenu Smart Tag to invoke the **Build Rad Menu** option. Add a few top level and child items to the menu. Set the **Skin** property to "Black".
2. Copy from the Telerik installation directory \Skins folder the file "menu.css" to the \Skins folder in your application. Within your application's \Skins folder, create a sub folder "\Black". Copy from the Telerik \Skins\Black folder to the \Black folder in your application both the file "Menu.Black.css" and the \Menu folder. Your project should now look something like the screenshot below:



3. In the designer, select the RadMenu and set **EnableEmbeddedBaseStyles** to **False** and **EnableEmbeddedSkins** to **False**. Now the RadMenu will display based off the CSS and images in your \Skins directory, not the embedded resources. Any changes you make to the CSS will now show up immediately in the control.
4. Open up "Menu.css" for editing. In the ".RadMenu" element change the "float" attribute to "right". The entire menu will not be right-justified on the web page.

[CSS] Editing Menu.css

```
.RadMenu
{
white-space:nowrap;
float:right;
position:relative;
}
```

5. Open up \Black\Menu.Black.css for editing. Locate the CSS selector ".RadMenu_Black .rmLink". Change the "color" attribute to "#DfDfDf". This brightens up the un-selected top-level font.

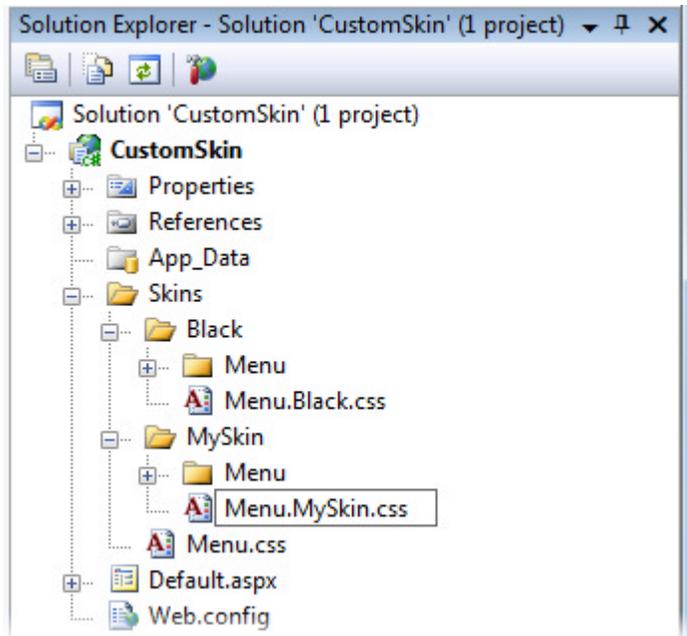
[CSS] Editing Menu

```
.RadMenu_Black .rmLink
{
line-height: 32px;
text-decoration: none;
color: #DfDfDf;
padding-left:3px;
}
```

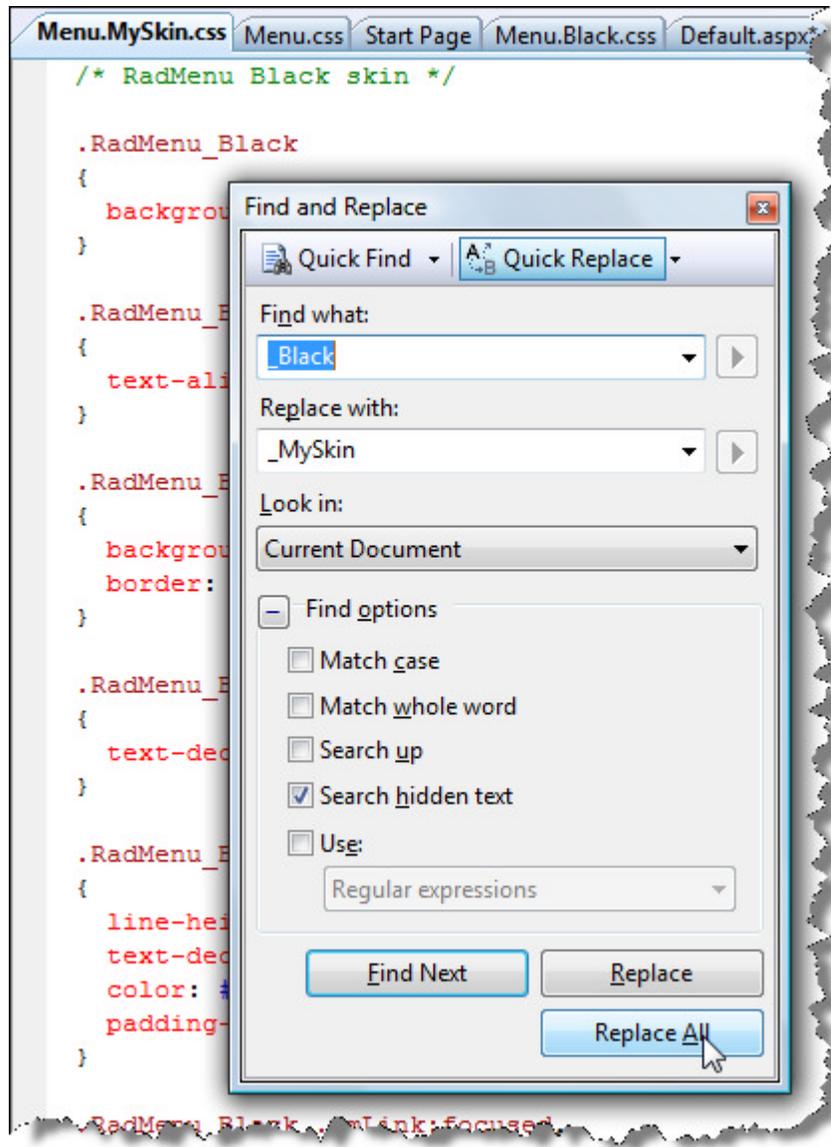
Create Your Own Skin

With just a little more work we can make our own unique skin with its own skin name.

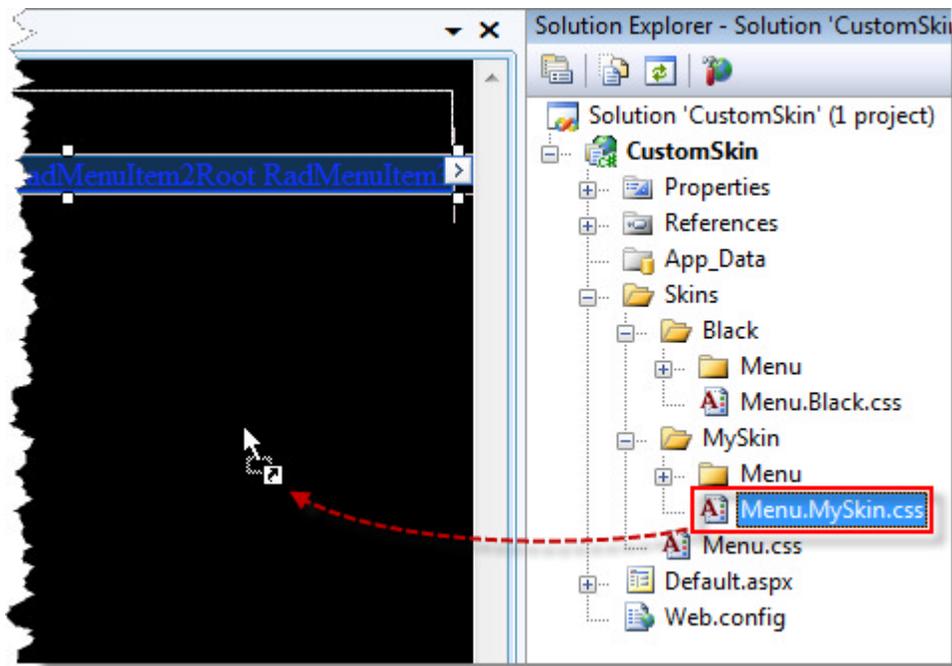
1. In the Solution Explorer, right-click the \Skin folder and select **Add | New Folder**. Name the new folder "MySkin". This folder will be a sibling to the \Black folder.
2. Copy the contents of the \Black folder to \MyFolder.
3. Rename the file "Menu.Black.css" to "Menu.MySkin.css".



4. Open Menu.MySkin.css for editing. Replace all "_Black" with "MySkin".
 - Press **Ctrl-H** to find and replace.
 - "Find What:" should be "_Black"
 - For "Replace with:" enter "_MySkin".
 - Click the **Replace All** button.



5. Set the **Skin** property of the RadMenu to "MySkin".
6. At this point you might expect to see your changes, but there's one last task. Drag "Menu.MySkin.css" to the page. *This will register the new css file. It's very easy to forget this step...*



10.7 Summary

In this chapter you learned how to use the built-in skins to provide a coherent, consistent style to your applications. You explored the general makeup of the skin CSS files and how the styles interact with the controls rendered in the browser. You learned multiple techniques for registering and assigning skins. You created your own custom skin starting with the "Black" skin as a basis.

11 Databinding

11.1 Objectives

- Introduce the bindable interfaces and Data Source controls.
- Get a feel for declarative data binding by a RadToolBar to a SqlDataSource.
- Review the data binding related properties and learn how to bind to multiple data sources at one time.
- Learn how to bind RadControls in code, starting with binding simple arrays and lists, then binding hierarchical data, business objects and LINQ data.
- Learn how to handle data binding related server events.

11.2 Introduction

Any industry-strength web application relies on database data, for example MS SQL, Oracle or MySQL. RadControls can automatically bind to data stores using standard Microsoft supplied data source controls **SqlDataSource**, **AccessDataSource**, **XmlDataSource**, **LinqDataSource** and **ObjectDataSource** or any **DataSourceControl** implementation. Once the control is bound you can use the data for display-only or configure the data source to support full CRUD (Create Read Update and Delete) operations. This chapter deals with some of the common ways you can work with data in your controls.

 Some of the more complex controls like RadChart, RadGrid and RadScheduler have additional properties, capabilities and UI helpers that will be explored within their own chapters.

You can bind RadControls to a data source that implements one of the following interfaces:

- **IEnumerable**: Supports simple iteration of a collection.
- **ICollection**: Extends IEnumerable and supports size, enumerator, and synchronization methods for collections.
- **IList**: Extends ICollection and is the base class for lists.
- **IBindingList**: Extends IList and supports binding to a data source.
- **IListSource**: Provides functionality to an object to return a list that can be bound to a data source.

Some of the implementations of these interfaces include generic lists, Arrays, ArrayLists, CollectionBase objects, and DataView/DataTable/DataSet objects. You can also bind to business objects in multi-tier scenarios using ObjectDataSource, bind to XML using XmlDataSource and, for best flexibility and performance in .NET 3.0, **LinqDataSource** fits the bill nicely.

You can bind most of the RadControls declaratively or at design-time and in server-side code. And of course you can couple AJAX requests with your server-side code for better performance. Many RadControls, such as RadGrid and RadTreeView, also allow binding in client code with data coming directly from a web service.

 It is best to use AJAX/server-code when some kind of process is involved and to use WebServices/client-code for data presentation. AJAX maintains the whole application state, providing integrity between the different components on the page, while Web services are pure, lightweight but have no sense of state.

The properties used to bind each of the RadControls are very similar. Once you know how it works for one control, you can apply that knowledge against the rest of the controls. There are some variations on this rule:

- Some of the more complex controls, particularly RadChart and RadGrid have additional properties used to map data to particular aspects of the specific control.
- Controls that are designed to display a hierarchy like RadTreeView and RadMenu have additional properties that define parent and child records forming the hierarchy. Controls such as RadComboBox or RadToolBar, are designed to work with flat data structures and have fewer properties as a result.
- Hierarchical controls also have a DataBindings that let you declaratively bind columns of the database to

arbitrary properties in the control.

RadControls that allow templates for completely free-form layout can also be bound declaratively using server tags with syntax similar to this:

[ASP.NET] Binding in Markup

```
<%# DataBinder.Eval(Container.DataItem, "ColumnName") %>
```

 Databinding in templates will be explored in the upcoming chapter on Templates.

These multiple binding techniques provide flexibility to let you populate RadControls in any web application building situation.

11.3 Getting Started

The easiest way to bind a RadControl is declaratively, right in the designer or through ASP.NET markup. The ASP.NET 2.0 Data Source controls allow you to simply set the control's DataSourceID property to the ID of a data source and "taa daa!" you not only can display the data, you can access and modify the data as well (depending on the particular functionality of the control you are working with). To set up this relationship between Data Source and RadControl you need:

- A **connection string** that defines the kind of database and its location. The connection string can be defined in the web.config file. For this walk-through we want to access standard Microsoft supplied "Northwind" data. The file Northwind.mdf and several other example databases are stored in the RadControls installation directory under:

```
\Telerik\RadControls for ASPNET AJAX<version>\Live Demos\App_Data
```

- A **Data Source** control that refers to the connection string. The Data Source defines a specific query that defines the table and columns to be retrieved. This walk-through will pull a product category name and ID from the Categories table in the Northwind database.
- The **RadControl DataSourceID** points to the ID of the Data Source on our web page and displays the data.
 -  RadChart, RadGrid and RadScheduler are complex controls that have additional properties and UI helpers that will be discussed further within their own chapters

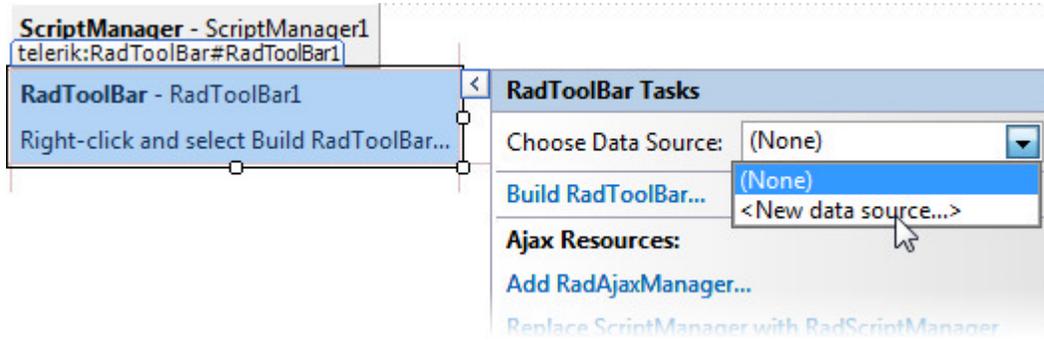
Simple Declarative Databinding



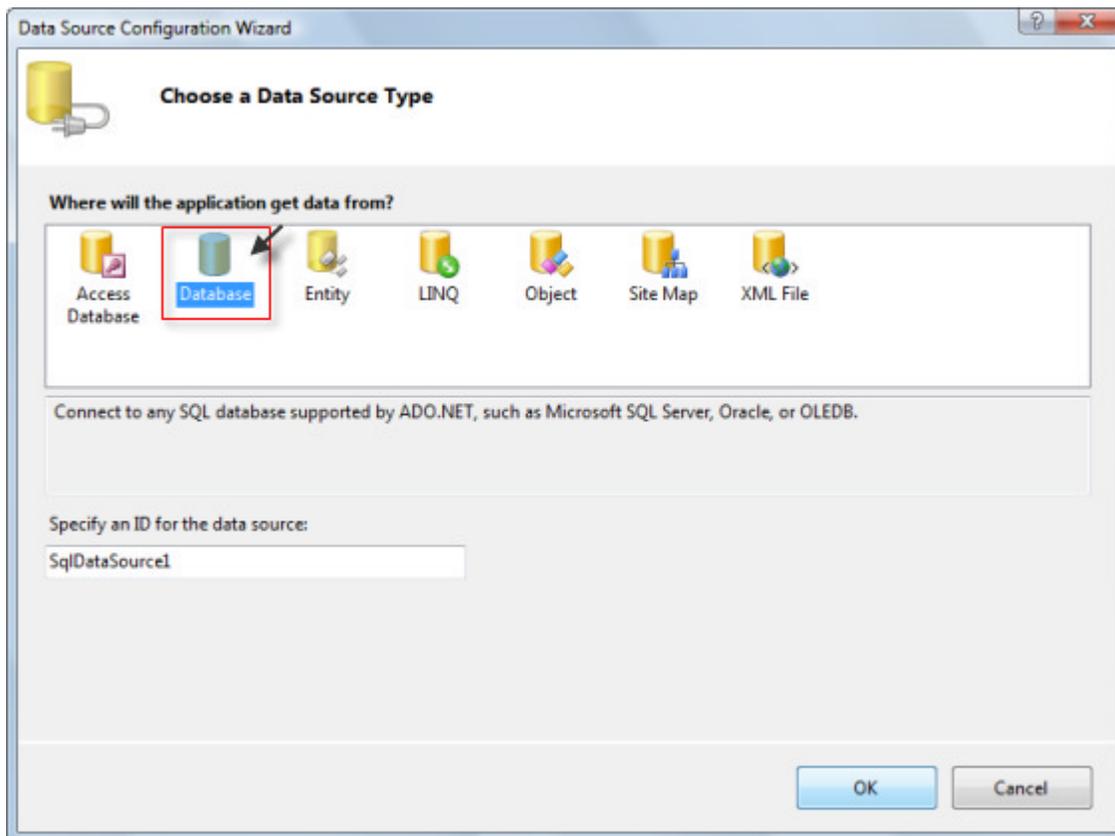
You can find the complete source for this project at:
 \VS Projects\Databinding\GettingStarted1

Let's start the walk-through using a RadToolBar control that by default displays data in a relatively simple, non-hierarchical form.

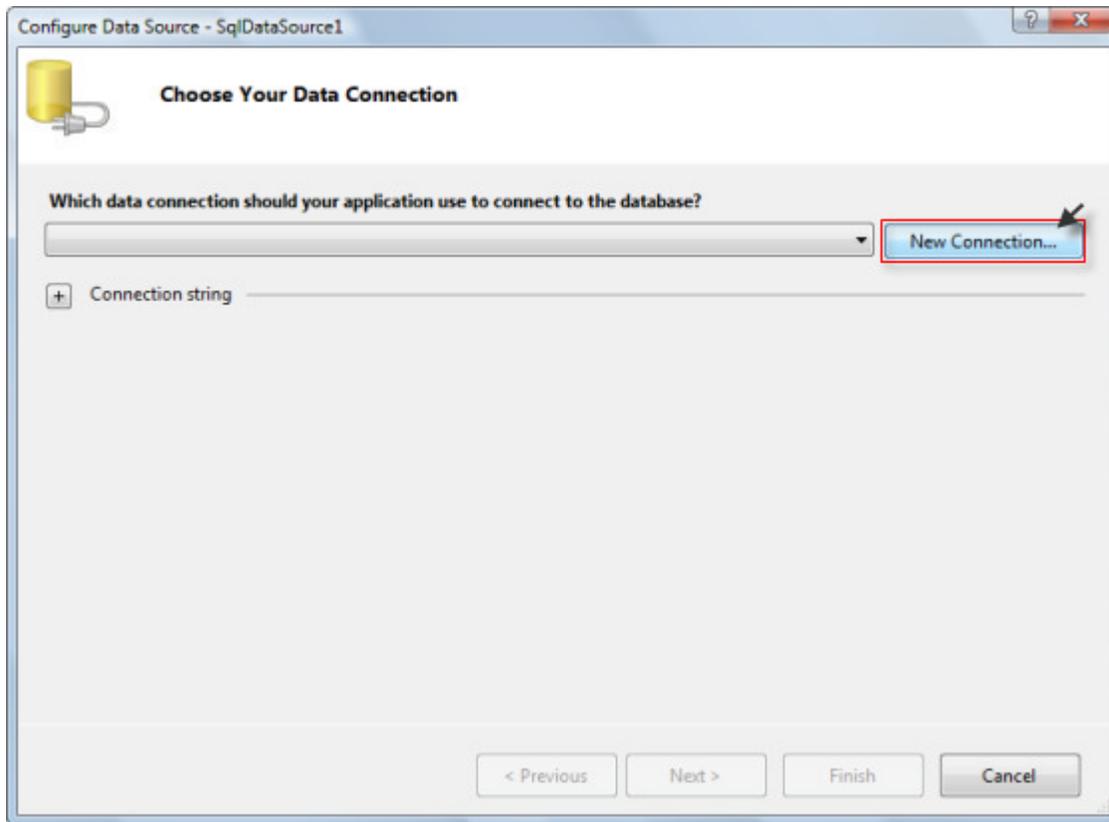
1. In a new web application add a **ScriptManager** control to the web page.
2. Add a **RadToolBar** to the web page.
3. From the RadToolBar Smart Tag, select **Choose Data Source** | **<New Data Source...>**. *This step will display the standard Data Source Configuration Wizard.*



4. In the Data Source Configuration Wizard "Choose a Data Source Type" page, select the **Database** option. Click the **OK** button.

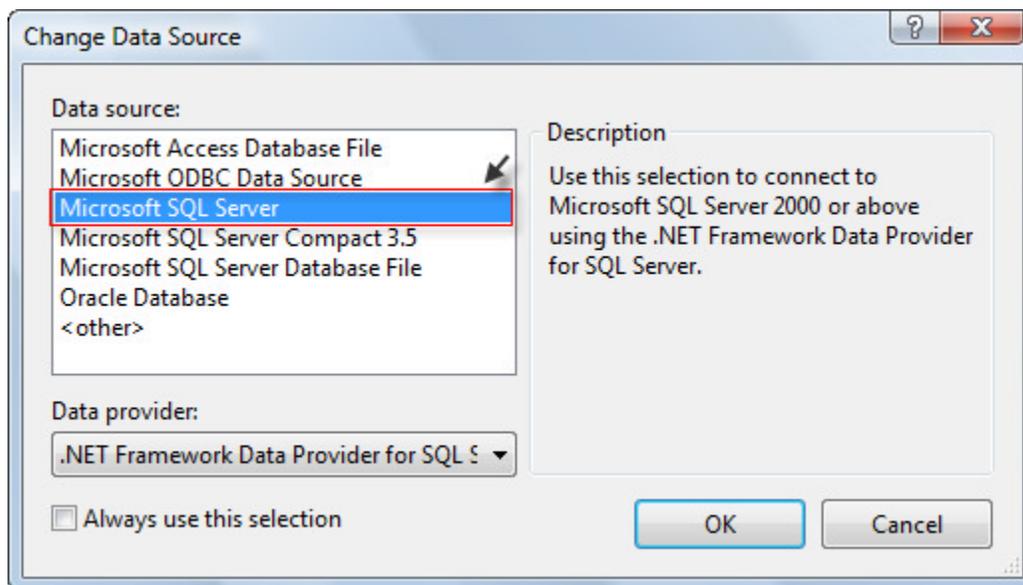


5. In the "Choose your Data Connection" page of the wizard click the **New Connection...** button. *This step will display the Add Connection dialog.*

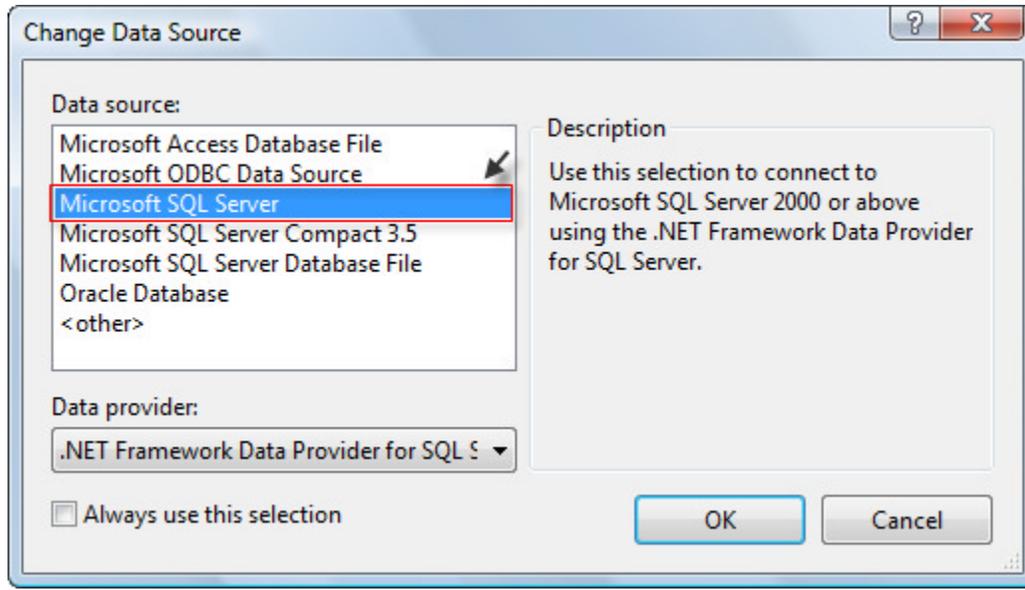


6. Create a new connection:

- In the Add Connection dialog, click the **Change...** button. *This step will display the Change Data Source dialog.*



- In the Change Data Source dialog, select the "Microsoft SQL Server" data source type and click the **OK** button. *This step will return you to the Add Connection dialog.*



- In the Add Connection dialog:
- Enter the **Server name** as ".\SQLEXPRESS".
- In the **Connect to a database** area of the dialog, click the **Attach to a Database File** option. Click the **Browse** button. Navigate to the directory where you installed RadControls for ASP.NET AJAX. Select the database file you want to use, e.g. "Northwind.mdb" and click the **Open** button to select the path.
- Click the **Test Connection** button to display a success alert if the settings are correctly entered.
- Click the **OK** button to close the Add Connection dialog.

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source:
Microsoft SQL Server (SqlClient) Change...

Server name:
.\SQLEXPRESS Refresh

Log on to the server

Use Windows Authentication
 Use SQL Server Authentication

User name:
Password:
 Save my password

Connect to a database

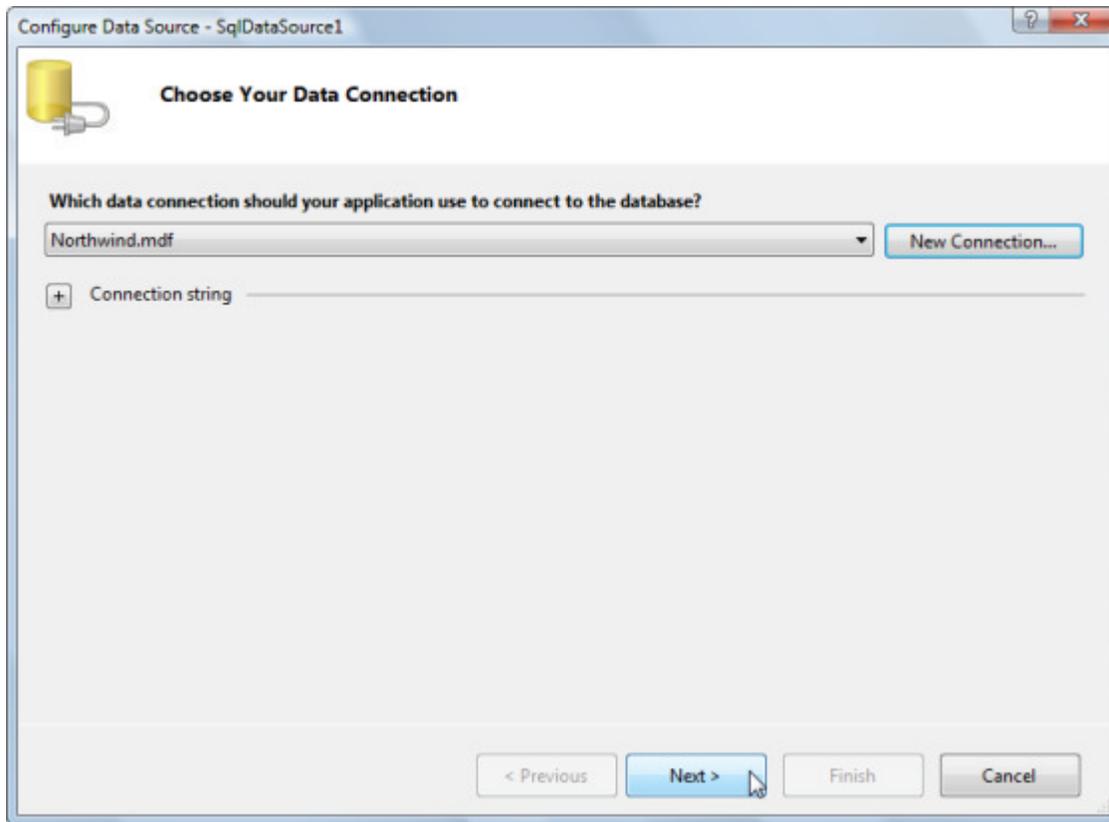
Select or enter a database name:

Attach a database file:
C:\Program Files\Telerik\RadControls for ASPNET Browse...
Logical name:

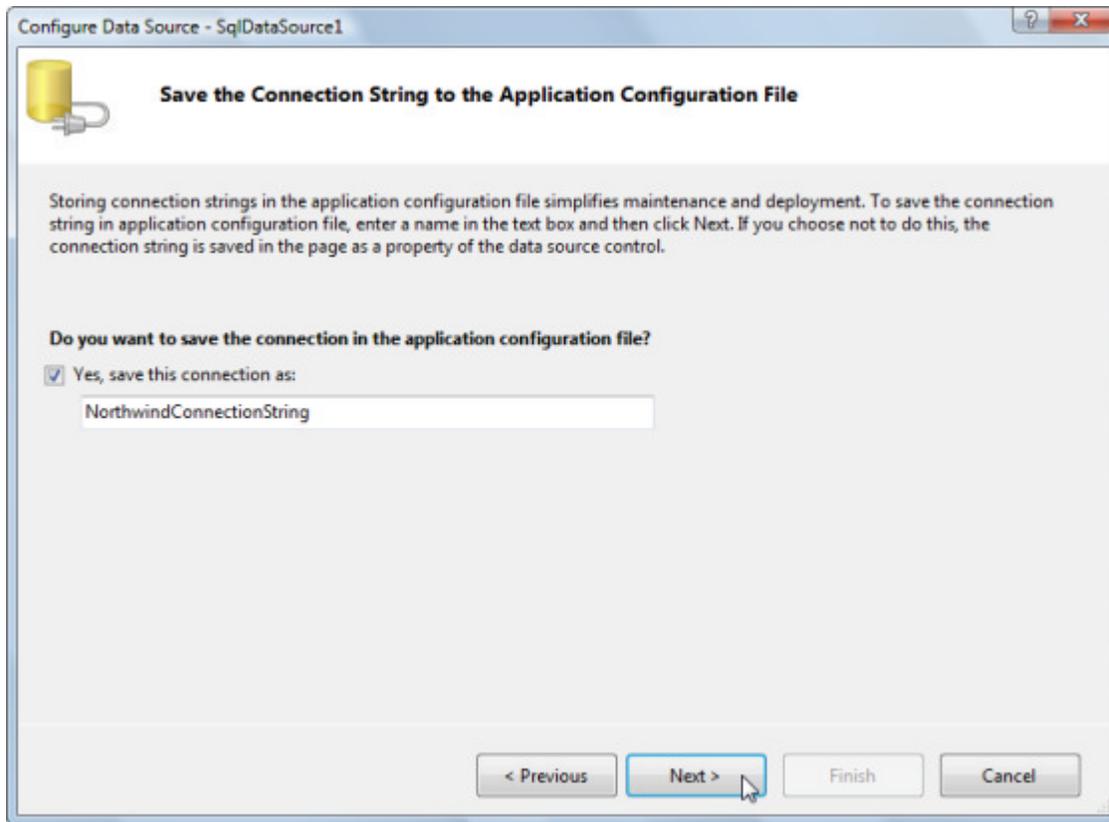
Advanced...

Test Connection OK Cancel

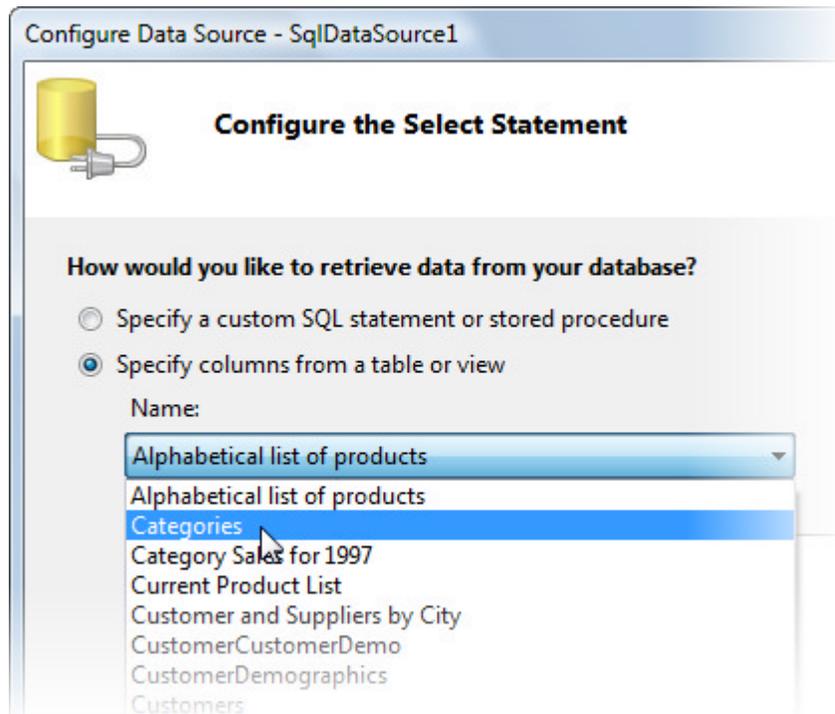
7. At the "Choose Your Data Connection" page of the wizard, click the **Next** button.



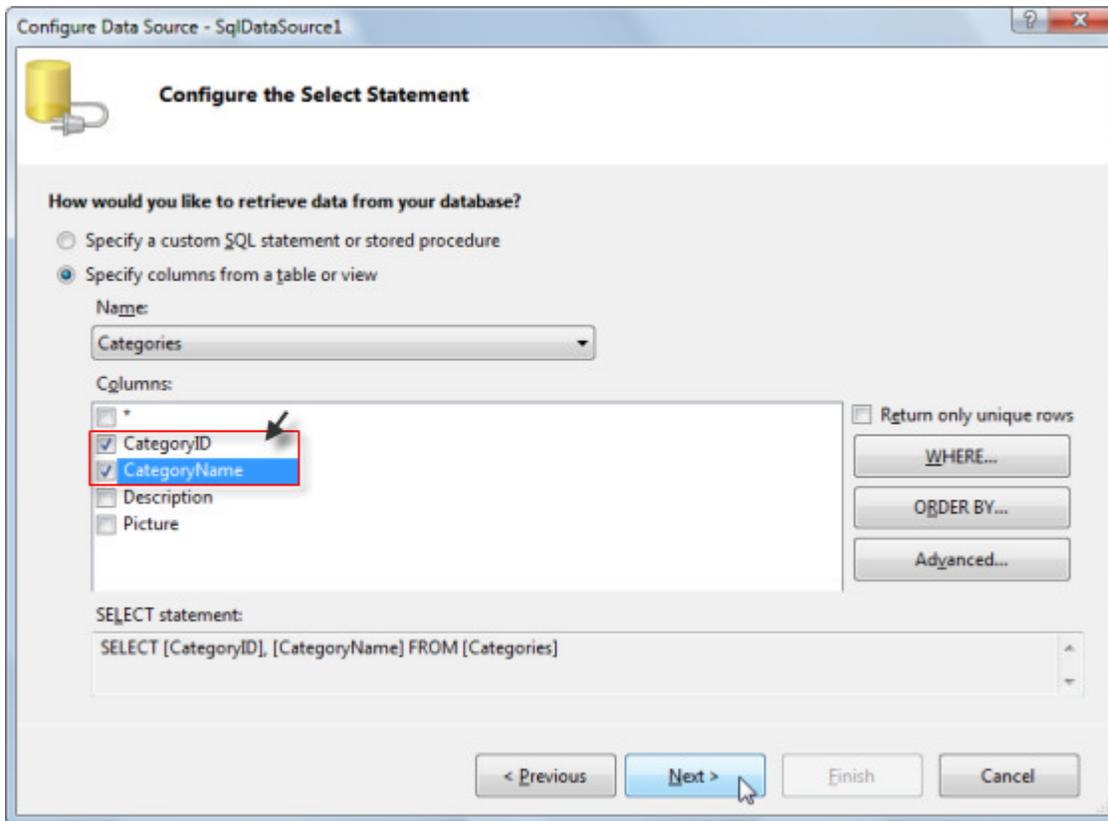
8. In the "Save the Connection String to the Application Configuration File", leave the defaults and click the Next button.



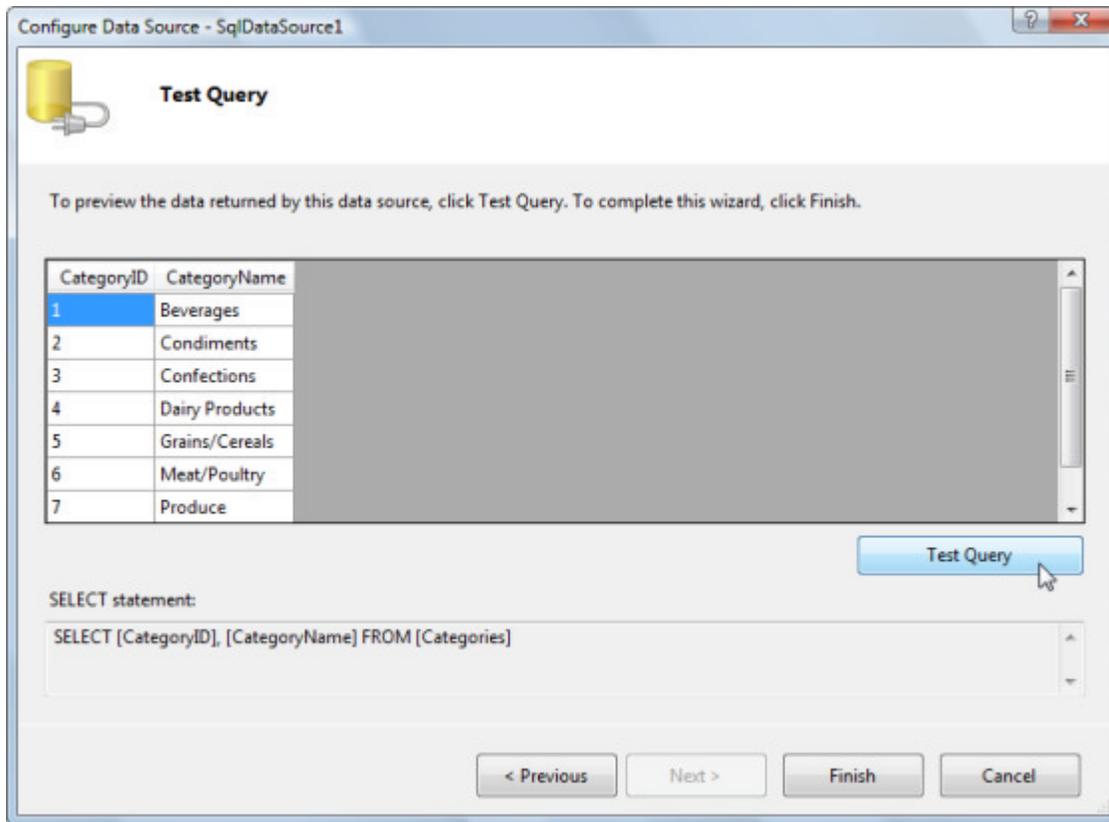
9. In the "Configure the Select Statement" of the wizard:
 1. Select the **Specify columns form a table or view** radio button option.
 2. Select "Categories" from the drop down list of table names.



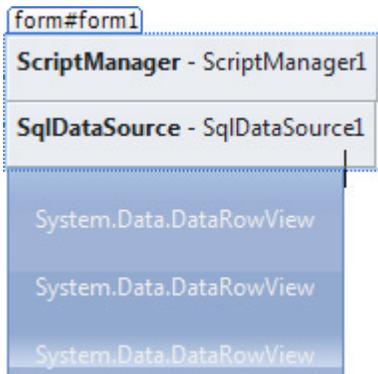
10. In the "Configure the Select Statement" page of the wizard, select "CategoryID" and "CategoryName" from the columns list. Click the **Next** button.



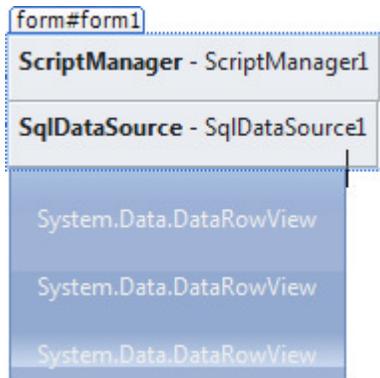
11. In the "Test Query" page of the wizard, click the **Test Query** button to see that the list of category names and ID's are listed. Click the **Finish** button to automatically create the `SqlDataSource` control and assign it to the `RadToolBar DataSourceID`.



- The design view of the page will look something like the example below. Notice that the `SqlDataSource` returns `DataRowView` objects and these are displayed because the `RadToolBar` doesn't know what columns should be used yet.



- In the Properties Window, set the `DataTextField` property to "CategoryName". The design view doesn't show the actual data yet, but does show a series of "abc" to indicate the column will have string data. **Note:** The `Skin` property here is set to "Web20".



14. Press **Ctrl-F5** to run the application. The category names show up as `RadToolBarButton` instances, one button per category name.



Data Properties

RadControls that can be bound to Data Sources use these basic properties:

- **DataSourceID:** If you are binding declaratively in ASP.NET markup (or setting the properties at design-time) you need to point `DataSourceID` at the ID of a Data Source control.
- **DataSource:** In complex or dynamic scenarios you may need to assign the Data Source at runtime. Instead of setting `DataSourceID`, leave that blank and assign the `DataSource` to the Data Source object (not the ID of the Data Source). You must also call the `DataBind()` method of the control after setting `DataSource`.



Gotcha! If you assign both `DataSourceID` at design-time and `DataSource` at runtime you get an error similar to "System.InvalidOperationException: Both DataSource and DataSourceID are defined on 'RadToolBar1'. Remove one definition."

- **DataMember:** If your Data Source is actually a `DataSet` object, how does the control know which table of the `DataSet` to bind to? `DataMember` specifies a table name of a `DataSet`. If `DataMember` is blank then the first table of the `DataSet` is used.
- **DataTextField:** You used this property in the last walk-through to bind a column to the `Text` property. The `DataTextField` column data is what the user will see in the user interface.
- **DataValueField:** This property is used to bind a column to the `Value` property of the control. You might typically use the `DataValueField` for ID column data while the `DataTextField` displays what the user will see.
- **DataNavigateUrlField:** Some of the navigation controls include this field to contain a column name that holds a URL. When a navigation item is clicked on the browser automatically navigates to the associated URL.

A few other properties fine-tune the data binding behavior of RadControls:

- **DataTextFormatString:** You can format the data automatically by defining a format string, e.g. "Category: {0}". Use the same formatting rules as used by the `String.Format()` method. The "0" column will represent the data found in the `DataTextField` column.
- **AppendDataBoundItems:** By default this property is false and data bound to the control will overwrite whatever data is already there. If `AppendDataBoundItems` is `True`, you can bind to multiple data sources or mix and match between items added at design-time and additional data introduced through binding. For example, you could add a tool bar button at design-time "My Button", then bind to "Categories" and finally

bind again to "Products". All these items would be displayed at one time in the tool bar. The binding at runtime to both category and product data sources will look like the example below. Note that each assignment of the **DataSource** property has to be followed by a call to the **DataBind()** method.

[VB] Binding to Multiple Data Sources

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' assign and bind the first data source
    RadToolBar1.DataSource = categoryDataSource
    RadToolBar1.DataBind()
    ' assign and bind the second data source
    RadToolBar1.DataSource = productDataSource
    RadToolBar1.DataBind()
End Sub
```

[C#] Binding to Multiple Data Sources

```
protected void Page_Load(object sender, EventArgs e)
{
    // assign and bind the first data source
    RadToolBar1.DataSource = categoryDataSource;
    RadToolBar1.DataBind();
    // assign and bind the second data source
    RadToolBar1.DataSource = productDataSource;
    RadToolBar1.DataBind();
}
```

Let's extend the previous tool bar example to include some of the data binding properties.



You can find the complete source for this project at:
 \VS Projects\Databinding\GettingStarted2

1. Use the previous example (or a copy of the previous example) as a starting point.
2. In the design-view of the page, from the RadToolBar Smart Tag select **Build RadToolBar...**
3. Add a single tool bar button and set the **Text** property of the button to "My Button". Set the **ToolTip** property to "This button defined at design-time".
4. Click **OK** to close the dialog.
5. In the Properties Window, set the **AppendDataBoundItems** property to **True**.
6. Set the **DataTextFormatString** property to "Category: {0}".
7. Set the **DataValueField** property to "CategoryID".
8. Add a **RadToolTipManager** control to the form. We will use tool tips display text and value data bound to each button.
9. In the design view of the page, select the RadToolBar and from the Properties window events tab  double-click the **ButtonDataBound** event. In the event handler that gets created for this event add the code below. *The code will set the tool tip text to use the category name and id from the bound row data.*

[VB] Handling the ButtonDataBound Event

```
Protected Sub RadToolBar1_ButtonDataBound(ByVal sender As Object, ByVal e As Telerik.Web.UI.RadToolBarButtonEventArgs)
    ' Get a reference to the row associated with the item
    Dim row As DataRowView = (TryCast(e.Button.DataItem, DataRowView))
```

```
' Assign the tooltip using the row data
e.Button.ToolTip = "Category Name: " + row("CategoryName") + " ID: " + row("CategoryID");
End Sub
```

[C#] Handling the ButtonDataBound Event

```
protected void RadToolBar1_ButtonDataBound(object sender,
    Telerik.Web.UI.RadToolBarButtonEventArgs e)
{
    // Get a reference to the row associated with the item
    DataRowView row = (e.Button.DataItem as DataRowView);
    // Assign the tooltip using the row data
    e.Button.ToolTip = "Category Name: " + row["CategoryName"] +
        " ID: " + row["CategoryID"];
}
```

10. Press **Ctrl-F5** to run the application. Notice that the button added at design-time remains along with the bound data added later. Run the mouse over the buttons to see the tool tips. You can see that both text and value data is available when the control is bound.



11.4 Binding Hierarchical Data

Several RadControls allow self-referencing data to display a hierarchical structure: RadMenu, RadContextMenu, RadTabStrip, RadPanelBar and RadTreeView are designed to show multiple levels of data. The data is defined using an "ID" column that identifies a given row and a "ParentID" that defines the parent row.

The small example shown below makes it easier to see. The "President" has the top spot in this view, has an ID of "1" and reports to no one so the ParentID is zero or null. The "Vice President" position reports to the "President" and has a ParentID of "1". Looking farther down the data we see that "CEO" and "CTO" have a ParentID of "2" and so refers to the "Vice President" and so on.

	ID	ParentID	Name
1	1	0	President
2	2	1	Vice President
3	3	2	CEO
4	4	2	CTO
5	5	4	Group PM
6	6	5	PM 1
7	7	5	PM 2

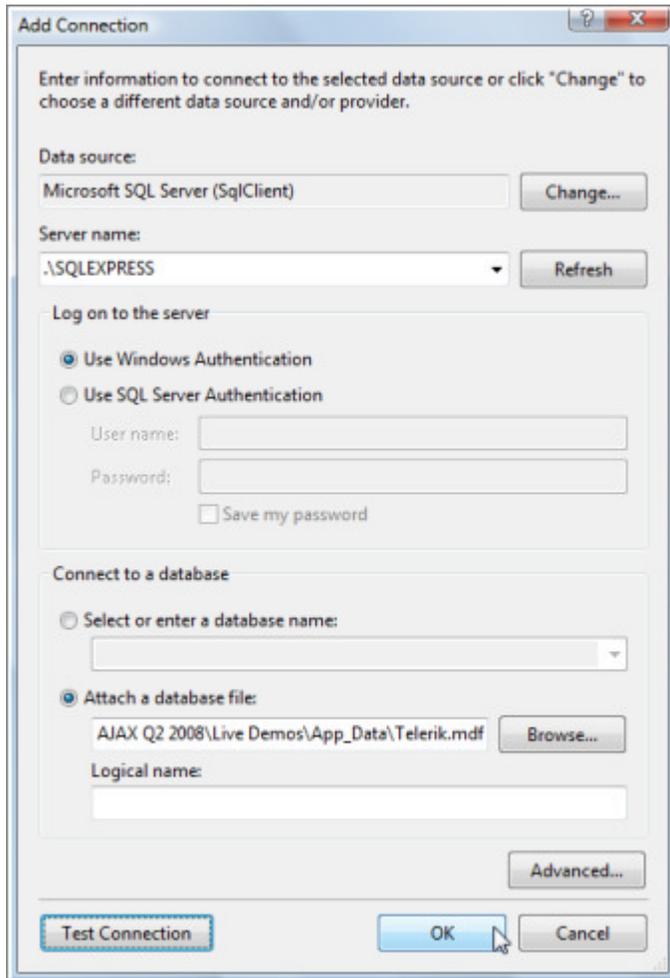
To hook this kind of data up to a RadControl you need to use the **DataFieldID** and **DataFieldParentID** properties. The **DataTextField** property is still used to display what the user actually sees in the browser. An additional property **MaxBindDepth** controls the number of levels that are included in the hierarchy. The default is "-1" and includes all levels of the hierarchy.

This walk-through hooks this data up to a RadTreeView.

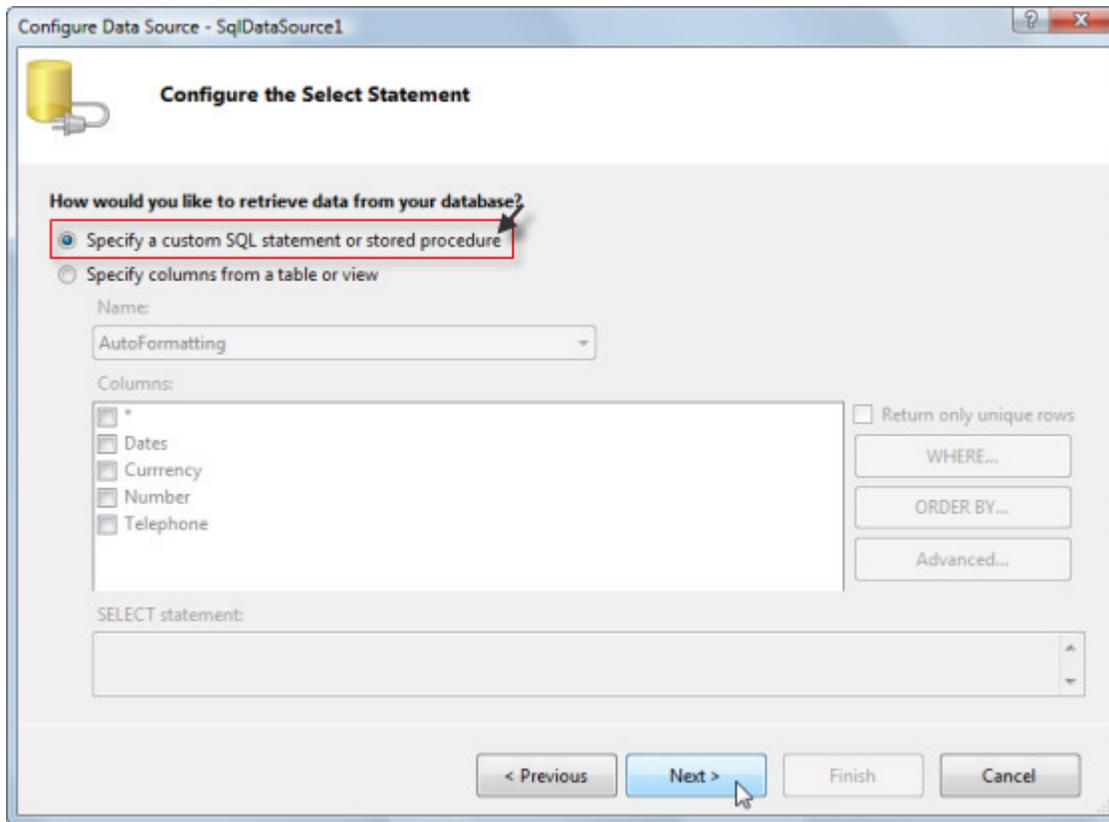


You can find the complete source for this project at:
\\VS Projects\Databinding\DataBindings

1. Create a new web project and add a ScriptManager to the default web page.
2. Add a RadTreeView to the form.
3. From the RadTreeView Smart Tag, select **Choose Data Source** | **<New Data Source...>**. From here, follow the same steps as the "Simple Declarative Binding" example up to the point where you see the Add Connection dialog. Instead of attaching to the Northwind.mdf database, attach the *Telerik.mdf* database found in the same folder.



4. Instead of selecting a table from the list, instead select **Specify a custom SQL statement or stored procedure**. *This step will display the Define Custom Statements or Stored Procedure page of the wizard.*



5. Enter the SQL below to the Select tab of the wizard page.

The general purpose of the query is to pull ID, ParentID and Name columns from the "SelfReferencing" table.

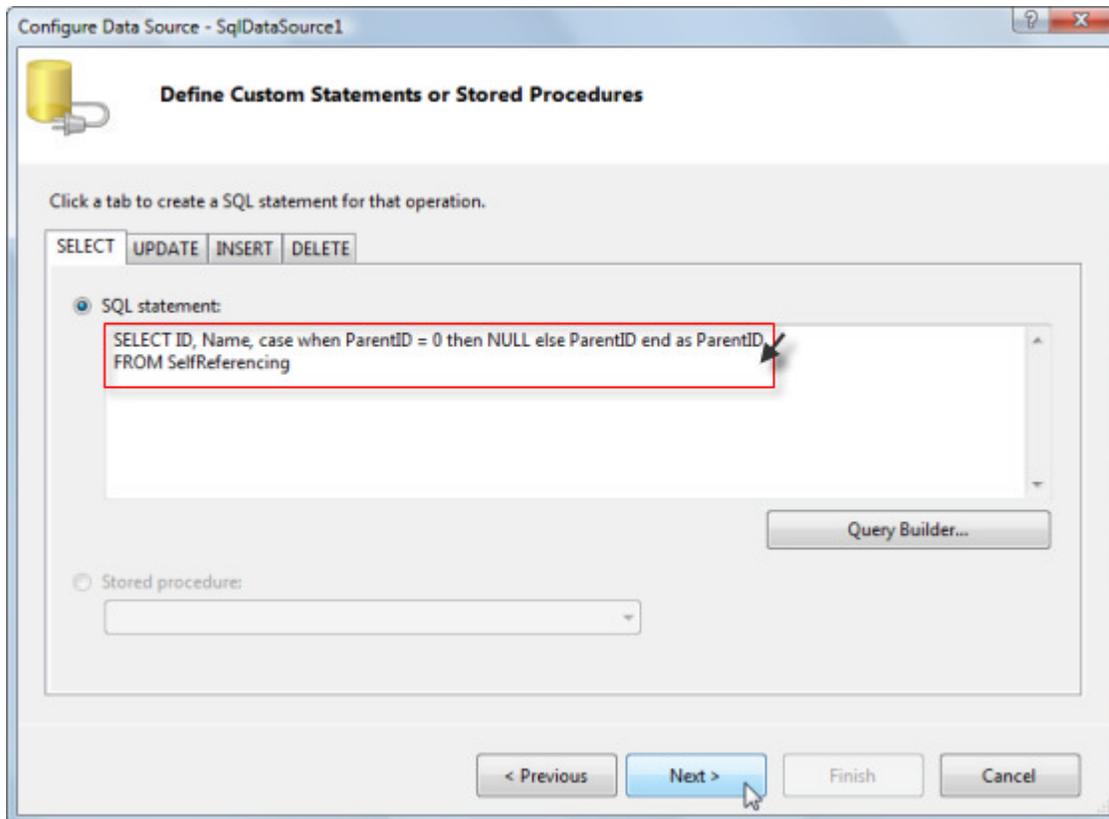
[T-SQL] Selecting from Table "SelfReferencing"

```
SELECT ID, NAME,
CASE WHEN ParentID = 0
THEN NULL ELSE ParentID
end AS ParentID
FROM SelfReferencing
```

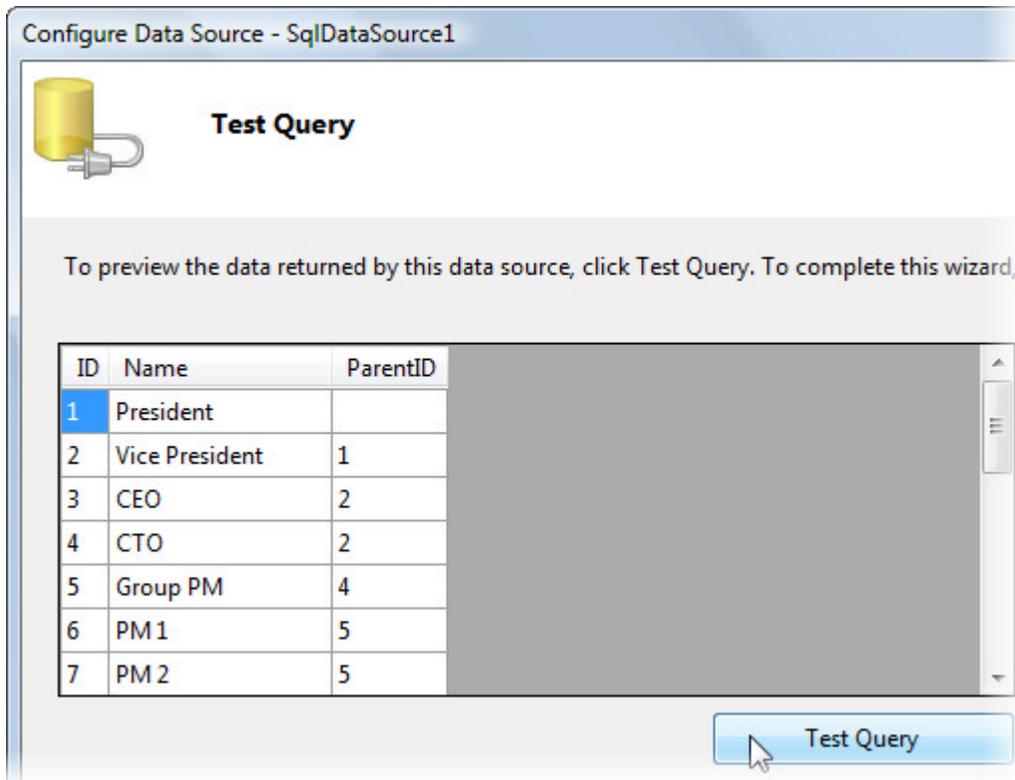


Gotcha! "This constraint cannot be enabled as not all values have corresponding parent values." This error may appear if a ParentID value doesn't point to an ID that exists in the table. The ParentID is allowed to be null though. If you look at the example data at the top of this lesson you'll see that the ParentID for "President" is "0". There is no ID of "0" in the table. So how do you define the top level value? You can either change the table data to be null, or change the select statement to look like the T-SQL above where a "case" statement checks if the value is "0" and changes it to null.

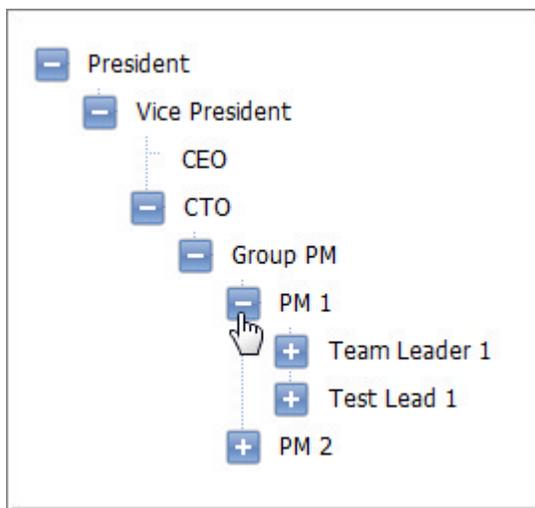
The select statement in the "Define Custom Statements or Stored Procedures" page of the wizard will look like the screenshot below.



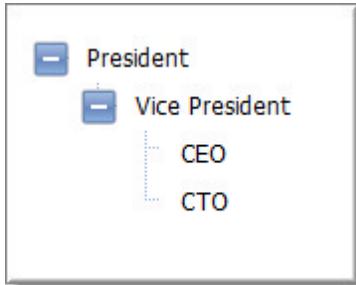
On the "Test Query" page of the wizard, click the Test Query button and observe the results. The "President" will have a ParentID that is null and the ParentIDs for the other records will refer to IDs that exist in the table.



6. In the Properties window, set the **DataFieldID** property to "ID" and the **DataFieldParentID** property to "ParentID".
7. Press **Ctrl-F5** to run the application. The tree view control displays the self-referencing data as a hierarchy.



8. Stop the application and change the **MaxBindDepth** property to "3". Re-run the application to see that only the first three levels are displayed in the tree view.

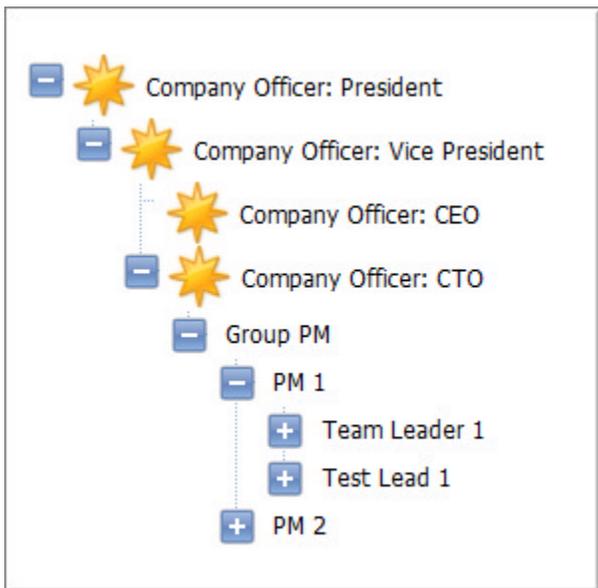


Using the DataBindings Property

With the **DataTextField**, **DataValueField** and **DataNavigateUrlField** you can map only specific properties to columns in the database. What if you want to bind to the **ImageUrl** or **ToolTip** of a control? You could use the **ItemDataBound** event handler (explained in the upcoming section on Server-Side coding) and that would provide a flexible approach to binding any property with any database column. *But*, you would lose the ability to define these relationships in ASP.NET markup.

The **DataBindings** property lets you map database columns directly to properties in any of the hierarchy-capable RadControls (RadTreeView, RadContextMenu, RadMenu, RadTabView, RadPanelBar). **DataBindings** is designed to be used directly within ASP.NET markup. Each binding has two attribute that can be defined: one is the name of a property, e.g. "ImageUrl" and the other is the property name with the suffix "Field", e.g. "ImageUrlField". The first is a hard-coded value that will be populated to the property. The second is the name of a column in the data source for the control. The binding may use a **FormatString** attribute that's used to present data from a column in a specific way. Also, each binding can use the **Depth** attribute to specify what levels of the hierarchy are bound to.

For example, let's take the previous "hierarchy" demo and add **DataBindings**. The completed tree view will show a star image next to the company officers in the diagram and also format the those same nodes with "Company Officer:".



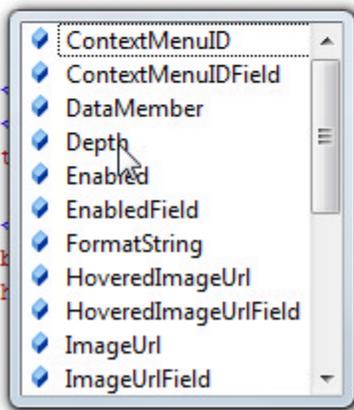
In the markup below shows the `<DataBindings>` element containing several `RadTreeNodeBinding` elements. Each element defines the `ImageUrl` that's hard-coded to an image path within the project, a `TextField` that still points to the "Name" column in the database and a `FormatString` property that formats the `TextField`. There are three bindings defined with `Depth` set to 0..2.

[ASP.NET] Defining DataBindings

```
<telerik:RadTreeView ID="RadTreeView1" runat="server" DataFieldID="ID"
DataFieldParentID="ParentID"
DataSourceID="SqlDataSource1" DataTextField="Name" Skin="Web20">
  <DataBindings>
    <telerik:RadTreeNodeBinding ImageUrl="\images\Annotation_new.png" TextField="Name"
FormatString="Company Officer: {0}" Depth="0" />
    <telerik:RadTreeNodeBinding ImageUrl="\images\Annotation_new.png" TextField="Name"
FormatString="Company Officer: {0}" Depth="1" />
    <telerik:RadTreeNodeBinding ImageUrl="\images\Annotation_new.png" TextField="Name"
FormatString="Company Officer: {0}" Depth="2" />
  </DataBindings>
  <ExpandAnimation Duration="100"></ExpandAnimation>
</telerik:RadTreeView>
```

You can find the other properties available for binding using Intellisense within the markup:

```
DataSourceID="SqlDataSource1" DataTextField="Name" Skin=
<DataBindings>
  <telerik:RadTreeNodeBinding
    ImageUrl="\images\Annotation_new.png"
    TextField="Name"
    FormatString="Company Officer: {0}"
    Depth="0" />
  <telerik:RadTreeNodeBinding
    ImageUrl="\images\Annotation_new.png"
    TextField="Name"
    FormatString="Company Officer: {0}"
    Depth="1" />
  <telerik:RadTreeNodeBinding
    ImageUrl="\images\Annotation_new.png"
    TextField="Name"
    FormatString="Company Officer: {0}"
    Depth="2" />
```



```
on="100"></ExpandAnimation>
```

11.5 Server-Side Programming

Binding in Server Code

Binding at runtime can be as simple as defining an array or generic list of strings, assigning it to the control's

DataSource and calling the DataBind() method. This gets you the basic data display in the control and not much else.



You can find the complete source for this project at:
 \VS Projects\Databinding\ServerBindingSimple

Binding Arrays and Lists

Both Array and List<> types implement IEnumerable and so can be used as fodder for binding. For example we can create a simple array of strings, assign it to the DataSource of a RadComboBox and call DataBind():

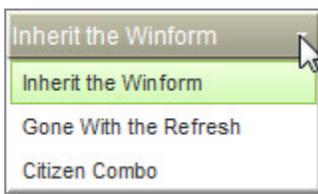
[VB] Binding to an Array of String

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        Dim movies As String() = New String() {"Inherit the Winform", "Gone With the Refresh",
"Citizen Combo"}
        RadComboBox1.DataSource = movies
        RadComboBox1.DataBind()
    End If
End Sub
```

[C#] Binding to an Array of String

```
IDataSource - system.web.ui
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        string[] movies = new string[]
        { "Inherit the Winform", "Gone With the Refresh", "Citizen Combo" };
        RadComboBox1.DataSource = movies;
        RadComboBox1.DataBind();
    }
}
```

The strings show up in the combo box but do not have associated values.



Likewise, we can bind to a generic List<> of strings:

[VB] Binding a Generic List

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        'string[] movies = new string[] { "Inherit the Winform", "Gone With the Refresh", "Citizen
Combo" };
        'RadComboBox1.DataSource = movies;
        'RadComboBox1.DataBind();
        ' Create and populate a generic list
        Dim movieMenu As New List(Of String)()
        movieMenu.Add("Show Times")
```

```
movieMenu.Add("Movies")
movieMenu.Add("Theaters")
' Bind the generic list to the RadComboBox
RadComboBox1.DataSource = movieMenu
RadComboBox1.DataBind()
End If
End Sub
```

[VB] Binding a Generic List

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        //string[] movies = new string[] { "Inherit the Winform", "Gone With the Refresh",
"Citizen Combo" };
        //RadComboBox1.DataSource = movies;
        //RadComboBox1.DataBind();
        // Create and populate a generic list
        List<string> movieMenu = new List<string>();
        movieMenu.Add("Show Times");
        movieMenu.Add("Movies");
        movieMenu.Add("Theaters");
        // Bind the generic list to the RadComboBox
        RadComboBox1.DataSource = movieMenu;
        RadComboBox1.DataBind();
    }
}
```

Again, the strings show up but the implementation is a bit bare because we have no associated Values. Also notice that we never assigned the `DataTextField` property. So we can see that if there's only a single column of data to work with, the control assumes you want to bind that column.



Binding Lists of Objects

Let's say we have an object with two properties, one string "Name" and a integer "ID". We can bind to a list or an array of these objects. Here's the object definition:

[VB] The MovieMenuItem Class

```
Public Class MovieMenuItem
    Public Sub New(ByVal name As String, ByVal id As Integer)
        _name = name
        _id = id
    End Sub
    Private _name As String
    Public Property Name() As String
        Get
            Return _name
        End Get
        Set
```

```

    _name = value
End Set
End Property
Private _id As Integer
Public Property ID() As Integer
Get
    Return _id
End Get
Set
    _id = value
End Set
End Property
End Class

```

[C#] The MovieMenuItem Class

```

public class MenuItem
{
    public MenuItem(string name, int id)
    {
        _name = name;
        _id = id;
    }
    private string _name;
    public string Name
    {
        get { return _name; }
        set { _name = value; }
    }
    private int _id;
    public int ID
    {
        get { return _id; }
        set { _id = value; }
    }
}

```

Now we can bind this the same way to the control's DataSource:

[VB] Binding to the list of MenuItem Objects

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        ' create list of MenuItem objects
        Dim movieItems As New List(Of MenuItem)()
        movieItems.Add(New MenuItem("Show Times", 1))
        movieItems.Add(New MenuItem("Movies", 2))
        movieItems.Add(New MenuItem("Theaters", 3))
        ' bind movie items List to combo box
        RadComboBox2.DataSource = movieItems
        RadComboBox2.DataBind()
    End If
End Sub

```

[C#] Binding to the list of MenuItem Objects

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)

```

```
{
    // create list of MenuItem objects
    List<MenuItem> movieItems = new List<MenuItem>();
    movieItems.Add(new MenuItem("Show Times", 1));
    movieItems.Add(new MenuItem("Movies", 2));
    movieItems.Add(new MenuItem("Theaters", 3));
    // bind movie items List to combo box
    RadComboBox2.DataSource = movieItems;
    RadComboBox2.DataBind();
}
}
```

But because there is more than a single property in the object, the results of binding show the object name in the combo box, not the property contents.



Again, if you assign the **DataTextField** the combo box is properly populated. You can change the example above to assign both **DataTextField** and **DataValueField**. Then set the **AutoPostBack** property to **True** and finally, create a **SelectedIndexChanged** event handler to display the text and value in the page title.

[VB] Creating, Binding and Retrieving Bound Data

```
Dim movieItems As New List(Of MenuItem)()
movieItems.Add(New MenuItem("Show Times", 1))
movieItems.Add(New MenuItem("Movies", 2))
movieItems.Add(New MenuItem("Theaters", 3))
' bind text and value to specific property names
RadComboBox2.DataTextField = "Name"
RadComboBox2.DataValueField = "ID"
RadComboBox2.AutoPostBack = True
RadComboBox2.DataSource = movieItems
RadComboBox2.DataBind()
```

'...

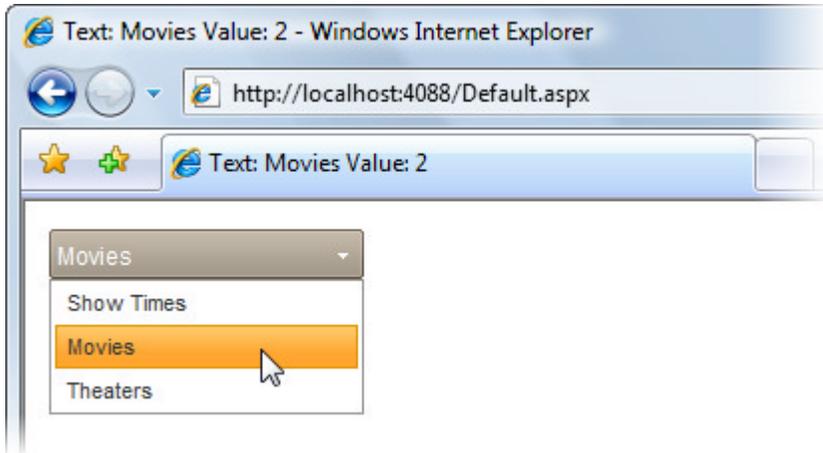
```
Protected Sub RadComboBox2_SelectedIndexChanged(ByVal o As Object, ByVal e As
RadComboBoxSelectedIndexChangedEventArgs)
    Me.Title = "Text: " + e.Text + " Value: " + e.Value
End Sub
```

[C#] Creating, Binding and Retrieving Bound Data

```
List<MenuItem> movieItems = new List<MenuItem>();
movieItems.Add(new MenuItem("Show Times", 1));
movieItems.Add(new MenuItem("Movies", 2));
movieItems.Add(new MenuItem("Theaters", 3));
// bind text and value to specific property names
RadComboBox2.DataTextField = "Name";
RadComboBox2.DataValueField = "ID";
RadComboBox2.AutoPostBack = true;
RadComboBox2.DataSource = movieItems;
RadComboBox2.DataBind();
//...
protected void RadComboBox2_SelectedIndexChanged(object o,
```

```
RadComboBoxSelectedIndexChangedEventArgs e)
{
    this.Title = "Text: " + e.Text + " Value: " + e.Value;
}
}
```

Now the data again displays correctly in the combo box and the text and values can be retrieved:



Server Events

Look for three basic events in most data bound RadControls.

- The **OnDataBinding** event fires first when the controls data binding expressions are about to be evaluated.
- **OnItemDataBound** fires when each item in a collection is bound to data. This event is control specific and has a different naming convention for some controls, i.e. RadTabStrip uses OnTabDataBound, RadTreeView uses OnNodeDataBound and tool bar has OnButtonDataBound. The general pattern for all of these OnFooDataBound events is that a custom set of event arguments are passed in to provide a reference.

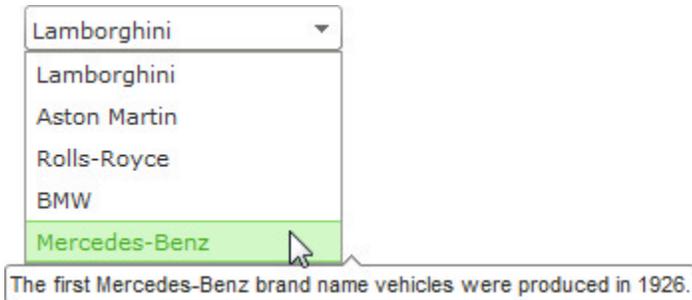
In the arguments is a property for the item being bound and inside the item object are the properties for the item (Text, Value, ToolTip, etc.) and a DataItem reference that lets you get at the underlying object being bound. When you bind to a SqlDataSource, the DataItem will be a DataRowView. If you bind a list of some custom object, then the DataItem will be that custom object type. This event gives you maximum flexibility to use the data against the item in whatever way you choose. ItemDataBound is used to populate properties that don't have directly binding support, e.g. ToolTip, Visible, Enabled, etc.

- The **OnDataBound** event fires when the control is finished binding its data. Use this event to guarantee that all the control's data is present.

Here's a short example that populates the tool tip for each item in a RadComboBox. The combo is bound to a list of "Car" objects: a simple object that has three properties, Name, ID and Comment.



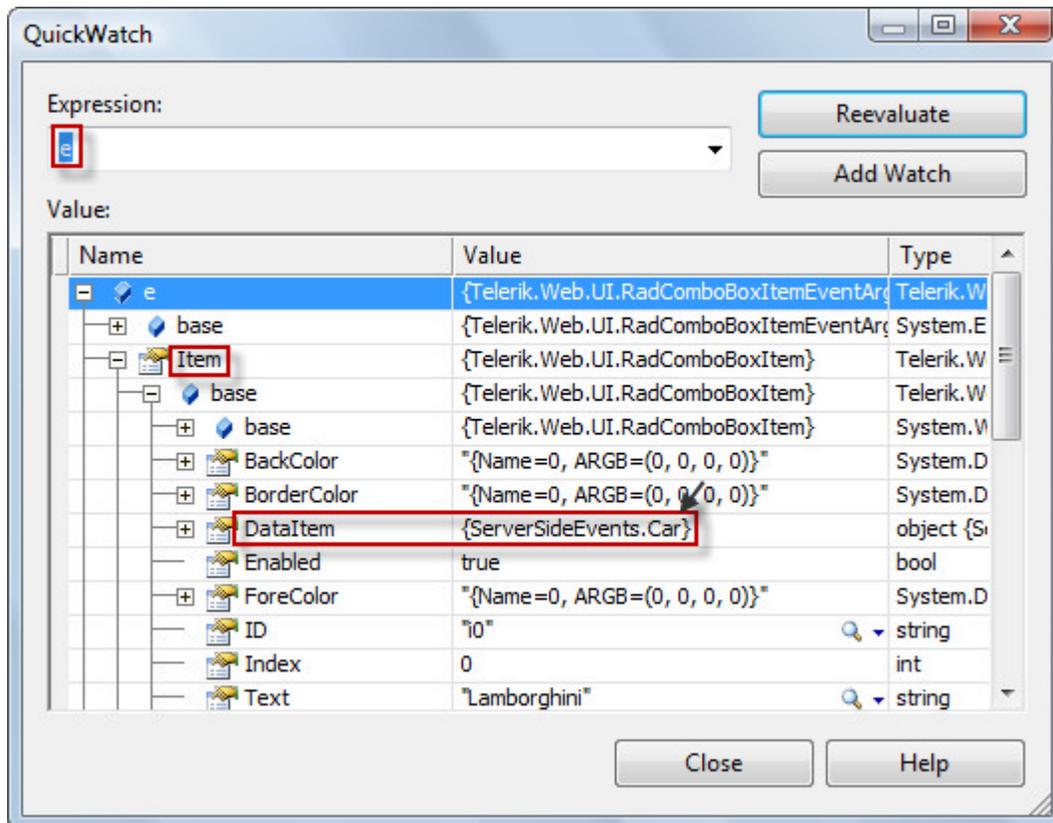
You can find the complete source for this project at:
 \VS Projects\DataBinding\ServerSideEvents



Inside the `ItemDataBound` we get the `RadComboBoxItemEventArgs` passed to the event handler. Remember that this parameter will be slightly different for each `RadControl` but the pattern will be quite similar. Inside `RadComboBoxItemEventArgs` you can find the `Item` property (or `Tab` or `Node` or `Button`). And within `Item` is `DataItem` that represents the chunk of data we just bound to. In this case the type is "Car" so `DataItem` is cast to be a `Car` type so that the `Comment` property is accessible.



You can also put a breakpoint on the first line of this handler and see the underlying type for `DataItem`. In the case of standard database data sources, its `DataRowView`.



In the `ItemDataBound` event the `DataItem` is retrieved and cast to type "Car", the `Comment` property is accessed and assigned to the `Item`'s `ToolTip` property.

[VB] Handling the `ItemDataBound` Event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Create a list of car objects
    Dim cars As New List(Of Car)()
    cars.Add(New Car("Lamborghini", 1, "Based in the small Italian village of SantAgata
    Bolognese, near Bologna"))
    cars.Add(New Car("Aston Martin", 1, "Headquarters are at Gaydon, England"))
```

```

cars.Add(New Car("Rolls-Royce", 1, "Founded in 1906 by Henry Royce and C.S. Rolls"))
cars.Add(New Car("BMW", 1, "Bayerische Motoren Werke"))
cars.Add(New Car("Mercedes-Benz", 1, "The first Mercedes-Benz brand name vehicles were
produced in 1926."))
' assign and bind
RadComboBox1.DataSource = cars
RadComboBox1.DataBind()
End Sub
Protected Sub RadComboBox1_ItemDataBound(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadComboBoxItemEventArgs)
' get the DataItem for the selected item and cast back
' to its original type to access its Comment property.
' Assign the comment to the item's ToolTip
e.Item.ToolTip = (TryCast(e.Item.DataItem, Car)).Comment
End Sub

```

[C#] Handling the ItemDataBound Event

```

protected void Page_Load(object sender, EventArgs e)
{
    // Create a list of car objects
    List<Car> cars = new List<Car>();
    cars.Add(new Car("Lamborghini", 1,
        "Based in the small Italian village of SantAgata Bolognese, near Bologna"));
    cars.Add(new Car("Aston Martin", 1,
        "Headquarters are at Gaydon, England"));
    cars.Add(new Car("Rolls-Royce", 1,
        "Founded in 1906 by Henry Royce and C.S. Rolls"));
    cars.Add(new Car("BMW", 1,
        "Bayerische Motoren Werke"));
    cars.Add(new Car("Mercedes-Benz", 1,
        "The first Mercedes-Benz brand name vehicles were produced in 1926."));
    // assign and bind
    RadComboBox1.DataSource = cars;
    RadComboBox1.DataBind();
}
protected void RadComboBox1_ItemDataBound(object sender,
Telerik.Web.UI.RadComboBoxItemEventArgs e)
{
    // get the DataItem for the selected item and cast back
    // to its original type to access its Comment property.
    // Assign the comment to the item's ToolTip
    e.Item.ToolTip = (e.Item.DataItem as Car).Comment;
}

```

11.6 Binding to Business Objects

The ASP.NET **ObjectDataSource** component lets you have declarative access to objects, rather than having to populate the object directly in code. **ObjectDataSource** is commonly used in multi-tier scenarios when you want to interact with objects that implement business logic.

ObjectDataSource can handle Select, Update, Delete, Insert operations. The object that **ObjectDataSource** consumes needs methods that match what these operations are looking for. The Select operation for example is looking for a method that returns an **IEnumerable**. If the class being consumed does not implement **IEnumerable**, **ObjectDataSource** wraps the result of the Select method as an **IEnumerable**. The expected parameters are explained briefly in the Configure Data Source... dialog available from the **ObjectDataSource** Smart Tag.

UI for ASP.NET AJAX

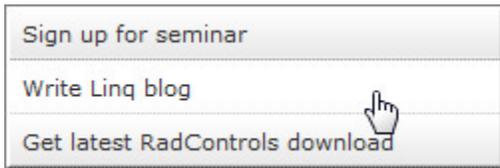
 Much of the information here concerning CRUD operations and defining parameters is true for all the Data Source controls.



You can find the complete source for this project at:

\\VS Projects\DataBinding\ObjectDataSource

The following shows a stub implementation of a task list displayed in RadPanelBar. When an item is clicked, it is deleted from the list.



We will create a Tasks object that contains a generic list of Tasks, can retrieve a list of tasks and can delete a task. The list of "Items" are stored in the Session in the "Items" accessor, if Session "Items" is null then a few sample items are created and fed into the list. **Note:** This use of the Session is merely a way to persist the data while keeping the size of the example down and avoiding data access issues not central to this topic.

The Task object itself is relatively trivial and has two properties, one for Name and one for ID.

[VB] Implementing Tasks and Task Objects

Public Class Tasks

' Stores a list of Tasks in the Session.

' This simulates a kind of persistent storage

Public ReadOnly Property Items() As List(Of Task)

Get

If HttpContext.Current.Session("Items") = Nothing Then

Dim result As New List(Of Task)()

result.Add(New Task("Sign up for seminar", 1))

result.Add(New Task("Write Linq blog", 2))

result.Add(New Task("Get latest RadControls download", 3))

HttpContext.Current.Session("Items") = result

End If

Return DirectCast(HttpContext.Current.Session("Items"), List(Of Task))

End Get

End Property

' method used by the object data source Select

Public Function GetMyData() As List(Of Task)

Return Me.Items

End Function

' method used by the object data source Delete

Public Sub DeleteMyData(ByVal id As Integer)

Dim foundTask As Task = Me.Items.Find()

Me.Items.Remove(foundTask)

End Sub

Public Sub stuff(ByVal test As String)

End Sub

End Class

#region Task object

Public Class Task

Public Sub New()

End Sub

Public Sub New(ByVal name As String, ByVal id As Integer)

_id = id

```

    _name = name
End Sub
Private _id As Integer
Public Property ID() As Integer
    Get
        Return _id
    End Get
    Set
        _id = value
    End Set
End Property
Private _name As String
Public Property Name() As String
    Get
        Return _name
    End Get
    Set
        _name = value
    End Set
End Property
End Class
#End Region

```

[C#] Implementing Tasks and Task Objects

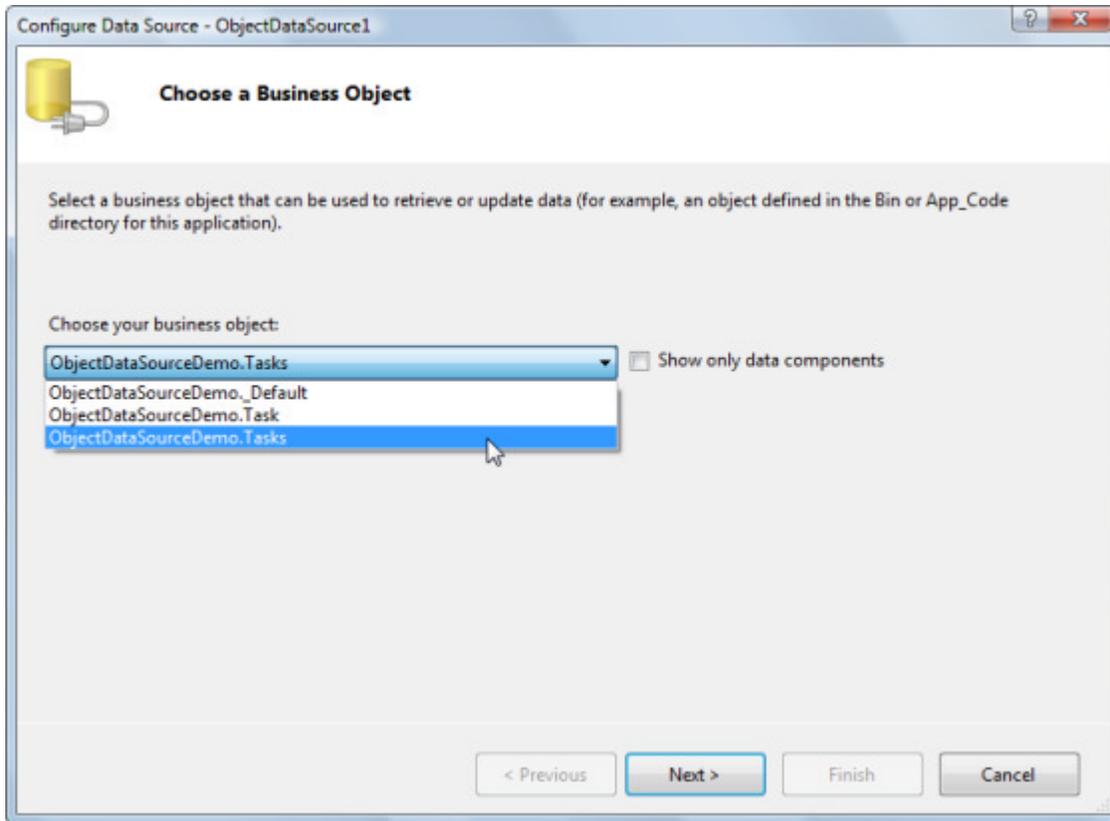
```

public class Tasks
{
    // Stores a list of Tasks in the Session.
    // This simulates a kind of persistent storage
    public List<Task> Items
    {
        get
        {
            if (HttpContext.Current.Session["Items"] == null)
            {
                List<Task> result = new List<Task>();
                result.Add(new Task("Sign up for seminar", 1));
                result.Add(new Task("Write Linq blog", 2));
                result.Add(new Task("Get latest RadControls download", 3));
                HttpContext.Current.Session["Items"] = result;
            }
            return (List<Task>)HttpContext.Current.Session["Items"];
        }
    }
    // method used by the object data source Select
    public List<Task> GetMyData()
    {
        return this.Items;
    }
    // method used by the object data source Delete
    public void DeleteMyData(int id)
    {
        Task foundTask = this.Items.Find(delegate (Task task)
        { return task.ID == id; });
        this.Items.Remove(foundTask);
    }
}

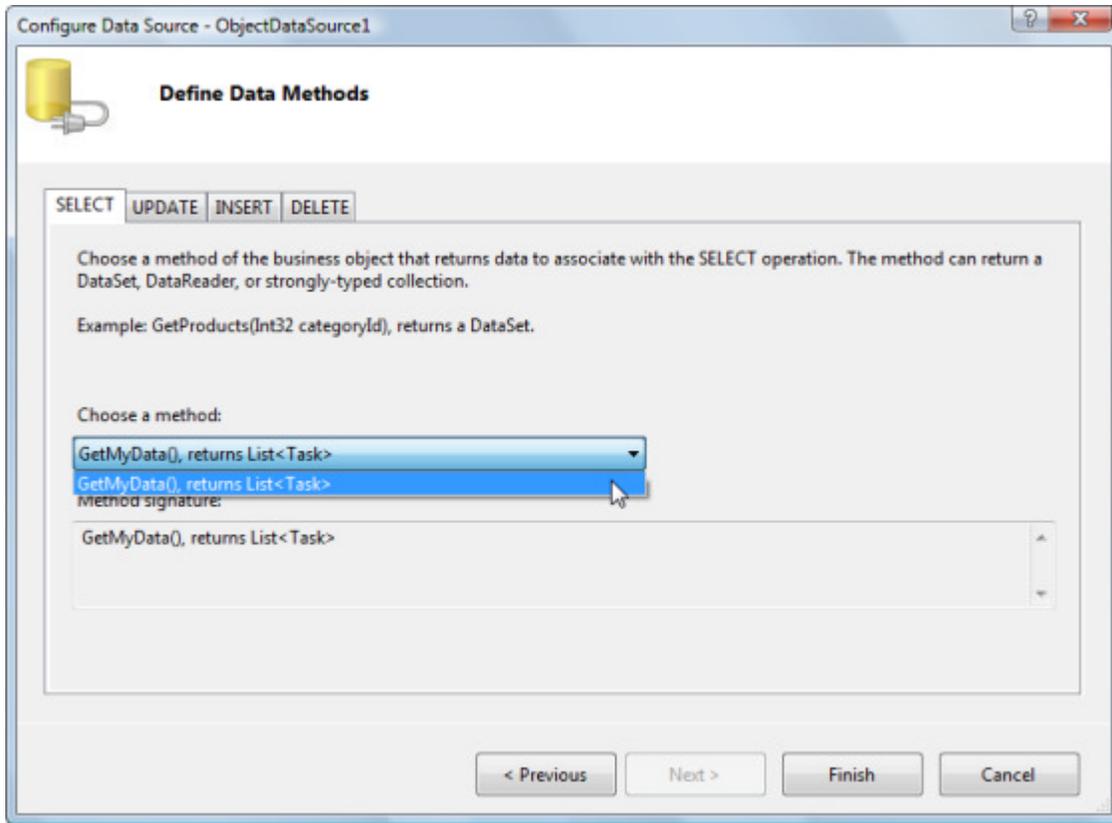
```

```
public void stuff(string test)
{
}
}
#region Task object
public class Task
{
public Task()
{
}
public Task(string name, int id)
{
    _id = id;
    _name = name;
}
private int _id;
public int ID
{
    get { return _id; }
    set { _id = value; }
}
private string _name;
public string Name
{
    get { return _name; }
    set { _name = value; }
}
}
#endregion
```

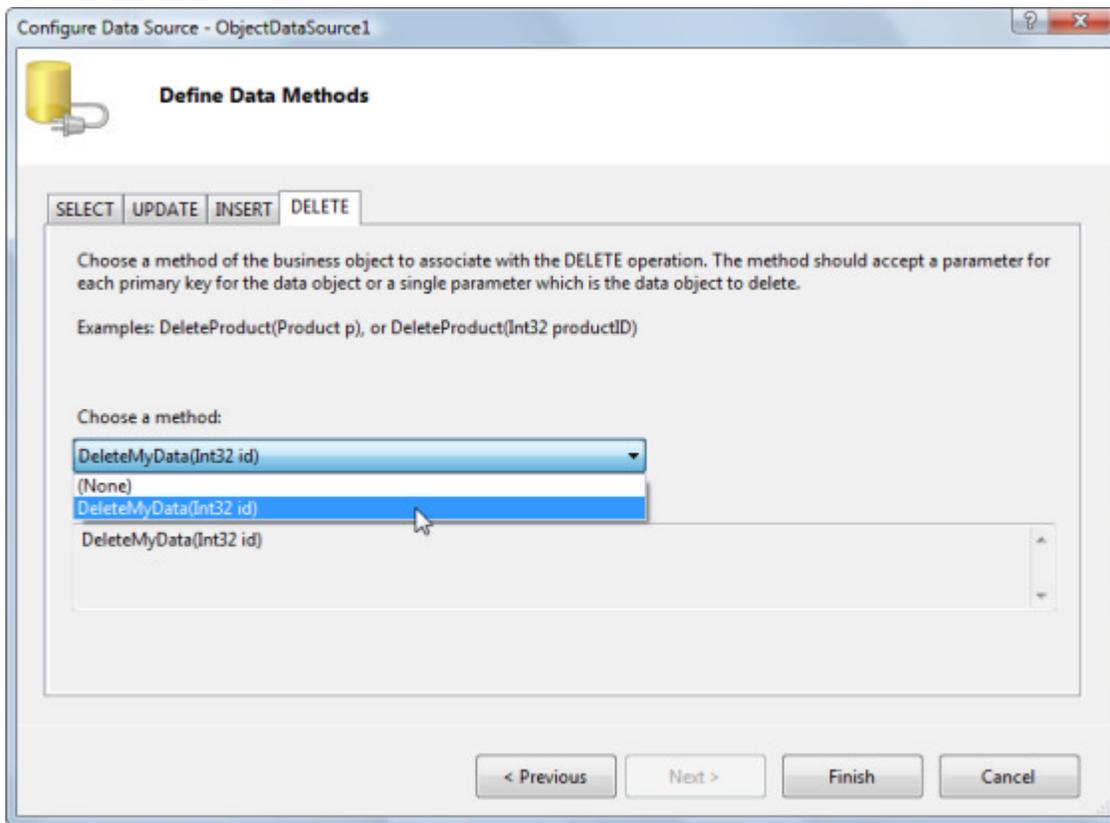
Using ObjectDataSource is much simpler than setting up the object itself. Most of the work can take place in the designer. You drop an ObjectDataSource from the Data tab of the ToolBox to the web page and click the **Configure Data Source...** option from the Smart Tag. In the "Choose a Business Object" page of the Configure Data Source wizard you get a list of available objects. You can see in the list "_default" which is the page class, "Task" is the single element of the Tasks list and "Tasks" which contains our generic list of Task objects.



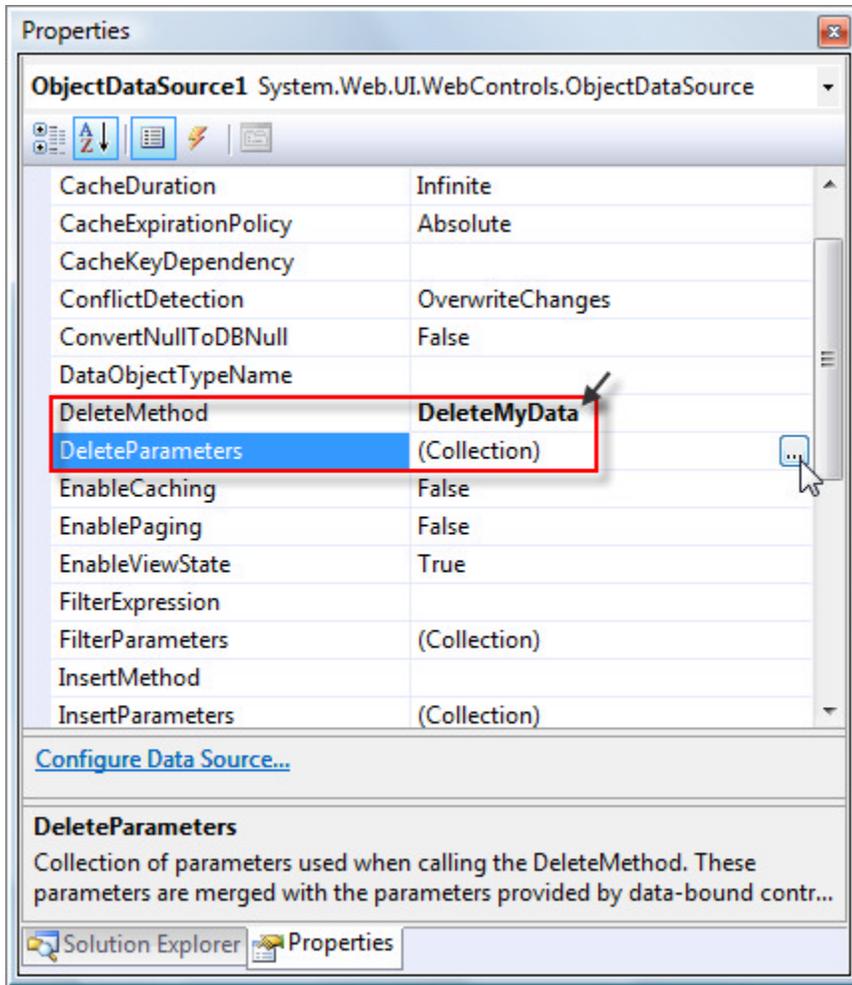
After selecting the business object to consume and hitting the Next button, the "Define Data Methods" page displays. There are tabs to help define each of the supported CRUD operations, Select Update, Insert and Delete. The selections here populate ObjectDataSource properties SelectMethod, UpdateMethod, InsertMethod and DeleteMethod that contain just the name of the methods without the parenthesis. In this example we just want to select and delete. The Select tab has a list of methods supported by the business object. In this case, only "GetMyData()" returns any values, and so only that method shows up in the list. If you had another method "GetMyString()" that returned a string for example, it too would show up in the list.



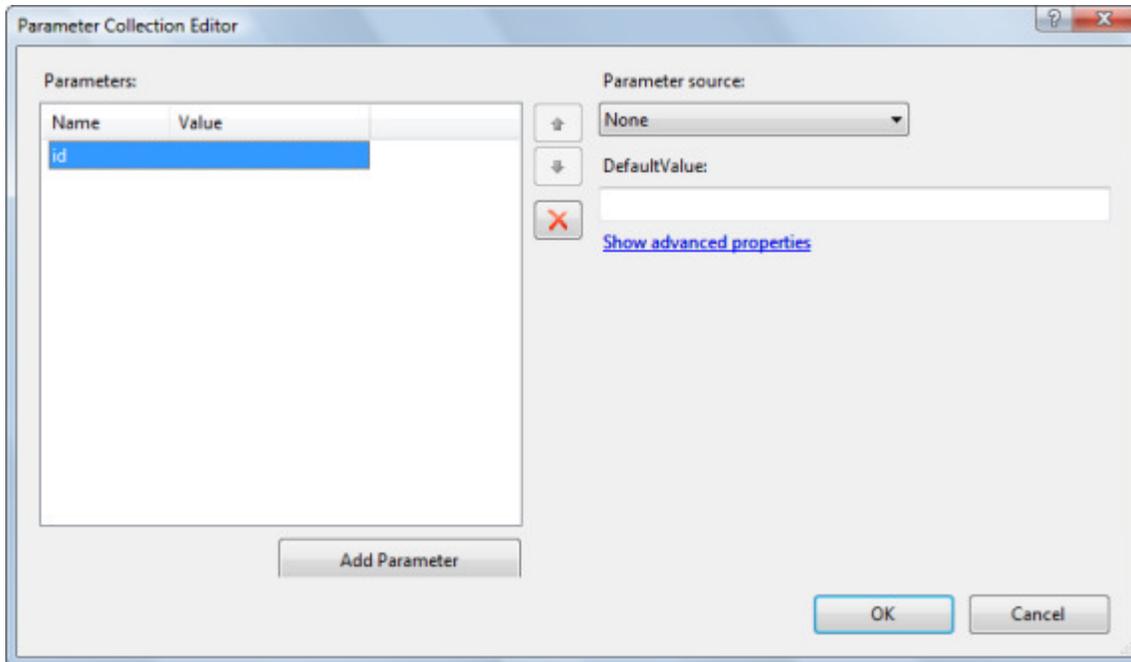
On the Delete tab of the "Define Data Methods" page, you can choose a method that accepts a parameter. Notice that `GetMyData()` doesn't show up in this list because it does not take a parameter.



After clicking the Finish button to complete the ObjectDataSource definition the next question is, "how does the delete operation know what record to delete?". Each of the method properties is coupled with a parameters collection property.



Clicking the **DeleteParameters** ellipses displays the Parameter Collection Editor where you define parameters used by the method. In some cases you can define that the "Parameter source" as a control with a particular property, but the collection editor does not always make this visible so we can leave the parameter source set to "None" and supply the parameter value in code.



Some RadControls, like RadGrid, can automatically trigger CRUD operations from a data source. You can also trigger data source methods directly in code. In this example we use the **ItemClick** event of a RadPanelBar. The DeleteParameters "id" member is supplied the clicked on value, then the ObjectDataSource Delete() method gets called.

The Select method is called implicitly by virtue of being bound to the control.

[VB] Handling the ItemClick Event

```
Protected Sub RadPanelBar1_ItemClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadPanelBarEventArgs)
    ' fill the delete parameter with the value for the clicked item
    ObjectDataSource1.DeleteParameters("id").DefaultValue = e.Item.Value
    ' delete the item in the data store and rebind
    ObjectDataSource1.Delete()
End Sub
```

[C#] Handling the ItemClick Event

```
protected void RadPanelBar1_ItemClick(object sender, Telerik.Web.UI.RadPanelBarEventArgs e)
{
    // fill the delete parameter with the value for the clicked item
    ObjectDataSource1.DeleteParameters["id"].DefaultValue = e.Item.Value;
    // delete the item in the data store and rebind
    ObjectDataSource1.Delete();
}
```

11.7 Binding to Linq

LINQ (Language Integrated Query) is a powerful new extension to the .NET framework that integrates query expressions as a part of your primary programming language (VB.NET or C# for example). LINQ provides huge improvements to data access performance, code elegance and brevity. Our use for data binding will be limited to the "LINQ to SQL" feature that lets us use query functionality (selects, aggregates, "where" clauses, etc) against ADO.NET data sources without leaving the comfort of our chosen .NET language.

For an introduction to LINQ see the MSDN article [LINQ: .NET Language-Integrated Query](#)

(<http://msdn.microsoft.com/en-us/library/bb308959.aspx>).

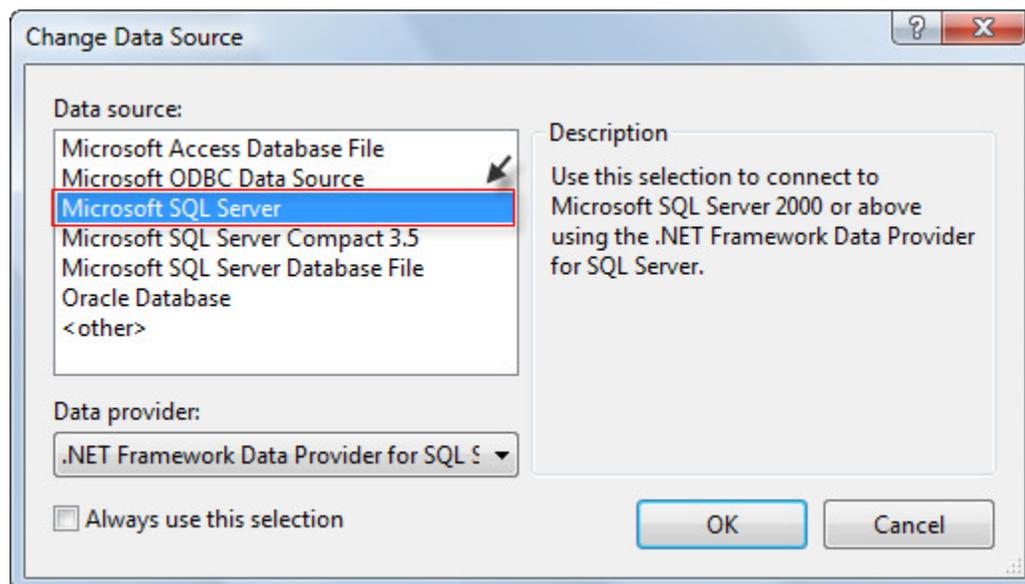
Using LinqDataSource

Our first walk-through will take place in the designer only. All the heavy lifting will be performed by Visual Studio and the RadControls.

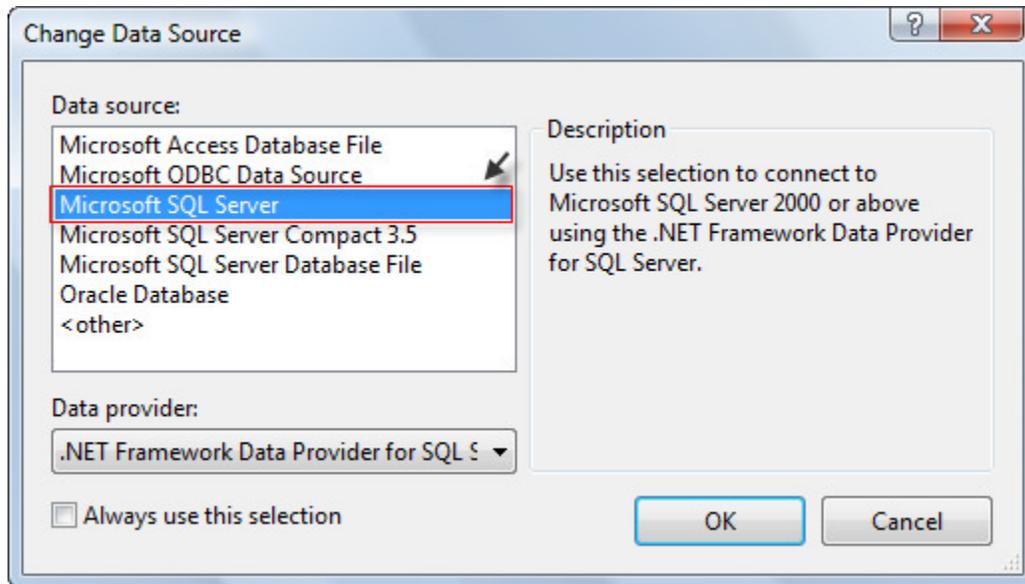


You can find the complete source for this project at:
\\VS Projects\Databinding\LinqDemo

1. In Visual Studio, create a new web application and add a **ScriptManager** to the default web page.
2. Open the Server Explorer (**View | Server Explorer**)
3. Create a connection to the Northwind database by right-clicking the **Data Connections** node of the tree and select **Add Connection...** from the context menu.
 - In the Add Connection dialog, click the **Change...** button. *This step will display the Change Data Source dialog.*



- In the Change Data Source dialog, select the "Microsoft SQL Server" data source type and click the **OK** button. *This step will return you to the Add Connection dialog.*



- In the Add Connection dialog:
- Enter the **Server name** as ".\SQLEXPRESS".
- In the **Connect to a database** area of the dialog, click the **Attach to a Database File** option. Click the **Browse** button. Navigate to the directory where you installed RadControls for ASP.NET AJAX. Select the database file you want to use, e.g. "Northwind.mdb" and click the **Open** button to select the path.
- Click the **Test Connection** button to display a success alert if the settings are correctly entered.
- Click the **OK** button to close the Add Connection dialog.

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient) Change...

Server name: .\SQLEXPRESS Refresh

Log on to the server

Use Windows Authentication

Use SQL Server Authentication

User name:

Password:

Save my password

Connect to a database

Select or enter a database name:

Attach a database file:

C:\Program Files\Telerik\RadControls for ASPNET Browse...

Logical name:

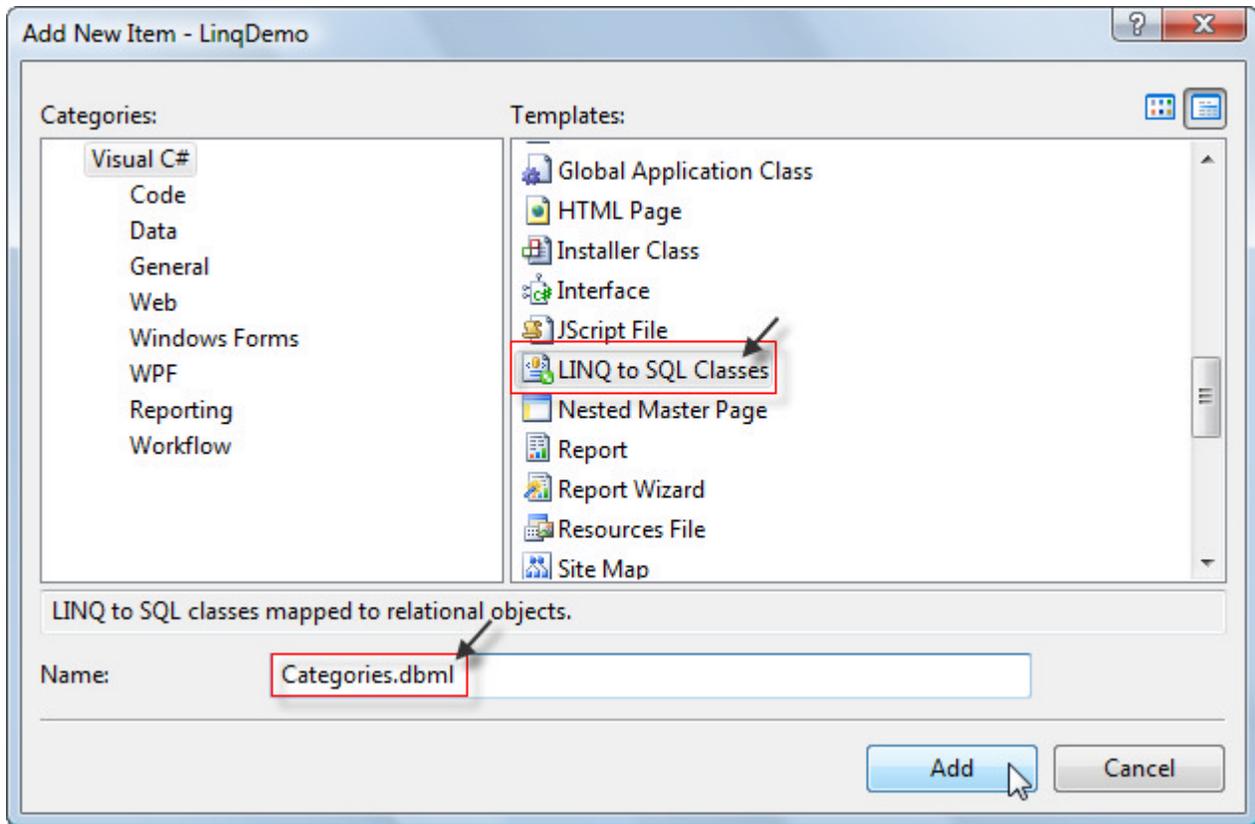
Advanced...

Test Connection OK Cancel

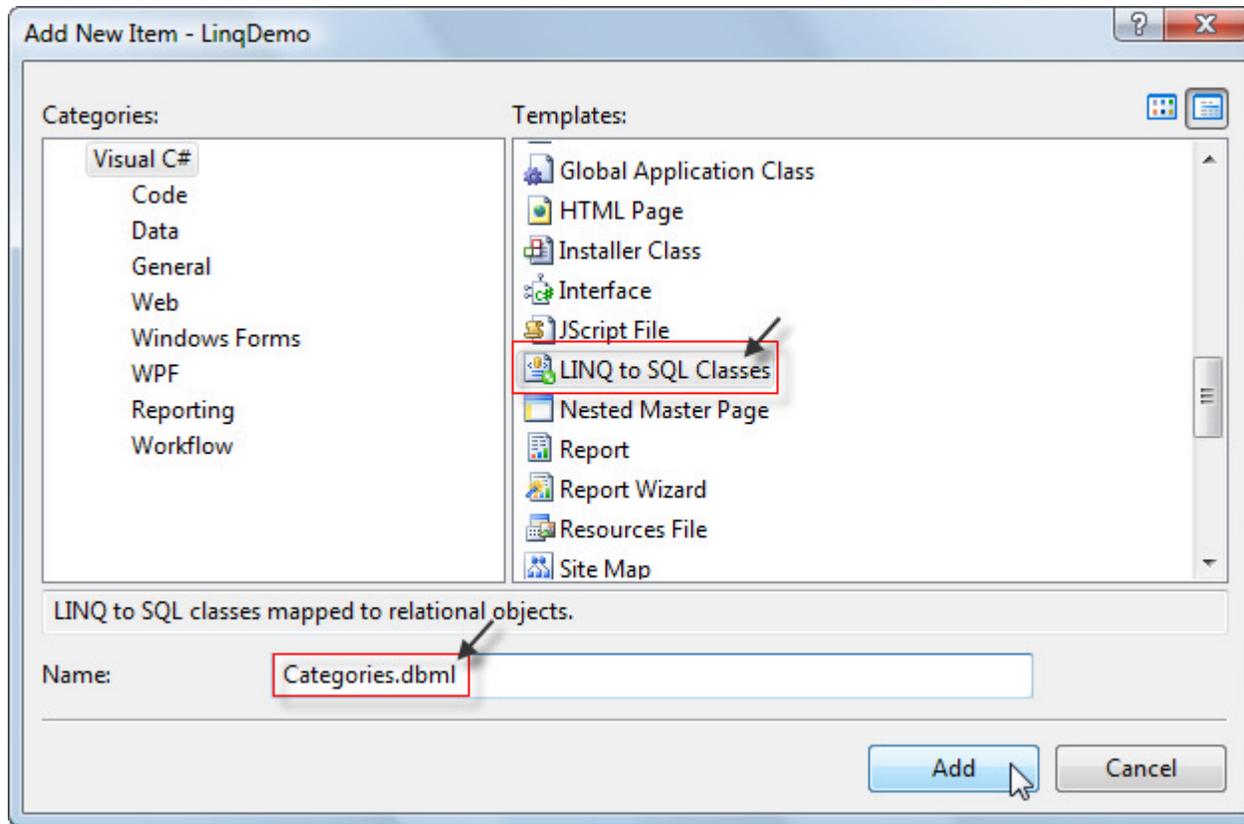


Gotcha! If you attempt to connect using the Server Explorer and get an error "Cannot open default database. Login failed": click the Advanced button on the Add Connection dialog. Set the User Instance property to True. This will redirect the connection to an instance of SQL Server running under the users account.

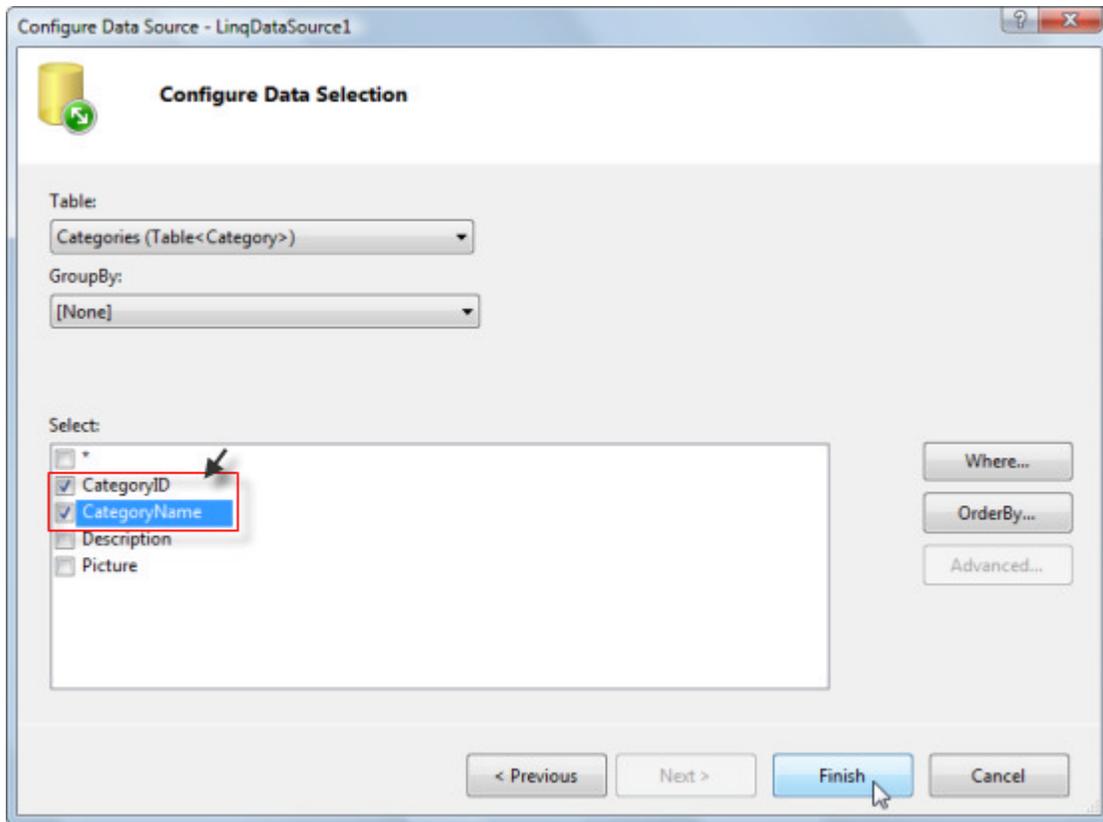
4. In the Solution Explorer, right-click the project and select **Add | New Item...**
5. Select the **LINQ to SQL Classes** template, name it Categories.dbml and click the **Add** button to close the dialog. *This step has created a DataContext object, a type that can be consumed by a LinqDataSource.*



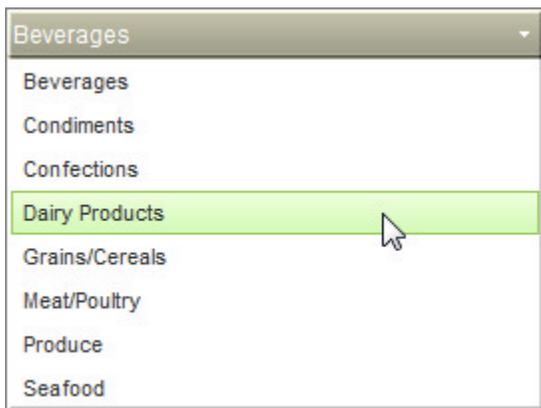
6. Drop a **LinqDataSource** component from the ToolBox Data tab to the default web page. Configure the data source:
 - Using the LinqDataSource Smart Tag, click the **Configure Data Source...** option.
 - In the "Choose a Context Object" page of the wizard, leave the default "CategoriesDataContext" and click the Next button.



- In the "Configure Data Selection" page of the wizard, select the "CategoryID and "CategoryName" columns and click the **Finish** button to close the wizard.



7. Drop a RadComboBox on the page. From the Properties window, set the DataSourceID to point to the LinqDataSource, **DataTextField** property to "CategoryName" and **DataValueField** to "CategoryID".
8. Press **Ctrl-F5** to run the application.



- ☑ The performance and ease of use here is very similar to other data source controls. As we will see in the chapter on RadGrid, the performance differences will really show in high-volume data traffic situations in the 1+ million records neighborhood.

Using LINQ Expression Results

LINQ expressions conveniently output `IEnumerable` objects that can be assigned as data sources. In this example below an array of string listing LINQ features is used as raw material for a LINQ expression. The expression selects members of the array that start with "LINQ". The results of the expression are assigned to a RadTabStrip DataSource and bound. The advantage here is that the LINQ expression provides tremendous

flexibility expressed succinctly.

LINQ to SQL

LINQ to XML

[VB] Assigning LINQ Expression Results

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' the raw data
    Dim LinqFeatures As String() = {"Lambda Expressions", "Expression Trees", "Predicates",
    "Projections", "Key extraction", "LINQ to SQL", _
    "LINQ to XML"}
    ' LINQ expression processes data and outputs IEnumerable
    Dim query As IEnumerable(Of String) = From s in LinqFeatures_
        Where s.StartsWith("LINQ")
        Order By s _
        Select s
    ' IEnumerable is assigned and bound
    RadTabStrip1.DataSource = query
    RadTabStrip1.DataBind()
End Sub
```

[C#] Assigning LINQ Expression Results

```
protected void Page_Load(object sender, EventArgs e)
{
    // the raw data
    string[] LinqFeatures =
    {
        "Lambda Expressions",
        "Expression Trees",
        "Predicates",
        "Projections",
        "Key extraction",
        "LINQ to SQL",
        "LINQ to XML"
    };
    // LINQ expression processes data and outputs IEnumerable
    IEnumerable<string> query = from s in LinqFeatures
        where s.StartsWith("LINQ")
        orderby s
        select s;
    // IEnumerable is assigned and bound
    RadTabStrip1.DataSource = query;
    RadTabStrip1.DataBind();
}
```

11.8 Summary

In this chapter we introduced the interfaces that RadControls can bind to and the task specific Data Source controls that can be used to bind declaratively. You built a simple declarative data binding example using RadToolBar with SqlDataSource. You learned in more detail how the data binding properties were used. You also learned how to bind to multiple data sources at one time.

In server-side code you learned how to bind simple arrays and lists, hierarchical data, business objects and LINQ data. You also learned how to handle data binding related server events.

12 Templates

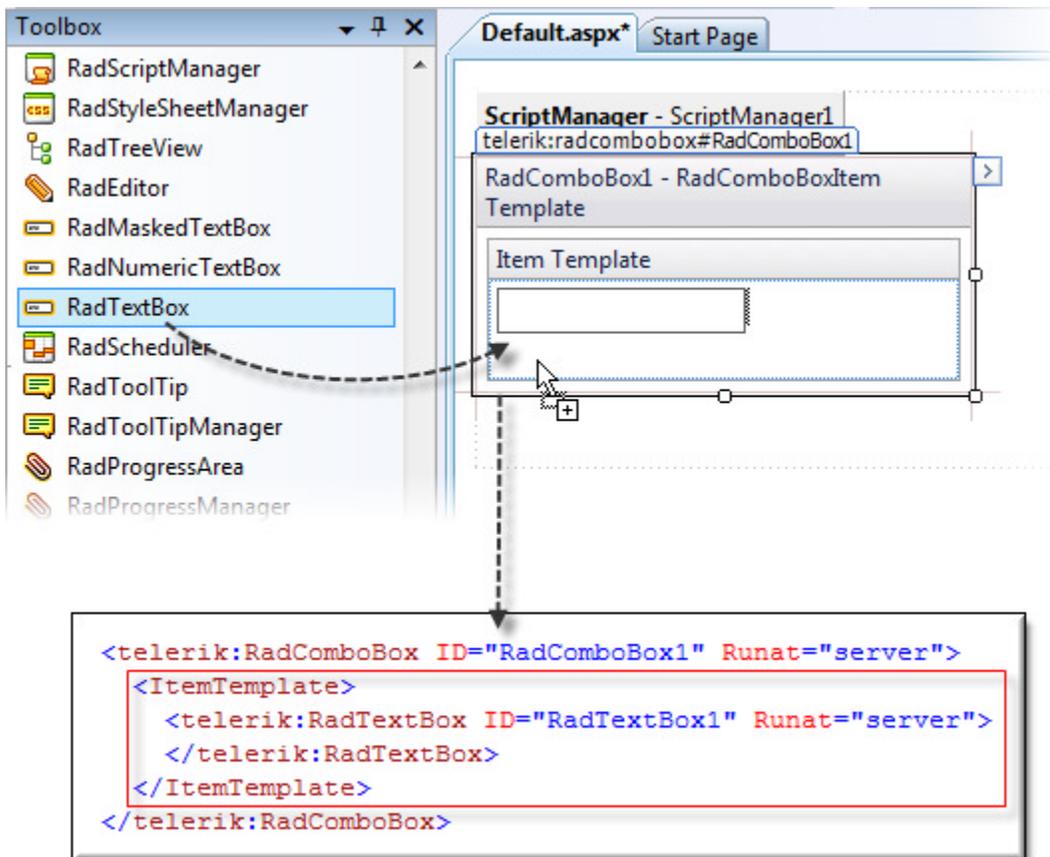
12.1 Objectives

- Learn the purpose and uses for templates in RadControls.
- Build a simple data-bound application using templates and data bound to elements within the template.
- Explore the details of binding expressions, starting with public server methods output directly to the browser and working through Container, DataItem, Eval() and Bind() methods.
- Learn how templates are presented within the design environment: multiple templates for complex controls like RadGrid, items templates for controls with multiple records (like the navigation controls) and single view templates for controls like RadRotator and RadPageView.
- Learn how to create custom templates on-the-fly.
- Learn how to find controls within templates on both the server and client.

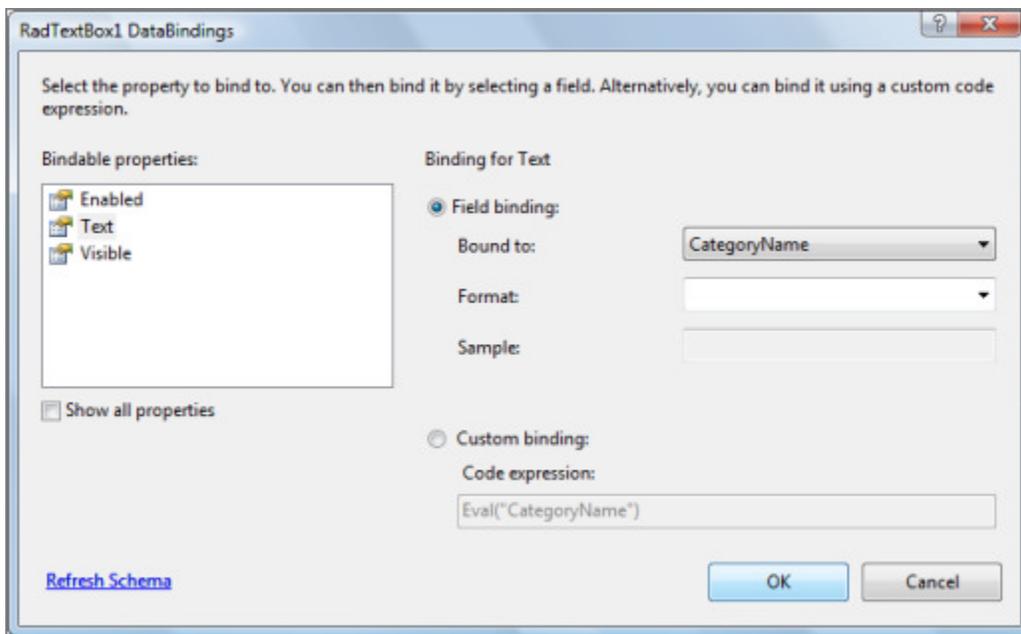
12.2 Introduction

Templates let you paint a completely unique layout within your RadControls. Any arbitrary HTML can be used to fill templates: standard ASP.NET controls, HTML elements and other RadControls can all be placed inside of templates. RadControls may have multiple templates, each defining a particular area of the control. For example, RadPanelBar has only an "Item" template representing each panel while RadGrid has templates for the "no records" message, the paging area and the master table view edit form.

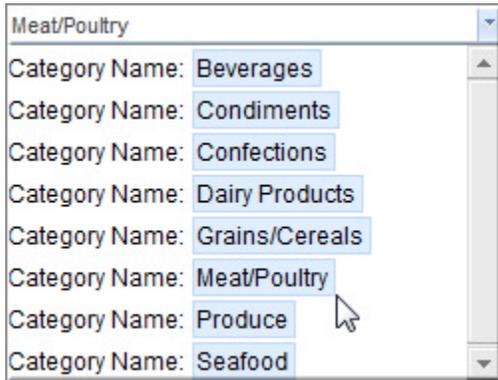
The screenshot below shows a RadComboBox in the designer with the template opened and a RadTextBox added to the template surface. The diagram also shows the resulting markup.



Anything within the template can be bound to data for display-only or editing. The RadTextBox within the RadComboBox is shown having the Text property bound to a "CategoryName" column in a database table:



The resulting combo/template/text box combination displays at runtime looking something like the example below:



12.3 Getting Started

Templates are a "bread sandwich" that you can fill with whatever you please. In this example we will fill a RadPanelBar with pictures, text and numbers from a furniture business database. When complete, the panel bar will look something like the screenshot below:

Wooden Table	
Mfg: <input type="text" value="Molzano Italy"/>	
Price: <input type="text" value="\$3,534.00"/>	
Quantity: <input type="text" value="3"/>	
Dimensions: <input type="text" value="1.2 x 0.8"/>	
Glass Table	
Mfg: <input type="text" value="Ovalia Germany"/>	
Price: <input type="text" value="\$546.00"/>	
Quantity: <input type="text" value="5"/>	
Dimensions: <input type="text" value="1.4 x 0.9"/>	
Black Leather Couch	
Mfg: <input type="text" value="Spidea England"/>	
Price: <input type="text" value="\$4,565.00"/>	
Quantity: <input type="text" value="4"/>	
Dimensions: <input type="text" value="2.6 x 0.7"/>	

Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Using the Solution Explorer, add a new Folder to your project and name it "Images".
3. Drag the contents of the "\VS Projects\Images\Stock" folder into your project's "Images" folder.
4. Locate the "Telerik.mdf" file in the "Live Demos\App_Data" folder under the folder where you installed RadControls for ASPNET AJAX. Drag this file into the "App_Data" folder of your project.
5. Open the "Web.config" file of your project. Add the standard Northwind connection string to your project by replacing the line

```
<connectionStrings />
```

with

```
<connectionStrings>
```

```
  <add name="TelerikConnectionString"
```

```
    connectionString="Data
```

```
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|Telerik.mdf;Integrated Security=True;User Instance=True"
```

```
    providerName="System.Data.SqlClient" />
```

```
</connectionStrings>
```

Add a data-bound Panel Bar

1. Drag a RadPanelBar from the Tool Box onto your Web page. Set its **Skin** property to "Telerik".
2. In the RadPanelBar Smart Tag, select "<New data source...>" from the **Choose Data Source** drop-down.
3. In the first page of the **DataSource Configuration Wizard**, select "Database" as the database type, and click **OK** to move to the next page.
4. On the **Choose Your Data Connection** page, select "TelerikConnectionString" from the drop-down list. Then click the **Next** button to continue.



Gotcha! If you don't see "TelerikConnectionString" in the drop down list, either something is wrong with the connection string (which if you copy it from above should not be an issue) or Telerik.mdf has not been copied to the App_data folder in your application. The file has to be present in the App_Data folder and the connection string must specify the file that is in that folder.

5. On the **Configure the Select Statement** page, make sure the "Specify columns from a table or view" radio button is selected, and then choose "Products" from the "Name" drop-down list.
6. Check "*" to return all columns.
7. Click the **Next** button, test the query if you wish, and then click **Finish**.
8. In the Properties Window for the RadPanelBar:
 - Set the **DataTextField** property to "ProductName". The product name will show up in the title bar of each panel.
 - Set the **DataValueField** property to "ID".
 - Also set the RadPanelBar **Width** property to "300px". This will accept the width of the items that will populate it.

Edit the Template

1. In the Source view of the page, add the styles below to the <head> tag.

The Image style floats the image to the right and provides space between the image and following label. The Frameset style provides more height to hold all the items.

[ASP.NET] Styles for Template Elements

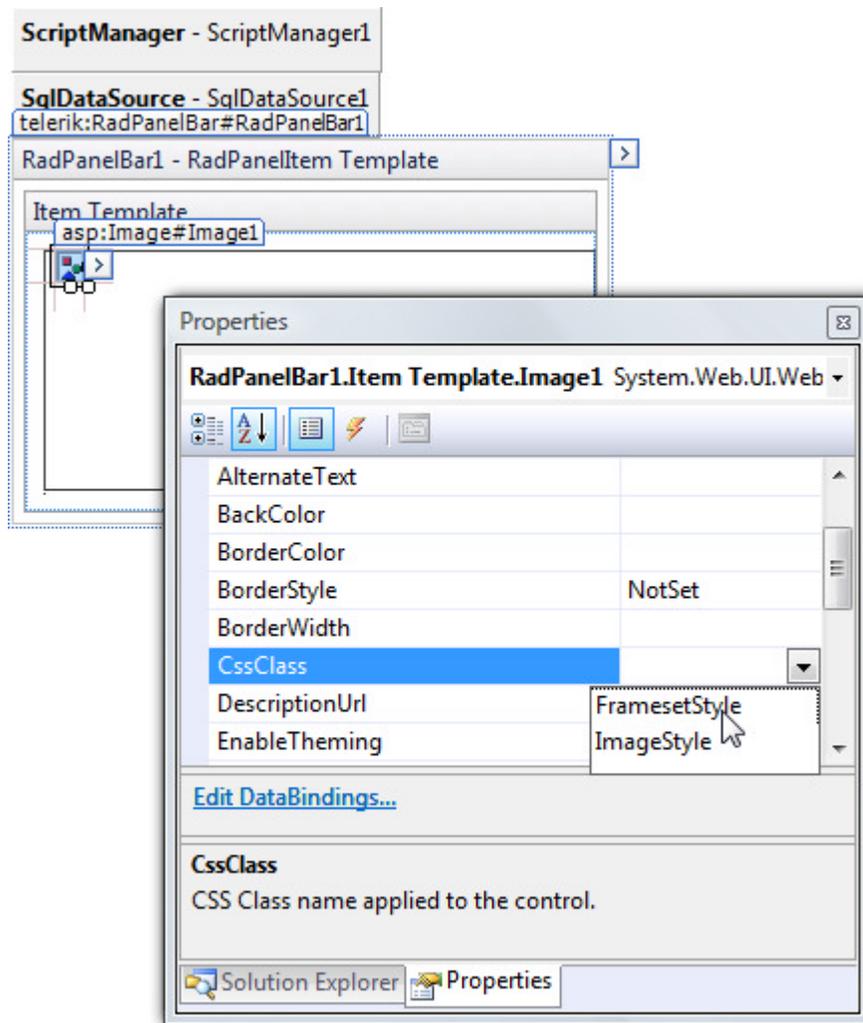
```
<style type="text/css">
  .ImageStyle
  {
    float: Right;
    margin: 5px;
  }
  .FramesetStyle
  {
    height: 120px;
    margin: 5px;
  }
</style>
```

2. Add a <frameset> element into the template. This will surround the other elements with a box and when styled, provide a margin for visual separation with neighboring panels.

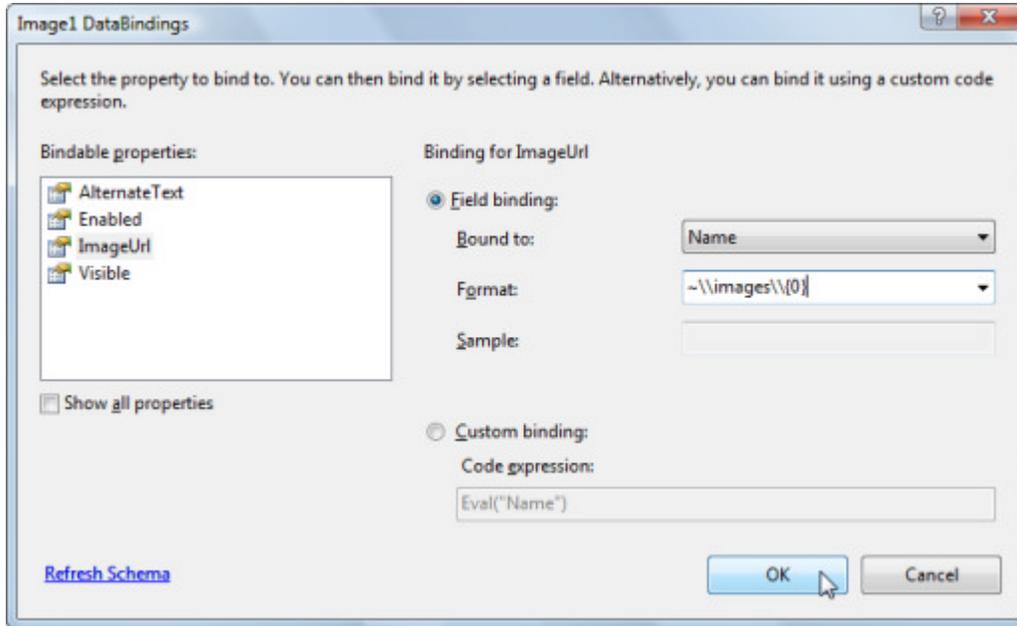
[ASP.NET] Adding the Frameset

```
<telerik:RadPanelBar ID="RadPanelBar1" runat="server" DataSourceID="SqlDataSource1"
  DataTextField="ProductName" Skin="Telerik" Width="300px">
  <ItemTemplate>
    <fieldset class="FramesetStyle">
      <!-- bound elements go here-->
    </fieldset>
  </ItemTemplate>
</telerik:RadPanelBar>
```

3. The controls to include in the template can all be added and data bound from the Design view of the page.
Using the RadPanelBar Smart Tag, select the Edit Templates option. Add and configure the following controls:
 - From the Standard tab of the Toolbox add an Image control. In the Properties window, set the **CssClass** property to "ImageStyle".



- Using the Image Smart Tag select **Edit Databindings....**
- In the DataBindings dialog, leave "ImageUrl" selected in the Bindable properties list. On the right side of the dialog, leave the **Field binding** radio button selected. In the **Bound to** drop down list, select "Name" ("Name" contains the name of an image that will be used in a URL path). In the **Format** entry add "-\\images\\{0}". Click **OK** to close the dialog and create the bindings in markup.



The output for this bound field will be placed in the Image controls ImageUrl property and look something like the markup example below. In the section on "Binding Expressions" we will explore what these binding expressions, i.e. <%= %> mean:

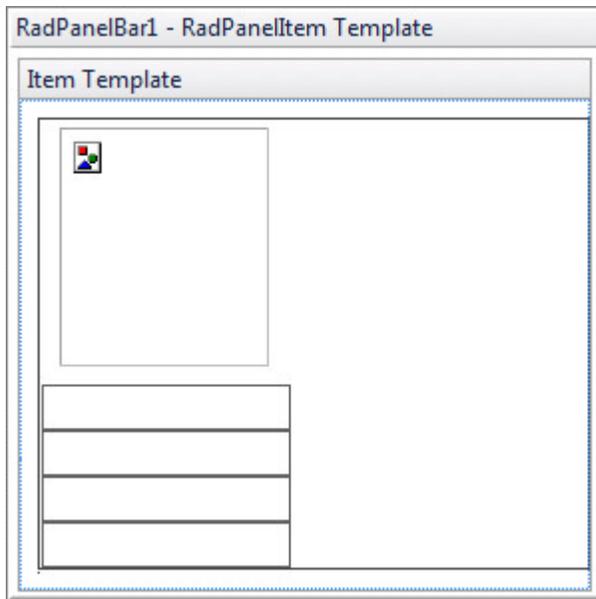
[ASP.NET] Bound ImageUrl Tag

```
<asp:Image ID="Image1" runat="server" CssClass="ImageStyle"
  ImageUrl='<%= Eval("Name", "~\\images\\{0}") %>' />
```



Gotcha! The ASP image path should be pre-pended with a tilde ("~") to indicate the path is relative to the project path. It will work without the tilde in Internet Explorer but fail to display in FireFox.

- Place your cursor just to the right of the Image control and press the **Enter** key four times. *This will place break ("**
**") tags that will separate the text box controls you will add next.*
- Drop **four RadTextBox** controls below the image, each one below the other. The design surface should now look something like this:



- Select the first RadTextBox and set the **Label** property to "Mfg:", the **Skin** to "Telerik" and **ReadOnly** to "true". Using the RadTextBox Smart Tag select **Edit Databindings....** Bind the Text property to the "Manufacturer" field. Leave the Format blank.
 - Configure the next three RadTextBox controls with these same settings but change the labels and fields to:
 - Label: "Price:", Field: "Price"
 - Label: "Quantity:", Field: "Quantity"
 - Label: "Dimensions:", Field: "Dimensions"
4. Press **Ctrl-F5** to run the application. Notice that the standard ASP:Image control has been bound to the data for the RadPanelBar and that the style has floated the image to the right side. All of the RadTextBox Text properties have been bound to columns in the table. The completed markup right now if you look at it is filled with "<%# %>" server tags with expressions like "Eval("Manufacturer")". This next section on Binding Expressions explains what these expressions are and how they are used.

[ASP.NET] The ItemTemplate Markup

```
<ItemTemplate>
<fieldset class="FramesetStyle">

    <!-- bound elements go here-->
    <asp:Image ID="Image1" runat="server" CssClass="ImageStyle"
        ImageUrl='<%= Eval("Name", "~\\images\\{0}") %>' />
    <telerik:RadTextBox ID="RadTextBox1" Runat="server" Label="Mfg:"
        LabelCssClass="radLabelCss_Telerik" Skin="Telerik" ReadOnly="true"
        Text='<%= Eval("Manufacturer") %>' Width="125px">
    </telerik:RadTextBox>
    <br />
    <telerik:RadTextBox ID="RadTextBox2" Runat="server" Label="Price:"
        LabelCssClass="radLabelCss_Telerik" Skin="Telerik" ReadOnly="true"
        Text='<%= Eval("Price") %>' Width="125px">
    </telerik:RadTextBox>
    <br />
    <telerik:RadTextBox ID="RadTextBox3" Runat="server" Label="Quantity:"
        LabelCssClass="radLabelCss_Telerik" Skin="Telerik" ReadOnly="true"
```

```

        Text='<%=# Eval("Quantity") %>' Width="125px">
    </telerik:RadTextBox>
    <br />
    <telerik:RadTextBox ID="RadTextBox4" Runat="server" Label="Dimensions:"
        LabelCssClass="radLabelCss_Telerik" Skin="Telerik" ReadOnly="true"
        Text='<%=# Eval("Dimensions") %>' Width="125px">
    </telerik:RadTextBox>
</fieldset>
</ItemTemplate>

```

12.4 Binding Expressions

With the ASP.NET 2.0 binding syntax you can bind any RadControl within your template to its underlying data. For that matter, you can output any public server-side method that returns a value. The syntax uses `<%=# %>` tags and looks something like this:

```
<%=# MyMethod() %>
```

For example, in the code-behind for the web page you can declare a method that returns a string:

[VB] Declaring the Public Method

```

Public Function MyMethod() As String
    Return "The time is: " + DateTime.Now.ToLongTimeString()
End Function

```

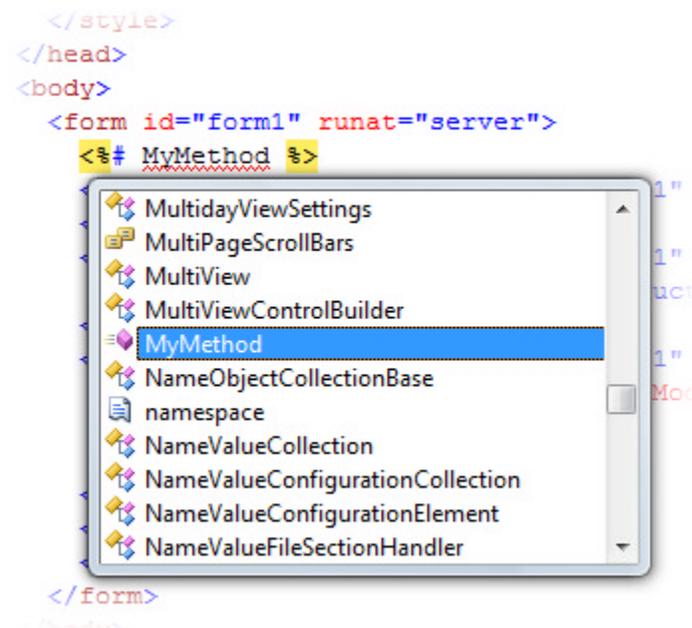
[C#] Declaring the Public Method

```

public string MyMethod()
{
    return "The time is: " + DateTime.Now.ToLongTimeString();
}

```

In the markup for the same page you can call that method within "`<%=# %>`" tags using Intellisense. Just put your cursor inside the tag and hit Ctl-Spacebar to get the drop down list of properties and methods available in the page context. Don't forget to add the curly braces after the method name, i.e. `MyMethod()` (but don't add a semi-colon).



If you ran this now, nothing would show in the browser. You need to call `DataBind()` on whatever context the

UI for ASP.NET AJAX

method is being called from. In this case that would be the page itself. In other situations the call to DataBind() might also be for a <div> tag surrounding your objects (if the div has an ID and runat="server"), DataBind() might apply to a RadControl or the DataBind() call might be implicit because the RadControl is declaratively bound.

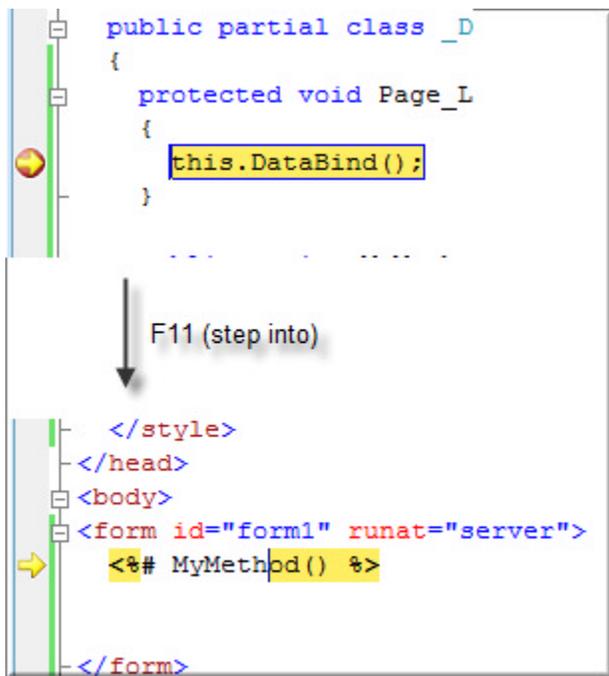
[VB] Calling DataBind()

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Me.DataBind()
End Sub
```

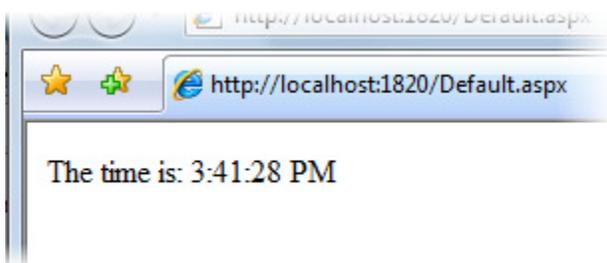
[C#] Calling DataBind()

```
protected void Page_Load(object sender, EventArgs e)
{
    this.DataBind();
}
```

We can actually debug starting from the DataBind() and stepping to the MyMethod() call by putting a break point on the DataBind() and pressing F11 to step into the server method called from within the markup:



...and pressing F5 to let the application continue, the output of the method shows up as a literal in the browser:



The point here is that there's nothing strange or mysterious going on in the server code that is embedded in markup. We have access to all the facilities available in code-behind. And the code itself will be in the language of the code-behind, i.e. if you were developing the code-behind in VB.NET, you can develop the code in the markup with VB.NET.

Container and DataItem

Typically you won't be using your own methods inside the markup. More often you will use "Container.DataItem" to access bound data from within the markup.

To see how this works, let's add a ScriptManager, SqlDataSource and RadPanelBar into the equation. In the markup below notice that the SqlDataSource is consuming the Products table from the Telerik database. The kind of data isn't critical to this demonstration, just so we have data that is bound to the control. The RadPanelBar is hooked up to the SqlDataSource declaratively through the DataSourceID property. Also notice that we have an <ItemTemplate> tag with nothing in it.

[ASP.NET] Form with ScriptManager, SqlDataSource and RadPanelBar

```
<form id="form1" runat="server">
  <asp:ScriptManager ID="ScriptManager1" runat="server">
  </asp:ScriptManager>

  <asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%= $ ConnectionStrings:TelerikConnectionString %>"
    SelectCommand="SELECT * FROM [Products]"></asp:SqlDataSource>

  <telerik:RadPanelBar ID="RadPanelBar1" runat="server" DataSourceID="SqlDataSource1"
    Skin="Hay" DataTextField="ProductName">
    <ItemTemplate>
    </ItemTemplate>
  </telerik:RadPanelBar>

</form>
```

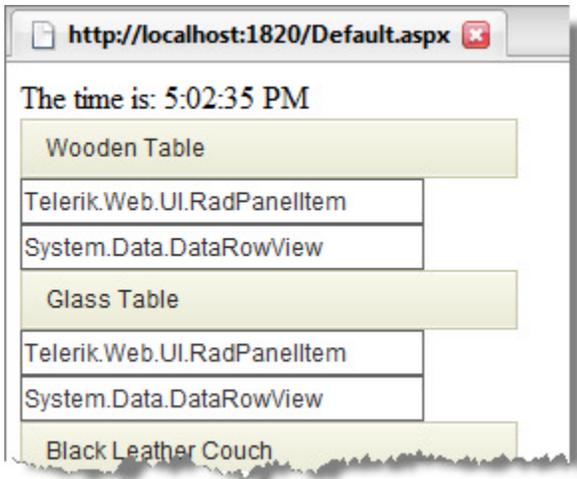
Controls can be added to the ItemTemplate and any of the properties can be populated using the <%= # %> syntax. The two objects in use constantly for this kind of markup that accesses bound data are **Container** and **DataItem**. Here we just output the class names of each using the ToString() method.

[ASP.NET] Adding Controls to the ItemTemplate

```
<ItemTemplate>
  <telerik:RadTextBox ID="RadTextBox1" runat="server" Width="200px"
    Text='<%= # Container.ToString() %>'></telerik:RadTextBox>
  <br />
  <telerik:RadTextBox ID="RadTextBox2" runat="server" Width="200px"
    Text='<%= # Container.DataItem.ToString() %>'></telerik:RadTextBox>
</ItemTemplate>
```

Container and Container.DataItem types are output as strings to the template area. You can see that "Container" is the item instance for a given row and "DataItem" is the underlying data for the row. These two objects correspond to objects introduced in the Data Binding chapter, e.Item and e.Item.DataItem, available from the ItemDataBound event arguments. You may also remember from this chapter that the DataItem may be some other type besides DataRowView depending on what kind of DataSource is bound to the control.

UI for ASP.NET AJAX



A typical use for these two objects is to populate the Text or Value properties of a control. To use Container you cast it to its actual runtime type and then use the methods of the type. In this case Container is a RadPanelItem so we cast Container as a RadPanelItem to get access to Text, Value and DataItem properties. The general rule here is that when a control is not data bound (ie when using statically declared items), you use Container, while with data-bound controls, you generally use Container.DataItem.

The example markup below shows that you can populate any of the control's properties using information from Container or DataItem. Text and Tooltip properties are populated here by casting Container.DataItem to a DataRowView and accessing the corresponding columns.

[ASP.NET] Populating Properties From Container.DataItem

```
<ItemTemplate>
  <telerik:RadTextBox ID="RadTextBox1" runat="server" Width="200px"
    Label="Manufacturer:" Skin="Hay"
    Text='<# (Container.DataItem as System.Data.DataRowView)["Manufacturer"] %>'
    Tooltip='<# (Container.DataItem as System.Data.DataRowView)["SalesRepresentative"] %>'>
  </telerik:RadTextBox>
</ItemTemplate>
```

The browser output for the bound RadTextBox looks like this screenshot where the Text is populated with the manufacturer name and the tool tip shows the sales representative name.



Eval and Bind

DataBinder.Eval() is a helper method that uses reflection to resolve the DataItem type. To make it even easier, DataBinder is the default context for data binding expressions so the syntax can be more succinct. The property assignments from the last example are shortened substantially:

[ASP.NET] Using Eval

```
Text='<%# Eval("Manufacturer") %>'
ToolTip='<%# Eval("SalesRepresentative") %>'>
```

It would be nice to pre-pend some additional text to the ToolTip so the hint is more descriptive, like this example screenshot:



Eval has an optional format parameter that allows substitution of a single value, making it easy to combine data values with other text:

[ASP.NET] Formatting with Eval()

```
ToolTip='<%# Eval("SalesRepresentative", "The sales rep is {0}") %>'>
```

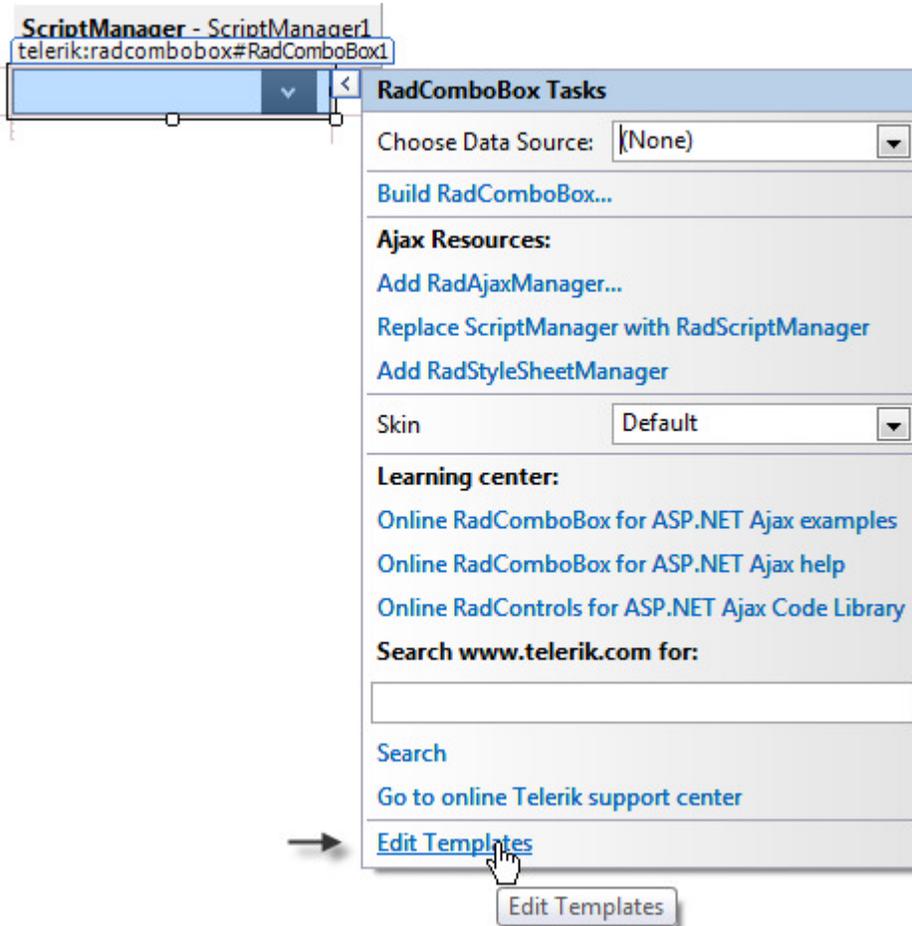
While Eval() is used for read-only access to the data. Bind() is a similar method used in place of Eval() for two-way data binding. Use Bind() when you want to update the data for controls containing values used as DataSource parameters.

[ASP.NET] Using Bind()

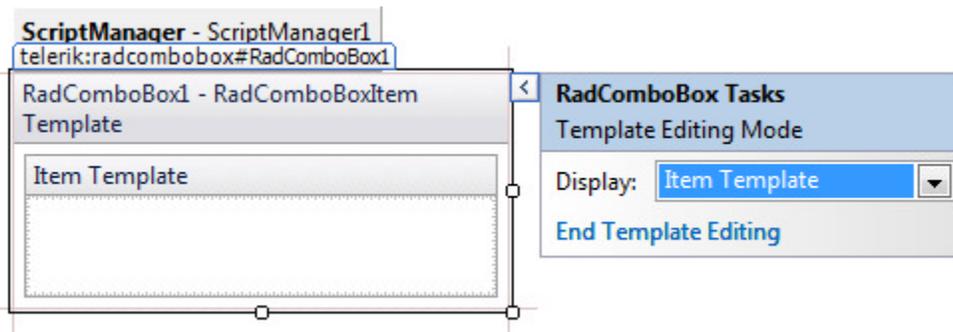
```
Text='<%# Bind("ProductName") %>'
```

12.5 Designer Interface

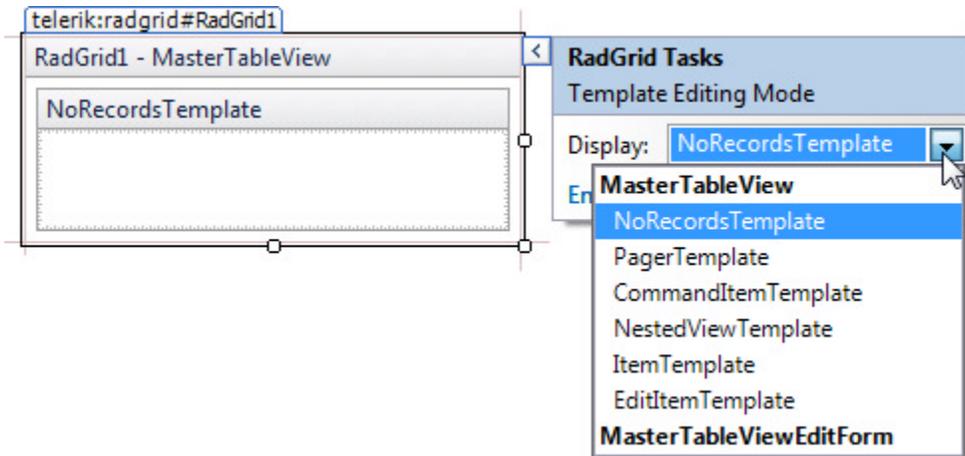
You can access template designers using the RadControl's Smart Tag **Edit Templates** options as this screenshot of a RadComboBox control shows:



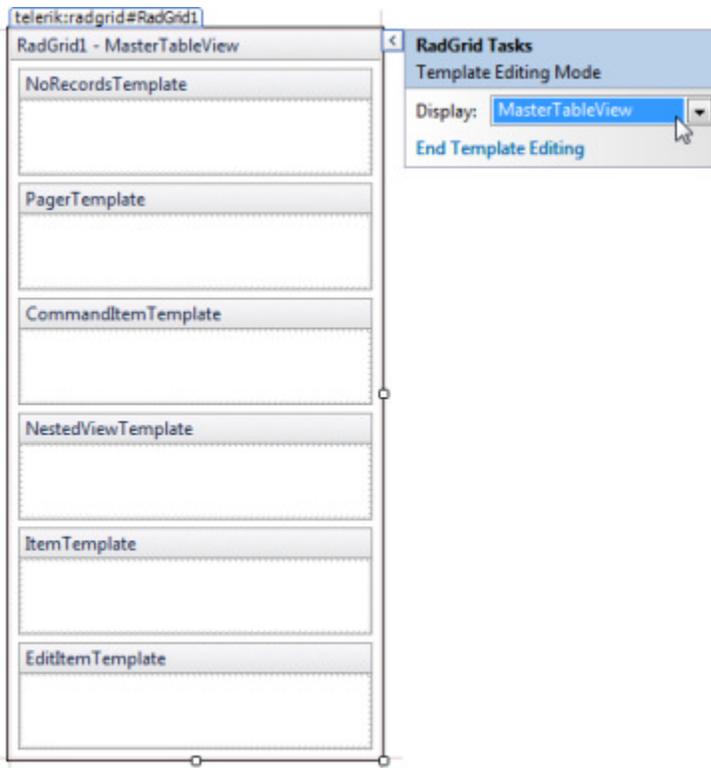
After clicking the **Edit Templates** option, the designer for the control switches to "template editing mode" where an area is reserved for dropping controls. Also notice that the Smart Tag now has an **End Template Editing** option.



While RadComboBox has a single template type "Item Template", other RadControls may have multiple template types. When in template editing mode the Smart Tag "Display" drop down list lets you navigate to different template types for editing. RadGrid for example lists templates for portions of the master table view.



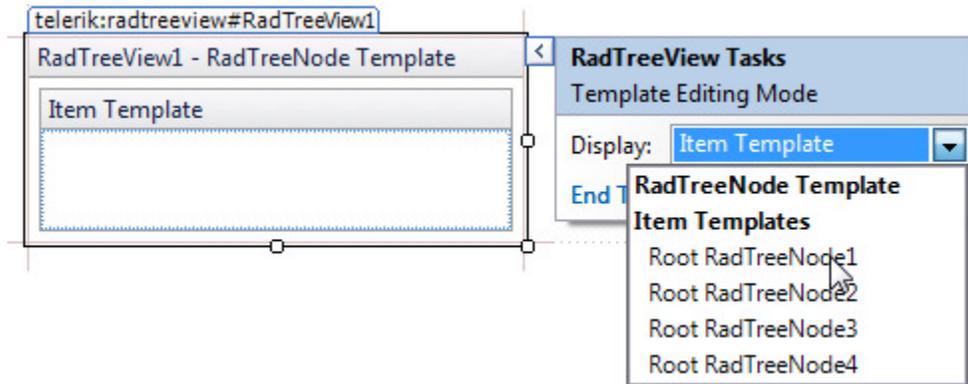
You can select individual templates or select one of the bolded items in the list to edit a collection of templates all at one time. Selecting the bold "MasterTableView" from the list shows all the templates under it in the list:



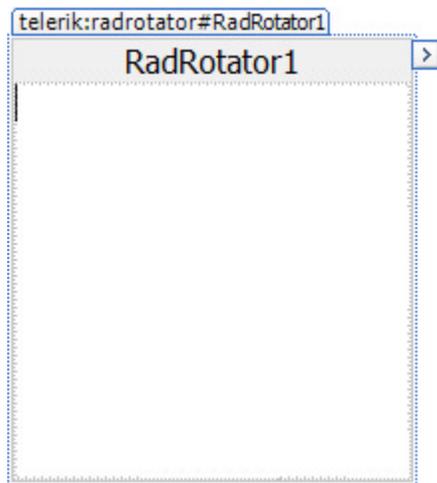
As you can see there is a fair amount of variation between how individual RadControls present templates. The commonality between them is that the Smart Tag displays an **Edit Templates** link when templates are available for editing, provides a list of templates that can be edited and finally the **End Template Editing** link ends editing. The key is to look for the cues on the Smart Tag.

For example, the RadToolBar doesn't show template related options until you have added buttons to the list. When you enter into edit template mode, the list of templates includes a template per button or split-button

(drop down's don't support templates). RadTreeView supports a general template for all nodes and can also have a template per specific node that overrides the general template:



RadControls that handle a single item such as RadRotator and RadPageView have no such "Edit Templates" Smart Tag options to begin with. Instead they are always in "template mode" as this screenshot of RadRotator shows:



12.6 Server-Side Programming

The most common programmatic task you're likely to perform on a template is locating some control *within* the template. You may want to retrieve properties from controls within a template, set properties or attach event handlers on the fly within code.

Finding Controls in a Template

For example, a RadToolBar has two elements, the first is a button with text "Add" and the second is a button that has an item template. The first button increments a text box located in the template of the second button:





You can find the complete source for this project at:
 \VS Projects\Templates\ServerSide

First, notice the markup for the second button defines a RadNumericTextBox within the button's template.

[ASP.NET] RadToolBar Markup

```
<telerik:RadToolBar ID="RadToolBar1" runat="server" OnButtonClick="RadToolBar1_ButtonClick">
  <Items>
    <telerik:RadToolBarButton Text="Add" />
    <telerik:RadToolBarButton Value="TemplateTextBox" runat="server">
      <ItemTemplate>
        <telerik:RadNumericTextBox ID="txtResultCount" runat="server" NumberFormat-
DecimalDigits="0"
          Value="0">
        </telerik:RadNumericTextBox>
      </ItemTemplate>
    </telerik:RadToolBarButton>
  </Items>
</telerik:RadToolBar>
```

Depending on how deep your control is buried, you may have to go through iterations of calling FindControl(), along the lines of:

```
MyBigContainer.FindControl("MyInsideContainer").FindControl
("TheControlImLookingFor");
```

The code below executes if the first button, "Add" is clicked. The second button is retrieved using the FindItemByValue() method. FindControl() is called against this second button and the returned control is cast to RadNumericTextBox. The text is retrieved, converted to int, incremented and placed back in the text of the control.

[VB] Finding the Embedded Control

```
Protected Sub RadToolBar1_ButtonClick(ByVal sender As Object, ByVal e As
RadToolBarEventArgs)
  If e.Item.Text.Equals("Add") Then
    Dim ButtonWithTemplate As Control = RadToolBar1.FindItemByValue("ButtonWithTemplate")
    Dim textbox As RadNumericTextBox = DirectCast(ButtonWithTemplate.FindControl
("txtResultCount"), RadNumericTextBox)
    textbox.Text = (Convert.ToInt32(textbox.Text) + 1).ToString()
  End If
End Sub
```

[C#] Finding the Embedded Control

```
protected void RadToolBar1_ButtonClick(object sender, RadToolBarEventArgs e)
{
  if (e.Item.Text.Equals("Add"))
  {
    Control ButtonWithTemplate = RadToolBar1.FindItemByValue("ButtonWithTemplate");
    RadNumericTextBox textbox = (RadNumericTextBox)ButtonWithTemplate.FindControl
("txtResultCount");
    textbox.Text = (Convert.ToInt32(textbox.Text) + 1).ToString();
  }
}
```

Creating a Custom Template

If the Templates at design time don't give you enough flexibility, perhaps because you need to create templates

UI for ASP.NET AJAX

dynamically for some reason, you can create your template and assign it at run time. Your template object only needs to implement the **ITemplate** interface. **ITemplate** comes from the **System.Web.UI** namespace and has a single method you need to implement, **InstantiateIn**. **InstantiateIn()** passes a single **Control** parameter, the container for the template. Within your implementation, you create your controls, set properties, hook up methods and add the controls to the template container. When you're hooking up your control events, you can assign the **DataBinding** event. The **DataBinding** event is introduced in the **System.Web.UI.Control** class and allows you to access the **DataItem** object and set your control properties based on the data.

This next example creates a simple template class and assigns it to a **RadTabStrip**. The tab strip will be bound to a table of country names. As each template is instantiated and bound, if a flag image with a filename corresponding to the country name is available, it's displayed in the tab strip.



Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Using the Solution Explorer, add a new Folder to your project and name it "Images".
3. Drag the contents of the "\VS Projects\Images\Flags" folder into your project's "Images" folder.
4. Locate the "Countries.mdf" file in the "Live Demos\App_Data" folder under the folder where you installed RadControls for ASPNET AJAX. Drag this file into the "App_Data" folder of your project.
5. Open the "Web.config" file of your project. Add a connection string to your project by replacing the line

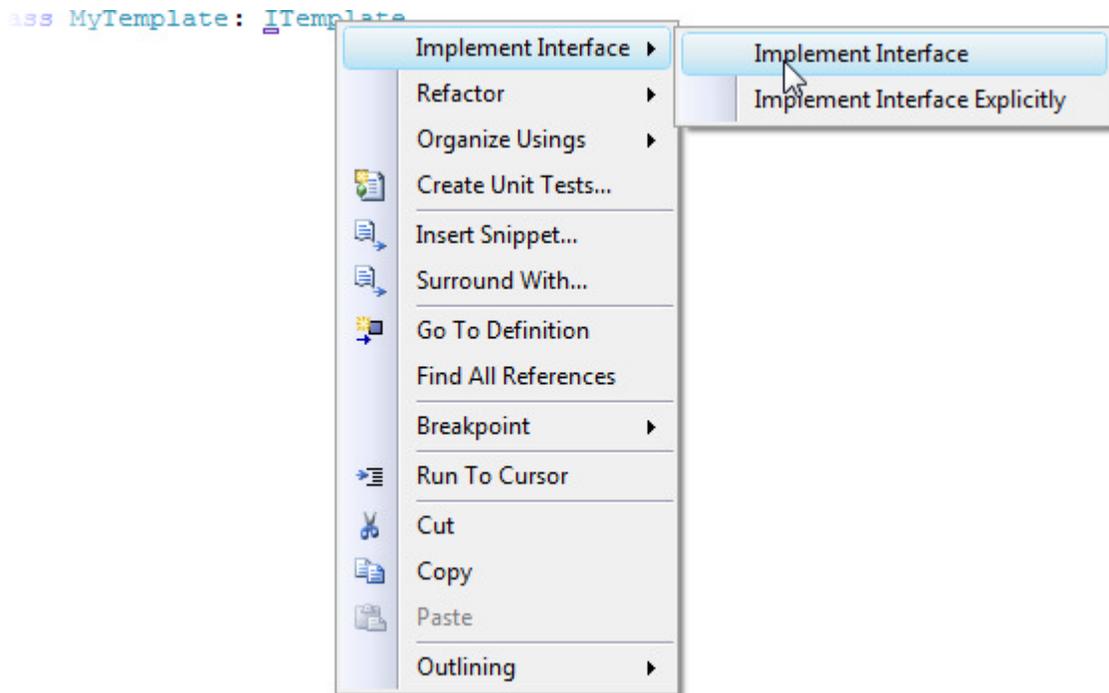
```
<connectionStrings />
with
<connectionStrings>
  <add name="TelerikConnectionString"
    connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|Countries.mdf;Integrated
Security=True;User Instance=True"
    providerName="System.Data.SqlClient" />
</connectionStrings>
```

Add a Data-bound Tab Strip

1. Drag a RadTabStrip from the Tool Box onto your Web page. Set its **Skin** property to "Office2007".
2. In the RadTabStrip Smart Tag, select "<New data source...>" from the **Choose Data Source** drop-down.
3. In the first page of the **DataSource Configuration Wizard**, select "Database" as the database type, and click **OK** to move to the next page.
4. On the **Choose Your Data Connection** page, select "CountriesConnectionString" from the drop-down list. Then click the **Next** button to continue.
5. On the **Configure the Select Statement** page, make sure the "Specify columns from a table or view" radio button is selected, and then choose "Countries" from the "Name" drop-down list.
6. Check "*" to return all columns.
7. Click the **Next** button, test the query if you wish, and then click **Finish**.
8. In the Properties Window for the RadTabStrip:
 - Set the **DataTextField** property to "Name".
 - Set the **DataValueField** property to "ID".
 - Set the **Orientation** property to "VerticalLeft".

Create a Custom Template Class

1. In the Solution Explorer, right-click the project and select **Add | Class...** from the context menu. Set the class file name to be "MyTemplate.cs".
2. Add the following namespaces to the "Imports" (VB) or "using" (C#) section of the code:
 - System.Web.UI
 - System.Data;
 - System.IO;
 - System.Web.UI.WebControls;
 - Telerik.Web.UI;
3. In the code-behind for the MyTemplate class, descend the class from ITemplate.
4. Right-click ITemplate and choose **Implement Interface | Implement Interface** from the context menu.



The class should look something like the example below:

[VB] Implementing the ITemplate Interface

```
Public Class MyTemplate
    Implements ITemplate
    #region ITemplate Members
    Public Sub InstantiateIn(ByVal container As Control)
        Throw New NotImplementedException()
    End Sub
    #End Region
End Class
```

[C#] Implementing the ITemplate Interface

```
public class MyTemplate: ITemplate
{
    #region ITemplate Members
    public void InstantiateIn(Control container)
    {
        throw new NotImplementedException();
    }
    #endregion
}
```

5. Implement the InstantiateIn() method.

This method creates a standard ASP Image control, hooks up the DataBinding event so we can extract the path name for the image and adds the image to the container's Controls collection. In this case the container is a RadTab. The code also scales the image down slightly to fit on the tab and adds a right margin to position the image.

A standard ASP Label control is also created and added to the containers Controls collection. Notice that

the label is also hooked up to the *DataBinding* event.

[VB] Implement the *InstantiateIn* Method

```
Public Sub InstantiateIn(ByVal container As Control)
    ' Create an image control, hook up to DataBinding
    ' event and add to the RadTab (the container in this case)
    Dim image As New System.Web.UI.WebControls.Image()
    image.Height = Unit.Pixel(20)
    image.Width = Unit.Pixel(25)
    image.Style.Add("Margin-right", "5px")
    AddHandler image.DataBinding, AddressOf image_DataBinding
    container.Controls.Add(image)
    ' Create a label, hook up to DataBinding event and add to
    ' the RadTab
    Dim label As New Label()
    AddHandler label.DataBinding, AddressOf label_DataBinding
    container.Controls.Add(label)
End Sub
```

[C#] Implement the *InstantiateIn* Method

```
public void InstantiateIn(Control container)
{
    // Create an image control, hook up to DataBinding
    // event and add to the RadTab (the container in this case)
    System.Web.UI.WebControls.Image image =
        new System.Web.UI.WebControls.Image();
    image.Height = Unit.Pixel(20);
    image.Width = Unit.Pixel(25);
    image.Style.Add("Margin-right", "5px");
    image.DataBinding += new EventHandler(image_DataBinding);
    container.Controls.Add(image);
    // Create a label, hook up to DataBinding event and add to
    // the RadTab
    Label label = new Label();
    label.DataBinding += new EventHandler(label_DataBinding);
    container.Controls.Add(label);
}
```

6. Implement the Image's *DataBinding* Event. Add the following code to the *MyTemplate* class.

*In this event, "sender" is the object being databound, i.e.. the Image control. The parent of sender is the container control. You can cast the container control to its runtime type and then access the *DataItem* property of the container.*

*At this point you have references to the object being data bound and the data that is being bound to it. After that we build a path that looks in the images directory for a "png" file with the same name as the country name. If found, the *ImageUrl* path is assigned.*

[VB] Implementing the *DataBinding* Event for Image

```
Sub image_DataBinding(ByVal sender As Object, ByVal e As EventArgs)
    Dim image As System.Web.UI.WebControls.Image = TryCast(sender,
System.Web.UI.WebControls.Image)
    Dim tab As RadTab = TryCast((TryCast(sender, Control)).Parent, RadTab)
    Dim row As DataRowView = TryCast(tab.DataItem, DataRowView)
```

```
Dim path As String = [String].Format("~\images\{0}.png", row("Name").ToString())
Dim physicalPath As String = System.Web.HttpContext.Current.Server.MapPath(path)
If Not File.Exists(physicalPath) Then
    image.Visible = False
Else
    image.ImageUrl = path
End If
End Sub
```

[C#] Implementing the DataBinding Event for Image

```
void image_DataBinding(object sender, EventArgs e)
{
    System.Web.UI.WebControls.Image image =
        sender as System.Web.UI.WebControls.Image;
    RadTab tab = (sender as Control).Parent as RadTab;
    DataRowView row = tab.DataItem as DataRowView;
    string path =
        String.Format("~\images\{0}.png", row["Name"].ToString());
    string physicalPath = System.Web.HttpContext.Current.Server.MapPath(path);
    if (!File.Exists(physicalPath))
    {
        image.Visible = false;
    }
    else
    {
        image.ImageUrl = path;
    }
}
```

7. Implement the label's DataBinding Event. Add the code below to the MyTemplate class.

Data binding for the Label follows the same pattern as for the Image but is less complex because we don't need to determine a URL. The Label is created, the data is retrieved for the current tab, the label text is assigned from the data. The Label font is also set to a blue color.

[VB] Implementing the DataBinding Event for Label

```
Sub label_DataBinding(ByVal sender As Object, ByVal e As EventArgs)
    Dim label As Label = TryCast(sender, Label)
    Dim tab As RadTab = TryCast(label.Parent, RadTab)
    Dim row As DataRowView = TryCast(tab.DataItem, DataRowView)
    label.Text = row("name").ToString()
    label.ForeColor = System.Drawing.Color.Blue
End Sub
```

[C#] Implementing the DataBinding Event for Label

```
void label_DataBinding(object sender, EventArgs e)
{
    Label label = sender as Label;
    RadTab tab = label.Parent as RadTab;
    DataRowView row = tab.DataItem as DataRowView;
    label.Text = row["name"].ToString();
    label.ForeColor = System.Drawing.Color.Blue;
}
```

8. Press Ctl-F5 to run the application. The flag graphics should show up in tabs that have available images,

namely "USA", "China", "England" and "Canada".

12.7 Client-Side Programming

You can find controls on the client-side using server code that is very similar to the server-side example. The basic idea is to take whatever code you have on the server and place it between `<% %>` tags within the JavaScript function. To output server generated values to the client, use `"<%= %>"` (including the = sign).

 `<%=` is a shortcut for `Response.Write()`.

This example first verifies that the "Add" button is clicked. Inside the "if" is where it gets interesting. Take a close look at the server tags `<% %>`. All this code executes before the page is even output to the browser. The entire first block between `<% %>` does not even show up in the browser source, but it all executes, finds the `RadNumericTextBox` and outputs the client id right between the `$find()` parenthesis.

[JavaScript] Getting a Reference to a Control in a Template on the Client

```
function clientButtonClicked(sender, args)
{
    // get a reference to the tool bar button, get it's text
    if (args.get_item().get_text() == "Add")
    {
        // run some code on the server. this evaluates well before any JavaScript executes.
        // this entire block within the server tags will not be visible in the output
        <%
            // get a reference to the second button, i.e. "ButtonWithTemplate"
            Control control = RadToolBar1.FindItemByValue("ButtonWithTemplate");
            // from the button reference, find the numeric text box
            RadNumericTextBox txtResultCount = control.FindControl("txtResultCount") as
RadNumericTextBox;
        %>
        // output the results of the server code as a parameter of the $find() method.
        // this will be visible
        var textBox = $find('<%= txtResultCount.ClientID %>');
        // retrieve, increment and reassign the text box value
        textBox.set_value(textBox.get_value() + 1);
        alert(textBox.get_value());
    }
}
```

Compare that with the actual output to the browser. The server code is over and done. The only thing that remains from the server is the client id for `RadNumericTextBox` inside the call to `$find()`. Here's a screenshot of the rendered script in a JavaScript debugger with two overlays showing the server tags before rendering:

```
// run some code on the server. this evaluates well before any JavaScript executes.
// this entire block within the server tags will not be visible in the output
<%
  // get a reference to the second button, i.e. "ButtonWithTemplate"
  Control control = RadToolBar1.FindControl("ButtonWithTemplate");
  // from the button reference, find the numeric text box
  RadNumericTextBox txtResultCount =
  |control.FindControl("txtResultCount") as RadNumericTextBox;
%>
```

```
12 {
13   // get a reference to the tool bar button, get it's text
14   if (args.get_item().get_text() == "Add")
15   {
16     // run some code on the server. this evaluates well before any JavaScript executes.
17     // this entire block within the server tags will not be visible in the output
18
19     // output the results of the server code as a parameter of the $find() method.
20     // this will be visible
21     var textBox = $find('RadToolBar1 | txtResultCount');
22     // retrieve, increment and reassign the text box value
23     textBox.set_value(textBox.get_value() + 1);
24     alert(textBox.get_value());
25   }
26 }
27
28 </script>
29
```

```
var textBox = $find('<%= txtResultCount.ClientID %>');
```

12.8 Summary

In this chapter you learned how templates are used within RadControls. First you built a simple application that used templates and data bound to elements within the template. We explored the details of binding expressions, starting with public server methods and working through Container, DataItem, Eval() and Bind() methods. You learned how to find controls in both server and client code.

13 ActiveSkill: Admin Page

13.1 Objectives

- Build the Admin home page structure and the code-behind necessary to dynamically load user controls.
- Create user controls to be loaded into the Admin home page.
- Create a new ActiveSkill skin based on the existing "Black" skin.

13.2 Introduction

If you remember in the "Getting Started" chapter we created the beginnings of an "Admin" page. In another chapter on RadAjax we talked about how to dynamically load user controls on-the-fly. In the skinning chapter we created a custom skin.

In this chapter we combine all of these techniques to fill out the functionality of the Admin home page so that it has:

- A tab strip that users can use to navigate between user controls
- A series of user controls for each database function required.
- A new custom "ActiveSkill" skin that will be applied to the entire application.

13.3 Build the Admin Page

The general layout of the page will have a slight black gradient along the top with a logo image positioned on the left. The lower part of the page will have a RadTabStrip on the left with tools for the administrator to use for tasks such as maintain questions. The tab strip will be placed on a sliding panel bar so that the "toolbox" can slide away out of sight to allow for more usable space. The right side of the lower page will contain dynamically loaded user controls that house the actual database maintenance functionality.

The Admin page is built in two steps. The first is to prepare the page by designing the page layout, adding styles, adding controls and setting properties. The second is to add the code-behind to handle the dynamically added controls.



You can find the complete source for this project at:
 \VS Projects\

Prepare the Page

1. In the Solution Explorer, add all the images found in the directory \VS Projects\Images\ActiveSkill\Admin to the \images directory in the ActiveSkillUI project.
2. In the AdminHome.aspx page design view add a ScriptManager control.
3. Add a RadAjaxManager control to the page.
4. Switch to the source view of the page and add the <style> tag contents below to the <head> tag of the page.

These styles will set the maximum height and width for the page elements. Also notice the "topDiv" style with its "header_strip.png". This image is a one pixel wide, 86 pixel tall image of a very slight gradient moving from dark to slightly lighter at the bottom.

[CSS] AdminHome Styles

```
<style type="text/css">

html, body, form
{
    height: 100%;
    width:100%;
    margin: 0px;
    padding: 0px;
    overflow: hidden;
}
#mainDiv
{
    height: 100%;
    width: 100%;
    background-color: Black;
}

#topDiv
{
    background-image: url(../Images/header_strip.png);
    background-repeat: repeat;
    height: 86px;
    width: 100%;
}

#bottomDiv
{
    height: 100%;
    width: 100%;
}

#divContent
{
    height: 100%;
    width: 100%;
}
</style>
```

5. Add the markup below to stake out the general layout structure of divs, splitter and panes. This markup should be placed inside the <form> just after the ScriptManager and RadAjaxManager.

Notice in "bottomDiv" that "SplitterMain" contains a vertical RadSplitter with two RadPanes, one for the tab strip area and the other for whatever context is triggered by the tab strip selection. Also notice the commented areas with "this goes here" notes. We will fill these in shortly.

[ASP.NET] The Top Level Layout of AdminHome

```
<div id="mainDiv" runat="server">
<div id="topDiv" >
    <!--logo image goes here-->
</div>
<div id="bottomDiv" runat="server">
    <telerik:RadSplitter ID="SplitterMain" runat="server" Height="100%" Width="100%"
        Items-Capacity="4" SplitBarsSize="">
        <telerik:RadPane ID="MainLeft" runat="server" Locked="True" Index="0" >
            <!--sliding zone, pane and RadTabstrip go here-->
```

```

</telerik:RadPane>
<telerik:RadPane ID="MainRight" runat="server" Height="" Index="1" Width="">
  <!--content div and placeholder go here to contain dynamic controls-->
</telerik:RadPane>
</telerik:RadSplitter>
</div>
</div>

```

6. Add an HTML tag to "topDiv". This image will contain the logo.

[ASP.NET] Adding the Image Tag to "topDiv"

```

<div id="topDiv">
  
</div>

```

7. To the top RadPane with ID "MainLeft", add a sliding zone, pane and tab strip.

[ASP.NET] Adding the Tab Strip and Sliding Zone

```

<telerik:RadPane ID="MainLeft" runat="server" Locked="True" Index="0" >

  <!--sliding zone, pane and rad tabstrip go here-->
  <telerik:RadSlidingZone ID="MasterSlidingZone" runat="server"
  DockedPaneId="MasterSlidingPane"
  Height="100%" Width="15px">
    <telerik:RadSlidingPane ID="MasterSlidingPane" runat="server" EnableDock="true"
  DockText="Tools"
  Title="Admin Tools" Height="100%" Scrolling="none" TabView="TextOnly" Width="160px":
    <telerik:RadTabStrip ID="tsMain" runat="server" Orientation="VerticalRight"
  SelectedIndex="0" Height="336px" Width="160px">
      </telerik:RadTabStrip>
    </telerik:RadSlidingPane>
  </telerik:RadSlidingZone>

</telerik:RadPane>

```

8. To the bottom RadPane with ID "MainRight", add a <div> tag with id "divContent" and a standard Placeholder control.

The Placeholder will be used to host user controls that will be loaded dynamically.

[ASP.NET] Add Content Div and Placeholder

```

<telerik:RadPane ID="MainRight" runat="server" Index="1" >

  <!--content div and placeholder go here to contain dynamic controls-->
  <div id="divContent" runat="server" style="height: 100%; width: 100%">
    <asp:Placeholder ID="Placeholder1" runat="server"></asp:Placeholder>
  </div>
</telerik:RadPane>

```

9. In the design view for the form, select the RadTabStrip, then click the Smart Tab Build RadTabStrip... option. Create tabs with the following Text, Value and ImageUrl Properties:
- Text "Categories", Value "Categories.ascx", ImageUrl "~/images/Categories.png".
 - Text "Questions", Value "Questions.ascx", ImageUrl "~/images/Questions.png".
 - Text "Create Exams", Value "CreateExams.ascx", ImageUrl "~/images/Exams.png".

- Text "Schedule", Value "ScheduleExams.ascx", ImageUrl "~/images/Schedule.png".

10. Use the RadAjaxManager Smart Tag to select the **Configure Ajax Manager** option. Set the RadTabStrip "tsMain" to update itself and the Placeholder1 control.

Writing the Code-Behind for the Admin Page

The code here will look similar to the examples from the RadAjax chapter that talked about dynamic user controls.

1. Add a new IASControl interface to the ActiveSkillBO project with class file name "ASControl.cs". Replace the code with the code shown below.

The interface has a single method FirstLoad() that is fired by the hosting page when the control should initialize itself. A dictionary is passed to this method with a collection of parameters. The parameter is not used in the "Admin" page and so this will always be passed null. In the "User" page we will pass parameters from the client and so this parameter will be used then.

[VB] The IASControl Interface

```
Imports Microsoft.VisualBasic
Imports System.Collections.Generic
Namespace Telerik.ActiveSkill.Common
    Public Interface IASControl
        Sub FirstLoad(ByVal args As Dictionary(Of String, String))
    End Interface
End Namespace
```

[C#] The IASControl Interface

```
using System.Collections.Generic;
namespace Telerik.ActiveSkill.Common
{
    public interface IASControl
    {
        void FirstLoad(Dictionary<string, string> args);
    }
}
```

2. Back in the ActiveSkillUI project, in the designer, double-click the RadTabStrip to create a TabClick event handler. Add the code below to the TabClick event handler.

This code LoadUserControl() to actually load the user control to the place holder. The first parameter is the parent control, the second is the CurrentControl path that will contain the path to the clicked on tab and the last parameter indicates that this is the first load (always the case when clicking the tab).

[VB] Handling the Tab Click

```
Protected Sub tsMain_TabClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadTabStripEventArgs)
    ' this always is a first load
    LoadUserControl(Placeholder1, CurrentControl, True)
End Sub
```

[C#] Handling the Tab Click

```
protected void tsMain_TabClick(object sender,
Telerik.Web.UI.RadTabStripEventArgs e)
```

```
{
  // this always is a first load
  LoadUserControl(PlaceHolder1, CurrentControl, true);
}
```

- In the code-behind for AdminHome.aspx, add namespace **Telerik.ActiveSkill.Common** to the "Imports" (VB) or "using" (C#) statements.
- Add the "CurrentControl" property. *CurrentControl tracks the last user control path loaded to the place holder.*

[VB] Adding Properties

```
' store the last selected control for reload
Private Const CurrentControlKey As String = "CurrentControlKey"
Private Property CurrentControl() As String
  Get
    Return If(ViewState(CurrentControlKey) Is Nothing, "", ViewState
(CurrentControlKey).ToString())
  End Get
  Set(ByVal value As String)
    ViewState(CurrentControlKey) = value
  End Set
End Property
```

[C#] Adding Properties

```
// store the last selected control for reload
private const string CurrentControlKey = "CurrentControlKey";
private string CurrentControl
{
  get
  {
    return ViewState[CurrentControlKey] == null ?
      "" : ViewState[CurrentControlKey].ToString();
  }
  set
  {
    ViewState[CurrentControlKey] = value;
  }
}
```

- Add the code below to load the user control into the Placeholder.

This method uses the parentControl LoadControl() method to load a user control, given a path to the ascx file. The path is passed to this method in the newControlPath parameter. isFirstLoad if false indicates we're just reloading the control because of a postback due to something on the page and that we're still looking at the same user control. If we click on a tab or it's not a postback, then isFirstLoad will be true.

[VB] Load the User Control

```
Private Function LoadUserControl(ByVal parentControl As Control, ByVal newControlPath As
String, ByVal isFirstLoad As Boolean) As Control
  ' Load the control and set its id
  Dim control As Control = Page.LoadControl(newControlPath)
```

```
control.ID = newControlPath
' the viewstate control will be out of sync with
' the previously loaded control. Temporarily shut off
' viewstate if this is the first load of the control
If isFirstLoad Then
    control.EnableViewState = False
End If
' add to the parent controls collection
parentControl.Controls.Add(control)
' if this is the first load (first time the page is loaded or
' a new tab has been clicked) enable the viewstate again. Forgetting to
' reenale the viewstate will controls to be loaded only once. Then
' call the FirstLoad() method of the web user control for first time
' loading tasks.
If isFirstLoad Then
    control.EnableViewState = True
    TryCast(control, IASControl).FirstLoad(Nothing)
End If
Return control
End Function
```

[C#] Load the User Control

```
private Control LoadUserControl(Control parentControl,
    string newControlPath, bool isFirstLoad)
{
    // Load the control and set its id
    Control control = Page.LoadControl(newControlPath);
    control.ID = newControlPath;
    // the viewstate control will be out of sync with
    // the previously loaded control. Temporarily shut off
    // viewstate if this is the first load of the control
    if (isFirstLoad)
    {
        control.EnableViewState = false;
    }
    // add to the parent controls collection
    parentControl.Controls.Add(control);
    // if this is the first load (first time the page is loaded or
    // a new tab has been clicked) enable the viewstate again. Forgetting to
    // reenale the viewstate will controls to be loaded only once. Then
    // call the FirstLoad() method of the web user control for first time
    // loading tasks.
    if (isFirstLoad)
    {
        control.EnableViewState = true;
        (control as IASControl).FirstLoad(null);
    }
    return control;
}
```

6. Handle the Page_Load event. Replace the Page_Load event handler with the code shown below.

The very first time the page loads, the selected tab value is saved off in CurrentControl. To determine if this is a control that simply needs to be reloaded or if we're navigating to a new tab the CurrentControl is compared with selected tab. If it's a new tab, then save it in CurrentControl and the TabClick event will

take care of calling `LoadUserControl`. If this is the same control and we need to reload, then call `LoadUserControl()`. This second condition will happen the first time the page is loaded and on any reloads that are not caused by a tabclick.

[VB] Handling the Page_Load Event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' if this is the first load of the page,
    ' set the CurrentControl to the selected tab value
    If (Not IsPostBack) Then
        CurrentControl = tsMain.SelectedTab.Value
    End If
    Dim isNewControl As Boolean = Not CurrentControl.Equals(tsMain.SelectedTab.Value)
    If isNewControl Then
        ' new control, so wait for the tabclick to load it
        CurrentControl = tsMain.SelectedTab.Value
    Else
        ' same control, reload it.
        LoadUserControl(Placeholder1, CurrentControl, (Not IsPostBack))
    End If
End Sub
```

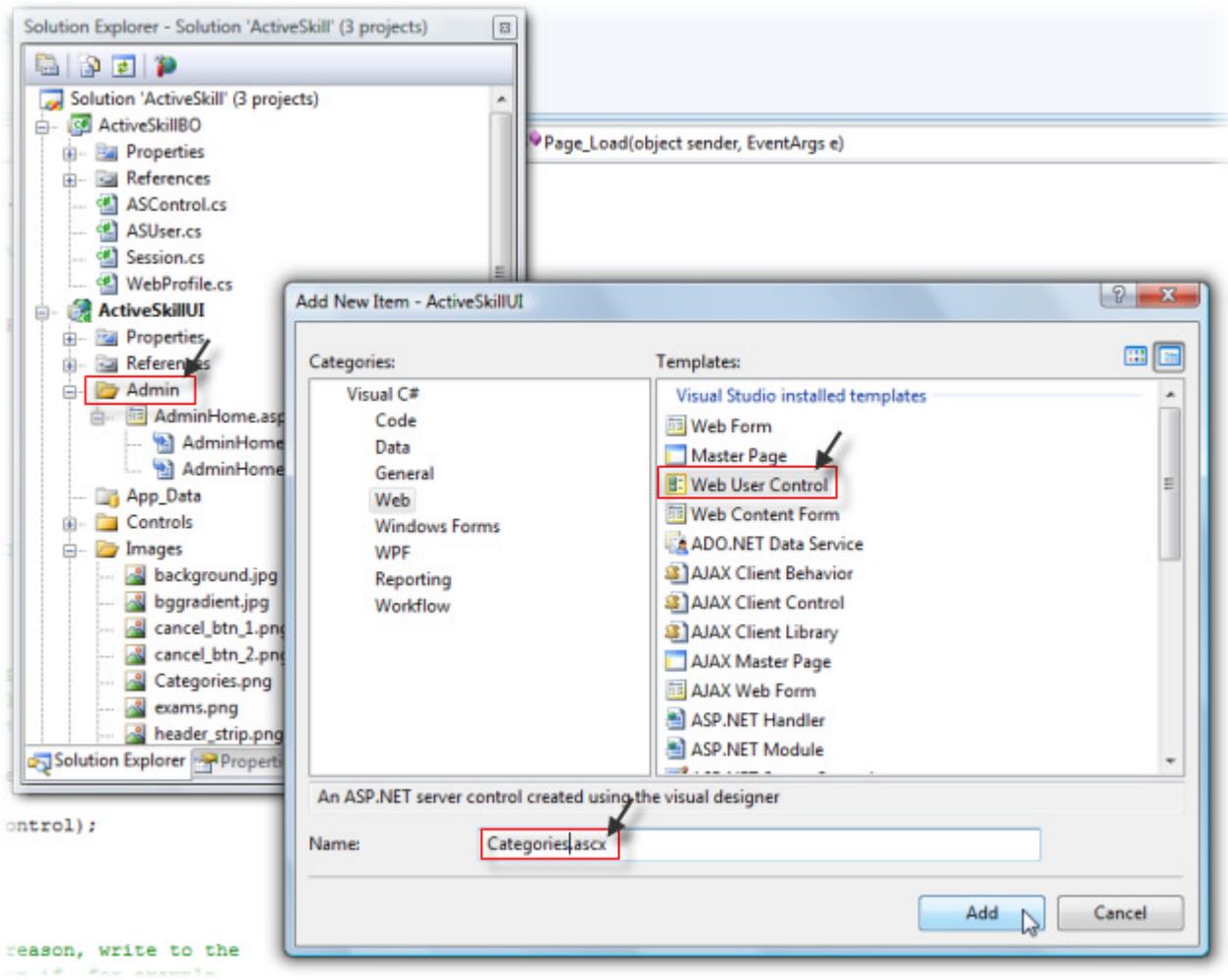
[C#] Handling the Page_Load Event

```
protected void Page_Load(object sender, EventArgs e)
{
    // if this is the first load of the page,
    // set the CurrentControl to the selected tab value
    if (!IsPostBack)
    {
        CurrentControl = tsMain.SelectedTab.Value;
    }
    bool isNewControl = !CurrentControl.Equals(tsMain.SelectedTab.Value);
    if (isNewControl)
        // new control, so wait for the tabclick to load it
        CurrentControl = tsMain.SelectedTab.Value;
    else
        // same control, reload it.
        LoadUserControl(Placeholder1, CurrentControl, !IsPostBack);
}
```

13.4 Create User Controls

We need user controls for each database maintenance operation we want to perform, i.e. maintenance for categories, exams, questions and scheduling of exams. In this phase we build those controls without filling out their ultimate functionality.

1. Using the Solution Explorer, right-click the Admin folder of the ActiveSkillUI project. Select **Add | New Item**. Select "Web User Control" from the list of item types. Set the file name to be "Categories.ascx" and click **Add**.

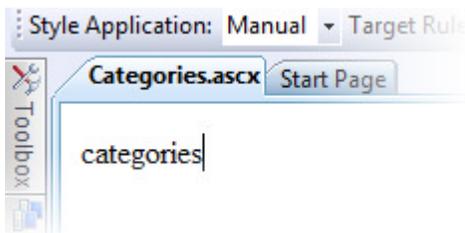


2. Add three more user controls to the \Admin directory: "CreateExams.ascx", "Questions.ascx", "ScheduleExams.ascx".

These file names must exactly match the Value properties of the RadTabStrip on the admin page.

3. In the design view of each user control, type in the name of the page.

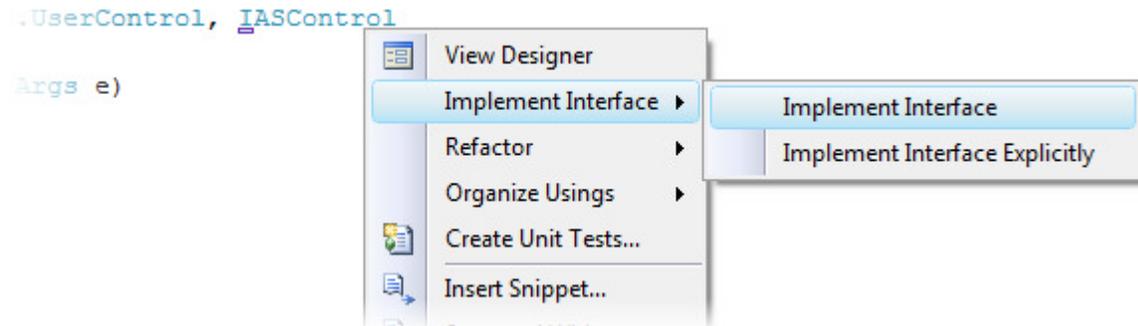
This is just so we can see what control is loaded and that the user control swapping mechanism is working as expected.



4. In the code-behind for the Categories.ascx user control:
 - o Add the Telerik.ActiveSkill.Common namespace to the "Imports" (VB) or "uses" (C#) sections of code.
 - o Add IASControl to the class declaration. Right-click IASControl and select **Implement Interface** from the context menu.
 - o Inside the implementation, remove the the line that throws an exception so that the FirstLoad()

method is empty (see code example below).

In the AdminHome page, the FirstLoad() method is being called and will fail if this interface is not present. Once the IASControl is part of the declaration of the user control class, it must be implemented. The implementation generated by the UI will contain a NotImplementedException that must be removed so that we can load each of the user controls without failing.



The resulting code for the user control should now look something like this example:

[VB] The UserControl with empty IASControl Implementation

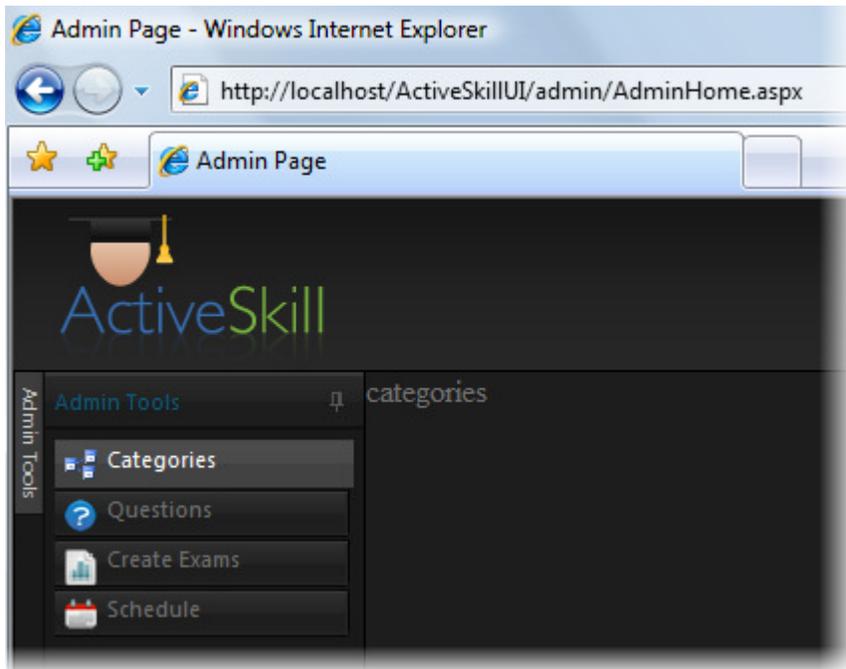
```
Imports Microsoft.VisualBasic
Imports System
Imports System.Collections.Generic
Imports Telerik.ActiveSkill.Common
Namespace Telerik.ActiveSkill.UI.Admin
    Partial Public Class Categories
        Inherits System.Web.UI.UserControl
        Implements IASControl
        Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
            End Sub
        #region IASControl Members
        Public Sub FirstLoad(ByVal args As Dictionary(Of String, String)) Implements
IASControl.FirstLoad
            End Sub
        #End Region
    End Class
End Namespace
```

[C#] The UserControl with empty IASControl Implementation

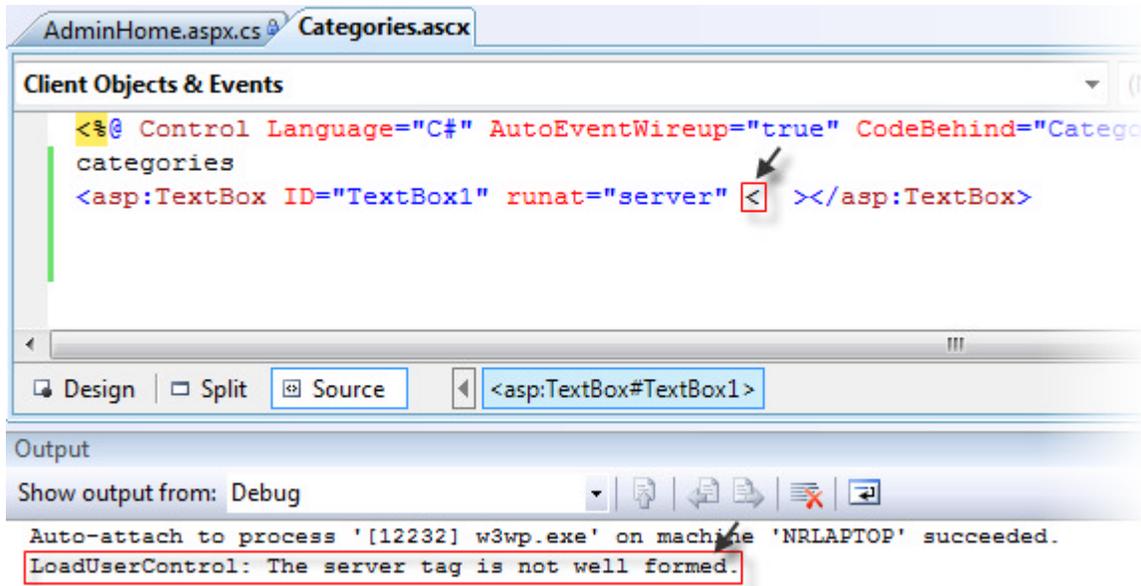
```
using System;
using System.Collections.Generic;
using Telerik.ActiveSkill.Common;
namespace Telerik.ActiveSkill.UI.Admin
{
    public partial class Categories : System.Web.UI.UserControl, IASControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        #region IASControl Members
```

```
public void FirstLoad(Dictionary<string, string> args)
{
}
#endregion
}
```

5. Repeat this last step for each of the user controls "CreateExams.ascx", "Questions.ascx" and "ScheduleExams.ascx".
6. Press **F5** to run the application. You should be able to click the tabs and have the names you typed in show up in the placeholder area.

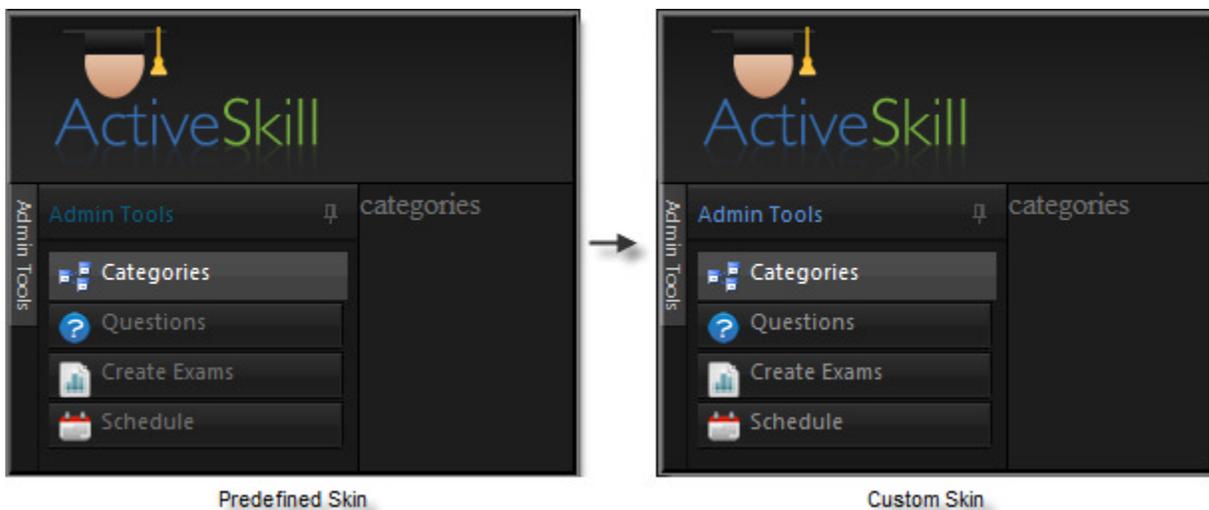


💡 If one of your user controls fails to load, don't forget to look at the Output window in Visual Studio. For example, if in the categories user control you place a textbox and then add another "less than" bracket to malformed the tag, this error will show up in the output window. You must run the application in debug mode to get the output window display of the error message.



13.5 Create ActiveSkill Skin

While the "Black" skin theme on this application is sleek, some of it is hard to read. And we may want to customize aspects of the skin later to more exactly correspond to our visual theme. For right now we want to set up a custom ActiveSkill skin now before we get too invested in the predefined skins. We will adjust the colors of the splitter "Admin Tools" title and the fonts for unselected tabs so that they are just slightly brighter. Take a look at the screen shot below that shows the user interface with the "Black" skin and again with our new "ActiveSkill" skin.



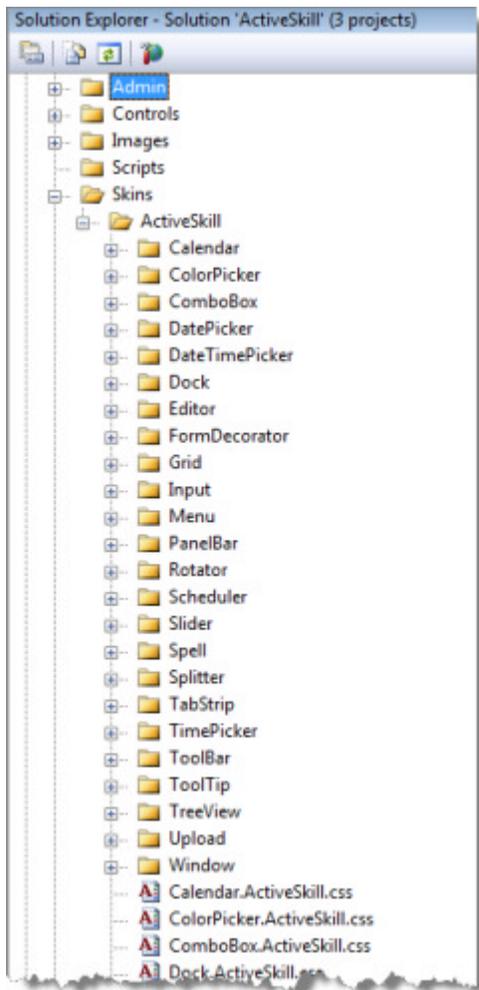
To create a new skin you typically start by copying an existing skin that's closest to the appearance you're looking for. You change the file names from something like "Calendar.Hay.css" to "Calendar.MyStyle.css".

UI for ASP.NET AJAX

Then you open each of the CSS files and change the instances of "_Hay" or ".Hay" to their "_MyStyle" and ".MyStyle" equivalents. From there you change the actual style specifics to get the look and feel you're after. In this case we've done the first two steps for you by copying the files for the "Black" skin, renaming the files and renaming the CSS selectors. The altered files are found in the \VS Projects\ActiveSkill Skins folder.

1. Copy the contents of \VS Projects\ActiveSkill Skins to your \Skins folder.

This step should create in your project the \Skins\ActiveSkill folder. The Skins folder will have sub-folders for each control and a series of CSS files, also for each control. The Skins directory should look something like the example below:



2. In the web.config file add the following appSettings to globally set the skin for the entire application. *This will set the skin for all RadControls that are not already defined by settings on the specific controls.*

[XML] Defining AppSettings for Skins in Web.Config

```
<appSettings>
  <!-- RadControls for ASP.NET AJAX Step By Step -->
  <add key="Telerik.Skin" value="ActiveSkill"/>
  <add key="Telerik.EnableEmbeddedSkins" value="false"/>
</appSettings>
```

3. Go through Login.aspx and Register.aspx and completely remove references to the `Skin` and `EnableEmbeddedSkins`.
4. Open the `Splitter.ActiveSkill.css` file for editing. Locate the string "slideHeaderDocked". Change the "color" attribute to "color:#598FD3;".
5. Open the `TabStrip.ActiveSkill.css` file for editing. Locate the string "color: #717171" and change the setting to "color: #919191".
6. Press **Ctrl-F5** to run the application. *The login and registration pages should look as they did before. The "AdminHome" page where the "Admin Tools" and unselected tab strip text will be slightly brighter.*

13.6 Summary

In this chapter we built the Admin Home page, starting with the general layout and adding the code-behind required to swap user controls dynamically. We created each of the user controls and tested the dynamic swapping behavior. Finally we created a new custom ActiveSkill skin based on the standard Telerik "Black" skin and configured the application to use that skin throughout.

14 RadAsyncUpload

14.1 Objectives

- Get acquainted with **RadAsyncUpload** as well as explore some of its basic features.
- Create a simple application to get confidence in using the RadAsyncUpload.
- Explore the RadAsyncUpload design time interface, including Smart Tag and the different skins that you can apply.
- Explore some of the most important properties that represent a sufficient part of the RadAsyncUpload functionality.
- Learn some server-side coding techniques as well as get acquainted with the server-side events of RadAsyncUpload.
- Explore some of the client-side methods and client-side events.
- Explore some advanced techniques, such as the three types of modules that RadAsyncUpload uses.

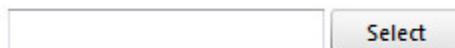
14.2 Introduction

General Information

RadAsyncUpload offers **asynchronous upload** capability while maintaining the look of the regular RadUpload control. The upload process requires that the files are uploaded to a custom handler and not to the hosting page. Files are stored in a temporary location **until a postback** occurs. The temporary location is cleaned-up automatically.

Internally, RadAsyncUpload can choose between three modules for uploading - IFrame, Flash and Silverlight. The module with higher priority is Silverlight. If there is no Silverlight installed on the client machine, RadAsyncUpload will utilize the Flash module. If there is no Flash as well, RadAsyncUpload will use the IFrame module which is supported out of the box on all browsers.

The control **supports Web Farm** scenarios. Upload progress is available in this scenario as long there is Flash/Silverlight installed on the client machine.



● PlusMinus.png × Remove



Temporary Files

RadAsyncUpload relies on saving temporary files to work. When posted, files are saved to the designated temp folder (**App_Data/RadUploadTemp** by default) with unique names. Once a postback occurs the **RadAsyncUpload** fires the **OnFileUploaded** event for each file. The target file is passed as part of the arguments to the event and can be set as either valid (default) or invalid. After the events fire, all files marked as valid are automatically saved to the **TargetFolder** if it's set.

Finally, all processed temporary files are deleted. Temporary files are also deleted after a set amount of time defined by the **TemporaryFileExpiration** property

Validation

Validation differs from **RadUpload** as it is now possible to validate size on the client, as long as there is Silverlight or Flash installed on the client's browser. If the validation fails, **RadAsyncUpload** will fire the **OnClientValidationFailed**.

Web Farms

In web farms, each server will need to use the **same MachineKey** as **RadAsyncUpload** uses it for encryption. Most web farms should already have their **MachineKeys** synchronized as this is the recommended approach for web farm deployment.

14.3 Getting Started

Getting Started

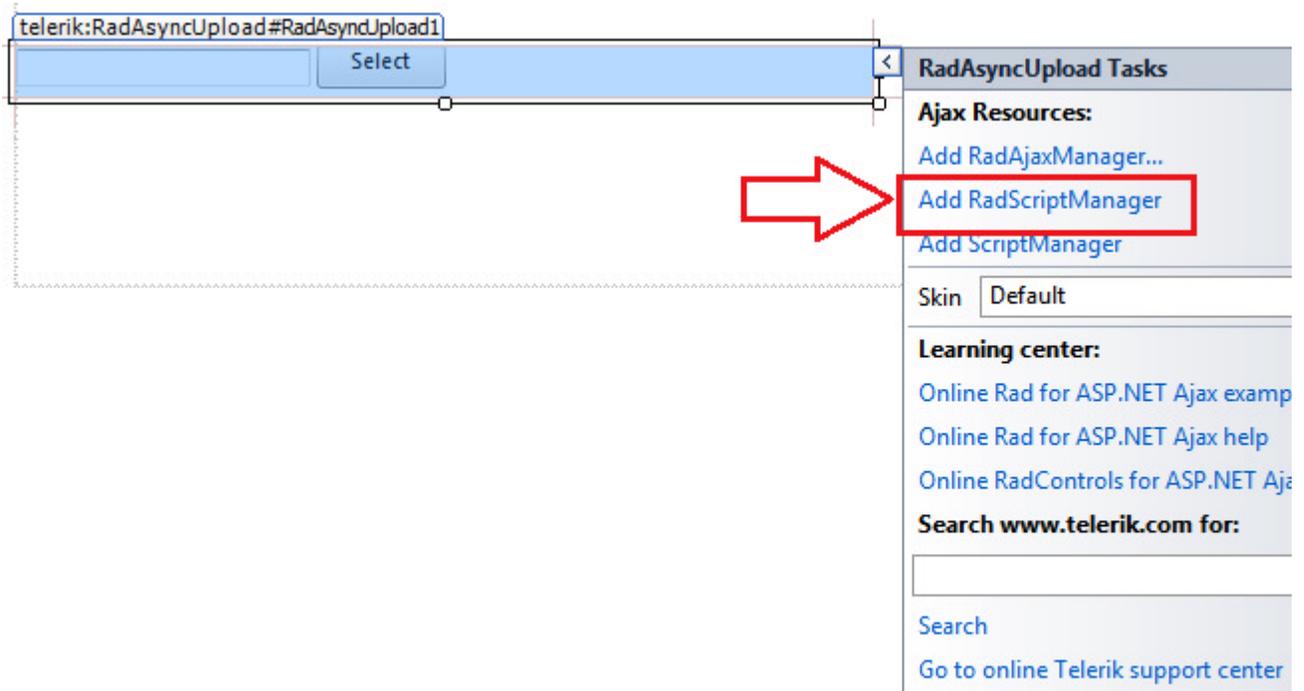
This tutorial will walk you through creating a Web page that uses **RadAsyncUpload** control. Following the steps below you can learn how to:

- Use **RadAsyncUpload** to upload files.
- Use skins to provide a consistent look & feel.

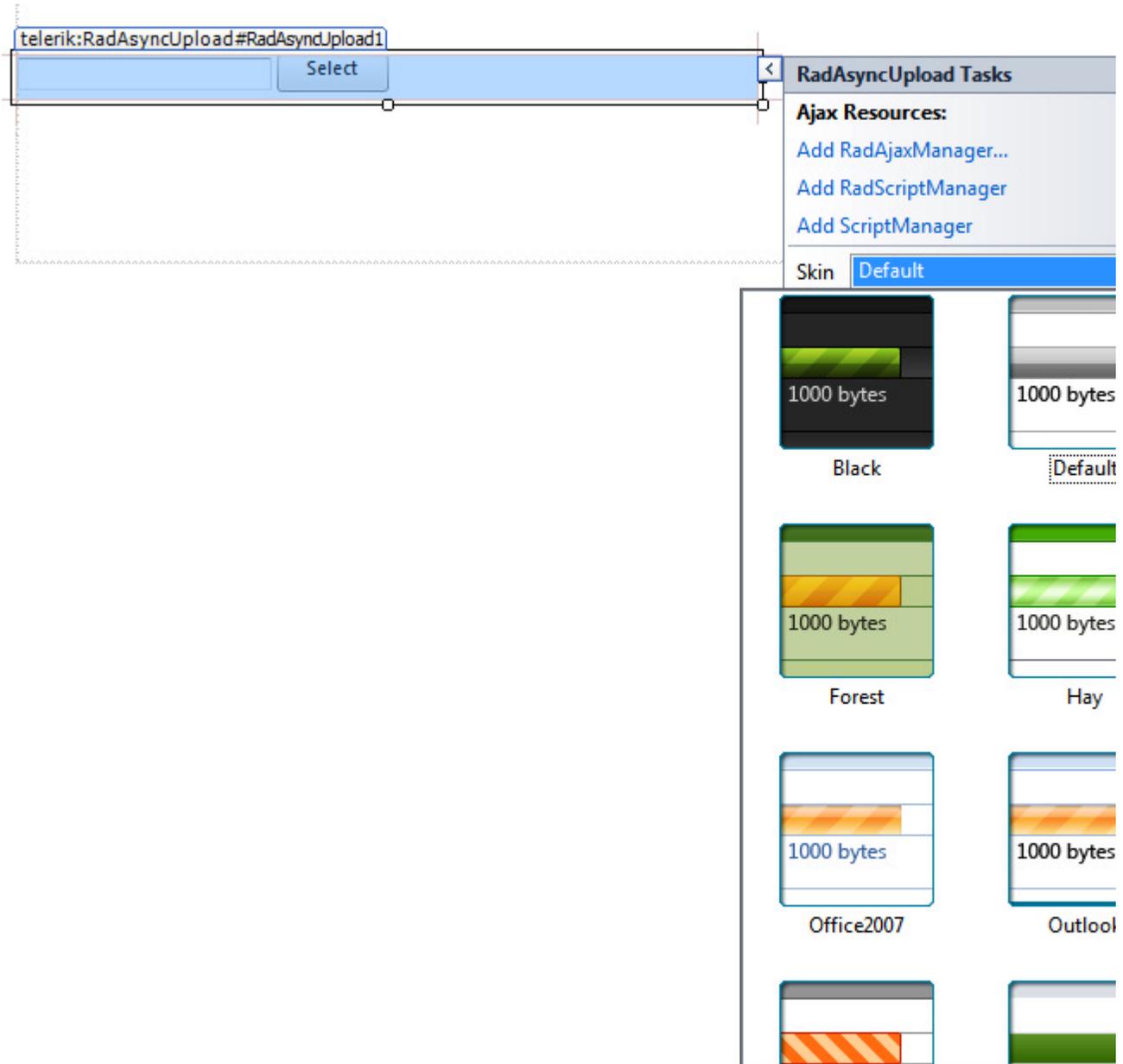


You can find the complete source for this project at:
`\VS Projects\AsyncUpload\GettingStarted`

1. We start by creating a new page and adding a **RadAsyncUpload** control to it.
2. Use the Smart Tag of the control to add **RadScriptManager** on the page:



3. It will automatically register the Telerik.Web.UI.WebResource.axd handler in the web.config file. This handler is used by both RadScriptManager and RadAsyncUpload.
4. You can change the Default skin of the RadAsyncUpload by choosing any of the predefined:



5. Set the **TargetFolder** property to the folder where you want the files to be automatically uploaded after the postback.

 The files are **automatically** and **asynchronously** uploaded to the folder specified by the **TemporaryFolder** property but are copied to the **TargetFolder** after the postback on the page.

By default the TemporaryFolder is set to **App_Data / RadUploadTemp** folder. The files are uploaded with randomly generated unique names.

[ASP.NET] Setting the target folder

```
<telerik:RadAsyncUpload ID="RadAsyncUpload1" TargetFolder="TargetLocation"
    runat="server"></telerik:RadAsyncUpload>
```

6. Finally, add a button on the page which will postback. Run the application and select a file - you will see

the loading image indicating that the file is uploaded asynchronously. During the upload the page is interactive to the user. After clicking on the postback button - the file is saved to the TargetFolder.

Ajax Processing

Here is a sample Web page that uses RadAsyncUpload control and Ajax Processing to upload files.



You can find the complete source for this project at:
\\VS Projects\AsyncUpload\AjaxProcessing



Keep in mind that RadAsyncUpload:

- Files are not directly uploaded to the page, but to a handler - Telerik.Web.UI.WebResource.axd.
- Uploaded files will be transferred to the TargetFolder when a **postback occurs** (meaning that you will need to add a "Submit" button, so that the files will be saved in the target folder).
- The page submission is not automatically blocked until the file upload completes.
- Uploaded files are stored in a temporary location - App_Data/RadUploadTemp by default.
- Windows and Forms Authentication require special handling.

14.4 Important Properties

Important Properties

AllowedFileExtensions	Gets or sets the allowed file extensions for uploading.
EnableInlineProgress	Specifies whether RadAsyncUpload displays an in-line progress next to each file being uploaded.
HttpHandlerUrl	Specifies the URL of the HTTP Handler for which the image will be served.
InitialFileInputsCount	Gets or sets the initial count of file input fields, which will appear in RadAsyncUpload (should be used only when MultipleSelection = "disabled")
InputSize	Gets or sets the size of the input field.
MaxFileInputsCount	Gets or sets maximum file input fields that can be added to the control(should be used only when MultipleSelection = "disabled")
MaxFileSize	Gets or sets the maximum file size allowed for uploading in bytes.
MultipleFileSelection	Specifies whether RadAsyncUpload allows selecting multiple files in the File Selection dialog.
TargetFolder	Gets or sets the virtual path of the folder where RadAsyncUpload will automatically save the valid files after the upload completes.
UploadConfiguration	Sets upload configuration that has additional information.
UploadedFiles	Provides access to the valid files uploaded by the RadAsyncUpload instance.

14.5 Upload Modules

RadAsyncUpload utilizes four different modules for file uploading - **Iframe**, **Flash**, **Silverlight** and **File Api**. The module with highest priority is Silverlight. If there is no Silverlight installed on the client machine, RadAsyncUpload will utilize the Flash module. If neither Flash nor Silverlight is installed - the IFrame module takes place.

Since the three modules are based on entirely different technologies there are slight differences in the approach they handle file uploads. The following information might be useful to developers implementing RadAsyncUpload in their scenarios:

1. How the **IFrame/Flash** module handles file uploads

The IFrame and Flash modules upload the selected file(s) using normal http post request. The iframe uses tag for file uploads whereas Flash uses the Flex FileReference object in order to upload files. The files are uploaded using Post HTTP request in absolutely the same manner as the normal and html form. On the server, there is no difference where you have used the normal upload or the Flash upload in order to upload the files. The files are buffered in the ASP.NET Temporary folder, not in the App_Data/RadUploadTemp folder. After the upload is completed, the files are automatically moved from ASP.NET temp to the Async Upload temporary folder, which is most commonly App_Data/RadUploadTemp. This temp folder can be set by the programmer to any folder on the system. To sum up, as the ASP.NET runtime intercepts the request, it uses the ASP.NET Temp folder in order to assemble the files there, and upon upload completion the latter are moved to the temporary folder.

2. How the **Silverlight** module handles file uploads

In contrast, we have designed the Silverlight upload in a different way. The Silverlight module is designed from scratch to handle very large file uploads; on the client it divides the file to be uploaded in many chunks, each of which is 2mb large. It then starts uploading the chunks one after another and manually assembling them inside our temp folder (not the ASP.NET one). Dividing the file into chunks works around the large file uploads limitation in IIS. The size of the file to upload is only limited by the max file size allowed by the server's operating system. Unfortunately, it is not possible to do that with IFrame/Flash because we have no control of the overall upload process

3. Starting from Q2 2011 RadAsyncUpload introduces the new **File Api** module.

This API is designed to be used in conjunction with other APIs and elements on the web platform, notably: XMLHttpRequest (e.g. with an overloaded send() method for File or Blob objects), postMessage, DataTransfer (part of the drag and drop API defined in [HTML5,]) and Web Workers. Additionally, it should be possible to programmatically obtain a list of files from the input element when it is in the File Upload state[HTML5]. The new module supports the following features:

- Multiple file upload.
- Stand alone progress monitoring (No http module used!).
- Support for the same events as the other modules.
- Upload cancellation.
- Upload files via chunks, effectively walking around the ASP.NET max files size limitation.
- Limitation of the File Api module - No file filtering, i.e. it is not possible to filter the select files dialog with the allowed extensions, because the latter functionality is not yet implemented by any browser. This is the only disadvantage of File API compared to Silverlight.

14.6 Server-Side Programming

The **RadAsyncUpload** control exposes the following server-side event:

FileUploaded occurs when the **RadAsyncUpload** is about to process an uploaded file. If there were multiple files uploaded, the FileUploaded event is going to fire for each and every file. The **FileUploaded** event handler

receives two arguments:

1. The **RadAsyncUpload** control that initiated the file upload. This argument is of type object, but can be cast to the **RadAsyncUpload** type.
2. An **FileUploadedEventArgs** object. It has three properties:
 - **IsValid** Allows you to specify whether the uploaded file is valid. If it is, **RadAsyncUpload** will automatically save it to the **TargetFolder**, if one is set.
 - **UploadedFile** Provides reference to the file uploaded.
 - **UploadResultContainer** object containing information sent from the **RadAsyncUpload** file handler.

The example below demonstrates how to prepare the uploaded file for sending as an e-mail attachment:

[C#] Prepare the uploaded file for sending as an e-mail attachment

```
void RadAsyncUpload1_FileUploaded(object sender, FileUploadedEventArgs e)
{
    e.IsValid = !CheckUploadedFileValidity();
    if (e.IsValid)
    {
        byte[] buffer = new byte[e.File.ContentLength];
        using (Stream str = e.File.InputStream)
        {
            str.Read(buffer, 0, e.File.ContentLength);
            var attachment = createAttachment(buffer);
            // more code
        }
    }
}
```

[VB] Prepare the uploaded file for sending as an e-mail attachment

```
Private Sub RadAsyncUpload1_FileUploaded(ByVal sender As Object, ByVal e As
FileUploadedEventArgs)
    e.IsValid = Not CheckUploadedFileValidity()
    If e.IsValid Then
        Dim buffer As Byte() = New Byte(e.File.ContentLength - 1) {}
        Using str As Stream = e.File.InputStream
            str.Read(buffer, 0, e.File.ContentLength)
            ' more code
            Dim attachment = createAttachment(buffer)
        End Using
    End If
End Sub
```

14.7 Client-Side Programming

RadAsyncUpload provides a flexible client-side API. You can easily interact with the **RadAsyncUpload**, object in the browser using their client-side object. This model lets you achieve tasks while avoiding the post-backs that would trigger file uploading.

Getting the Client-Side Object

RadAsyncUpload creates client-side objects with the **ClientID** of the server-side object. You can obtain a reference using the **\$find()** method, as shown in the following JavaScript code:

[JavaScript] Getting the client-side object

```
var upload = $find("<%= RadAsyncUpload1.ClientID %>");
```

Calling Client-Side Method

Once you have access to a client-side object, you can use it to call its client-side methods, as shown in the following examples:

[ASP.NET] Calling client-side `confirmDeletes()` method

```
<telerik:radupload id="RadUpload1" runat="server"
onclientfileuploadremoving="confirmDeletes"></telerik:radupload>
```

[JavaScript] Defining the client-side `confirmDeletes()` method

```
function confirmDeletes(sender, eventArgs) {
    if (!confirm("Are you sure you want to delete the selected row?"))
        eventArgs.set_cancel(true);
}
```

Client-Side Events

`RadAsyncUpload` support the following client-side events:

- *OnClientAdded* occurs when a row has just been added to the `RadAsyncUpload` control.
- *OnClientFileSelected* occurs when a file is selected in a file input control.
- *OnClientFilesSelected* occurs when files(s) are selected. These event can be cancelled, which will erase the selected files collection.
- *OnClientFileUploading* occurs when a file upload has started uploading.
- *OnClientFileUploaded* occurs when a file has finished uploading.
- *OnClientValidation* occurs when a uploaded file is about to be removed from the uploaded files collection.
- *OnClientFileUploadRemoving* occurs before a selected file is about to be removed from the uploaded files collection. The event can be cancelled.
- *OnClientFileUploadRemoved* occurs when a uploaded file is about is removed from the uploaded files collection.
- *OnClientFileUploadFailed* occurs when a file upload has failed due to an HTTP or server-side error.
- *OnClientProgressUpdating* occurs each time the in-line progress indicator is being updated.

To use these events, simply write a javascript function that can be called when the event occurs. Then assign the name of the javascript function as the value of the the corresponding property (just like the example above).

14.8 Summary

In this chapter you learned about the `RadAsyncUpload` control and became acquainted with some of the powerful features that it provides. You created a simple application using the control, and explored the server-side properties of `RadAsyncUpload`.

Additionally, you examined the client-side properties of `RadAsyncUpload`, which should prove useful in the event that you need to add to the functionality of the control.

Finally, you explored the interaction between `RadAsyncUpload` and various modules

15 RadComboBox

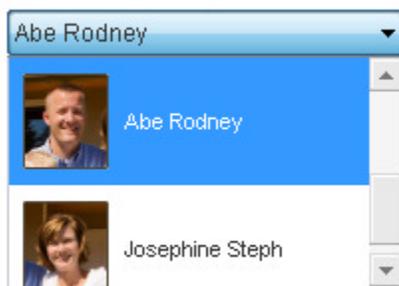
15.1 Objectives

- Explore the features of the **RadComboBox** control.
- Create a simple application to get confidence in using the combo box and to see the difference between static items and data-bound items.
- Explore the combo box design time interface, including Smart Tag, Properties Window, Property Builder, and Template Design surface.
- Explore principal properties and groups of properties where most of the functionality is found.
- Learn to use special combo-box features such as templates, custom attributes, and load-on-demand.
- Learn server-side coding techniques, including an exploration of how to work with the items collection, an examination of important server-side events, and a look at the properties and methods for sorting the drop-down list.
- Explore some of the client-side methods for working with the items collection and some of the client-side events, with special attention to the events you can use with the load-on-demand feature.
- Explore some advanced techniques, such as implementing custom sort criteria, adding input controls to an item template, and enabling virtual scrolling without allowing custom text to be entered, and implementing a Web service to supply items on demand.

15.2 Introduction

You are probably already familiar with the ASP.NET **DropDownList** control. It lets users select options from a drop-down list of items. The **RadComboBox** control is a similar control, but it gives you far more power and flexibility.

Like all RadControls, RadComboBox lets you assign a **Skin** to instantly change the appearance to one that harmonizes with the overall look-and-feel of your Web page. Your control over the look-and-feel does not stop there. You can also add images to the items in the drop-down list, or even use templates for complete control over the appearance of items. You can even add animated effects to the way the list drops down and closes back up.



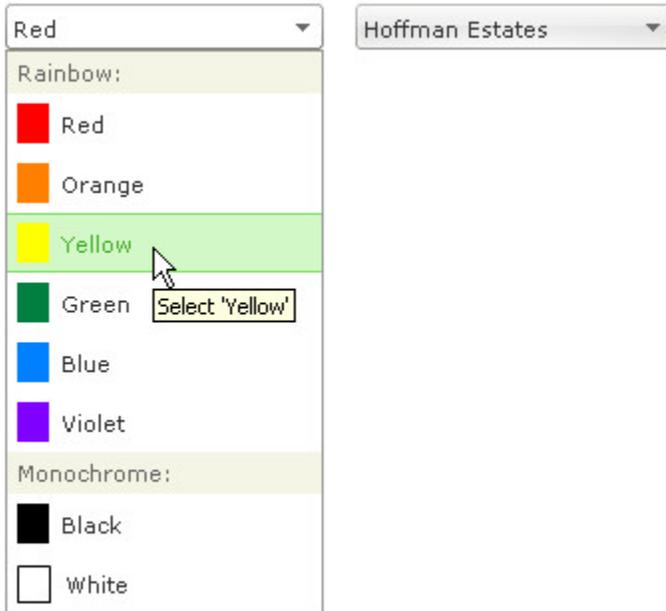
Unlike the ASP.NET **DropDownList** control, which restricts users to selecting only items from the list, **RadComboBox** can optionally allow users to type in entries that are not in the list. When the combo box is configured to let users enter custom text in this manner, you can provide an "empty" message to the drop-down list, similar to the empty message that you saw on the input RadControls.

If you need to work with a very long list of items, RadComboBox can help users navigate that list with its auto-complete feature, which automatically scrolls the list and highlights the first item that matches the currently entered text. Alternately, you can use the filtering feature, which limits the list to items that contain the currently entered text, and highlights the matching text. You can even configure the combo box to load items

on demand, so that your application does not need to download all items at once.

15.3 Getting Started

In this walk-through you will become familiar with the **RadComboBox** control. You will create two combo boxes: one with statically declared items, and one with items loaded from a database.



You can find the complete source for this project at:
 \VS Projects\ComboBox\GettingStarted

Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Using the Solution Explorer, add a new Folder to your project and name it "Images".
3. Drag the contents of the "\VS Projects\Images\Colors" folder into your project's "Images" folder.
4. Locate the "Northwind.mdf" file in the "Live Demos\App_Data" folder under the folder where you installed RadControls for ASPNET AJAX. Drag this file into the "App_Data" folder of your project.
5. Open the "Web.config" file of your project. Add the standard Northwind connection string to your project by replacing the line


```
<connectionStrings />
```

 with


```
<connectionStrings>
  <add name="NorthwindConnectionString" connectionString="Data
  Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|Northwind.mdf;Integrated
  Security=True;User Instance=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Add a combo box with statically declared items

1. Drag a **RadComboBox** control from the Tool Box onto your Web page. Set its **Skin** property to "Telerik".

- From the RadComboBox Smart Tag, choose **Build RadComboBox...** to bring up the **RadComboBox Item Builder**.
- In the Item Builder, click the "Add Item" button () to add a new item to the combo box.
- Using the properties pane on the right,
 - Set the **Text** property to "Rainbow:".
 - Set the **IsSeparator** property to true.
- Click the "Add Item" button again to add a second item. Set its properties as follows:
 - Set the **Text** property to "Red".
 - Set the **ToolTip** property to "Select 'Red'".
 - Set the **ImageUrl** property to "~/Images/Red.png".
- Add another item and set its properties as follows:
 - Set the **Text** property to "Orange".
 - Set the **ToolTip** property to "Select 'Orange'".
 - Set the **ImageUrl** property to "~/Images/Orange.png".
- Repeat this process for four more buttons, using the colors "Yellow", "Green", "Blue", and "Violet".
- Add another item and set its **Text** property to "Monochrome:" and its **IsSeparator** property to true.
- Add another item and set its properties as follows:
 - Set the **Text** property to "Black".
 - Set the **ToolTip** property to "Select 'Black'".
 - Set the **ImageUrl** property to "~/Images/Black.png".
- Add a last "color" item, using the color "White".
- Click OK to exit the Item Builder.

Add a data-bound combo box

- Drag a second **RadComboBox** from the Tool Box onto your Web page. Set its **Skin** property to "Telerik".
- In the RadComboBox Smart Tag, select "<New data source...>" from the **Choose Data Source** drop-down.
- In the first page of the **Data Source Configuration Wizard**, select "Database" as the application type, and click **OK** to move to the next page.
- On the **Choose Your Data Connection** page, select "NorthwindConnectionString" from the drop-down list. Then click the **Next** button to continue.
- On the **Configure the Select Statement** page, make sure the "Specify columns from a table or view" radio button is selected, and then choose "Territories" from the "Name" drop-down list.
- Check all three fields of the Territories table, and then click the **Where** button to add a WHERE clause to the SELECT statement.
- Select the "RegionID" column from the **Column** drop-down, leave the operator as "=", and select "None" from the **Source** drop down. In **Parameter properties** on the right, set the **Value** to "2".
- Click the **Add** button to add the Where clause, and then Click **OK**.
- Back on the **Configure the Select Statement** page, click the **Next** button, test the query if you wish, and then click **Finish**.
- In the Properties Window for the second combo box,

- Set the `DataTextField` property to "TerritoryDescription".
- Set the `DataValueField` property to "TerritoryID".

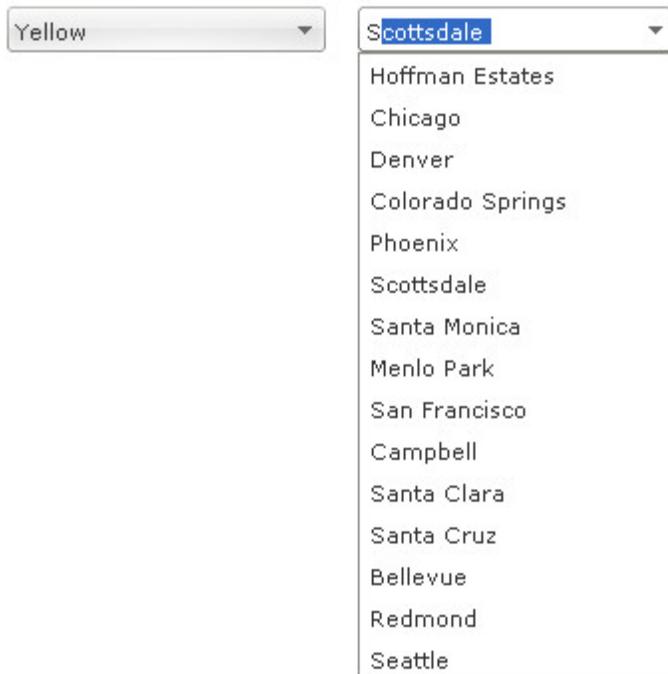
The combo box is now bound to the data source.

11. While you are in the Properties Window, set a few more properties:

- Expand the `ExpandAnimation` property and set the `Type` sub-property to "OutBounce".
- Expand the `CollapseAnimation` property and set the `Type` sub-property to "InBounce".
- Set the `MarkFirstMatch` property to true.

Run the application

1. Press Ctrl-F5 to run the application.
2. Click the drop-down arrow to expand the first combo box. Note the images next to items, the tool tips when you hover the mouse over items, and the non-selectable separator items.
3. Expand the second combo box. Note the animated effect you added. Type an "S" in the text area and notice how the `MarkFirstMatch` property causes the text to change, with the unentered portion selected.

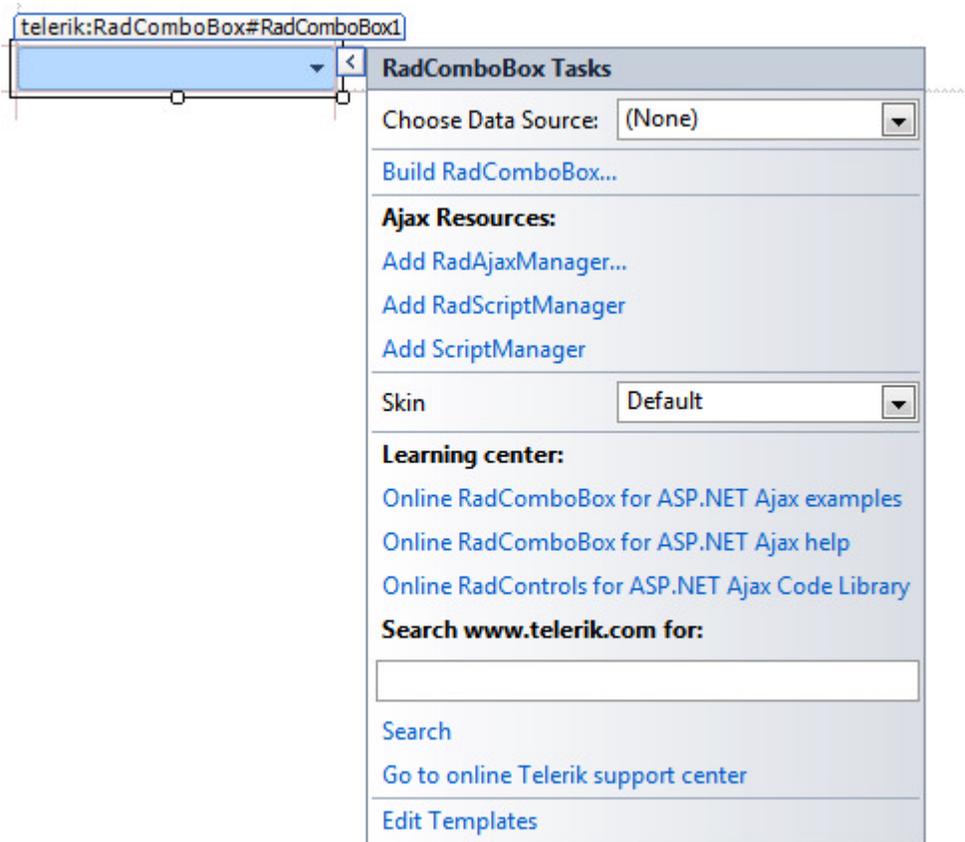


15.4 Designer Interface

In the Visual Studio designer, you can configure the `RadComboBox` control using the Smart Tag, the Properties Window, and the `RadComboBox` Item Builder. In addition, you can add templates using the Template Design surface.

Smart Tag

The `RadComboBox` Smart Tag contains a few control-specific entries in addition to the standard Ajax Resources, Skin selection, and Learning center sections.



At the top of the Smart Tag, the **Choose Data Source** drop-down lets you bind the combo box to a data source control already on the Web page, or launch the **Data Source Configuration Wizard** to create and configure a new data source control. Once the combo box is bound to a data source, the Smart Tag changes to include **Configure Data Source** and **Refresh Schema** links beneath the Choose Data Source drop-down.

Below the Choose Data Source drop-down, the **Build RadComboBox...** link brings up the **RadComboBox Item Builder**, where you can add statically declared items to the combo box.

At the bottom of the Smart Tag, the **Edit Templates** link lets you bring up the **Template Design Surface**, where you can design a template for combo box items.

Properties Window

At design time, you can use the Properties Window to configure almost every aspect of the combo box. (A notable exception is the creation of templates.) As with the other controls we have seen, let us look at the most important properties of the combo box.

Specifying Items

Probably the most important property of the combo box is the one that specifies what items appear in the drop-down list. What property you choose for this task depends on whether you want to load items from a database:

- If you want to load items from a database, RadComboBox supports the standard data-binding properties (**DataSourceID** and **DataMember**), which you have already seen in the chapter on Data Binding. That chapter also introduced you to the **AppendDataBoundItems** property, which lets you combine data-bound items with statically declared items, and the alternate approach to binding of using the **DataSource** property and the **DataBind** method in the code-behind. When binding RadComboBox to a data source, use

the **DataTextField**, **DataTextFormatString**, and **DataValueField** properties to map records from the data source to properties of the combo box items.

- If you want to use statically declared items, you can use the **Items** property to bring up the **RadComboBox Item Builder**. The Item Builder is described in more detail below.

Specifying the behavior of the text area

The **AllowCustomText** property specifies whether the user can enter text into the text area of the combo box that is not in the drop-down list. When **AllowCustomText** is true, you can use the **EmptyMessage** property to specify a prompt string when there is no text assigned. This behaves like the **EmptyMessage** property you saw on the input controls such as **RadTextBox**.

The **MarkFirstMatch** property turns on the auto-complete feature of the combo box. You already saw this feature briefly in the Getting Started project. **MarkFirstMatch** interacts with the **AllowCustomText** property as follows:

- When **MarkFirstMatch** is false, the **AllowCustomText** property has a major effect on the behavior of the combo box. When **AllowCustomText** is false, typing a character in the text box selects the first item from the drop-down list that starts with the entered character. If another item starts with the same character, it can be selected by typing the character a second time. When **AllowCustomText** is true, typing in the text area selects exactly the text that is typed, regardless of whether it matches a string in the list.
- When **MarkFirstMatch** is true, typing a character in the text box always selects the first item from the drop-down list that matches the character. However, the text area remains editable, so that more characters can be typed, further limiting the number of strings that match. If **AllowCustomText** is false, any characters that do not match a string in the list are ignored. If **AllowCustomText** is true, entering a character that does not match any items in the list causes the combo box to behave in the same way as when the auto-complete feature is not turned on.

When **MarkFirstMatch** is true, two other properties influence the behavior of the auto-complete feature.

- **AutoCompleteSeparator** enables the user to select multiple items from the list. It specifies the character that separates list items. Typically, **AutoCompleteSeparator** is set to a character such as "," or ";". After the user enters the separator character, the next character typed begins a new selected item, and the list items are matched to the text that follows the separator.
- **EnableTextSelection** controls whether the unentered text of the matched list item is selected when the combo box adds it to the edit box as a result of the auto-complete feature. When **EnableTextSelection** is true, the unentered text is selected. This is most useful for a single-selection combo box (with no **AutoCompleteSeparator**), because the next character that the user types simply refines the search. When **EnableTextSelection** is false, the rest of the matched list item is not selected, and the cursor appears at the end of the string. This is most useful for a multi-selection combo box, as the cursor is placed conveniently for typing the separator character.

Another feature that is similar to the auto-complete feature is the use of filters. You can turn on filtering by setting the **Filter** property to "StartsWith" or "Contains". When **Filter** is "StartsWith", typing in the text area causes the list to filter out all items that do not start with the string in the text area. The portion of list items that matches the string in the text area is highlighted. When **Filter** is "Contains", the combo box uses a broader criterion for matching the string in the text area. If the entered text falls anywhere within a list item, it is kept in the list. Once again, the matching text is highlighted.

✎ Unlike the **MarkFirstMatch** property, which uses the **IsCaseSensitive** property to determine whether to match items in a case-sensitive manner, the **Filter** property is always case-insensitive.

Specifying the behavior of the drop-down list

You can add animated effects to the way the drop-down list opens and closes by setting the **ExpandAnimation** and **CollapseAnimation** properties. Both of these properties have two sub-properties: **Type**, which identifies the desired animated effect, and **Duration**, which specifies how long, in milliseconds, the effect lasts.

By default, the drop-down list opens when the user clicks in the text area or on the drop-down arrow and closes

when the user selects an item in the list, clicks the drop-down arrow a second time or moves focus to another control on the page. The **ShowDropDownOnTextBoxClick** and **CloseDropDownOnBlur** properties let you change these defaults by restricting the drop-down opening to the use of the arrow and allowing the list to remain open when another control gets focus.

The **ChangeTextOnKeyboardNavigation** property specifies whether traversing the list using the arrow keys changes the selected item, or whether the user must use the mouse or Enter key to select an item.

Specifying appearance

As with all RadControls, you can use the **Skin** property to change the overall look of the combo box. In addition, a number of properties influence the layout of combo box parts. The **Width** property specifies the width of the text area plus drop-down arrow, while the **DropDownWidth** specifies the width of the drop-down list. The **OffsetX** and **OffsetY** properties let you control the position of the drop-down list. The **Height** property lets you control the height of the drop-down list.

💡 When you use the **Filter** feature, it is a good idea to explicitly set the **Height** property. Otherwise, the drop-down list will be sized based on its contents the first time it opens, and if that is a small filtered list, the combo box can become hard to use.

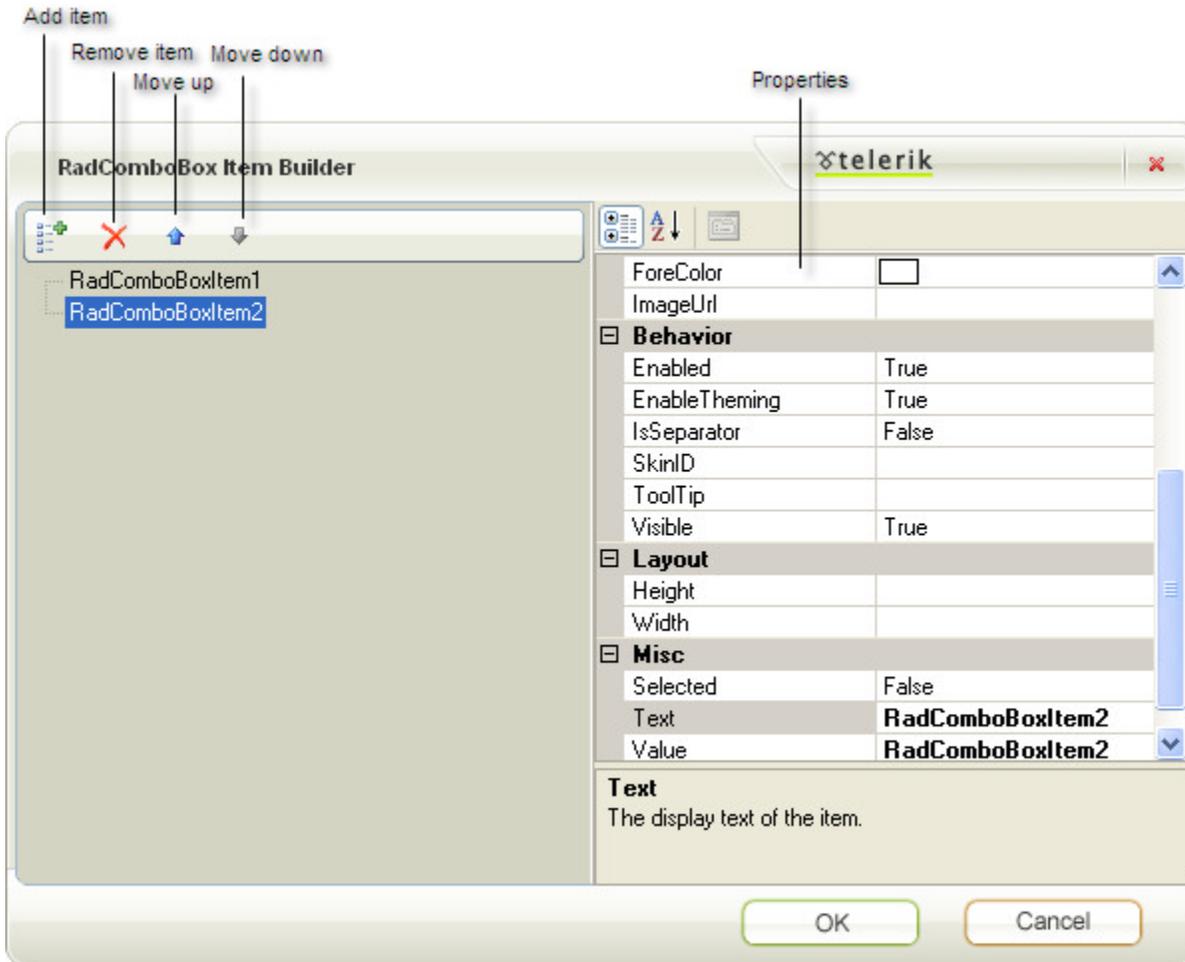
You can hide the drop-down arrow by setting the **ShowToggleImage** property to false. When hiding the drop-down arrow, be sure that the **ShowDropDownOnTextBoxClick** property is true, or the user will not be able to open the drop-down list! You can move the drop-down arrow so that it appears to the left of the text box by changing the **RadComboBoxImagePosition** property to "Left".

💡 To configure the combo box for a right-to-left locale, don't bother using the **RadComboBoxImagePosition** property. Instead, add the **dir="rtl"** attribute to the combo box. The **dir="rtl"** attribute moves the position of item text and images as well, and even moves punctuation characters in the item text to the other side.

RadComboBox Item Builder

RadComboBox lets you edit the list of statically defined items using the **RadComboBox Item Builder**. This item builder is very similar to the Property Builder dialogs you looked at in the chapter on Navigation controls, or, for that matter, to the Property Builder dialogs for any of the controls that maintain a static collection of items. Display the item builder either from the Smart Tag or by clicking the ellipsis button on the **Items** property in the Properties Window.

Below is a screen shot of the RadComboBox Item Builder. Use the buttons on the upper left to add items to the drop-down list or to remove or reposition the selected item. To edit the text of an item in-line, select it with the mouse, then click it a second time. You can select any of the items and set item properties using the properties pane on the right of the dialog. Typically, you will set the **Text** property first.



Each item has its own set of properties: **Text** is the string that displays in the drop-down list, **IsSeparator** specifies whether the user can select the item, **ImageUrl** is the path to an image file that will display next to the **Text**, and **DisabledImageUrl** is the path to an image file to use when the item is disabled. These four properties control the basic appearance of the list items. You may also want to set the **ToolTip** property to assign a tool tip for the item, or use the **Selected** and **Enabled** properties to specify the state of the item when the Web page first loads. Another common property is the **Value** property, which associates a value with the item that you can then use when programming with the combo box.

Template Design surface

You can use the Smart Tag or context menu to bring up the Template Design surface, where you can create an item template. The item template applies to all items in the combo box, and lets you customize the appearance of items for even more control than when using the item properties.

We will look at combo box templates in more detail in the next section (Control Specifics).

15.5 Control Specifics

Templates

RadComboBox supports three types of template: **ItemTemplate**, **HeaderTemplate**, and **FooterTemplate**. The item template is used for displaying each item in the drop-down list, while the header template and footer template appear at the top and bottom of the drop-down list.

Using Templates to create a multi-column combo box

The following example uses an item template with controls that are bound to the **DataItem** of each combo box item. In addition to the item template, it also shows the use of header and footer templates.

The combo box in this example displays items in a multi-column format. The header template labels the columns of the drop-down list. The footer template gives the text associated with the last selected item. (The footer text matches the text area of the combo box unless the user navigates the drop-down list using the arrow keys.)

Dr. Andrew Fuller (Vice President, Sales)		
Last Name	First Name	Title
Davolio	Nancy	Sales Representative
Fuller	Andrew	Vice President, Sales
Leverling	Janet	Sales Representative
Peacock	Margaret	Sales Representative
Buchanan	Steven	Sales Manager
Suyama	Michael	Sales Representative
King	Robert	Sales Representative
Callahan	Laura	Inside Sales Coordinator
Dodsworth	Anne	Sales Representative
Mr. Michael Suyama (Sales Representative)		



You can find the complete source for this project at:
\\VS Projects\ComboBox\Templates

Both the **HeaderTemplate** and the **ItemTemplate** are formatted using a `<table>` with fixed width columns. This way, the header labels line up with the columns in the drop-down list. The **FooterTemplate** starts out empty, as the text it displays is populated in client-side code.

[ASP.NET] ComboBox with templates

```
<telerik:RadComboBox ID="RadComboBox1" Runat="server"
  DropDownWidth="350px" Width="250px" Skin="Outlook"
  DataSourceID="SqlDataSource1" HighlightTemplatedItems="True"
  onitemdatabound="RadComboBox1_ItemDataBound"
  onclientslectedindexchanged="IndexChanged"
  onclientload="InitComboFooter" >
  <HeaderTemplate>
    <table style="width: 315px; text-align: left">
      <tr>
        <td style="width: 95px;">Last Name</td>
        <td style="width: 95px;">First Name</td>
        <td style="width: 125px;">Title</td>
      </tr>
    </table>
  </HeaderTemplate>
```

```

<FooterTemplate>
</FooterTemplate>
<ItemTemplate>
  <table style="width: 315px; text-align: left">
    <tr>
      <td style="width: 95px;">
        <#%# DataBinder.Eval(Container.DataItem, "LastName") %>
      </td>
      <td style="width: 95px;">
        <#%# DataBinder.Eval(Container.DataItem, "FirstName") %>
      </td>
      <td style="width: 125px;">
        <#%# DataBinder.Eval(Container.DataItem, "Title") %>
      </td>
    </tr>
  </table>
</ItemTemplate>
</telerik:RadComboBox>

```

You may have noticed in the declaration above that the combo box has three event handlers, a server-side **ItemDataBound** event handler, and two client-side event handlers (**OnClientLoad** and **OnClientSelectedIndexChanged**).

The server-side **ItemDataBound** event handler is used to combine the fields from each data item and use them to set the **Text** property of the item:

[VB] Setting the Text property in ItemDataBound

```

Protected Sub RadComboBox1_ItemDataBound(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadComboBoxItemEventArgs) Handles RadComboBox1.ItemDataBound
  'Use the data item to set the combo box item's Text
  e.Item.Text = (DirectCast(e.Item.DataItem, DataRowView))("TitleOfCourtesy").ToString
().Trim() _
  + " " + (DirectCast(e.Item.DataItem, DataRowView))("FirstName").ToString().Trim() _
  + " " + (DirectCast(e.Item.DataItem, DataRowView))("LastName").ToString().Trim() _
  + " (" + (DirectCast(e.Item.DataItem, DataRowView))("Title").ToString().Trim() + ")"
End Sub

```

[C#] Setting the Text property in ItemDataBound

```

protected void RadComboBox1_ItemDataBound(object sender, RadComboBoxItemEventArgs e)
{
  // Use the data item to set the combo box item's Text
  e.Item.Text = ((DataRowView)e.Item.DataItem)["TitleOfCourtesy"].ToString().Trim() + " " +
  ((DataRowView)e.Item.DataItem)["FirstName"].ToString().Trim() + " " +
  ((DataRowView)e.Item.DataItem)["LastName"].ToString().Trim() + " (" +
  ((DataRowView)e.Item.DataItem)["Title"].ToString().Trim() + ")";
}

```

The two client-side event handlers set the text of the footer to the text of the selected item. The **OnClientLoad** handler ("InitComboFooter") initializes the footer so that it displays the text of the combo box. The **OnClientSelectedIndexChanged** ("IndexChanged") updates the footer when an item is selected.

[JavaScript] Updating the footer

```

function InitComboFooter(sender) {
  // get a reference to the footer DOM element
  var footer = sender._getFooterElement();
  // set its innerHTML to the initial text of the combo box
  footer.innerHTML = sender.get_text();
}

```

```
}  
function IndexChanged(sender, args) {  
    // get a reference to the footer DOM element  
    var footer = sender._getFooterElement();  
    // set its innerHTML to the selected item text  
    footer.innerHTML = args.get_item().get_text();  
}
```

Using ItemTemplate to replace the drop-down list with a control

In the last example, the item template was used to format information associated with each item in the combo box. Some controls that you might want to include in an item template, however, represent all the possible choices. Such controls include calendars, color pickers, tree views, or various custom controls. When including such a control in the item template, you do not want a list that repeats the control. Rather, you want to replace the list with a single embedded control. The following walk-through illustrates how this can be accomplished.

This walk-through shows how to use the ItemTemplate to replace the drop-down list with a RadCalendar control. It accomplishes this by using a single statically-declared item.



You can find the complete source for this project at:
\\VS Projects\ComboBox\TemplateControl

1. Create a Web Application and add a ScriptManager to the default page.
2. Add a **RadComboBox** to the default page.
 - o Set the **Skin** property to "Hay".
 - o Set the **ShowDropDownOnTextboxClick** property to false.
 - o Set the **DropDownWidth** property to "230px".
3. Bring up the **RadComboBox Item Builder** and add a single item with its **Text** property set to an empty string.
4. Click on the **Edit Templates** link in the Smart Tag to bring up the Template Design Surface.
5. Drag a **RadCalendar** control from the Tool Box onto the Template Design Surface.
 - o Set the **AutoPostBack** property to true
 - o Set the **EnableMultiSelect** property to false.

- Set the **ShowRowHeaders** property to false.
 - Set the **TitleFormat** property to "MMM yyyy".
6. Go to the code-behind for the default page. At the top of the page, in addition to an **Imports** or **using** statement for "Telerik.Web.UI", add one for "Telerik.Web.UI.Calendar". Then add the following method:

[VB] SelectionChanged

```
Protected Sub RadCalendar1_SelectionChanged(ByVal sender As Object, ByVal e As SelectedDatesEventArgs)
    Dim calendar As RadCalendar = DirectCast(sender, RadCalendar)
    'Update the combo box item Text
    'The combo box will then assign its value accordingly
    RadComboBox1.SelectedItem.Text = [String].Format("{0}/{1}/{2}",
calendar.SelectedDate.Month, calendar.SelectedDate.Day, calendar.SelectedDate.Year)
End Sub
```

[C#] SelectionChanged

```
protected void RadCalendar1_SelectionChanged(object sender,
Telerik.Web.UI.Calendar.SelectedDatesEventArgs e)
{
    RadCalendar calendar = (RadCalendar)sender;
    // Update the combo box item Text
    // The combo box will then assign its value accordingly
    RadComboBox1.SelectedItem.Text = String.Format("{0}/{1}/{2}", calendar.SelectedDate.Mo
calendar.SelectedDate.Day, calendar.SelectedDate.Year);
}
```

Note that this event handler assigns the **Text** property of the single combo box item.

7. Back in the Design window for the Web page, assign the method you just created as the **SelectionChanged** event handler of the **RadCalendar** control in the template, and then end template editing.
8. Press **Ctrl-F5** to run the application. When you expand the drop-down list, the combo box displays the calendar, and when you select a date, the selected text of the combo box updates to the value you assigned in the event handler. However, you will also notice that there are some rough edges. Before the **SelectionChanged** event handler sets the text, a different value sometimes appears, and there is a jarring disruption when the page reloads on the postback. Shut down the running application so that we can fix these irritations.
9. Bring up the Template Design Surface again, and on the **RadCalendar** control's Properties Window, expand the **ClientEvents** property. Set the **OnDateSelecting** sub-property to "OnDateSelecting" and the **OnDateClick** sub-property to "OnDateClick". Then end template editing.
10. Switch to the Source window, and add the following script block to the form.

[ASP.NET] Calendar client-side events

```
<script type="text/javascript">
// prevent the click from propagating up
// to the combo box item
function OnDateClick(sender, args) {
    args.get_domEvent().stopPropagation();
}
// Before the calendar causes a postback
// call attachDropDown so that the postback
// can be converted to a callback
function OnDateSelecting(sender, args) {
    if (args.get_isSelecting()) {
        var combo = $find("RadComboBox1");
        combo.attachDropDown();
    }
}
```

```
}  
else // just cancel de-selection of the old date  
    args.set_cancel(true);  
}  
</script>
```

The first function ("OnDateClick") prevents the click event from bubbling up to the combo box. This prevents the effect where you saw a different value in the combo box text box before the server-side event handler updated the text. The second function ("OnDateSelecting") calls the client-side `attachDropDown()` method before the calendar generates a postback.



Gotcha! The `attachDropDown()` method is required to allow postbacks initiated by any controls inside the item template of a combo box to be converted into asynchronous AJAX callbacks.



As an alternate approach to using the `attachDropDown()` method, you can use the client-side events of the calendar to update the combo box text rather than the server-side `SelectionChanged` event.

- Return to the Design view. Drag a **RadAjaxManager** from the Tool Box onto the Web page. Configure the AJAX manager so that requests can be initiated by the combo box item. Set the combo box as the updated control.

[ASP.NET] RadAjaxManager

```
<telerik:RadAjaxManager ID="RadAjaxManager1" runat="server">  
  <AjaxSettings>  
    <telerik:AjaxSetting AjaxControlID="i0">  
      <UpdatedControls>  
        <telerik:AjaxUpdatedControl ControlID="RadComboBox1" />  
      </UpdatedControls>  
    </telerik:AjaxSetting>  
  </AjaxSettings>  
</telerik:RadAjaxManager>
```

- Press Ctrl-F5 to run the application again. Now, when you select a date in the calendar, the combo box updates smoothly.

Custom Attributes

`RadComboBox` supports the use of custom attributes. You have already seen custom attributes in the chapter on Data Binding, and saw how they could be set in an `ItemDataBound` event handler. You can also set custom attributes declaratively when using statically-declared items.

The following example illustrates using declarative custom attributes with a combo box. The custom attributes are used to bind elements in an item template.





You can find the complete source for this project at:
 \VS Projects\ComboBox\CustomAttributes

The Combo box declares its items statically. The ItemTemplate includes a <div> with its background color set to the "Color" custom attribute of each item.

[ASP.NET] Combo box with custom attributes

```
<telerik:RadComboBox ID="RadComboBox1" Runat="server"
Skin="Office2007" >
<Items>
  <telerik:RadComboBoxItem runat="server"
    Text="Red" Value="Red" Color="#ff3333" />
  <telerik:RadComboBoxItem runat="server"
    Text="Orange" Value="Orange" Color="#ff9933" />
  <telerik:RadComboBoxItem runat="server"
    Text="Yellow" Value="Yellow" Color="#ffff33" />
  <telerik:RadComboBoxItem runat="server"
    Text="Green" Value="Green" Color="#33cc66" />
  <telerik:RadComboBoxItem runat="server"
    Text="Blue" Value="Blue" Color="#0099ff" />
  <telerik:RadComboBoxItem runat="server"
    Text="Violet" Value="Violet" Color="#9900ff" />
</Items>
<ItemTemplate>
  <div style='background-color:<%=# DataBinder.Eval(Container, "Attributes['Color']") %>;
height: 20px;' >
    <%=# DataBinder.Eval(Container, "Text") %>
  </div>
</ItemTemplate>
</telerik:RadComboBox>
```

In the code-behind, the **Page_Load** event handler calls the **DataBind** method for each item:

[VB] Calling DataBind on Page_Load

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
  Dim i As Integer = 0
  'call DataBind for each combo box item
  'so that template controls can access its properties
  While i < RadComboBox1.Items.Count
    RadComboBox1.Items(i).DataBind()
    i = i + 1
  End While
End Sub
```

[C#] Calling DataBind on Page_Load

```
protected void Page_Load(object sender, EventArgs e)
{
  // call DataBind for each combo box item
  // so that template controls can access its properties
  for (int i = 0; i < RadComboBox1.Items.Count; i++)
  {
    RadComboBox1.Items[i].DataBind();
  }
}
```

Gotcha! When using an item template with statically declared items, always call the **DataBind()**



method for each item when the page loads. Otherwise, the template controls cannot access the item properties.

Load-on-demand

When the data source for a combo box includes a very large number of items, it may be impractical to display them all at once. For one thing, too many entries make it hard for the user to find the correct choice, and for another, loading all of those elements at once can hurt performance. The first issue may be solved by using filters, but the only way to solve the second is to load items only as they are needed.

To enable the combo box to load items only as they are needed, set the **EnableLoadOnDemand** property to true. When load-on-demand is enabled, the combo box always allows the user to enter custom text, regardless of the value of the **AllowCustomText** property. This is necessary so that the user can enter text that does not appear in the drop-down list, which then causes the combo box to load matching items.



Gotcha! When **EnableLoadOnDemand** is true, do not try to read the **Items** property of the combo box in the code-behind. Due to performance issues, the **Items** property is not updated to reflect items loaded on demand.

The load-on-demand mechanism works as follows:

1. The user types in the text area of the combo box or clicks on the drop-down arrow.
2. In response, the combo box automatically generates an AJAX callback to request a new list. The new list of items can be supplied either by a server-side **ItemsRequested** event handler or by a Web Service that is identified by the **WebServiceSettings** property.
3. The drop-down list opens, displaying the new list of items.

One important thing to notice here is that the load-on-demand mechanism uses an AJAX callback, not a postback. If the **ItemsRequested** event handler makes any changes to controls on the Web page (other than the combo box list of items), they are immediately lost, because they are not identified as updated controls for the callback. Another consequence is that any code in the **Page_Load** event handler that is protected by a check of the **Page.IsPostBack** property will execute every time the combo box requests a new list of items, since callbacks do not change the **IsPostBack** property.

💡 To prevent code in the **Page_Load** event handler from executing every time the combo box requests a new list of items, check the **Page.IsCallback** property as well as the **Page.IsPostBack** property.

Loading matching items

The next example illustrates the use of the "Load on demand" mechanism. When the user types in the text area of the combo box, the combo box automatically clears the drop-down list. New items are added by an **ItemsRequested** event handler.

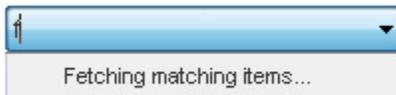




You can find the complete source for this project at:
 \VS Projects\ComboBox\LoadOnDemand

To enable load-on-demand, the RadComboBox control has its **EnableLoadOnDemand** property set to true. In addition, the **EnableItemCaching** property is set to true. When item caching is enabled, the combo box caches the item lists that are returned after each item request. The next time that the same text is entered in the text area (for example if the user types some text and then hits the Backspace key), the combo box does not generate a new callback; instead, it reloads the list of items from the cache. This reduces Web traffic, which can improve performance if the list of items is very large or the user types the same string repeatedly.

Another property set on the combo box is the **LoadingMessage** property. This is the message that appears in the drop-down list while the combo box is waiting for the list of items to appear. Because item caching is turned on, this message only appears the first time any given string is typed in the combo box.



The **ItemsRequested** event handler checks whether the combo box contains any text, and if so, it generates a database query to retrieve matching items and adds them to the combo box. In this example, checking for an empty text string is probably not that important, but in cases where the number of items is huge, you would probably not want to handle the request that fetches the entire data set.

✎ After the **ItemsRequested** event handler creates new items and adds them to the **Items** property of the combo box, it causes the thread to sleep for 100 milliseconds less than the item request timeout period. Obviously you would not want to do this in a real application, but this example inserts the command so that you can see the effects of item caching.

[VB] Adding items in the ItemsRequested event handler

```
Protected Sub RadComboBox1_ItemsRequested(ByVal o As Object, ByVal e As
RadComboBoxItemsRequestedEventArgs)
    ' only add items if there is text to match
    If e.Text = [String].Empty Then
        ' open a connection to the database
        Dim dbCon As New SqlConnection(NWConnectionString)
        dbCon.Open()
        ' the query uses LIKE to match the entered text
        Dim sql As String = "SELECT * from Customers WHERE CompanyName LIKE '" + e.Text + "%'"
        ' create a data adapter and use it to fetch data into a table
        Dim adapter As New SqlDataAdapter(sql, dbCon)
        Dim dt As New DataTable()
        adapter.Fill(dt)
        dbCon.Close()
        ' use the table to add items to the combo box
        For Each row As DataRow In dt.Rows
            Dim item As New RadComboBoxItem(row("CompanyName").ToString())
            RadComboBox1.Items.Add(item)
        Next
        'Simulate a lengthy process so that the effects of caching can be seen
        Threading.Thread.Sleep(RadComboBox1.ItemRequestTimeout - 100)
    End If
End Sub
```

[C#] Adding items in the ItemsRequested event handler

```
protected void RadComboBox1_ItemsRequested(object o, RadComboBoxItemsRequestedEventArgs e)
{
```

```
// only add items if there is text to match
if (e.Text == String.Empty)
{
    // open a connection to the database
    SqlConnection dbCon = new SqlConnection(NWConnectionString);
    dbCon.Open();
    // the query uses LIKE to match the entered text
    string sql = "SELECT * from Customers WHERE CompanyName LIKE '" + e.Text + "%'";
    // create a data adapter and use it to fetch data into a table
    SqlDataAdapter adapter = new SqlDataAdapter(sql, dbCon);
    DataTable dt = new DataTable();
    adapter.Fill(dt);
    dbCon.Close();
    // use the table to add items to the combo box
    foreach (DataRow row in dt.Rows)
    {
        RadComboBoxItem item = new RadComboBoxItem(row["CompanyName"].ToString());
        RadComboBox1.Items.Add(item);
    }
    // Simulate a lengthy process so that the effects of caching can be seen
    System.Threading.Thread.Sleep(RadComboBox1.ItemRequestTimeout - 100);
}
}
```

Virtual Scrolling and Show More Results box

Another use of the load-on-demand mechanism is to introduce virtual scrolling. Virtual scrolling is when the combo box only loads a subset of its items each time an item request is made. If the user scrolls to the bottom of the drop-down list, another callback is made to fetch more items.

As an alternative to virtual scrolling (or in addition to it), the combo box can display a footer at the bottom of the drop-down list that includes a link for fetching additional items. This footer is called the "Show More Results" box.



- When Virtual Scrolling or the Show More Results box is enabled, the combo box does **not** clear the items list before generating a callback to request items. This is in contrast to the way load-on-demand works when these features are not enabled.

The following example enables both Virtual Scrolling and the Show More Results box. The `EnableVirtualScrolling` property turns on the virtual scrolling feature, while the `ShowMoreResultsBox` property causes the combo box to display the "Show More Results" footer.



Gotcha! When you set either the `EnableVirtualScrolling` property or the `ShowMoreResultsBox` property to true, do *not* enable item caching. Setting the `EnableItemCaching` property to true prevents the callback for additional items that is required by virtual scrolling or the show more results box.

The `ItemsRequested` event handler in this example is similar to the one used in the last example, except that this time, the query only fetches the number of items that will appear in the combo box. For very large data sets, optimizing the query to fetch only the required records can improve performance.

In addition, the event handler uses a few additional properties of the event arguments object.

- It checks the `e.NumberOfItems` property to determine the number of items already loaded in the combo box.
- It sets the `e.EndOfItems` property when it detects that there are no more items to fetch. By setting `e.EndOfItems`, the event handler turns off the virtual scrolling mechanism. (`e.EndOfItems` does not affect the Show More Results box)
- It sets the `e.Message` property to a string describing the items that were fetched. The `e.Message` property sets the text that appears in the Show More Results box. If the event handler does not set this text, the Show More Results box includes only the link.



You can find the complete source for this project at:
 \VS Projects\ComboBox\VirtualScrolling

[VB] Adding the next batch of items in the `ItemsRequested` handler

```
Protected Sub RadComboBox1_ItemsRequested(ByVal o As Object, ByVal e As
RadComboBoxItemsRequestedEventArgs) Handles RadComboBox1.ItemsRequested
    Dim connectionString As String = ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString
    ' open a connection to the database
    Dim dbCon As New SqlConnection(connectionString)
    Try
        dbCon.Open()
        Dim itemsPerRequest As Integer = 10
        Dim itemOffset As Integer = e.NumberOfItems
        Dim endOffset As Integer = itemOffset + itemsPerRequest
        ' the query only fetches the required number of records
        Dim sql As String = "SELECT top " + endOffset.ToString() + "ProductName FROM [Products
by Category] where ProductName LIKE '" + e.Text + "%'"
        ' create a data adapter and use it to fetch data into a table
        Dim adapter As New SqlDataAdapter(sql, dbCon)
        Dim dt As New DataTable()
        adapter.Fill(dt)
        ' if we did not get the requested number of records
        ' we have reached the end of the data set
        ' set EndOfItems to signal this
        If dt.Rows.Count < endOffset Then
            e.EndOfItems = True
        End If
        ' if there are no items, set the message for the more results box
        If dt.Rows.Count = 0 Then
            e.Message = "No items"
```

```
Else
    ' use the table to add new items to the combo box
    If dt.Rows.Count > itemOffset Then
        Dim i As Integer = itemOffset
        While i < dt.Rows.Count
            RadComboBox1.Items.Add(New RadComboBoxItem(dt.Rows(i)("ProductName").ToString()))
            System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
        End While
    End If
    ' set the message for the more results box
    e.Message = [String].Format("Items <b>1</b>-<b>{0}</b>", dt.Rows.Count.ToString())
    If e.EndOfItems Then
        e.Message += [String].Format(" out of <b>{0}</b>", dt.Rows.Count.ToString())
    End If
End If
Catch
    e.Message = "No items available"
Finally
    dbCon.Close()
End Try
End Sub
```

[CS] Adding the next batch of items in the ItemsRequested handler

```
protected void RadComboBox1_ItemsRequested(object o, RadComboBoxItemsRequestedEventArgs e)
{
    string connectionString = ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].ConnectionString;
    // open a connection to the database
    SqlConnection dbCon = new SqlConnection(connectionString);
    try
    {
        dbCon.Open();

        int itemsPerRequest = 10;
        int itemOffset = e.NumberOfItems;
        int endOffset = itemOffset + itemsPerRequest;
        // the query only fetches the required number of records
        string sql = "SELECT top " + endOffset.ToString() + "ProductName FROM [Products by
Category] where ProductName LIKE '" + e.Text + "%'";
        // create a data adapter and use it to fetch data into a table
        SqlDataAdapter adapter = new SqlDataAdapter(sql, dbCon);
        DataTable dt = new DataTable();
        adapter.Fill(dt);
        // if we did not get the requested number of records
        // we have reached the end of the data set
        // set EndOfItems to signal this
        if (dt.Rows.Count < endOffset)
            e.EndOfItems = true;
        // if there are no items, set the message for the more results box
        if (dt.Rows.Count == 0)
            e.Message = "No items";
        else
        {
            // use the table to add new items to the combo box
            if (dt.Rows.Count > itemOffset)
```

```

    {
        for (int i = itemOffset; i < dt.Rows.Count; i++)
        {
            RadComboBox1.Items.Add(new RadComboBoxItem(dt.Rows[i]["ProductName"].ToString
        ());
        }
    }
    // set the message for the more results box
    e.Message = String.Format("Items <b>1</b>-<b>{0}</b>", dt.Rows.Count.ToString());
    if (e.EndOfItems)
        e.Message += String.Format(" out of <b>{0}</b>", dt.Rows.Count.ToString());
}
}
catch
{
    e.Message = "No items available";
}
finally
{
    dbCon.Close();
}
}

```

15.6 Server-Side Programming

Working with the Items collection

RadComboBox supports a number of methods for locating items in the drop-down list. These are

- **FindItemByText**, which returns a reference to an item given the value of its **Text** property.
- **FindItemByValue**, which returns a reference to an item given the value of its **Value** property.
- **FindItemIndexByText**, which returns the index of an item given the value of its **Text** property.
- **FindItemIndexByValue**, which returns the index of an item given the value of its **Value** property.

Once you have located an item, you can use its properties to change its value, select it, disable it, delete it, and so on.

The following example illustrates some of these methods. It uses the **FindItemByText** method to determine whether the string a user types in the text area of a combo box is already in the drop-down list. If not, it adds an item to the list with its **Text** property set to the new string and its **Value** property set to reflect the position of the new item. Next, the **Value** property of the item in the drop-down list (either the matching item that was found or the newly entered item) is passed to the **FindItemIndexByValue** method of a second combo box. If an item with a matching value is found, that corresponding item is selected.



You can find the complete source for this project at:
\\VS Projects\\ComboBox\\ServerSide

[VB] Working with items

```
Protected Sub RadComboBox1_TextChanged(ByVal sender As Object, ByVal e As EventArgs) Handles RadComboBox1.TextChanged
    ' If the string is not in combo box 1, add it
    Dim item As RadComboBoxItem = RadComboBox1.FindItemByText(RadComboBox1.Text)
    If item Is Nothing Then
        ' create a new item with Text set to the text in the input area
        ' and value set to the position of the new item
        item = New RadComboBoxItem(RadComboBox1.Text, (RadComboBox1.Items.Count + 1).ToString())
        ' add it to combo box 1
        RadComboBox1.Items.Add(item)
    End If
    ' If an item with the same value is in combo box 2, select it
    Dim value As String = item.Value
    Dim index As Integer = RadComboBox2.FindItemIndexByValue(value)
    If RadComboBox2.SelectedIndex <> index AndAlso index >= 0 Then
        RadComboBox2.Items(index).Selected = True
    End If
End Sub
```

[C#] Working with items

```
protected void RadComboBox1_TextChanged(object sender, EventArgs e)
{
    // If the string is not in combo box 1, add it
    RadComboBoxItem item = RadComboBox1.FindItemByText(RadComboBox1.Text);
    if (item == null)
    {
```

```

    // create a new item with Text set to the text in the input area
    // and value set to the position of the new item
    item = new RadComboBoxItem(RadComboBox1.Text, (RadComboBox1.Items.Count + 1).ToString
());
    // add it to combo box 1
    RadComboBox1.Items.Add(item);
}
// If an item with the same value is in combo box 2, select it
string value = item.Value;
int index = RadComboBox2.FindItemIndexByValue(value);
if (RadComboBox2.SelectedIndex != index && index >= 0)
    RadComboBox2.Items[index].Selected = true;
}

```

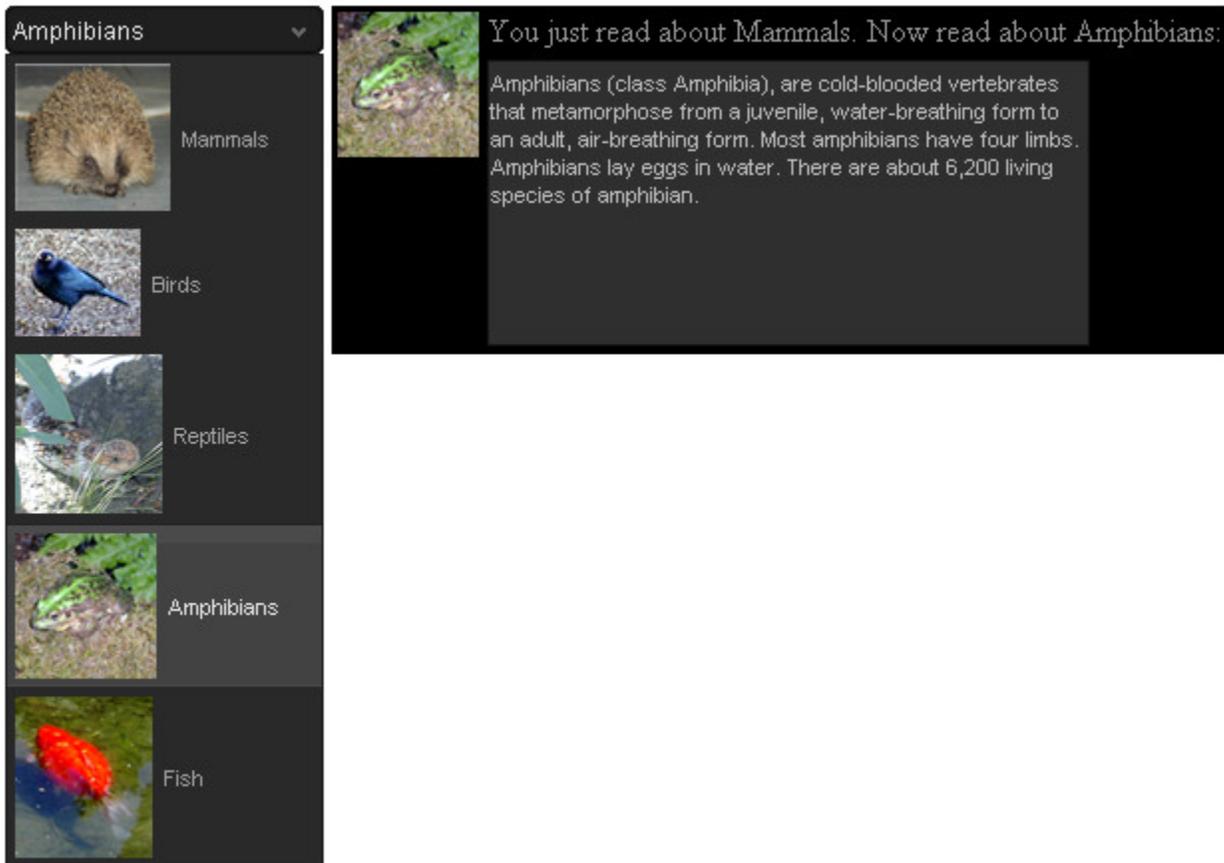
Note that the code above illustrates two properties for determining whether an item is selected. The **SelectedIndex** property of the combo box is used to identify the position of the currently selected item, and the **Selected** property of a combo box item is used to select it. You can use either of these properties to change the current selection of a combo box.

Responding to selection changes

The previous example used the **TextChanged** event handler to respond when the user changed the text of the combo box. This event occurs when focus leaves the text area of the combo box after its text changes, either because the user typed a string or because the user selected an item from the drop-down list. **TextChanged** is useful for responding to selection changes in combo boxes, such as the one in the previous example, that have the **AllowCustomText** property set to true. However, as you may have noticed in the last example, the event handler has to use the properties of the combo box to determine the current values. Furthermore, there is no way to determine which item, if any, was previously selected.

When **AllowCustomText** is not set to true, it can be more useful to use the **SelectedIndexChanged** event instead. The **SelectedIndexChanged** event arguments object has properties that let you access the **Text** and **Value** of the last selected item, as well as properties for the **Text** and **Value** of the item that was just selected.

The following example illustrates using the **SelectedIndexChanged** event. The event handler uses the event arguments to update a label that gives both the text of the last selected item and the text of the currently selected item, as well as to update a text box to reflect the current selection.



You can find the complete source for this project at:
\\VS Projects\ComboBox\ServerSelectedIndex

[VB] Updating controls in the SelectedIndexChanged event

```
Protected Sub RadComboBox1_SelectedIndexChanged(ByVal o As Object, ByVal e As  
RadComboBoxSelectedIndexChangedEventArgs) Handles RadComboBox1.SelectedIndexChanged  
    ' Display the last selection and the new selection in the label  
    Label1.Text = "You just read about " + e.OldText + ". Now read about " + e.Text + ": "  
    ' update the image to the one associated with the current item  
    Image1.ImageUrl = RadComboBox1.SelectedItem.ImageUrl  
    ' use the new value to index the appropriate blurb.  
    RadTextBox1.Text = Blurbs(System.Int16.Parse(e.Value))  
End Sub
```

[C#] Updating controls in the SelectedIndexChanged event

```
protected void RadComboBox1_SelectedIndexChanged(object o,  
RadComboBoxSelectedIndexChangedEventArgs e)  
{  
    // Display the last selection and the new selection in the label  
    Label1.Text = "You just read about " + e.OldText + ". Now read about " + e.Text + ": ";  
    // update the image to the one associated with the current item  
    Image1.ImageUrl = RadComboBox1.SelectedItem.ImageUrl;  
    // use the new value to index the appropriate blurb.
```

```
RadTextBox1.Text = Blurbs[System.Int16.Parse(e.Value)];
}
```

Sorting items

You can sort the items of RadComboBox to make it easier for users to locate the item they want in the drop-down list. To enable sorting, set the **Sort** property of the combo box to "Ascending" or "Descending", depending on the order you want. By default, sorting is case sensitive, but you can change that by setting the **SortCaseSensitive** property to false.

When the Sort property is set to "Ascending" or "Descending", you can sort the items by calling the combo box's **SortItems** method, or by calling the **Sort** method of the **Items** collection. Both methods do the same thing: they sort the items based on their **Text** property values.

The following walk-through illustrates the use of the **Sort** and **SortItems** methods.



You can find the complete source for this project at:
 \VS Projects\ComboBox\Sorting

Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Locate the "Northwind.mdf" and "Northwind_log.ldf" files in the "Live Demos\App_Data" folder under the folder where you installed RadControls for ASPNET AJAX. Drag these files into the "App_Data" folder of your project.
3. Open the "Web.config" file of your project. Add the standard Northwind connection string to your project by replacing the line


```
<connectionStrings />
```

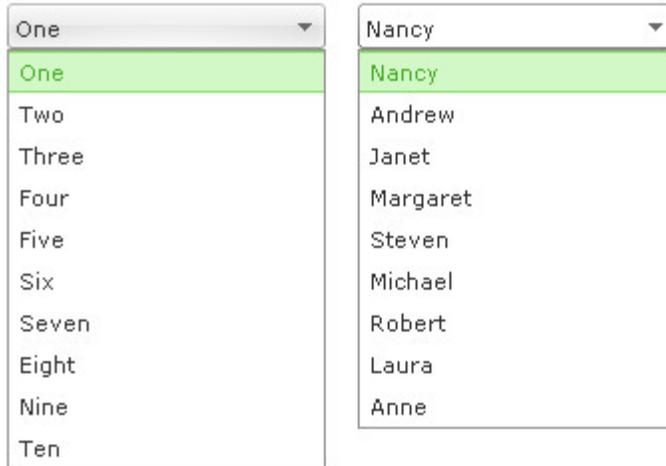
 with


```
<connectionStrings>
  <add name="NorthwindConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|Northwind.mdf;Integrated
Security=True;User Instance=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Add the combo boxes

1. Drag a RadComboBox from the Tool Box onto your Web page. Set the **Skin** property to "Telerik" and the **CloseDropDownOnBlur** property to false.
2. Bring up the RadComboBox Item Builder, and add ten items to the combo box with Text properties set to "One", "Two", "Three", "Four", "Five", "Six", "Seven", "Eight", "Nine", and "Ten" (in that order).
3. Drag a second RadComboBox from the Tool Box onto your Web page. Set the **Skin** property to "Telerik".
4. Using the Smart Tag of the second RadComboBox control, select "<New data source...>" from the **Choose Data Source** drop-down.
5. In the first page of the **Data Source Configuration Wizard**, select "Database" as the application type, and click **OK** to move to the next page.
6. On the **Choose Your Data Connection** page, select "NorthwindConnectionString" from the drop-down list. Then click the **Next** button to continue.
7. On the **Configure the Select Statement** page, make sure the "Specify columns from a table or view" radio button is selected, and then choose "Employees" from the "Name" drop-down list.
8. Check the "FirstName" field of the "Employees" table, and then click the **Next** button to continue.
9. Test the query if you wish, and then click **Finish**.

- In the Properties Window for the second combo box, Set the DataTextField property to "FirstName".
- Click Ctrl-F5 to run the application. Open the first combo box, and then the second. Note that the items are not sorted.



Sort the drop-down lists

- You can sort statically declared items in a **Page_Load** event handler. Add the following code the **Page_Load** event handler. (Remember to first add an Imports or using statement for "Telerik.Web.UI!")
[VB] Sorting statically declared items

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
    'We only need to sort items the first time the page is loaded
    If Not IsPostBack Then
        'Sort the statically-bound items in ascending order
        RadComboBox1.Sort = RadComboBoxSort.Ascending
        RadComboBox1.SortItems()
    End If
End Sub
```

[C#] Sorting statically declared items

```
protected void Page_Load(object sender, EventArgs e)
{
    // We only need to sort items the first time the page is loaded
    if (!IsPostBack)
    {
        // Sort the statically-bound items in ascending order
        RadComboBox1.Sort = RadComboBoxSort.Ascending;
        RadComboBox1.SortItems();
    }
}
```

- To sort the items in a data-bound combo box, you must use the **DataBound** event, which occurs once all the items have been added to the combo box.
 Do not confuse the **DataBound** event, which occurs once all items are loaded, with the **ItemDataBound** event, which occurs after each item is loaded.

Because we sorted the last set of items using an ascending order, this time, set the sort order to "Descending". This time, also, use the **Sort** method of the **Items** collection rather than the **SortItems** method you used for the other combo box.

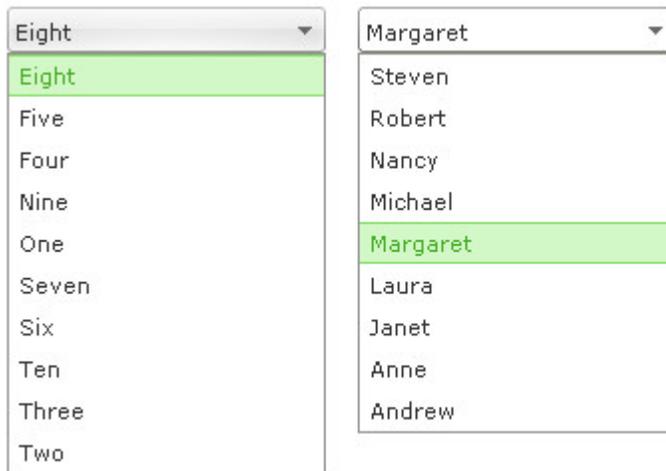
[VB] Sorting data-bound items

```
Protected Sub RadComboBox2_DataBound(ByVal sender As Object, ByVal e As EventArgs) Handles
RadComboBox2.DataBound
    'Data-bound items must be sorted in the DataBound event handler
    'Sort them in descending order
    RadComboBox2.Sort = RadComboBoxSort.Descending
    RadComboBox2.Items.Sort()
End Sub
```

[CS] Sorting data-bound items

```
protected void RadComboBox2_DataBound(object sender, EventArgs e)
{
    // Data-bound items must be sorted in the DataBound event handler
    // Sort them in descending order
    RadComboBox2.Sort = RadComboBoxSort.Descending;
    RadComboBox2.Items.Sort();
}
```

- Run the application again. This time, note that the items have been sorted:



💡 You can also sort items using a custom sort criterion. See the How-To section for an example of how this is done.

15.7 Client-Side Programming

Working with the items collection

You do not need to rely on server-side code to change the items of the drop-down list. You can also add or remove items and change their properties in client-side code. You can use the client-side `findItemsByText()` and `findItemsByValue()` methods to locate items, just like you can use the similarly named methods on the server side. The client-side object for the combo box has a `get_items()` method that provides access to the items collection. You can use the methods of the items collection to add or delete items, or iterate through the items collection.

The following example illustrates the use of some of these client-side methods. When the Web page loads, it contains an empty combo box and a text box with an associated button. When the user enters a string in the text box and clicks the button, a client-side script checks whether the string already appears in the drop-down list of the combo box. If so, it deletes it. If not, it adds it.



You can find the complete source for this project at:
\\VS Projects\\ComboBox\\ClientSide

[JavaScript] Manipulating the items collection

```
function ChangeDropDownList(sender) {  
    // check whether the text box has any text  
    var text = sender.get_value();  
    if (text.trim() != "") {  
        // get a reference to the combo box  
        var combo = $find("<%= RadComboBox1.ClientID %>");  
        // find the matching combo box item  
        var item = combo.findItemByText(text);  
        // call trackChanges so that the changes can persist  
        combo.trackChanges();  
        // if the item is in the list, remove it  
        if (item) {  
            combo.get_items().remove(item);  
        }  
        else {  
            // if the item is not in the list, add it  
            var comboItem = new Telerik.Web.UI.RadComboBoxItem();  
            comboItem.set_text(text);  
            combo.get_items().add(comboItem);  
            comboItem.select();  
        }  
        // commit the changes  
        combo.commitChanges();  
    }  
}
```

Notice in the code above that before making any changes to the items collection, the function calls the combo box's `trackChanges()` method. After the changes are complete, it calls the `commitChanges()` method. These two methods are important when working with the items collection. If you do not surround any changes that you make to the items collection with calls to `trackChanges()` and `commitChanges()`, the changes are lost the next time the page executes a postback.



Any changes made in client-side code after the call to `trackChanges()` can be reviewed in server-side code by reading the server-side `ClientChanges` property. `ClientChanges` is a collection of `ClientOperation` objects that describe each client-side change that was made.

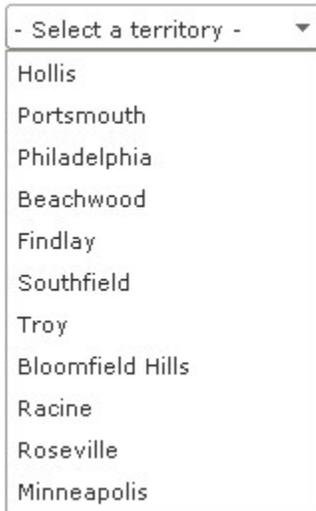
Using client-side events

In addition to a powerful set of client-side methods for manipulating the combo box and its items collection, you can also make use of a wealth of client-side events to respond to just about any change that occurs. You

have already seen some of these events used: in the first Templates example, the **OnClientLoad** and **OnClientSelectedIndexChanged** events were used to maintain the footer template. The following example illustrates some of the other client-side events.

This example uses a combo box with its **AllowCustomText** property set to true and its **EmptyMessage** property set to "- Select a territory -". When the combo box gets focus, the **OnClientFocus** event handler automatically opens the drop-down list if the user has not selected a value.

Normally, when the combo box gets focus, the **EmptyMessage** string disappears. To prevent this from happening, the **OnClientDropDownOpened** event handler sets the text to the value of **EmptyMessage** if it is an empty string. The **OnClientDropDownClosed** event handler restores the text to an empty string when the drop down closes if the user has not changed the value.



You can find the complete source for this project at:
 \VS Projects\ComboBox\ClientEvents

[ASP.NET] ComboBox with client events

```
<script type="text/javascript">
// OnClientFocus handler
function OpenDropDownOnFocus(sender, args) {
    // if the text is not set
    if (sender.get_text() == "");
        // open the drop down list
        sender.showDropDown();
}
// OnClientDropDownOpened handler
function RestoreEmptyMessage(sender, args) {
    // if the text is not set
    if (sender.get_text() == "")
        // set it back to the empty message
        sender.set_text(sender.get_emptyMessage());
}
// OnClientDropDownClosed handler
function RestoreEmptyString(sender, args) {
    // if the text is still the empty message
    if (sender.get_text() == sender.get_emptyMessage())
        // set it back to an empty string
```

```
        sender.set_text("");
    }
</script>
<telerik:RadComboBox ID="RadComboBox1" runat="server"
    Skin="Gray" DataSourceID="SqlDataSource1"
    DataTextField="TerritoryDescription" DataValueField="TerritoryID"
    AllowCustomText="true" EmptyMessage="- Select a territory -"
    OnClientFocus="OpenDropDownOnFocus"
    OnClientDropDownOpened="RestoreEmptyMessage"
    OnClientDropDownClosed="RestoreEmptyString">
</telerik:RadComboBox>
```

Using client events with load-on-demand

When using the load-on-demand feature (**EnableLoadOnDemand** set to true), there are three client-side events that surround the callback to populate the drop-down list.

- Before the callback occurs, the **OnClientItemsRequesting** event lets you provide additional context information to the event handler or Web service that supplies items, or even cancel the event to prevent the callback from occurring.
- If the callback fails for some reason, the **OnClientItemsRequestFailed** event lets you provide your own response, in addition to or instead of the default error message.
- If the callback is successful, the **OnClientItemsRequested** event lets you provide your own post-processing.

The following example illustrates the use of these three events. The example uses two combo boxes: one bound to a data source and the other getting its items using the load-on-demand mechanism. The combo box that loads its items on demand uses the currently selected item in the first combo box to provide additional context information to the server-side **ItemsRequested** handler. As a result, the items list of the second combo box displays only items associated with the selected item in the first combo box. Entering text in the second combo box further limits this list to items that match the entered text.



You can find the complete source for this project at:
\\VS Projects\\ComboBox\\ClientLoadOnDemand

Before looking at the load-on-demand related events, let us first look at the first combo box. When the user changes the selected item in the first combo box, the drop-down list and text of the second combo box become invalid. To handle this, the first combo box has an **OnClientSelectedIndexChanged** handler that clears the text and drop-down list of the second combo box. As an added nicety, the event handler moves focus to the text area of the second combo box:

[JavaScript] Clearing the second combo box when the first changes

```
function SelectionChanged(sender, args) {
    // when combo1 changes its selection,
    // we need to clear combo 2
    var combo2 = $find("<%= RadComboBox2.ClientID %>");
    // clear the current list of items
    combo2.clearItems();
    // set the text to an empty string
    combo2.set_text("");
    // move focus to the text area of the second combo
    combo2.get_inputDomElement().focus();
}
```

The load-on-demand related events occur on the second combo box, which loads its items on demand. The first of these events that gets called is **OnClientItemsRequesting**. This event handler looks up the currently selected value on the first combo box, and adds that as context information to the item request:

[JavaScript] Adding context information to the items request

```
function ItemsRequesting(sender, args) {
    // set the context to the value of the selected item in combo 1
    // the value holds the category ID
    var combo1 = $find("<%= RadComboBox1.ClientID %>");
    var item = combo1.get_selectedItem();
    args.get_context()["Category"] = item.get_value();
}
```

On the server, the server-side **ItemsRequested** event handler uses both the text of the second combo box and the context information that was added in the **OnClientItemsRequesting** handler:

[VB] Servicing the items request on the server

```
Protected Sub RadComboBox2_ItemsRequested(ByVal o As Object, ByVal e As
RadComboBoxItemsRequestedEventArgs) Handles RadComboBox2.ItemsRequested
    ' build the query from the context and text values
    Dim query As New StringBuilder("SELECT ProductID, ProductName from Products WHERE")
    query.Append(" CategoryID = ")
    query.Append(e.Context("Category").ToString())
    query.Append(" AND ProductName LIKE '")
    query.Append(e.Text)
    query.Append("%' ")
    ' open a connection to the database
    Dim dbCon As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
    Try
        dbCon.Open()
        ' create a data adapter and use it to fetch data into a table
        Dim adapter As New SqlDataAdapter(query.ToString(), dbCon)
        Dim dt As New DataTable()
        adapter.Fill(dt)
        ' use the table to add items to the combo box
        For Each row As DataRow In dt.Rows
            Dim item As New RadComboBoxItem(row("ProductName").ToString(), row
("ProductID").ToString())
            RadComboBox2.Items.Add(item)
            If item.Value = "5" Then
                Throw New Exception("Simulated error")
            End If
        Next
    Finally
        dbCon.Close()
```

```
End Try
End Sub
```

[C#] Servicing the items request on the server

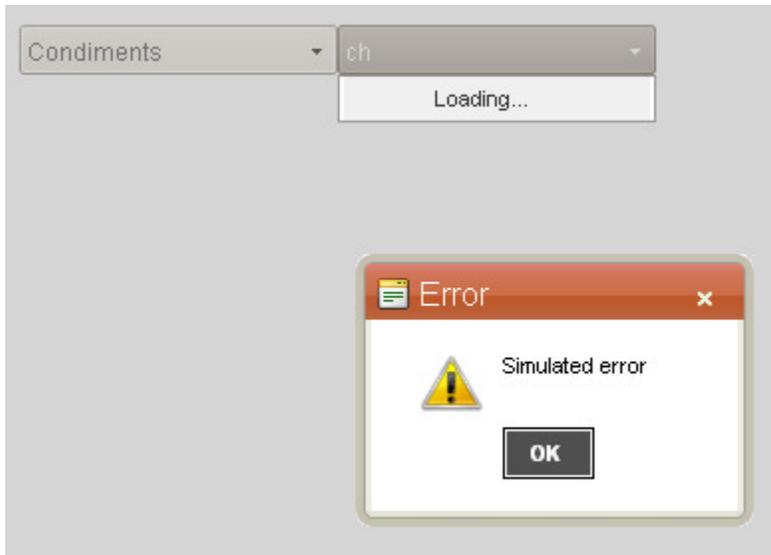
```
protected void RadComboBox2_ItemsRequested(object o, RadComboBoxItemsRequestedEventArgs e)
{
    // build the query from the context and text values
    StringBuilder query = new StringBuilder("SELECT ProductID, ProductName from Products
WHERE");
    query.Append(" CategoryID = ");
    query.Append(e.Context["Category"].ToString());
    query.Append(" AND ProductName LIKE '");
    query.Append(e.Text);
    query.Append("%'");
    // open a connection to the database
    SqlConnection dbCon = new SqlConnection(ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].ConnectionString);
    try
    {
        dbCon.Open();
        // create a data adapter and use it to fetch data into a table
        SqlDataAdapter adapter = new SqlDataAdapter(query.ToString(), dbCon);
        DataTable dt = new DataTable();
        adapter.Fill(dt);
        // use the table to add items to the combo box
        foreach (DataRow row in dt.Rows)
        {
            RadComboBoxItem item = new RadComboBoxItem(row["ProductName"].ToString(), row
["ProductID"].ToString());
            RadComboBox2.Items.Add(item);
            if (item.Value == "5")
                throw new Exception("Simulated error");
        }
    }
    finally
    {
        dbCon.Close();
    }
}
```

Note that the code shown above generates an exception if the list includes an item with product ID set to 5. This is included to simulate an error when fetching items so that you can see the effects of the **OnClientItemsRequestFailed** event handler. This event handler cancels the default error handling (which uses the browser-based alert dialog to display an error message), and instead uses the **radalert()** function to display the error message in a dialog that uses the same skin as the combo box:

[JavaScript] Replacing the default error handling

```
function ItemsRequestFailed(sender, args) {
    // use the rad alert to match the skin
    radalert(args.get_errorMessage(), 200, 75, "Error" );
    // set cancel to suppress the default error message
    args.set_cancel(true);
}
```

 To trigger the simulated error and see the effects of this error handler, select "Condiments" in the first combo box, and open the drop-down list of the second combo box when its text is an empty string or "c".



The last client-side event related to load-on-demand is **OnClientItemsRequested**, which occurs once the drop-down list is populated with new items. In this example, the event handler selects the first item in the list if the user has already entered text before opening the drop-down list:

[JavaScript] Selecting the first item in the new list

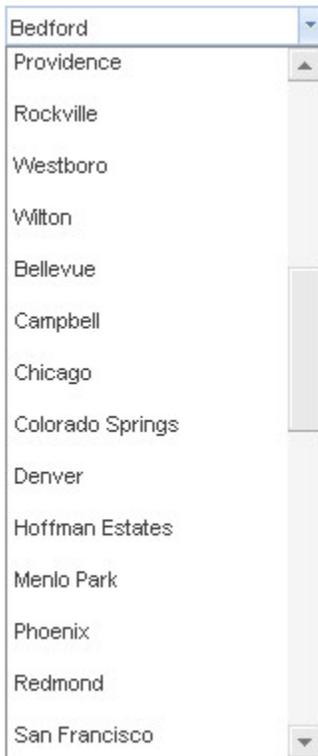
```
function ItemsRequested(sender, args) {
    // get the new list of items
    var items = sender.get_items();
    // if the user has started entering some text
    // and the new list of items is not empty
    if (sender.get_text() != "" && items.get_count() > 0)
        // select the first item
        items.getItem(0).select();
}
```

15.8 How To

Implementing a custom sort

As you saw in the section on Server-Side Programming, you can sort the items in the drop-down list by calling the **Sort** method of the **Items** collection or the **SortItems** method of the combo box. Both of these methods sort the items by the value of their **Text** property. However, both the **Sort** method and the **SortItems** method have an overloaded version that takes an argument of type **IComparer**. By creating a class that implements **IComparer** and passing an instance of your class to the **Sort** or **SortItems** method, you can sort the items in the drop-down list using your own custom algorithm.

The following example illustrates sorting items using a custom algorithm. The custom algorithm sorts items by a custom attribute ("Region"), and within region, by the value of the **Text** property.



You can find the complete source for this project at:
\\VS Projects\ComboBox\HowToCustomSort

To implement the custom sort, the application first defines a class that implements the **IComparer** interface. The **IComparer** interface defines a single method, **Compare**, which compares two objects:

[VB] CustomSort class

```
Public Class CustomSort
    Implements IComparer
    Public Function Compare(ByVal x As Object, ByVal y As Object) As Integer Implements
IComparer.Compare
        Dim p1 As RadComboBoxItem, p2 As RadComboBoxItem
        ' first make sure the items are of type RadComboBoxItem
        If TypeOf x Is RadComboBoxItem Then
            p1 = TryCast(x, RadComboBoxItem)
        Else
            Throw New ArgumentException("Object is not of type RadComboBoxItem.")
        End If
        If TypeOf y Is RadComboBoxItem Then
            p2 = TryCast(y, RadComboBoxItem)
        Else
            Throw New ArgumentException("Object is not of type RadComboBoxItem.")
        End If
        ' get the combo box
        Dim combo As RadComboBox = p1.ComboBoxParent
        ' get the Region attribute for each item
        Dim a1 As String = p1.Attributes("Region")
```

```

Dim a2 As String = p2.Attributes("Region")
' if the attribute does not exist, use an empty string
If a1 Is Nothing Then a1 = ""
If a2 Is Nothing Then a2 = ""
' sort by region
Dim cmp As Integer = String.Compare(a1, a2, Not combo.SortCaseSensitive)
' if regions match, sort by text within region
If cmp = 0 Then
    cmp = String.Compare(p1.Text, p2.Text, Not combo.SortCaseSensitive)
End If
' for a descending sort, reverse the comparison value
If combo.Sort = RadComboBoxSort.Descending Then
    cmp = cmp * -1
End If
Return cmp
End Function
End Class

```

[C#] CustomSort class

```

public class CustomSort : IComparer
{
    public int Compare(object x, object y)
    {
        RadComboBoxItem p1, p2;
        // first make sure the items are of type RadComboBoxItem
        if (x is RadComboBoxItem)
            p1 = x as RadComboBoxItem;
        else
            throw new ArgumentException("Object is not of type RadComboBoxItem.");
        if (y is RadComboBoxItem)
            p2 = y as RadComboBoxItem;
        else
            throw new ArgumentException("Object is not of type RadComboBoxItem.");

        // get the combo box
        RadComboBox combo = p1.ComboBoxParent;
        // get the Region attribute for each item
        string a1 = p1.Attributes["Region"];
        string a2 = p2.Attributes["Region"];
        // if the attribute does not exist, use an empty string
        if (a1 == null)
            a1 = "";
        if (a2 == null)
            a2 = "";
        // sort by region
        int cmp = String.Compare(a1, a2, !combo.SortCaseSensitive);
        // if regions match, sort by text within region
        if (cmp == 0)
            cmp = String.Compare(p1.Text, p2.Text, !combo.SortCaseSensitive);
        // for a descending sort, reverse the comparison value
        if (combo.Sort == RadComboBoxSort.Descending)
            cmp = cmp * -1;
        return cmp;
    }
}

```

The **Compare** method casts both objects to **RadComboBoxItem**. It then retrieves a reference to the combo box that contains them, so that it can look up the **Sort** and **SortCaseSensitive** properties. It compares the items based on their "Region" attribute, and if the regions match, it compares them by their **Text** property.

The default Web page contains a data-bound combo box. Each item sets the "Region" custom attribute in the **ItemDataBound** event handler:

[VB] Assigning custom attributes in ItemDataBound

```
Protected Sub RadComboBox1_ItemDataBound(ByVal sender As Object, _
                                         ByVal e As RadComboBoxItemEventArgs) _
                                         Handles RadComboBox1.ItemDataBound
    ' add a custom attribute for the region
    Dim dataSourceRow As DataRowView = DirectCast(e.Item.DataItem, DataRowView)
    e.Item.Attributes("Region") = dataSourceRow("RegionID").ToString()
End Sub
```

[C#] Assigning custom attributes in ItemDataBound

```
protected void RadComboBox1_ItemDataBound(object sender, RadComboBoxItemEventArgs e)
{
    // add a custom attribute for the region
    DataRowView dataSourceRow = (DataRowView)e.Item.DataItem;
    e.Item.Attributes["Region"] = dataSourceRow["RegionID"].ToString();
}
```

In the **DataBound** event handler, the application calls the **SortItems** method, passing in an instance of the **CustomSort** class:

[VB] Sorting items in the DataBound handler

```
Protected Sub RadComboBox1_DataBound(ByVal sender As Object, _
                                     ByVal e As EventArgs) _
                                     Handles RadComboBox1.DataBound
    RadComboBox1.Sort = RadComboBoxSort.Ascending
    Dim sort As IComparer = TryCast(New CustomSort(), IComparer)
    RadComboBox1.SortItems(sort)
End Sub
```

[C#] Sorting items in the DataBound handler

```
protected void RadComboBox1_DataBound(object sender, EventArgs e)
{
    RadComboBox1.Sort = RadComboBoxSort.Ascending;
    IComparer sort = new CustomSort() as IComparer;
    RadComboBox1.SortItems(sort);
}
```

Using an input control in an item template

In the section on Control Specifics, you built a Web page that included a **RadCalendar** control in the item template. In that example project, the combo box closed as soon as the user clicked on the calendar. There are times, however, when you may not want the drop-down list to close when the user clicks on a control in the item template. For example, if the item template includes an input control, the control would be unusable if the drop-down list closed as soon as the user clicked in it. However, if you simply add an input control to the item template, this is exactly what happens. The drop-down list closes because the mouse click event bubbles up from the embedded control to the combo box item, and when the combo box item receives the mouse click, it closes the drop-down list. To make the input control usable, you must therefore prevent the mouse click event from bubbling up to the combo box item.

The following example illustrates how this is done. The Web page contains a combo box that lets the user

select a date from its drop-down list, or, using the last item in the list, enter a custom date value that falls within an allowable range.

To implement this, the combo box uses an item template with a **RadDateInput** in it. For all but the last item, the date input control has its **ReadOnly** property set to true. On the last item, the **ReadOnly** property is set to "False", allowing the user to enter a custom value, and has its **MinDate** property set to enforce a range. The template suppresses mouse click events from bubbling up when the click occurs in the last text box.



You can find the complete source for this project at:
 \VS Projects\ComboBox\HowToInputInTemplate

The declaration for the combo box includes a custom attribute to indicate the "ReadOnly" status of items:

[ASP.NET] RadComboBox declaration

```
<telerik:RadComboBox ID="RadComboBox1" runat="server" >
  <Items>
    <telerik:RadComboBoxItem runat="server" Text="1/1/2005" Value="1/1/2005"
ReadOnly="True" />
    <telerik:RadComboBoxItem runat="server" Text="1/1/2006" Value="1/1/2006"
ReadOnly="True" />
    <telerik:RadComboBoxItem runat="server" Text="1/1/2007" Value="1/1/2007"
ReadOnly="True" />
    <telerik:RadComboBoxItem runat="server" Text="" ReadOnly="False" />
  </Items>
  <ItemTemplate>
    <div onclick="ClickTemplate(event)">
      <telerik:RadDateInput ID="RadDateInput1" runat="server" Width="97%" >
        <ClientEvents OnValueChanged="ValueChanged" OnFocus="FocusItem" />
      </telerik:RadDateInput>
    </div>
  </ItemTemplate>
</telerik:RadComboBox>
```

Note that the **RadDateInput** control in the template does not have its **SelectedDate**, **ReadOnly**, or **MinDate** properties set. This is done in the **Page_Load** event handler:

[VB] Initializing template controls

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
  Dim i As Integer = 0
  While i < RadComboBox1.Items.Count
    Dim item As RadComboBoxItem = RadComboBox1.Items(i)
    ' get the date input inside the template
    Dim di As RadDateInput = DirectCast(RadComboBox1.Items(i).FindControl("RadDateInput1"),
RadDateInput)
    If Not di Is Nothing Then
      ' set the ReadOnly property to match the item attribute
```

```
di.[ReadOnly] = (item.Attributes("ReadOnly") <> "False")
If item.Text <> "" Then
    ' Set the SelectedDate to the item text
    Dim [date] As DateTime = DateTime.Parse(item.Text)
    di.SelectedDate = [date]
End If
If Not di.[ReadOnly] Then
    ' if the item is editable, save its client ID in a hidden field
    HiddenField1.Value = di.ClientID + "_text"
    ' and give it a minimum date
    di.MinDate = DateTime.Parse("1/1/2008")
End If
End If
System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
End While
End Sub
```

[C#] Initializing template controls

```
protected void Page_Load(object sender, EventArgs e)
{
    for (int i = 0; i < RadComboBox1.Items.Count; i++)
    {
        RadComboBoxItem item = RadComboBox1.Items[i];
        // get the date input inside the template
        RadDateInput di = (RadDateInput)RadComboBox1.Items[i].FindControl("RadDateInput1");
        if (di != null)
        {
            // set the ReadOnly property to match the item attribute
            di.ReadOnly = (item.Attributes["ReadOnly"] != "False");
            if (item.Text != "")
            {
                // Set the SelectedDate to the item text
                DateTime date = DateTime.Parse(item.Text);
                di.SelectedDate = date;
            }
            if (!di.ReadOnly)
            {
                // if the item is editable, save its client ID in a hidden field
                HiddenField1.Value = di.ClientID + "_text";
                // and give it a minimum date
                di.MinDate = DateTime.Parse("1/1/2008");
            }
        }
    }
}
```

The **Page_Load** event handler iterates through the items in the combo box, and for each one, uses the **FindControl** method to locate the date input control in the template. It sets the **ReadOnly** and **SelectedDate** properties after parsing the **ReadOnly** attribute and **Text** property of the combo box item. Finally, for the item that permits editing, it stores the client ID of the input area in a hidden field and sets the **MinDate** property to limit the range of permissible values.

If you look back at the declaration of the combo box, you will see that the **ItemTemplate** includes a **<div>** element with an **onclick** handler that contains the rest of the template. This **onclick** handler stops the click event from bubbling up to the combo box item if it originated from the control whose id matches the value of the hidden field (that is, the date input that permits editing).

[JavaScript] Canceling event bubbling

```
function ClickTemplate(event) {
    // when <div> is clicked,
    // check whether the click originated from the editable item
    var hf = $get("HiddenField1");
    if (event.srcElement.id == hf.value)
        // if so, don't let the click through to the combo box item
        event.cancelBubble = true;
}
```

This allows the drop-down list to remain open when the user clicks on the last item. However, it also prevents the text from getting set or the drop-down list from closing when the user finishes editing. To perform these tasks, the date input control has a handler for the client-side **OnValueChanged** event:

[JavaScript] Setting the text and closing the drop-down

```
function ValueChanged(sender, args) {
    // when user edits the date input control
    var combo = $find("<%= RadComboBox1.ClientID %>");
    // set the combo box text
    combo.set_text(args.get_newValue());
    // and close the drop-down list
    combo.hideDropDown();
}
```

What happens when the user enters a value in the date input control, then selects another combo box item, and then returns to the previously entered value? Because the value of the date input control did not change, the combo box text is not updated. To handle this situation, the date input control has a handler for the client-side **OnFocus** event:

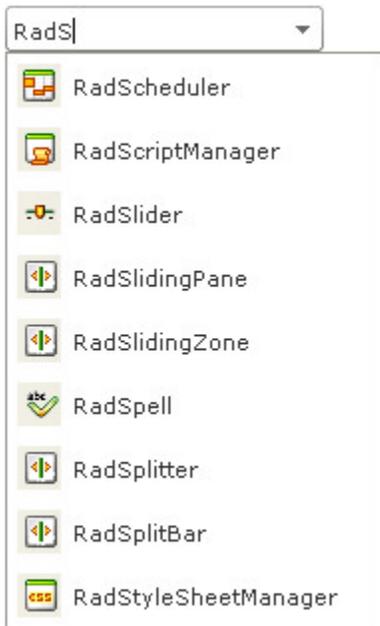
[JavaScript] Setting the combo box text on focus

```
function FocusItem(sender, args) {
    // if the editable item gets focus and already has a value
    if (sender.get_value() != "" && !sender.isReadOnly()) {
        var combo = $find("<%= RadComboBox1.ClientID %>");
        // update the combo box text
        combo.set_text(sender.get_value());
    }
}
```

Limiting item requests with Load-on-demand

In the first load-on-demand example, the **ItemsRequested** event handler did not add any items to the combo box when its **Text** property was an empty string. This was not really necessary for the sample database, but represented a situation where the data set was so large that it was not reasonable to fetch items until there was more of a filter. However, the callback to request items was still generated. To reduce this extra (and unnecessary) traffic, you can prevent the callback from occurring when the text box contains a string that should not generate a request (such as a string that is too short).

The following example illustrates how this is done. The Web page contains a combo box that displays a choice of RadControls which are loaded on demand:



Because each control name begins with the string "Rad", it does not make sense to generate a callback unless the text is an empty string or a string of at least four characters. A client-side **ItemsRequesting** event handler ensures that the callback only occurs when it makes sense, by cancelling the request when the text is one, two, or three characters long.

[JavaScript] Cancelling the request in the ItemsRequesting handler

```
function ItemsRequesting(sender, args) {  
    if ((sender.get_text().length < 4) && (sender.get_text() != ""))  
        args.set_cancel(true);  
}
```



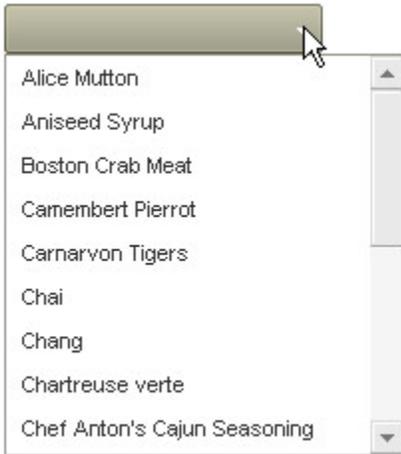
You can find the complete source for this project at:
\\VS Projects\ComboBox\HowToLimitItemRequests

Enabling virtual scrolling without allowing custom text

The RadComboBox virtual scrolling mechanism is typically used with the load-on-demand feature. The **EnableLoadOnDemand** property, however, forces the combo box to allow custom text, regardless of the value of the **AllowCustomText** property. This feature is required so that the load-on-demand mechanism can fetch items that are not already in the list.

What if you want to use virtual scrolling, but only allow the user to select items from the drop-down list? If you set the **EnableLoadOnDemand** to false, you can then set the **AllowCustomText** property to false as well. It is possible to enable the virtual scrolling mechanism even when **EnableLoadOnDemand** is false, but the drop-down list does not immediately fill when it opens. If **ShowMoreResultsBox** is true, the user is presented with an empty drop-down list and unlabelled Show More Results box, and must click on the box to fetch the first set of items. If only **EnableVirtualScrolling** is true, the situation is even worse: the empty list has no scroll bar, so there is no way to trigger the virtual scrolling.

The following example shows how to populate the drop-down list with the first set of items on a combo box that uses only virtual scrolling without a Show More Results box.



To populate the drop-down list when it first opens, the application uses an **OnClientDropDownOpening** event handler. The event handler calls the **requestItems()** method, which initiates a callback to fetch the first set of items.

[JavaScript] Initiating an item request when the drop-down opens

```
function OnClientDropDownOpeningHandler(sender, args) {
  // initiate an item request
  // first parameter is the text
  // second parameter is whether to append items
  sender.requestItems("", true);
}
```

Because virtual scrolling requires a scroll bar to be present, the **Height** property is set so that the first set of items do not all fit in the drop-down list. This ensures that a scroll bar appears when the first set of items is loaded.

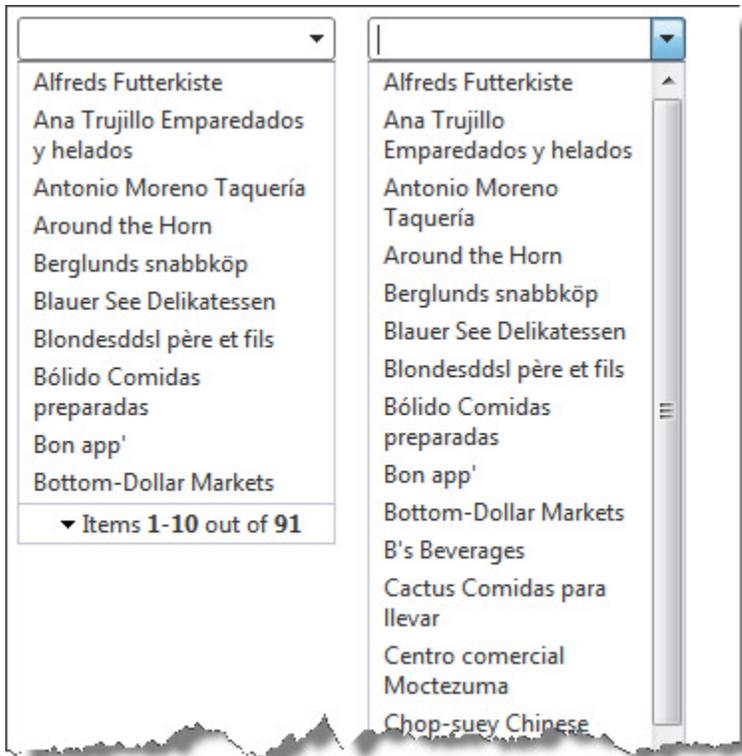
[ASP.NET] Combo box declaration

```
<telerik:RadComboBox ID="RadComboBox1" runat="server"
  Skin="Hay" DropDownWidth="200px" Height="200px"
  AllowCustomText="false" EnableVirtualScrolling="true"
  OnClientDropDownOpening="OnClientDropDownOpeningHandler"
  OnItemsRequested="RadComboBox1_ItemsRequested" >
</telerik:RadComboBox>
```



You can find the complete source for this project at:
 \VS Projects\ComboBox\HowToVirtualScrollNoCustomText

Implementing a Web service to load items on demand



Instead of implementing a server-side `ItemsRequested` event handler, the combo box can load items on demand from a Web service. To configure the combo box to use a Web service, set the `WebServiceSettings` property, giving the name of the Web Method and the path to the Web Service:

[ASP.NET] Combo box that uses a Web service for load-on-demand

```
<telerik:RadComboBox ID="RadComboBox1" Runat="server" Skin="Vista"
  EnableLoadOnDemand="True" ShowMoreResultsBox="True" >
  <WebServiceSettings Method="GetCompanyNames" Path="ComboWebService.asmx" />
</telerik:RadComboBox>
```

When implementing the Web Service, any Web Method that supports the load-on-demand mechanism must take a single parameter, which supplies context information about the request. This parameter is of type `RadComboBoxContext`, but it can also be cast to `IDictionary` to read context information that is added in an `OnClientItemsRequesting` event handler.



Gotcha! Because the context parameter is an `IDictionary` value, you cannot run the Web Service directly from within Visual Studio. When running the example, select `Default.aspx` before hitting F5 or Ctrl-F5.

You have a choice of two types for the return value:

- You can always return a value of type `RadComboBoxData`. This type includes an `Items` property that lists information about each item that is supplied. In addition, your Web Method can set the `Message` and `EndOfItems` properties if you are supporting virtual scrolling.
- If your Web Method does not support virtual scrolling, you can return an array of `RadComboBoxItemData` objects. (This is the type of the `Items` property of `RadComboBoxData`.)

The following example illustrates these different options. It implements a Web Service with two Web Methods, `GetCustomers` and `GetCompanyNames`. Both produce a list of items from the Northwind `Customers` table.



You can find the complete source for this project at:
 \VS Projects\ComboBox\HowToWebService

The Web Service itself includes an attribute to allow it to be called from AJAX script:

[VB] Enabling calls from AJAX script

```
<System.Web.Script.Services.ScriptService(>
```

[C#] Enabling calls from AJAX script

```
[System.Web.Script.Services.ScriptService]
```



Gotcha! Be sure to include the **ScriptService** attribute. If it is missing, all calls from the combo box to the Web Service will fail.

The first Web Method ("GetCustomers") expects the user to supply context information in an **OnClientItemsRequesting** handler: therefore, it casts the context parameter to **IDictionary**. Because it does not support virtual scrolling, it returns an array of **RadComboBoxItemData**.

[VB] Servicing requests that include extra context information

```
<WebMethod(> _
Public Function GetCustomers(ByVal context As RadComboBoxContext) As RadComboBoxItemData()
    ' create the connection
    Dim connection As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
    ' read the filter from context
    Dim filterString As String = (TryCast(context, IDictionary(Of String, Object)))
("FilterString").ToString()
    ' create a command, fetching customers that match filterString
    Dim selectCommand As New SqlCommand(" SELECT * FROM Customers WHERE CompanyName LIKE '" +
filterString + "%'", connection)
    ' fill a data table with the customers
    Dim adapter As New SqlDataAdapter(selectCommand)
    Dim customers As New DataTable()
    adapter.Fill(customers)
    ' create a list of RadComboBoxItemData to hold new items
    Dim result As New List(Of RadComboBoxItemData)(customers.Rows.Count)
    ' create an item for every row in the table
    For Each row As DataRow In customers.Rows
        Dim itemData As New RadComboBoxItemData()
        itemData.Text = row("CompanyName").ToString()
        itemData.Value = row("CompanyName").ToString()
        result.Add(itemData)
    Next
    ' convert the list to an array, and return it
    Return result.ToArray()
End Function
```

[C#] Servicing requests that include extra context information

```
[WebMethod]
public RadComboBoxItemData[] GetCustomers(RadComboBoxContext context)
```

```
{
    // create the connection
    SqlConnection connection = new SqlConnection(ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].ConnectionString);
    // read the filter from context
    string filterString = (context as IDictionary<string, object>)[ "FilterString" ].ToString();
    // create a command, fetching customers that match filterString
    SqlCommand selectCommand =
        new SqlCommand(@" SELECT * FROM Customers
            WHERE CompanyName LIKE '" + filterString + "%'", connection);
    // fill a data table with the customers
    SqlDataAdapter adapter = new SqlDataAdapter(selectCommand);
    DataTable customers = new DataTable();
    adapter.Fill(customers);
    // create a list of RadComboBoxItemData to hold new items
    List<RadComboBoxItemData> result = new List<RadComboBoxItemData>(customers.Rows.Count);
    // create an item for every row in the table
    foreach (DataRow row in customers.Rows)
    {
        RadComboBoxItemData itemData = new RadComboBoxItemData();
        itemData.Text = row["CompanyName"].ToString();
        itemData.Value = row["CompanyName"].ToString();
        result.Add(itemData);
    }
    // convert the list to an array, and return it
    return result.ToArray();
}
```

The second Web method second ("GetCompanyNames") supports virtual scrolling: it therefore returns a value of type **RadComboBoxItemData**. Because it does not rely on any extra context information, it uses a parameter of type **RadComboBoxContext**.

[VB] Servicing requests that use virtual scrolling

```
<WebMethod()> _
Public Function GetCompanyNames(ByVal context As RadComboBoxContext) As RadComboBoxData
    ' use the Text property of the context object to form the SELECT statement
    Dim sql As String = "SELECT * from Customers WHERE CompanyName LIKE '" + context.Text +
"%'"
    ' fill a data table using the SELECT statement
    Dim adapter As New SqlDataAdapter(sql, ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
    Dim data As New DataTable()
    adapter.Fill(data)
    ' declare the list to hold items we return
    Dim result As List(Of RadComboBoxItemData) = Nothing
    ' Create a RadComboBoxData object to pass virtual scrolling information
    Dim comboData As New RadComboBoxData()
    Try
        ' calculate the number of records we need
        ' based on the current number of items
        Dim itemsPerRequest As Integer = 10
        Dim itemOffset As Integer = context.NumberOfItems
        Dim endOffset As Integer = itemOffset + itemsPerRequest
        ' adjust endOffset if there are not enough rows
        If endOffset > data.Rows.Count Then
            endOffset = data.Rows.Count
        End If
    End Try
    Return result
End Function
```

```

End If
' set EndOfItems to reflect whether we are supplying the last items
If endOffset = data.Rows.Count Then
    comboData.EndOfItems = True
Else
    comboData.EndOfItems = False
End If
' create the list to hold the items we will return
result = New List(Of RadComboBoxItemData)(endOffset - itemOffset)
' iterate through the rows of the table, creating and adding items
Dim i As Integer = itemOffset
While i < endOffset
    Dim itemData As New RadComboBoxItemData()
    itemData.Text = data.Rows(i)("CompanyName").ToString()
    itemData.Value = data.Rows(i)("CompanyName").ToString()
    result.Add(itemData)
    System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
End While
' add a message for the more results box to the comboData
If data.Rows.Count > 0 Then
    comboData.Message = [String].Format("Items <b>1</b>-<b>{0}</b> out of <b>{1}</b>",
endOffset.ToString(), data.Rows.Count.ToString())
Else
    comboData.Message = "No matches"
End If
Catch e As Exception
    comboData.Message = e.Message
End Try
' add the list of items to the comboData and return it
comboData.Items = result.ToArray()
Return comboData
End Function

```

[C#] Servicing requests that use virtual scrolling

```

[WebMethod]
public RadComboBoxData GetCompanyNames(RadComboBoxContext context)
{
    // use the Text property of the context object to form the SELECT statement
    string sql = "SELECT * from Customers WHERE CompanyName LIKE '" + context.Text + "%'";
    // fill a data table using the SELECT statement
    SqlDataAdapter adapter = new SqlDataAdapter(sql,
        ConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString);
    DataTable data = new DataTable();
    adapter.Fill(data);
    // declare the list to hold items we return
    List<RadComboBoxItemData> result = null;
    // Create a RadComboBoxData object to pass virtual scrolling information
    RadComboBoxData comboData = new RadComboBoxData();
    try
    {
        // calculate the number of records we need
        // based on the current number of items
        int itemsPerRequest = 10;
        int itemOffset = context.NumberOfItems;
        int endOffset = itemOffset + itemsPerRequest;
    }
}

```

```

// adjust endOffset if there are not enough rows
if (endOffset > data.Rows.Count)
{
    endOffset = data.Rows.Count;
}
// set EndOfItems to reflect whether we are supplying the last items
if (endOffset == data.Rows.Count)
{
    comboData.EndOfItems = true;
}
else
{
    comboData.EndOfItems = false;
}
// create the list to hold the items we will return
result = new List<RadComboBoxItemData>(endOffset - itemOffset);
// iterate through the rows of the table, creating and adding items
for (int i = itemOffset; i < endOffset; i++)
{
    RadComboBoxItemData itemData = new RadComboBoxItemData();
    itemData.Text = data.Rows[i]["CompanyName"].ToString();
    itemData.Value = data.Rows[i]["CompanyName"].ToString();
    result.Add(itemData);
}
// add a message for the more results box to the comboData
if (data.Rows.Count > 0)
{
    comboData.Message = String.Format("Items <b>1</b>-<b>{0}</b> out of <b>{1}</b>",
endOffset.ToString(), data.Rows.Count.ToString());
}
else
{
    comboData.Message = "No matches";
}
}
catch (Exception e)
{
    comboData.Message = e.Message;
}
// add the list of items to the comboData and return it
comboData.Items = result.ToArray();
return comboData;
}

```

15.9 Summary

In this chapter you looked at the RadComboBox control and saw some of the powerful features it provides. You created a simple application that populated one combo box with statically declared items and another with items loaded from a data source. At the same time, you looked at some properties of the combo box and combo box items.

You looked at the design time support for the combo box and explored many of the properties and groups of properties you can use to configure the combo box at design time. You learned about the different types of template you can use with a combo box, and how to work with combo box custom attributes. You also learned about the load-on-demand mechanism, and saw how it can be used with virtual scrolling or a "More Results" box to improve performance.

You learned some of the server-side properties and methods, especially those for working with the items in the drop-down list. You looked at some of the important server-side events, including those that respond when the selected text changes or that service the load-on-demand mechanism. You learned when and how to sort the drop-down list in server-side code.

You explored some of the client-side methods for working with the items collection, and used some of the important client-side events, including those for responding to selection changes, opening and closing the drop-down list, and the events surrounding the load-on-demand mechanism.

Finally, you learned some advanced techniques, including implementing custom sort criteria, keeping the drop-down list open when an item template includes input controls, controlling when the load-on-demand mechanism fetches items, enabling virtual scrolling when not allowing custom text, and creating a Web service for loading items on demand.

16 RadTreeView

16.1 Objectives

- Explore the features of the **RadTreeView** control.
- Create a simple application to develop confidence in using the tree view and to see how to build a node hierarchy either statically or using data supplied from a data source.
- Explore the tree view design time interface, including Smart Tag, Properties Window, Property Builder, Collection Editors and Template Design surface.
- Explore principal properties and groups of properties where most of the functionality is found.
- Learn to use special tree view features such as node editing, check boxes, drag-and-drop, and node context menus.
- Learn server-side coding techniques, including traversing the node hierarchy to make a change to all parent nodes or all child nodes, building the tree view dynamically in code to accommodate data from multiple database tables, and handling server-side events.
- Explore some of the client-side methods of the tree node and tree view client-side objects, learn to implement the 'radio button' pattern for state changes, and learn to expand the number of client side events you can respond to by accessing the DOM object for the tree view.
- Learn to wrap the text that appears on tree view nodes and to add controls directly to tree nodes without using templates.
- Explore the different options for using load-on-demand to improve performance.

16.2 Introduction

You are undoubtedly aware of tree view controls, which appear in many desktop applications. For example, the Solution Explorer in Visual Studio is a tree view control that lets you navigate the components of your project and perform basic operations such as adding and deleting new components. **RadTreeView** lets you add the same capability to your Web applications. It displays a hierarchy of items that you can declare statically or load from a data source.



RadTreeView combines the highly efficient rendering of RadControls for ASP.NET AJAX with a powerful set of features. You can use the familiar skinning capabilities to make the tree view fit in with the look and feel of

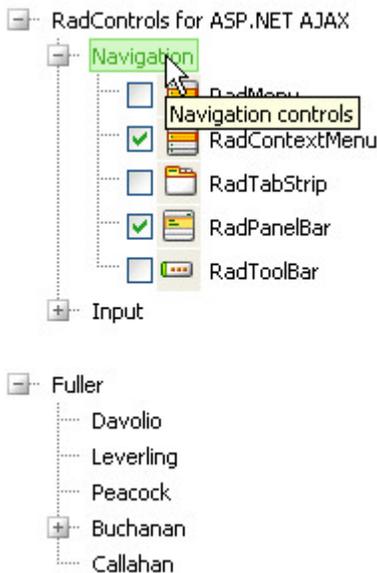
your Web site, as well adjusting the appearance using styles or item templates. You can add images to the nodes without using templates, and even specify different images depending on the state of the node.

The capabilities of RadTreeView extend beyond simply changing the appearance. You can configure the tree view to allow multiple nodes to be selected at a time. The tree view can be augmented with check boxes on nodes, or configured to allow users to edit the nodes. There is built-in support for adding context menus to nodes, based on the **RadContextMenu** control you learned about in the chapter on Navigation controls. RadTreeView also supports drag-and-drop operations: you can let users drag nodes to new positions in the tree view, to another tree view, or to a another element on the page of an entirely different type. If your tree view has many items, you can improve performance by using its load-on-demand mechanism, similar to the load-on-demand feature you saw with **RadComboBox**.

In addition, RadTreeView offers the rich client-side API that you can expect from all RadControls, and a wealth of server-side events and methods.

16.3 Getting Started

In this walk-through you will become familiar with the **RadTreeView** control. You will create two tree views: one with statically declared nodes, and one with nodes loaded from a database.



You can find the complete source for this project at:
 \VS Projects\TreeView\GettingStarted

Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Using the Solution Explorer, add a new Folder to your project and name it "Images".
3. From the "\VS Projects\Images\Controls" folder drag the "RadMenu.png", "RadPanelBar.png", "RadTabStrip.png", "RadTextBox.png" and "RadToolBar.png" files to your project's **Images** folder.
4. Locate the "Northwind.mdf" file and drag it into the "App_Data" folder of your project.
5. Open the "Web.config" file of your project. Add the standard Northwind connection string to your project by replacing the line

```
<connectionStrings />
with
  <connectionStrings>
    <add name="NorthwindConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|Northwind.mdf;Integrated
Security=True;User Instance=True" providerName="System.Data.SqlClient" />
  </connectionStrings>
```

Add a tree view with statically declared nodes

1. Drag a `RadTreeView` control from the Tool Box onto your Web page. Set its `Skin` property to "Telerik".
2. From the `RadTreeView` Smart Tag, choose **Build RadTreeView...** to bring up the **RadTreeView Item Builder**. The Item Builder should look familiar: you have seen similar Item Builders on the navigation controls.
3. In the Item Builder, click the "Add root item" button to add a root node to the tree view.
4. Using the properties pane on the right,
 - Set the `Text` property to "RadControls for ASP.NET AJAX".
 - Set the `Checkable` property to false.
5. Click the "Add child item" button again to add a child node to the root node. Set its properties as follows:
 - Set the `Text` property to "Navigation".
 - Set the `Checkable` property to false.
 - Set the `ToolTip` property to "Navigation controls".
6. With the "Navigation" node selected, click the "Add child item" button five times to add five child nodes.
 - On the first, set the `Text` property to "RadMenu" and the `ImageUrl` property to "~/Images/RadMenu.png".
 - On the second, set the `Text` property to "RadContextMenu" and the `ImageUrl` property to "~/Images/RadMenu.png".
 - On the third, set the `Text` property to "RadTabStrip" and the `ImageUrl` property to "~/Images/RadTabStrip.png".
 - On the fourth, set the `Text` property to "RadPanelBar" and the `ImageUrl` property to "~/Images/RadPanelBar.png".
 - On the fifth, set the `Text` property to "RadToolBar" and the `ImageUrl` property to "~/Images/RadToolBar.png".
7. Select the root item and click the "Add child item" button to add a second child node to the root node.
 - Set the `Text` property to "Input".
 - Set the `Checkable` property to false.
 - Set the `ToolTip` property to "Input controls".
8. With the "Input" node selected, click the "Add child item" button four times to add four child nodes.
 - On the first, set the `Text` property to "RadTextBox" and the `ImageUrl` property to "~/Images/RadTextBox.png".
 - On the second, set the `Text` property to "RadMaskedTextBox" and the `ImageUrl` property to "~/Images/RadTextBox.png".
 - On the third, set the `Text` property to "RadNumericTextBox" and the `ImageUrl` property to "~/Images/RadTextBox.png".

- On the fourth, set the **Text** property to "RadDateInput" and the **ImageUrl** property to "~/Images/RadTextBox.png".
9. Click **OK** to exit the Item Builder.
 10. Using the Properties Window, set the **CheckBoxes** and **SingleExpandPath** properties to true.

Add a data-bound tree view

1. In the designer, hit the Enter key to add a line break, and then drag a second **RadTreeView** control from the Tool Box onto your Web page. Set its **Skin** property to "Telerik".
2. In the RadTreeView Smart Tag, select "<New data source...>" from the **Choose Data Source** drop-down.
3. In the first page of the **Data Source Configuration Wizard**, select "Database" as the application type, and click **OK** to move to the next page.
4. On the **Choose Your Data Connection** page, select "NorthwindConnectionString" from the drop-down list. Then click the **Next** button to continue.
5. On the **Configure the Select Statement** page, make sure the "Specify columns from a table or view" radio button is selected, and then choose "Employees" from the "Name" drop-down list.
6. Check the "EmployeeID", "LastName" and "ReportsTo" fields. Then click the **Next** button to continue.
7. Test the query if you wish, and then click **Finish**.
8. In the Properties Window for the second tree view,
 - Set the **DataTextField** property to "LastName".
 - Set the **DataFieldID** property to "EmployeeID".
 - Set the **DataFieldParentID** property to "ReportsTo".

Run the application

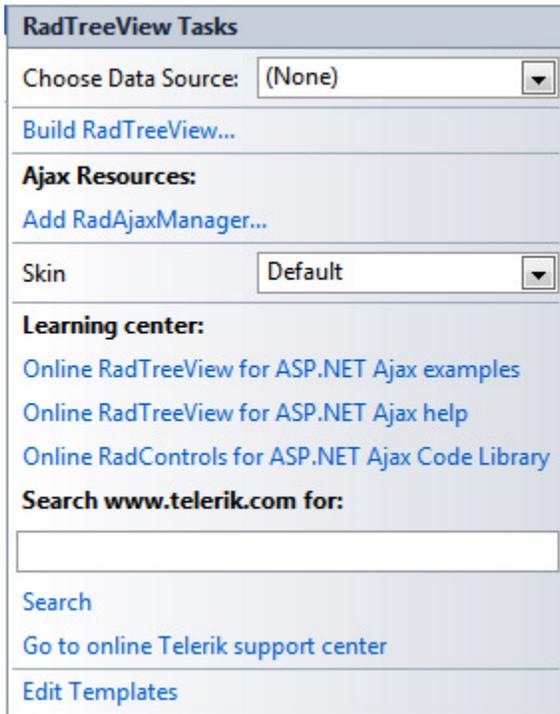
1. Press Ctrl-F5 to run the application.
2. Click the "+" buttons to expand nodes on the tree view controls. On the first tree view, note that only the leaf items have check boxes. This is because the **Checkable** property of the other nodes was set to false. Note that you can't expand both the "Navigation" and "Input" nodes at the same time. This is because of the **SingleExpandPath** property.
3. On the second tree view, note that the items form a hierarchy, although they all came from the same table. This hierarchy is built using the **DataFieldID** and **DataFieldParentID** properties of items.

16.4 Designer Interface

In the Visual Studio designer, you can configure the RadTreeView control using the Smart Tag, the Properties Window, and the RadTreeView Item Builder. In addition, you can add data bindings using the NavigationItemBinding collection editor, add context menus using the RadTreeViewContextMenu collection editor, and add templates using the Template Design surface.

Smart Tag

The RadTreeView Smart Tag looks like the typical Smart Tag of a RadControl that contains items which can be either statically declared or loaded from a data source:



At the top of the Smart Tag, the **Choose Data Source** drop-down to bind the tree view and the **Build RadTreeView** link to bring up the Item Builder should be familiar by now. So should the standard **Ajax Resources**, **Skin Selection**, and **Learning Center** items. Because you can define item templates for the tree view, there is also an **Edit Templates** link to bring up the Template Design Surface.

If you bind the tree view to a data source, the Smart Tag changes to its bound version:



The bound Smart Tag lets you change the data source, reconfigure the current data source, or refresh the schema. In addition, there is a link to bring up the **NavigationItemBinding Collection Editor**. This collection should be familiar to you from the Data Binding chapter.

The bound Smart Tag still contains the Edit Templates item to bring up the Template Design Surface.

Properties Window

At design time, you can use the Properties Window to configure almost every aspect of the tree view, with the exception of templates. As before, let us look at the most important properties.

Specifying Items

Probably the most important property of the tree view is the one that specifies what items appear and their hierarchical relationships. What properties you choose for this task depends on whether you want to load items from a data source:

- If you want to load items from a data source, you can use the the standard data-binding properties

(**DataSourceID** and **DataMember**), or use the **DataSource** property and **DataBind** method in the code-behind. When binding **RadTreeView** to a data source, you can use the **DataTextField**, **DataTextFormatString**, **DataValueField**, and **DataNavigationUrlField** properties to map fields from the data source to properties of the nodes, or use the **DataBindings** property to map even more node properties.

- If you want to establish a hierarchical relationship between nodes, use the **DataFieldID** and **DataFieldParentID** properties. When setting up a hierarchy in this way, you can use the **MaxDataBindDepth** property to limit the depth of the hierarchy.
- When using an inherently hierarchical data source such as an **XmlDataSource** or **SiteMapDataSource**, there is no need to use the **DataFieldID** and **DataFieldParentID** properties. The hierarchy is automatically honored by the tree view.
- If you want to use statically declared items, you can use the **Nodes** property to bring up the **RadTreeView Item Builder**, or you can switch to the Source view and define the structure directly in the mark-up.
- If you want to use both data-bound and statically-declared items, set the **AppendDataBoundItems** property to true.

Other tree view properties

In addition to the **Skin** property, you can affect the look-and-feel of the tree view by setting the **ShowLineImages** property to control whether the tree view displays lines connecting its nodes, and setting the **ExpandAnimation** and **CollapseAnimation** properties to specify animated effects when the nodes are expanded or collapsed.

Basic behavior is controlled by the **MultipleSelect** property, which specifies whether the user can select more than one node at a time, and the **SingleExpandPath**, which allows only one node at any level to be expanded at a time.

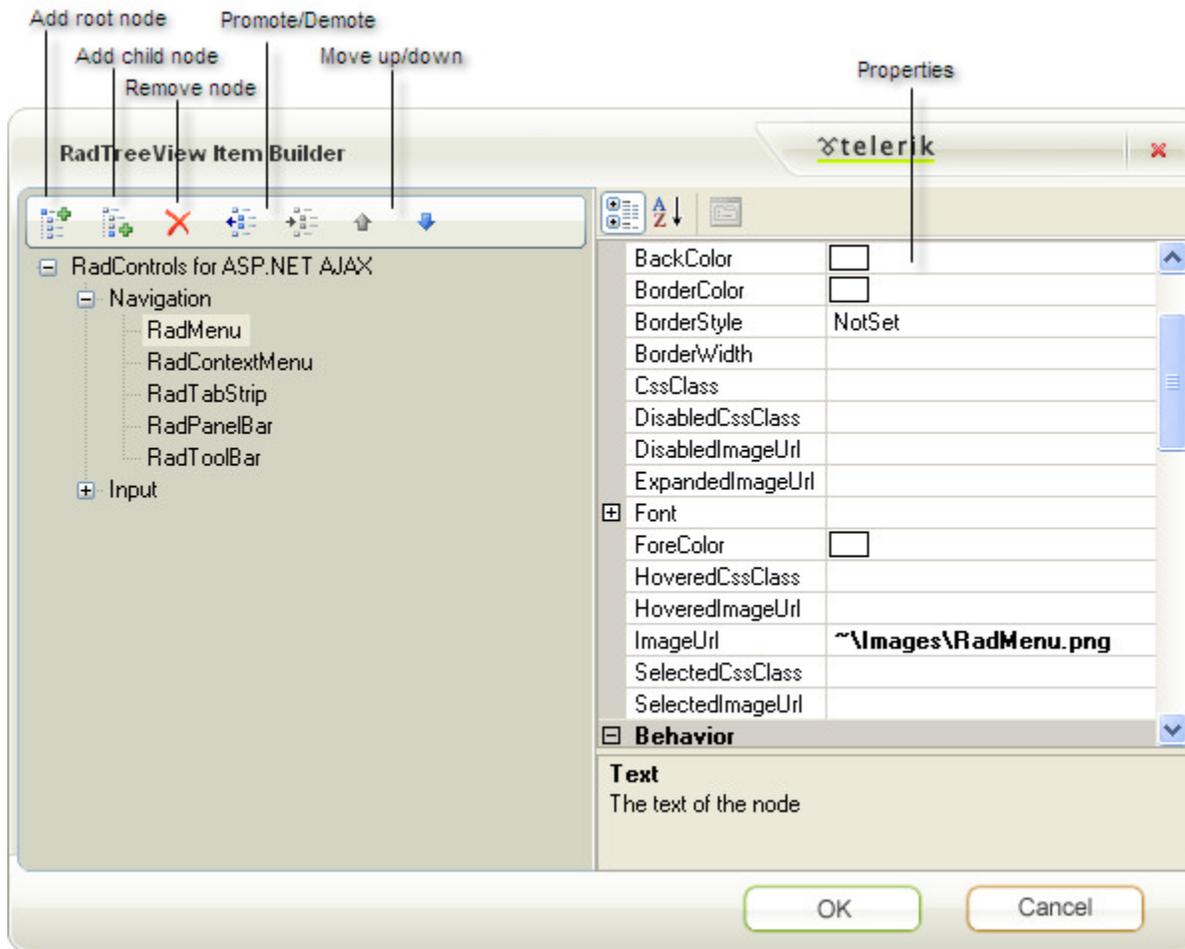
In addition, a few properties can be used to enable or disable **RadTreeView** features.

- The **AllowNodeEditing** property makes nodes editable. When **AllowNodeEditing** is true, users can edit the text of nodes by clicking on a node a second time once it is selected (the same way you can edit the item text in **RadControl** Item Builder dialogs).
- The **CheckBoxes** property adds check boxes to the nodes. The check boxes feature is described in more detail later in this chapter.
- The **EnableDragAndDrop** property allows the user to drag nodes and drop them on other nodes, or on other elements of the Web page. The drag-and-drop feature is also described in more detail later.
- The **ContextMenus** property lets you bring up the **RadTreeViewContextMenus** Collection Editor, where you can define the context menus that are used by items.

RadTreeVeiv Item Builder

RadTreeView lets you edit the list of statically defined nodes using the **RadTreeView Item Builder**. This item builder is very similar to the hierarchical Property Builder dialogs you looked at in the chapter on Navigation controls. Display the item builder either from the Smart Tag or by clicking the ellipsis button on the **Nodes** property in the Properties Window.

Below is a screen shot of the **RadTreeView** Item Builder. Use the buttons on the upper left to build or edit the node hierarchy. You can select any of the nodes and set its properties using the properties pane on the right of the dialog. Typically, you will set the **Text** property first.



Each node has its own set of properties: **Text** is the string that represents the node, **ToolTip** is the tool tip for the node, and **Value** is a value associated with the node. You can add images to a node by setting the **ImageUrl**, **DisabledImageUrl**, **ExpandedImageUrl**, **HoveredImageUrl** and **SelectedImageUrl** properties. If any of the other image properties is not set, the node uses the **ImageUrl** property as the image default. You may also want to use the **Selected**, **Enabled**, and **Expanded** properties to specify the state of the item when the Web page first loads. The **NavigateUrl** and **Target** properties let you use the node to navigate to another Web page.

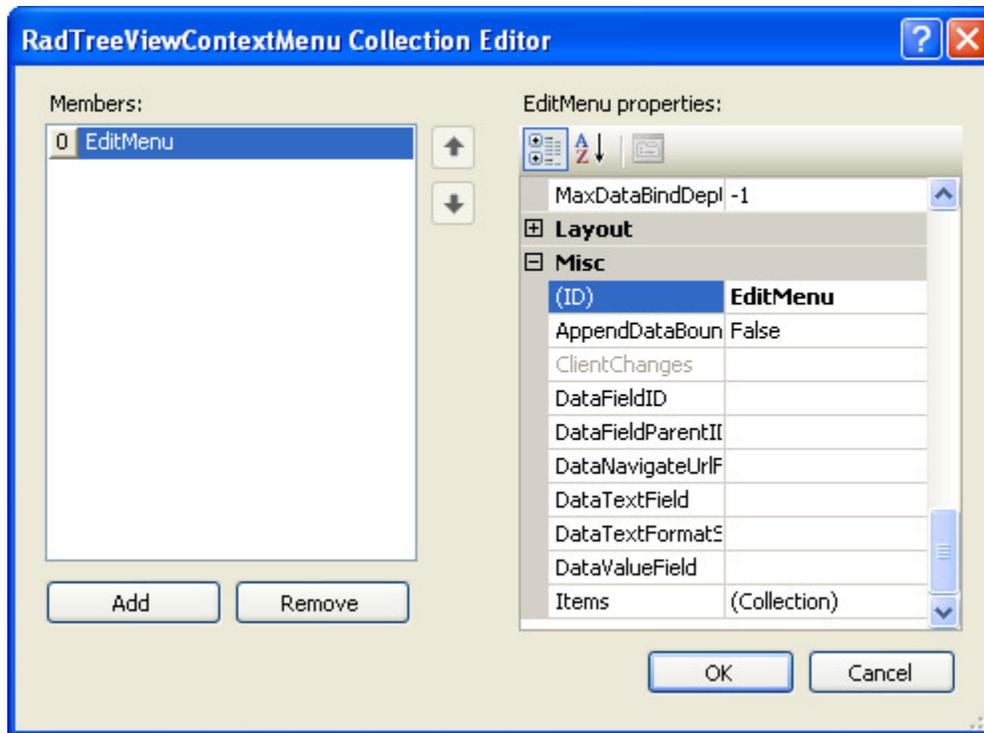
A few properties determine whether the node participates in specific features that are enabled at the tree view level:

- The **AllowEdit** property controls whether the node can be edited when the tree view's **AllowNodeEditing** property is true.
- The **Checkable** property controls whether the node gets a check box when the tree view's **CheckBoxes** property is true. The **Checked** property specifies whether the check box is checked when the page first loads.
- The **AllowDrag** and **AllowDrop** properties specify how the node participates in the drag-and-drop feature when the tree view's **EnableDragAndDrop** property is true.
- The **EnableContextMenu** property specifies whether the node displays a context menu when the user right clicks and the **ContextMenus** property collection defines a set of context menus. **ContextMenuID** specifies which context menu to use.

Collection Editors

RadTreeView uses two associated collection editors, the **NavigationItemBinding Collection Editor**, which is used to edit the **DataBindings** property collection, and the **RadTreeViewContextMenu Collection Editor**, which is used to edit the **ContextMenus** property collection.

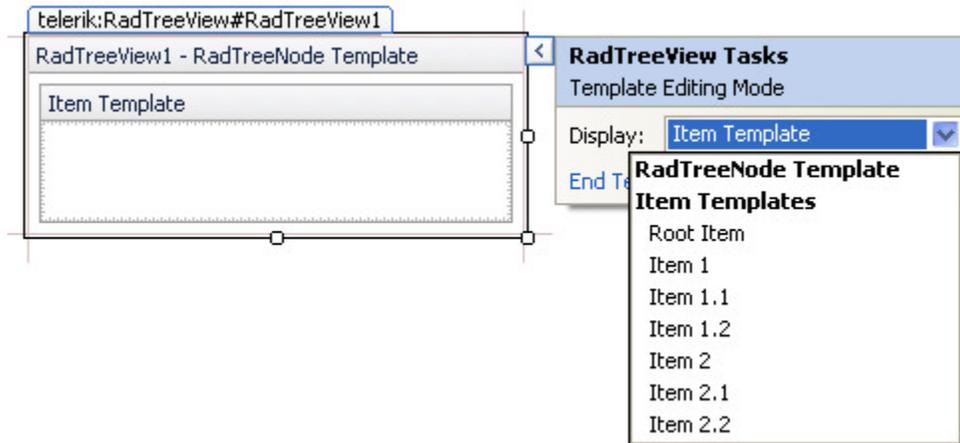
You have already seen how the NavigationItemBinding Collection Editor works in the Data Binding chapter. Let us look briefly at the **RadTreeViewContextMenu Collection Editor**.



When using the RadTreeViewContextMenu Collection Editor, the order of items does not matter, because each node is associated with a context menu by its **ContextMenuID** property. To make this work, you must take note of the **ID** property of each context menu in the collection, as this is the string that should be assigned to the **ContextMenuID** property. Another important property is the **Items** property, which can be used to bring up the **RadContextMenu Item Builder**, where you can define the items in the context menu. You may also wish to set the **Skin** property for the menu, as this is not inherited from the tree view.

Template Design surface

You can use the tree view's Smart Tag or context menu to bring up the Template Design surface, where you can create item templates. RadTreeView supports two types of template: a global RadTreeNode template that affects all nodes in the tree view, and individual item templates, which are associated with specific nodes in the Nodes collection. A drop-down control on the RadTreeView Smart Tag (when it is in template editing mode) lets you specify which template you want to edit:



When a tree view includes both a RadTreeNode template and individual item templates, the item templates have priority over the RadTreeNode template. That is, the RadTreeNode template is used for every node that does not have its own item template.

16.5 Control Specifics

Check boxes

When the **CheckBoxes** property of RadTreeView is true, the tree view adds check boxes to any nodes that have the **Checkable** property set to true (the default). For any node with Checkable set to true, the **Checked** property indicates its state.

In the code-behind, you can check all of the checkable nodes in a tree view by calling its **CheckAllNodes** method. Similarly, you can set all nodes to the unchecked state by calling the **ClearCheckedNodes** method. You can quickly find all of the checked nodes in a tree view by reading the **CheckedNodes** property.



Gotcha! Do not try checking nodes in the code-behind by adding them to the **CheckedNodes** property collection. Instead, check nodes by setting their **Checked** property to true.

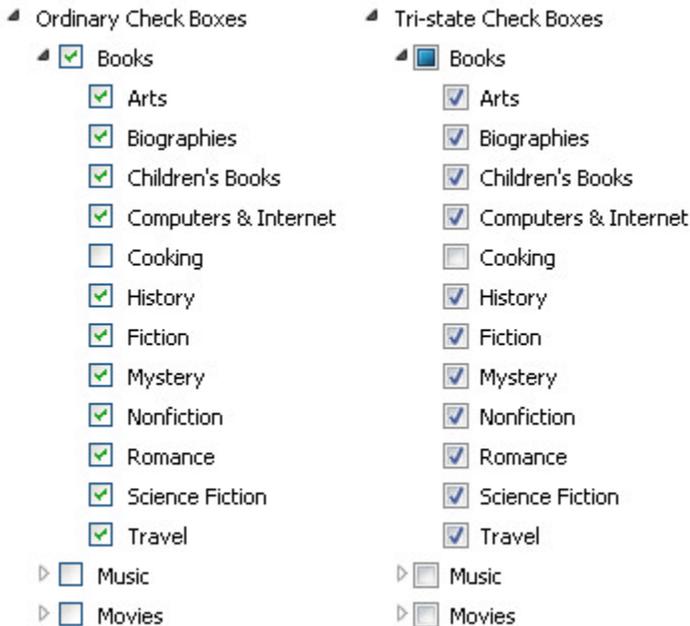
By default, all check boxes behave independently. However, in some applications, you may want to implement a cascading effect. That is, when a parent node is checked, all of its children are automatically checked as well, and when a parent node is unchecked, all of its children inherit that state. Achieving this effect is simple: just set the **CheckChildNodes** property to true.

You can convert the check boxes in the tree view from ordinary two-state check boxes to tri-state check boxes by setting the **TriStateCheckBoxes** property to true. Tri-state check boxes have three states: "Checked", "Unchecked", and "Indeterminate". The state of a tri-state check box is intended to reflect the state of the check boxes on child nodes. When all the child nodes of a tri-state check box are checked, the tri-state check box is checked. When they are all unchecked, the tri-state check box is unchecked. When the child nodes are a mix of checked and unchecked nodes, the tri-state check box is in an indeterminate state. Changing the checked state of a child node automatically changes the state of all parent tri-state check boxes.

Because of the relationship between the child nodes and parent nodes, set the **CheckChildNodes** property to true when using tri-state check boxes. Otherwise, changing the checked state of parent nodes does not make much sense. After a postback, the state will always revert to one that reflects the checked state of child nodes.

The following walk-through explores some of the features of RadTreeView check boxes. It creates a Web page with two tree views on it, one with ordinary check boxes, and one with tri-state check boxes. Both have the **CheckChildNodes** property set to true so that child nodes are updated when the parent node is checked. In addition, an event handler automatically expands or contracts the child nodes of a node when its check box is

checked or unchecked.



You can find the complete source for this project at:
 \VS Projects\TreeView\CheckBoxes

Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Drag an HTML table from the Tool Box onto the Web page and delete rows and columns until it has a single row with two cells. Set the **valign** attribute of both cells to "top". Set the **Width** attribute of the first cell to "200px".
3. Using the Solution Explorer, add an XML file to the project named "Entertainment.xml".
4. Add the following XML to the "Entertainment.xml" file.

Entertainment.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<Categories>
  <Category Name="Books">
    <SubCategory Name="Arts" />
    <SubCategory Name="Biographies" />
    <SubCategory Name="Children's Books" />
    <SubCategory Name="Computers & Internet" />
    <SubCategory Name="Cooking" />
    <SubCategory Name="History" />
    <SubCategory Name="Fiction" />
    <SubCategory Name="Mystery" />
    <SubCategory Name="Nonfiction" />
    <SubCategory Name="Romance" />
    <SubCategory Name="Science Fiction" />
    <SubCategory Name="Travel" />
  </Category>

```

```
<Category Name="Music">
  <SubCategory Name="Alternative" />
  <SubCategory Name="Blues" />
  <SubCategory Name="Children's Music" />
  <SubCategory Name="Classical" />
  <SubCategory Name="Country" />
  <SubCategory Name="Dance" />
  <SubCategory Name="Folk" />
  <SubCategory Name="Hard Rock" />
  <SubCategory Name="Jazz" />
  <SubCategory Name="Soundtracks" />
</Category>
<Category Name="Movies">
  <SubCategory Name="Action" />
  <SubCategory Name="Animation" />
  <SubCategory Name="Classics" />
  <SubCategory Name="Comedy" />
  <SubCategory Name="Documentary" />
  <SubCategory Name="Drama" />
  <SubCategory Name="Horror" />
  <SubCategory Name="Musicals" />
  <SubCategory Name="Mystery" />
  <SubCategory Name="Westerns" />
</Category>
</Categories>
```

5. Save "Entertainment.xml" and return to the designer for your default Web page.

Add the first tree view

1. Drag a **RadAjaxPanel** from the Tool Box into the first cell of the table. Remove the values of the **Width** and **Height** properties.
2. Drag a **RadTreeView** control from the Tool Box onto the RadAjaxPanel. Set its **Skin** property to "Vista".
3. In the RadTreeView Smart Tag, select "<New data source...>" from the **Choose Data Source** drop-down. The Data Source Configuration Wizard appears.
4. In the Data Source Configuration Wizard, select "XML File" and click OK to continue.
5. On the Configure Data Source page, set the **Data File** to "~/Entertainment.xml", and click OK to exit.
6. In the RadTreeView Smart Tag, select "Edit RadTreeView Data Bindings...".
7. In the NavigationItemBinding Collection Editor, add two data bindings.
 1. On the first, set the **TextField** property to "Name".
 2. On the second, set the **Depth** property to 0 and the **Text** property to "Ordinary Check Boxes".
8. Using the Properties Window, set the **CheckBoxes** and **CheckChildNodes** properties to true. At this point, the RadTreeView declaration should look like the following:

[ASP.NET] RadTreeView with CheckBoxes and CheckChildNodes

```
<telerik:RadTreeView ID="RadTreeView1" runat="server"
  Skin="Vista" DataSourceID="XmlDataSource1"
  CheckBoxes="True" CheckChildNodes="True">
  <DataBindings>
    <telerik:RadTreeNodeBinding TextField="Name" />
    <telerik:RadTreeNodeBinding Depth="0" Text="Ordinary Check Boxes" />
  </DataBindings>
```

```
</telerik:RadTreeView>
```

Add the second tree view

1. Copy the **RadAjaxPanel** in the first cell of the table and paste it into the second cell.
2. Move to the Source view.
3. Change the **DataSourceID** property of the second tree view to "XmlDataSource1" and delete the second data source component.
4. Change the **Text** property on the second **RadTreeNodeBinding** from "Ordinary Check Boxes" to "Tri-state Check Boxes".
5. Return to Design view.
6. Using the Properties Window, set the **TriStateCheckBoxes** property of the second tree view to true.

Add event handlers

1. To remove the check box from the root node and start it in the expanded state, add the following **DataBound** event handler to both tree views. Be sure to add an **Imports** or **using** statement for "Telerik.Web.UI" first!

[VB] Adjusting the root node

```
Protected Sub RadTreeView1_DataBound(ByVal sender As Object, ByVal e As EventArgs) _
    Handles RadTreeView1.DataBound, RadTreeView2.DataBound
    Dim node As RadTreeNode = (TryCast(sender, RadTreeView)).Nodes(0)
    node.Checkable = False
    node.Expanded = True
End Sub
```

[C#] Adjusting the root node

```
protected void RadTreeView1_DataBound(object sender, EventArgs e)
{
    RadTreeNode node = (sender as RadTreeView).Nodes[0];
    node.Checkable = false;
    node.Expanded = true;
}
```

2. To respond when the user checks or unchecks a node in the tree view, add the following **NodeCheck** event handler to both tree views:

[VB] Expanding or contracting a node when checked or unchecked

```
Protected Sub RadTreeView1_NodeCheck(ByVal sender As Object, ByVal e As
RadTreeNodeEventArgs) _
    Handles RadTreeView1.NodeCheck, RadTreeView2.NodeCheck
    If e.Node.Nodes.Count > 0 Then
        e.Node.Expanded = e.Node.Checked
    End If
End Sub
```

[C#] Expanding or contracting a node when checked or unchecked

```
protected void RadTreeView1_NodeCheck(object sender, RadTreeNodeEventArgs e)
{
    if (e.Node.Nodes.Count > 0)
        e.Node.Expanded = e.Node.Checked;
}
```

Run the application

1. Press Ctrl-F5 to run the application.
2. Note the difference between the appearance of the ordinary check boxes and the tri-state check boxes. The ordinary check boxes are standard HTML `<input>` elements of type "checkbox", while the tri-state check boxes are rendered as `` elements. (This means that you can change the appearance of the ordinary check boxes using `RadFormDecorator`, but not the tri-state check boxes.)
3. Check one of the check boxes in each tree view. Note that the node automatically expands because of the `NodeCheck` event handler. All child nodes are checked because of the `CheckChildNodes` property.
4. Uncheck one of the leaf nodes. Note that this effects only the child node when using ordinary check boxes, but changes the parent to an "indeterminate" state when using tri-state check boxes.

Drag-and-Drop

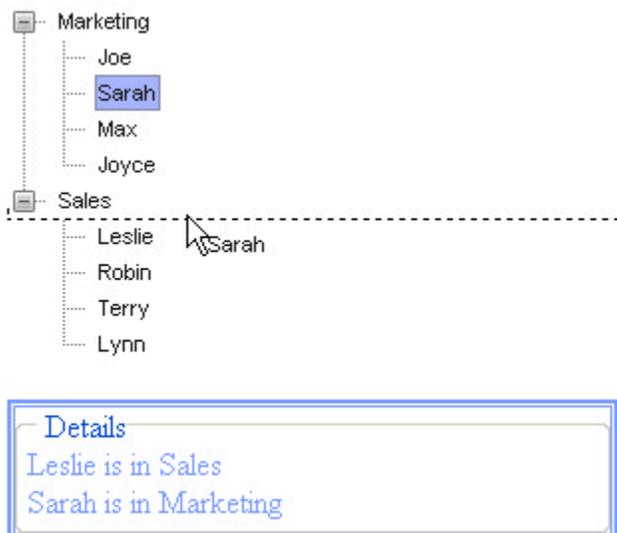
To enable the drag-and-drop feature, set the `EnableDragAndDrop` property of the tree view control to true. When drag-and-drop is enabled, the user can drag any node that has its `AllowDrag` property set to true and drop it on any node that has its `AllowDrop` property set to true. The node can also be dropped on any element on the page that has its `id` attribute set, including the nodes of other tree views, as long as the other tree view has `EnableDragAndDrop` set to true and the node has `AllowDrop` set to true.

✍ If the `MultipleSelect` property of the tree view is set to true, the user can drag multiple nodes at a time.

By default, when the user drops a node onto a tree view, it can only be dropped on another node. If you want to let the user drop a node between two other nodes, set the `EnableDragAndDropBetweenNodes` property to true. When `EnableDragAndDropBetweenNodes` is true and the user drags a node between two nodes on the tree view, the tree view displays a line where the node will land if dropped.

When the user drops a node (or nodes) on a valid drop location (a node with `AllowDrop` set to true or an element with an `id` attribute), the server-side `NodeDrop` event handler is called. This is where you must handle the drop event, implementing whatever changes the action implies.

The following example illustrates how to implement drag-and-drop. It contains a tree view control with drag-and-drop enabled and an ASP.NET panel. When the user drags a node from one position in the tree view to another, the node is removed from its old position and added to the new one. When the user drags a node from the tree view to the panel, a label inside the panel is updated to indicate where the node came from.



You can find the complete source for this project at:
`\VS Projects\TreeView\DragAndDrop`



To enable drag-and-drop, the tree view has **EnableDragAndDrop** set to true. It also has **EnableDragAndDropBetweenNodes** set to true, to allow nodes to be dragged between other nodes, as shown in the screen shot above.

The **NodeDrop** event occurs when a drag operation ends on a valid target:

[VB] NodeDrop event handler

```
Protected Sub RadTreeView1_NodeDrop(ByVal sender As Object, _
    ByVal e As RadTreeNodeDragDropEventArgs) _
    Handles RadTreeView1.NodeDrop
    Dim sourceNode As RadTreeNode = e.SourceDragNode
    Dim destNode As RadTreeNode = e.DestDragNode
    ' don't allow a node to be dropped on itself
    If sourceNode.Equals(destNode) Then
        Return
    End If
    ' destNode is set if dropping on another node of the tree
    If Not destNode Is Nothing Then
        ' remove the node from its current parent
        sourceNode.Owner.Nodes.Remove(sourceNode)
        If destNode.Level = 0 Then
            ' dropping on a group node
            ' insert in the first position
            destNode.Nodes.Insert(0, sourceNode)
        Else
            ' dropping on a leaf node
            Select Case e.DropPosition
                Case RadTreeViewDropPosition.Over, RadTreeViewDropPosition.Below
                    ' add after
                    destNode.InsertAfter(sourceNode)
                Exit Select
                Case RadTreeViewDropPosition.Above
                    ' add before
                    destNode.InsertBefore(sourceNode)
                Exit Select
            End Select
        End If
    ElseIf e.HtmlElementID = "Panel1" Then
        ' node was dropped on the panel
        If Label1.Text <> "" Then Label1.Text += "<br>"
        Label1.Text += String.Format("{0} is in {1}", sourceNode.Text,
sourceNode.ParentNode.Text)
    End If
End Sub
```

[C#] NodeDrop event handler

```
protected void RadTreeView1_NodeDrop(object sender, RadTreeNodeDragDropEventArgs e)
{
    RadTreeNode sourceNode = e.SourceDragNode;
    RadTreeNode destNode = e.DestDragNode;
    // don't allow a node to be dropped on itself
    if (sourceNode.Equals(destNode)) return;
    // destNode is set if dropping on another node of the tree
```

```
if (destNode != null)
{
    // remove the node from its current parent
    sourceNode.Owner.Nodes.Remove(sourceNode);
    if (destNode.Level == 0) // dropping on a group node
    {
        // insert in the first position
        destNode.Nodes.Insert(0, sourceNode);
    }
    else // dropping on a leaf node
    {
        switch (e.DropPosition)
        {
            case RadTreeViewDropPosition.Over:
            case RadTreeViewDropPosition.Below:
                // add after
                destNode.InsertAfter(sourceNode);
                break;
            case RadTreeViewDropPosition.Above:
                // add before
                destNode.InsertBefore(sourceNode);
                break;
        }
    }
}
// node was dropped on the panel
else if (e.HtmlElementID == "Panel1")
{
    if (Label1.Text != "")
        Label1.Text += "<br>";
    Label1.Text += string.Format("{0} is in {1}", sourceNode.Text,
sourceNode.ParentNode.Text);
}
}
```

The event handler first checks that a node is not being dropped on itself, as that would have no impact. It then looks at the **DestDragNode** argument, which is set when the node is dropped on another node of the tree view (including between nodes). If the destination node is one of the two group nodes, the dragged node (**e.SourceDragNode**) is added to the beginning of the destination node's collection of nodes. If the destination node is another leaf node, the event handler checks the **DropPosition** argument to determine whether the node was dropped above, below, or on the destination node, and then adds it to the parent node in the indicated position.

If the destination node is not set, then the node was dragged onto an element with an id (the panel). In that case, the event handler checks the **HtmlElementID** argument to verify that the node was dropped on the panel, and if so, updates a label inside the panel to display the dropped node's text and its parent node's text.

 This example uses a tree view with **MultipleSelect** set to false. If **MultipleSelect** were true, the event handler would need to use the **DraggedNodes** argument instead of the **SourceDragNode** argument.

If the application were run with only the server-side **NodeDrop** handler, it would work for nodes that were dropped inside the tree view, but not for nodes dropped on the panel. This is because the panel is not rendered as a single element with the id "Panel1". It is rendered as a pair of nested <div> elements with a <fieldset> element inside them. What looks like dropping on the panel is actually dropping on the <fieldset> element. Because the <fieldset> element does not have an id, it is not a valid drop target, and the **NodeDrop** handler never gets called.

To allow the **NodeDrop** handler to get called for the panel, the application uses a client-side

OnClientNodeDropping event handler. This client-side event always occurs, regardless of whether the target is valid.

[JavaScript] OnClientNodeDropping handler

```
function nodeDropping(sender, args) {
    // set target to the element on which the node is dropped
    var target = args.get_htmlElement();
    // check whether target is in the panel or tree view
    // by working up the parent chain to a known element
    while (target) {
        var targetID = target.id;
        var className = target.className;
        // we reached the tree view -- this is a good target
        if (targetID == "RadTreeView1")
            return;
        // the "between nodes" lines are not actually in the tree view,
        // but they have class names that begin "rtDrop"
        else if (className.startsWith("rtDrop"))
            return;
        // we are inside the panel -- this is a good target
        else if (targetID == "Panel1") {
            args.set_htmlElement(target);
            return;
        }
        target = target.parentNode;
    }
    // we were not in a good target, cancel the drop
    args.set_cancel(true);
}
```

This event handler starts with the element that is the actual drop target and moves up the change of parent elements until it reaches the tree view, the panel, or runs out of parent elements. The tree view and panel elements can be recognized by their client-side **id**. Because the tree view has **EnableDragAndDropBetweenNodes** set to true, the event handler must also recognize a drop between nodes. This lands on a <div> element that is not actually inside the tree view. To recognize this element, the event handler must use the class name, which begins with the string "rtDrop".

When the event handler detects that the drop target is in the tree view (or between lines), it simply exits, because the **NodeDrop** event will be correctly called by default. If the drop target is inside the panel, the event handler changes the drop target to the <div> with the proper **id**, so that the **NodeDrop** event handler can be called.

Context Menus

To add context menus to the nodes of the tree view, you must first add the context menus to the **ContextMenu** property collection of the tree view. Then, assign the **ContextMenuID** property of each node to the **ID** of the desired context menu. Use the **EnableContextMenu** property of each node to enable or disable context menu for that item.

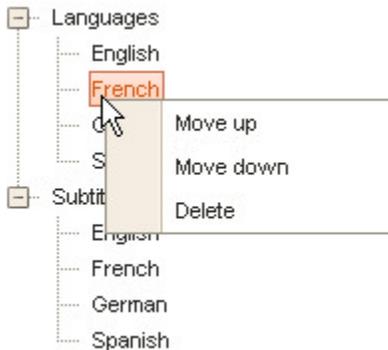


Gotcha! You must add the context menus to the **ContextMenu** property collection. Adding a separate **RadContextMenu** control to the page and setting the **ContextMenuID** property of a tree view node to its **ID** does not work.

When the user clicks on an item in the context menu, you can implement the response by using the server-side **ContextMenuItemClick** event, or by using the client-side **OnClientContextMenuItemClicking** or **OnClientContextMenuItemClicked** event.

UI for ASP.NET AJAX

The following example creates a tree view with two context menus: one for internal nodes and another for leaf nodes. It implements a `ContextMenuItemClick` event handler to respond when the user invokes the context menu on leaf nodes, and an `OnClientContextMenuItemClicking` event handler to respond when the user invokes the context menu on internal nodes.



You can find the complete source for this project at:

`\VS Projects\TreeView\ContextMenus`

The tree view includes the definitions of the two context menus in its `ContextMenus` property collection. Each node sets its `ContextMenuID` property to the ID of the context menu it uses:

[ASP.NET] Tree view with context menus

```
<telerik:RadTreeView ID="RadTreeView1" Runat="server" Skin="Sunset"
oncontextmenuItemclick="RadTreeView1_ContextMenuItemClick"
OnClientContextMenuItemClicking="HandleInternalNodeContextMenu" >
<ContextMenus>
  <telerik:RadTreeViewContextMenu ID="LeafMenu" runat="server" Skin="Sunset">
    <Items>
      <telerik:RadMenuItem runat="server" Text="Move up" />
      <telerik:RadMenuItem runat="server" Text="Move down" />
      <telerik:RadMenuItem runat="server" Text="Delete" />
    </Items>
  </telerik:RadTreeViewContextMenu>
  <telerik:RadTreeViewContextMenu ID="InternalNodeMenu" runat="server" Skin="Sunset">
    <Items>
      <telerik:RadMenuItem runat="server" Text="Reset nodes" />
    </Items>
  </telerik:RadTreeViewContextMenu>
</ContextMenus>
<Nodes>
  <telerik:RadTreeNode runat="server"
Text="Languages" Expanded="true" ContextMenuID="InternalNodeMenu" >
    <Nodes>
      <telerik:RadTreeNode runat="server" Text="English" ContextMenuID="LeafMenu" >
      </telerik:RadTreeNode>
      ...
    </Nodes>
  </telerik:RadTreeNode>
  ...
</Nodes>
</telerik:RadTreeView>
```

The `ContextMenuItemClick` event handler responds in the code-behind to the items on the LeafMenu:

[VB] ContextMenuItemClick handler

```
Protected Sub RadTreeView1_ContextMenuItemClick(ByVal sender As Object, _
    ByVal e As RadTreeViewContextMenuEventArgs) _
    Handles RadTreeView1.ContextMenuItemClick
    ' get the collection that holds the node who owns the menu
    Dim collection As RadTreeNodeCollection = e.Node.ParentNode.Nodes
    ' get the index of the node
    Dim curIndex As Integer = e.Node.Index
    Select Case e.MenuItem.Text
    Case "Move up"
        If curIndex > 0 Then
            ' only move up if not at top
            ' remove the node and insert in new position
            collection.Remove(e.Node)
            collection.Insert(curIndex - 1, e.Node)
        End If
    Case "Move down"
        If curIndex < collection.Count - 1 Then
            ' only move down if not at end
            ' remove the node and insert in new position
            collection.Remove(e.Node)
            collection.Insert(curIndex + 1, e.Node)
        End If
    Case "Delete"
        ' remove the node
        collection.RemoveAt(curIndex)
    End Select
End Sub
```

[C#] ContextMenuItemClick handler

```
protected void RadTreeView1_ContextMenuItemClick(object sender,
RadTreeViewContextMenuEventArgs e)
{
    // get the collection that holds the node who owns the menu
    RadTreeNodeCollection collection = e.Node.ParentNode.Nodes;
    // get the index of the node
    int curIndex = e.Node.Index;

    switch (e.MenuItem.Text)
    {
        case "Move up":
            if (curIndex > 0) // only move up if not at top
            {
                // remove the node and insert in new position
                collection.Remove(e.Node);
                collection.Insert(curIndex - 1, e.Node);
            }
            break;
        case "Move down":
            if (curIndex < collection.Count - 1) // only move down if not at end
```

```
    {
        // remove the node and insert in new position
        collection.Remove(e.Node);
        collection.Insert(curIndex + 1, e.Node);
    }
    break;
case "Delete":
    // remove the node
    collection.RemoveAt(curIndex);
    break;
}
}
```

The event arguments provide a reference to the node that was right clicked (**e.Node**) and to the menu item on the context menu (**e.Menuitem**).

Because there is a server-side **ContextMenuItemClick** handler, client-side handling should be done in an **OnClientContextMenuItemClicking** handler rather than an **OnClientContextMenuItemClicked** handler. This way, the handler can cancel the postback to the server for any menu items it handles.

[JavaScript] OnClientContextMenuItemClicking handler

```
// create a new node and add it to the node collection
function AddNodeToList(list, text) {
    var node = new Telerik.Web.UI.RadTreeNode();
    node.set_text(text);
    list.add(node);
}
function HandleInternalNodeContextMenu(sender, args) {
    var node = args.get_node();
    // handle the "Reset nodes" menu item
    if (args.get_menuItem().get_text() == "Reset nodes") {
        // track changes so they will persist
        sender.trackChanges();
        var nodeList = node.get_nodes();
        // clear the node's children
        nodeList.clear();
        // add a new set to the node collection
        AddNodeToList(nodeList, "English");
        AddNodeToList(nodeList, "French");
        AddNodeToList(nodeList, "German");
        AddNodeToList(nodeList, "Spanish");
        // commit the changes
        sender.commitChanges();
        // cancel the event to prevent the postback
        args.set_cancel(true);
    }
}
```

The client-side handler also has access to the node that was right clicked (by calling **args.get_node()**) and to the menu item that was invoked (by calling **args.get_menuitem()**).

 Note that because the nodes of the tree view are being changed on the client, the event handler calls the tree view's **trackChanges()** and **commitChanges()** methods. These perform the same functions as the methods with the same names that you saw on the combo box.

16.6 Server Side Programming

When working with RadTreeView in the code-behind, you can leverage what you have learned already from other controls. In the chapter on Navigation controls, you were introduced to controls that have similarly hierarchical items collections, and the techniques for manipulating those items is similar: the main difference is that in RadTreeView, the items collection is called **Nodes** rather than **Items**.

Traversing the node hierarchy

A common technique when working with RadTreeView is to perform some task for all the ancestors of a node or all the descendants of a node. When traversing up the hierarchy, from child node to parent node, use the **ParentNode** property to locate the parent of a node. When traversing down the hierarchy, from parent node to child nodes, use the **Nodes** property collection. At any step in either of these processes, you can use a node's **Level** property to ascertain the depth of a node in the hierarchy.

The following example illustrates how to traverse the node hierarchy. The Web page includes a text box, a button, and a tree view. When the user enters a string into the text box and clicks the button, the tree view selects the node that matches the string, and expands all of its parent and child nodes, collapsing any other nodes that were previously expanded.



You can find the complete source for this project at:
 \VS Projects\TreeView\ServerSide

The button's **Click** handler starts out by collapsing all nodes in the tree view, so that in the end, only the path of the specified node will be open. Then it locates the node that matches the text in the text box and selects it. Next the Click handler traverses the node hierarchy from the specified node, first upwards, to expand all ancestor nodes, and then downwards, to expand all descendants.

[VB] Expanding nodes upward and downward

```
Private Sub ExpandAllChildren(ByVal node As RadTreeNode)
```

```
' if the node has no children, we are done
If node.Nodes.Count > 0 Then
    ' expand the node
    node.Expanded = True
    ' recurse for each of the child nodes
    For Each child As RadTreeNode In node.Nodes
        ExpandAllChildren(child)
    Next
End If
End Sub
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
Button1.Click
    ' collapse any open nodes
    RadTreeView1.CollapseAllNodes()
    ' look up the name the user entered
    Dim controlName As String = RadTextBox1.Text
    ' find the node in the tree view
    Dim node As RadTreeNode = RadTreeView1.FindNodeByText(controlName)
    If Not node Is Nothing Then
        ' select the node
        node.Selected = True
        ' traverse the hierarchy upwards, expanding parent nodes
        Dim parentNode As RadTreeNode = node.ParentNode
        While Not parentNode Is Nothing
            parentNode.Expanded = True
            parentNode = parentNode.ParentNode
        End While
        ' if this is not a leaf node, expand all child nodes
        If node.Level <> 3 Then
            ExpandAllChildren(node)
        End If
    End If
End Sub
```

[C#] Expanding nodes upward and downward

```
private void ExpandAllChildren(RadTreeNode node)
{
    // if the node has no children, we are done
    if (node.Nodes.Count > 0)
    {
        // expand the node
        node.Expanded = true;
        // recurse for each of the child nodes
        for (int i = 0; i < node.Nodes.Count; i++)
        {
            ExpandAllChildren(node.Nodes[i]);
        }
    }
}
protected void Button1_Click(object sender, EventArgs e)
{
    // collapse any open nodes
    RadTreeView1.CollapseAllNodes();
    // look up the name the user entered
    string controlName = RadTextBox1.Text;
```

```

// find the node in the tree view
RadTreeNode node = RadTreeView1.FindNodeByText(controlName);
if (node != null)
{
    // select the node
    node.Selected = true;
    // traverse the hierarchy upwards, expanding parent nodes
    RadTreeNode parentNode = node.ParentNode;
    while (parentNode != null)
    {
        parentNode.Expanded = true;
        parentNode = parentNode.ParentNode;
    }
    // if this is not a leaf node, expand all child nodes
    if (node.Level != 3)
        ExpandAllChildren(node);
}
}

```

When traversing the hierarchy upwards, the simplest approach is a while loop, with the looping variable iterated using the **ParentNode** property. When the loop reaches a root node, the **ParentNode** property is **Nothing** (null), and the loop stops.

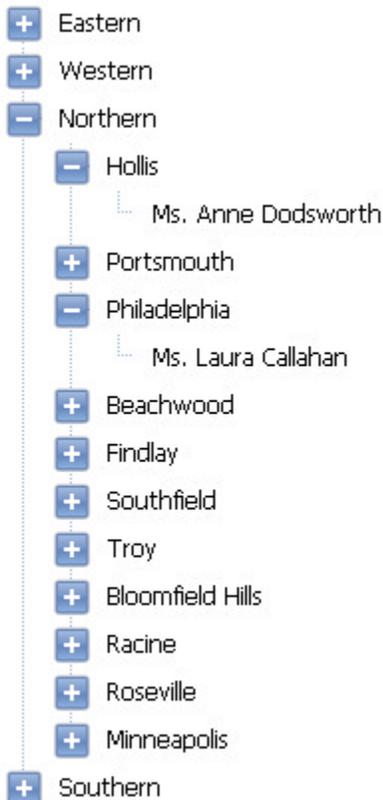
When traversing the hierarchy downwards, you can either use nested loops, or you can use a recursive procedure. The code above uses a recursive procedure ("ExpandAllChildren"), because the depth of the tree below the selected node can vary.

Building the tree view when binding to related tables

You have already seen examples of binding the tree view control to a data source. In the Getting Started example, you defined the hierarchy by using the **DataFieldID** and **DataFieldParentID** properties of the tree view, while in the Check Boxes example, you bound the tree view to an **XmlDataSource**, which was inherently hierarchical. In many cases, however, your database may be structured in such a way that the nodes at each level of the hierarchy need to come from a different table. There are two approaches for handling this:

- The **SELECT** statement of your data source can flatten all of the tables into a single table, using one or more **JOIN** or **UNION** clauses.
- You can build the tree view node hierarchy in the **Page_Load** event handler.

The following example illustrates the latter approach. In the **Page_Load** event handler, it builds the node hierarchy of a tree view from three separate data tables ("Regions", "Territories", and "Employees").



You can find the complete source for this project at:
\\VS Projects\TreeView\ServerRelatedTables

The **Page_Load** event handler only needs to generate the node hierarchy the first time the page loads:

[VB] Page_Load event handler

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
    ' we only need to generate the table when the page first loads
    If Not IsPostBack Then
        GenerateTreeView()
    End If
End Sub
```

[C#] Page_Load event handler

```
protected void Page_Load(object sender, EventArgs e)
{
    // we only need to generate the table when the page first loads
    if (!IsPostBack)
        GenerateTreeView();
}
```

The code to generate the node hierarchy first creates a data set with the three related tables. It then traverses each table in the data set, building the node hierarchy as it goes. For descendant nodes, the data table is accessed using a **Relation**, so that only the nodes that should descend from the parent node are traversed.

[VB] Building the node hierarchy

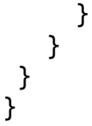
```

Private Function GenerateDataSet() As DataSet
    ' open a connection to the database
    Dim dbCon As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
    dbCon.Open()
    ' create a data set to hold the three tables
    Dim ds As New DataSet()
    ' create the tables
    Dim regions As New DataTable("Regions")
    Dim territories As New DataTable("Territories")
    Dim employees As New DataTable("Employees")
    ' create adapters to fill the tables
    Dim regionAdapter As New SqlDataAdapter("SELECT * FROM Region", dbCon)
    Dim territoryAdapter As New SqlDataAdapter("SELECT * FROM Territories", dbCon)
    Dim employeeAdapter As New SqlDataAdapter("SELECT * FROM Employees inner join
EmployeeTerritories on Employees.EmployeeID = EmployeeTerritories.EmployeeID", dbCon)
    regionAdapter.Fill(regions)
    territoryAdapter.Fill(territories)
    employeeAdapter.Fill(employees)
    ' now add the filled tables to the data set
    ds.Tables.Add(regions)
    ds.Tables.Add(territories)
    ds.Tables.Add(employees)
    ' create relations to express the links between tables
    ds.Relations.Add("RegionTerritory", ds.Tables("Regions").Columns("RegionID"), ds.Tables
("Territories").Columns("RegionID"))
    ds.Relations.Add("TerritoryEmployee", ds.Tables("Territories").Columns("TerritoryID"),
ds.Tables("Employees").Columns("TerritoryID"))
    Return ds
End Function
Private Sub GenerateTreeView()
    ' generate the data set with three tables and their relations
    Dim ds As DataSet = GenerateDataSet()
    ' now build the node hierarchy, starting at the top
    For Each regionRow As DataRow In ds.Tables("Regions").Rows
        ' create a root node for each region
        Dim regionNode As New RadTreeNode(regionRow("RegionDescription").ToString())
        RadTreeView1.Nodes.Add(regionNode)
        ' use the RegionTerritory relation to find the child rows
        For Each territoryRow As DataRow In regionRow.GetChildRows("RegionTerritory")
            ' create a territory node for each territory in the region
            Dim territoryNode As New RadTreeNode(territoryRow("TerritoryDescription").ToString())
            regionNode.Nodes.Add(territoryNode)
            ' use the TerritoryEmployee relation to find the child rows
            For Each employeeRow As DataRow In territoryRow.GetChildRows("TerritoryEmployee")
                ' create an employee node for each employee that covers the territory
                Dim employeeNode As New RadTreeNode(employeeRow("TitleOfCourtesy").ToString() + " "
+ employeeRow("FirstName").ToString() + " " + employeeRow("LastName").ToString())
                territoryNode.Nodes.Add(employeeNode)
            Next
        Next
    Next
End Sub

```

[C#] Building the node hierarchy

```
private DataSet GenerateDataSet()
{
    // open a connection to the database
    SqlConnection dbCon = new SqlConnection(ConfigurationManager.ConnectionStrings
["NorthwindConnectionString"].ConnectionString);
    dbCon.Open();
    // create a data set to hold the three tables
    DataSet ds = new DataSet();
    // create the tables
    DataTable regions = new DataTable("Regions");
    DataTable territories = new DataTable("Territories");
    DataTable employees = new DataTable("Employees");
    // create adapters to fill the tables
    SqlDataAdapter regionAdapter = new SqlDataAdapter("SELECT * FROM Region", dbCon);
    SqlDataAdapter territoryAdapter = new SqlDataAdapter("SELECT * FROM Territories", dbCon);
    SqlDataAdapter employeeAdapter = new SqlDataAdapter("SELECT * FROM Employees inner join
EmployeeTerritories on Employees.EmployeeID = EmployeeTerritories.EmployeeID", dbCon);
    regionAdapter.Fill(regions);
    territoryAdapter.Fill(territories);
    employeeAdapter.Fill(employees);
    // now add the filled tables to the data set
    ds.Tables.Add(regions);
    ds.Tables.Add(territories);
    ds.Tables.Add(employees);
    // create relations to express the links between tables
    ds.Relations.Add("RegionTerritory", ds.Tables["Regions"].Columns["RegionID"], ds.Tables
["Territories"].Columns["RegionID"]);
    ds.Relations.Add("TerritoryEmployee", ds.Tables["Territories"].Columns["TerritoryID"],
ds.Tables["Employees"].Columns["TerritoryID"]);
    return ds;
}
private void GenerateTreeView()
{
    // generate the data set with three tables and their relations
    DataSet ds = GenerateDataSet();
    // now build the node hierarchy, starting at the top
    foreach (DataRow regionRow in ds.Tables["Regions"].Rows)
    {
        // create a root node for each region
        RadTreeNode regionNode = new RadTreeNode(regionRow["RegionDescription"].ToString());
        RadTreeView1.Nodes.Add(regionNode);
        // use the RegionTerritory relation to find the child rows
        foreach (DataRow territoryRow in regionRow.GetChildRows("RegionTerritory"))
        {
            // create a territory node for each territory in the region
            RadTreeNode territoryNode = new RadTreeNode(territoryRow
["TerritoryDescription"].ToString());
            regionNode.Nodes.Add(territoryNode);
            // use the TerritoryEmployee relation to find the child rows
            foreach (DataRow employeeRow in territoryRow.GetChildRows("TerritoryEmployee"))
            {
                // create an employee node for each employee that covers the territory
                RadTreeNode employeeNode = new RadTreeNode(employeeRow["TitleOfCourtesy"].ToString()
+ " " + employeeRow["FirstName"].ToString() + " " + employeeRow["LastName"].ToString());
                territoryNode.Nodes.Add(employeeNode);
            }
        }
    }
}
```



Server-side events

RadTreeView supports a wealth of server-side events for responding in the code-behind when the user interacts with the tree view. You have already seen some of these: The **NodeCheck** event that occurs when the user changes the state of a check box on a node, the **NodeDrop** event that occurs when the user drops a dragged node, and the **ContextMenuItemClick** event that occurs when the user selects an item in the context menu of a node.

NodeClick

One of the most useful server-side events is the **NodeClick** event, which fires when the user clicks on a node.



Gotcha! When using the **NodeClick** event, do not assign a value to the node's **NavigateUrl** property. When **NavigateUrl** is set, clicking on a node causes a different page to be loaded in the same or another browser window. The page redirection means that the **NodeClick** event will not fire.

The following example illustrates one use of the **NodeClick** event. The Web page contains a tree view control and a Placeholder. When the user clicks on a leaf node of the tree view, the associated text is fetched from the server and added to the Placeholder.



There are 10 kinds of people in this world.
Those who understand binary, and those
who don't.



You can find the complete source for this project at:
[\VS Projects\TreeView\ServerNodeClick](#)

When the **NodeClick** event handler is assigned, all nodes automatically trigger a postback when the user clicks on them. To prevent the postback from occurring on internal nodes, the **PostBack** property for those nodes is set to false.

[ASP.NET] Suppressing the postback on internal nodes

```
<telerik:RadTreeView ID="RadTreeView1" runat="server" Skin="Office2007"
  OnNodeClick="RadTreeView1_NodeClick" >
  <Nodes>
    <telerik:RadTreeNode runat="server"
      Text="Nursery Rhymes"
      ExpandedImageUrl="~/Images/FolderOpen.gif" ImageUrl="~/Images/folder.gif"
      PostBack="False" >
      <Nodes>
        <telerik:RadTreeNode runat="server" Text="Jack and Jill" />
        <telerik:RadTreeNode runat="server" Text="Little Bo-Peep" />
        <telerik:RadTreeNode runat="server" Text="Little Jack Horner" />
      </Nodes>
    </telerik:RadTreeNode>
    <telerik:RadTreeNode runat="server"
      Text="Random"

```

```
ExpandedImageUrl="~/Images/FolderOpen.gif" ImageUrl="~/Images/folder.gif"
PostBack="False" >
<Nodes>
  <telerik:RadTreeNode runat="server" Text="Joke" />
  <telerik:RadTreeNode runat="server" Text="Quote" />
</Nodes>
</telerik:RadTreeNode>
</Nodes>
</telerik:RadTreeView>
```

In the code-behind, the **NodeClick** event handler uses the **Text** property of the clicked node to locate the nursery rhyme, joke, or quote to add to the placeholder. It then generates and adds literal controls to the placeholder for the current selection.

[VB] NodeClick event handler

```
Protected Sub RadTreeView1_NodeClick(ByVal sender As Object, _
                                     ByVal e As RadTreeNodeEventArgs) _
    Handles RadTreeView1.NodeClick

    Dim r As Random
    ' initialize textToDisplay
    Dim textToDisplay As String() = {}
    ' Fetch the lines of text for the clicked node
    Select Case e.Node.Text
        Case "Jack and Jill"
            textToDisplay = JackAndJill
            Exit Select
        Case "Little Bo-Peep"
            textToDisplay = LittleBoPeep
            Exit Select
        Case "Little Jack Horner"
            textToDisplay = LittleJackHorner
            Exit Select
        Case "Joke"
            ' select a joke at random
            r = New Random(DateTime.Now.Millisecond)
            textToDisplay = Jokes(r.[Next](Jokes.Length))
            Exit Select
        Case "Quote"
            ' select a quote at random
            r = New Random(DateTime.Now.Millisecond)
            textToDisplay = Quotes(r.[Next](Quotes.Length))
            Exit Select
    End Select
    ' Add each line of text to the placeholder as a Literal control
    ' Add a break between lines (two breaks for random jokes or quotes)
    For Each s As String In textToDisplay
        Placeholder1.Controls.Add(New LiteralControl(s))
        Placeholder1.Controls.Add(New LiteralControl("<br/>"))
        If e.Node.ParentNode.Text = "Random" Then
            Placeholder1.Controls.Add(New LiteralControl("<br/>"))
        End If
    Next
End Sub
```

[C#] NodeClick event handler

```
protected void RadTreeView1_NodeClick(object sender, RadTreeNodeEventArgs e)
```

```

{
    Random r;
    // initialize textToDisplay
    string[] textToDisplay = {};
    // Fetch the lines of text for the clicked node
    switch (e.Node.Text)
    {
        case "Jack and Jill":
            textToDisplay = JackAndJill;
            break;
        case "Little Bo-Peep":
            textToDisplay = LittleBoPeep;
            break;
        case "Little Jack Horner":
            textToDisplay = LittleJackHorner;
            break;
        case "Joke":
            // select a joke at random
            r = new Random(DateTime.Now.Millisecond);
            textToDisplay = Jokes[r.Next(Jokes.Length)];
            break;
        case "Quote":
            // select a quote at random
            r = new Random(DateTime.Now.Millisecond);
            textToDisplay = Quotes[r.Next(Quotes.Length)];
            break;
    }
    // Add each line of text to the placeholder as a Literal control
    // Add a break between lines (two breaks for random jokes or quotes)
    foreach (string s in textToDisplay)
    {
        Placeholder1.Controls.Add(new LiteralControl(s));
        Placeholder1.Controls.Add(new LiteralControl("<br/>"));
        if (e.Node.ParentNode.Text == "Random")
            Placeholder1.Controls.Add(new LiteralControl("<br/>"));
    }
}

```

NodeEdit

When node editing is enabled (the **AllowNodeEditing** property is true), the tree view automatically permits the user to edit the **Text** property of nodes. This is true even if you use an item template (in that case, when the user finishes editing the text, any controls in the template that are bound to the item's **Text** or the field specified by **DataTextField** are automatically updated.)

Unfortunately, any changes that the user makes by editing nodes do not persist after a postback. To allow the edits to persist, you must handle the **NodeEdit** event.

The following walk-through illustrates this issue.



You can find the complete source for this project at:
 \VS Projects\TreeView\ServerNodeEdit

1. Create a new ASP.NET Web Application and drag a **ScriptManager** from the Tool Box onto the Web page.
2. Drag a **Button** from the Tool Box onto the Web page. Set its **Text** property to "Postback".

3. Drag a **RadFormDecorator** from the Tool Box onto the Web page and set its **Skin** property to "Black".
4. Drag a **RadTreeView** from the Tool Box onto the Web page and set its **Skin** property to "Black".
5. Bring up the **RadTreeView Item Builder** and add some nodes at random. You do not need to set any node properties.
6. Set the **AllowNodeEditing** property of the tree view to true.
7. Press Ctrl-F5 to run the application.
8. Click on a node and edit its text. You can do this to as many nodes as you like.



9. Click the "Postback" button. After the postback, all the nodes revert to their old **Text** values!



10. To allow the new text values to persist, add the following **NodeEdit** event handler:

[VB] NodeEdit event handler

```
Protected Sub RadTreeView1_NodeEdit(ByVal sender As Object, _  
    ByVal e As RadTreeNodeEditEventArgs) _  
    Handles RadTreeView1.NodeEdit  
    ' Assign the Text property of the node  
    ' so that the new value persists  
    e.Node.Text = e.Text  
End Sub
```

[C#] NodeEdit event handler

```
protected void RadTreeView1_NodeEdit(object sender, RadTreeNodeEditEventArgs e)  
{  
    // Assign the Text property of the node  
    // so that the new value persists  
    e.Node.Text = e.Text;  
}
```

11. Before trying out this change, add a **RadAjaxManager** to the Web page, and configure it so that postbacks initiated by the tree view cause the tree view to update.
12. Press Ctrl-F5 again. This time, if you edit the Text of some nodes and hit the "Postback" button, the change persists.

 Another good use of the **NodeEdit** event is to save the user's edits back to a database or other permanent storage.

16.7 Client-Side Programming

You have already seen some examples of working with the client-side API of RadTreeView. In the drag-and-drop example, you saw how to use the **OnClientNodeDropping** event to assign an id to drop targets, or cancel the postback for the **NodeDrop** event. In the context menus example, you saw how to use the **OnClientContextMenuItemClicking** event, and to use client side methods to alter the node hierarchy. That example showed that, just like RadComboBox, RadTreeView has **trackChanges()** and **commitChanges()** methods, which must surround any client-side code that makes changes to the node hierarchy if those changes are to persist.

Using the tree node object

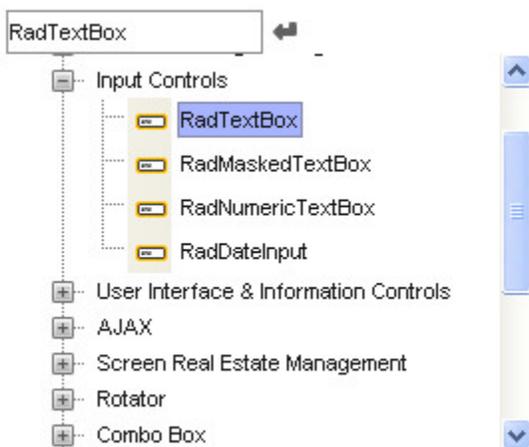
In addition to methods for getting and setting node properties, the client-side RadTreeNode object has a number of methods for changing the state of the node. These methods tend to come in pairs:

- The **expand()** and **collapse()** methods let you change whether the node is expanded. (You can also use the **toggle()** method to change the node's state from one to the other.)
- The **select()** and **unselect()** methods let you change whether the node is selected.
- The **check()** and **uncheck()** methods let you change the checked state of a check box on the node.
- The **startEdit()** and **endEdit()** methods let you switch the node in and out of edit mode.
- The **highlight()** and **unhighlight()** methods let you change whether the node is highlighted.

Another useful method is the **scrollIntoView()** method, which scrolls the tree view so that the node becomes visible.

 The **scrollIntoView()** method scrolls the tree view so that the node's current position is in view. However, this method does not expand any of the node's parent nodes. If one of the parent nodes is not expanded, the node will still not be visible, even after a call to **scrollIntoView()**.

The following example illustrates a number of the methods of the client-side **RadTreeNode** object. The Web page has a **RadTextBox** and a **RadTreeView** on it. When the user enters a string in the text box and clicks its button, the tree view expands all parents of the matching node, selects it, and scrolls it into view.



You can find the complete source for this project at:
 \VS Projects\TreeView\ClientSide

The text box's **OnButtonClick** client-side event handler first locates the matching node by calling the tree

view's `findNodeByText()` method. It then traverses the node hierarchy upward, making sure all of the node's parents are expanded. Then it scrolls the node into view and selects it.

[JavaScript] Using tree node methods

```
function FindNode(sender, args) {  
    // get a reference to the tree view  
    var treeView = $find("<%= RadTreeView1.ClientID %>");  
    // locate the node  
    var node = treeView.findNodeByText(sender.get_value());  
    if (node != null) {  
        // make sure the nodes parent's are all expanded  
        var parent = node.get_parent();  
        while (parent != treeView) {  
            parent.expand();  
            parent = parent.get_parent();  
        }  
        // scroll the node into view --  
        // this should happen AFTER expanding parent nodes  
        node.scrollIntoView();  
        // and select it  
        node.select();  
    }  
}
```

Note that the `scrollIntoView()` method is called *after* the parent nodes are expanded. This is important, because expanding the parent nodes can change the position of the node within the tree view.

Establishing a 'radio button' pattern

You have already looked at examples that traverse the node hierarchy to apply a change to all parents of a node or all children of a node. Another common technique is to establish a 'radio button' pattern on a set of sibling nodes.

The following example establishes a 'radio button' pattern on the checked state of the nodes in a tree view.

- ▾ Section
 - First Class
 - Executive Class
 - Business Class
 - Economy
- ▾ Seat Preference
 - Window
 - Aisle
- ▾ Meals
 - No Restrictions
 - Diabetic
 - Gluten Free
 - Kosher
 - Vegetarian
 - Vegan



You can find the complete source for this project at:
 \VS Projects\TreeView\ClientRadioPattern

To establish a 'radio button' pattern, the application supplies a handler to the **OnClientNodeChecked** event. The event handler calls the **get_parent()** method to get a reference to the parent of a checked node. This method can return either another node, or the tree view object itself (although in this example it is always another node). Whichever one is returned, the object has a **get_nodes()** method to access the node collection that contains the node that was just checked. By iterating the node collection, the event handler can uncheck every node except the one that was just checked.

[JavaScript] Establishing a 'radio button' pattern

```
function NodeChecked(sender, args) {
  // uncheck all siblings when a node is checked
  // first get the parent node
  var parentNode = args.get_node().get_parent();
  // iterate through all its children
  var siblings = parentNode.get_nodes();
  for (var i = 0; i < siblings.get_count(); i++) {
    var sibling = siblings.getNode(i);
    // uncheck all but the node that was just checked
    if (args.get_node() != sibling)
      sibling.uncheck();
  }
}
```

Using the tree view object

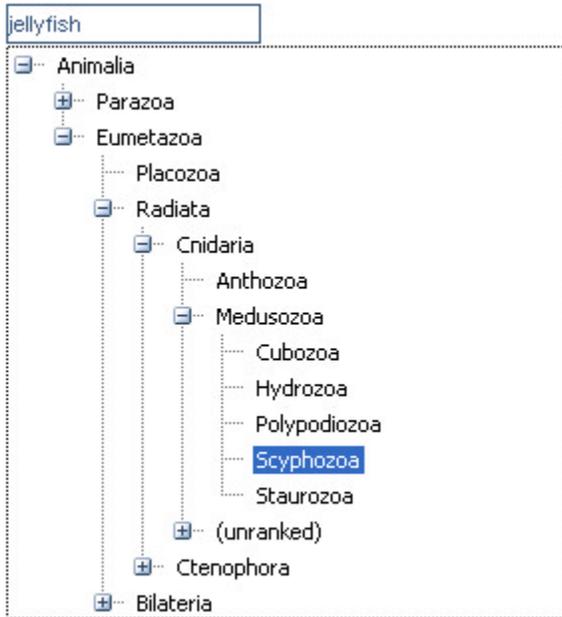
The RadTreeView client-side object, like the server-side object, has a number of methods for locating nodes. In addition to the **findNodeByText()** method, which you saw in the tree node example above, the tree view also supports the **findNodeByValue()** method to locate nodes by the Value property, and a **findNodeByAttribute()** method, which locates nodes based on custom attributes. (Unlike the server-side object, on the client side there is no method for locating a node by its `NavigateUrl` property.)

In addition to the **get_nodes()** method, which returns the root nodes of the tree view, the tree view also has a **get_allNodes()** method, which returns all of the nodes in the entire tree view.

These methods are illustrated in the following example. In addition, the example illustrates how you can use the **get_element()** method to access the DOM element for the tree view to add a handler to client-side events that are not available on the RadTreeView client-side object (in this case, the **onfocus** event).

The example is similar to the one shown above for working with tree nodes. As in that example, the Web page contains a text box and a tree view. The tree view displays the top levels of a taxonomic tree for animal classifications. In this example, the user does not need to press a button to locate the node named by the value in the text box; the tree view automatically locates, selects, and displays the node when it gains focus, so all the user needs to do is tab to the tree view.

Another difference is that the tree view searches not just the text of its nodes, but also for matches to custom attributes that have been added to nodes. (The custom attributes give the common names associated with nodes in the tree.)



You can find the complete source for this project at:
\\VS Projects\\TreeView\\ClientTreeView

The tree view has an **OnClientLoad** event handler which accesses the DOM element for the tree view and attaches a handler to its **onfocus** event. It also saves a reference to the tree view object in a global variable, so that there is no need for a call to the **Sys.Application.findComponent()** method (**\$find()**) every time the application needs to call one of the tree view methods.

[JavaScript] Attaching a handler to the onfocus event of the DOM object

```
var tree;
function OnLoad(sender) {
    // save a reference to the tree view for later
    tree = sender;
    // add the event handler to the onfocus event of the tree view
    var treeDiv = sender.get_element();
    treeDiv.onfocus = function() { FindNode(); };
}
```

Before looking at the **FindNode** function, which handles the **onfocus** event of the tree view, let us first look at two helper functions that illustrate the difference between the results from the **get_nodes()** method and the **get_allNodes()** method.

The **get_nodes()** method returns a tree node collection object. To iterate through all the nodes in this collection, use its **getNode()** method to access nodes and its **get_count()** method to get the number of nodes. (The node collection also has methods for inserting and removing nodes, which we are not using here.)

[JavaScript] Iterating nodes returned by get_nodes()

```
function ExpandRootNodes() {
    // get the nodes collection
    var rootNodes = tree.get_nodes();
    var index;
    // iterate the nodes collection, expanding every node
```

```

for (index = 0; index < rootNodes.get_count(); index++) {
    var node = rootNodes.getNode(index);
    if (!node.get_expanded()) {
        node.expand();
    }
}
}
}

```

The `get_allNodes()` method, by contrast, returns an array. To iterate through all the nodes in the array, simply index into the array using square brackets (`[]`), and use the `length` property to get the number of nodes.

[JavaScript] Iterating nodes returned by `get_allNodes()`

```

function CollapseAllNodes() {
    // get all the nodes
    var allNodes = tree.get_allNodes();
    var index;
    // iterate the nodes array, collapsing them
    for (index = 0; index < allNodes.length; index++) {
        var node = allNodes[index];
        if (node.get_expanded()) {
            node.collapse();
        }
    }
}
}

```

Now let us look at the `FindNode` function. This function is quite similar to the function of the same name that was used in the tree node example above. The main differences are

- In addition to using the `findNodeByText()` method to match the text of a node, it also calls the `findNodeByAttribute()` method to look for a match to the "CommonName" or "AltName" custom attributes.
- It calls the `ExpandRootNodes()` function shown above to expand the root nodes if no match is found.
- It calls the `CollapseAllNodes()` function shown above before opening a path to the matching node so that any other nodes in the tree view are collapsed.
- It does not call the `scrollIntoView()` method, but rather, just lets the tree view grow to accommodate its contents.

[JavaScript] Locating a node by text and custom attribute

```

function FindNode() {
    // get a reference to the text box
    var textBox = $find("<%= RadTextBox1.ClientID %>");
    // if no text specified, just expand the root nodes
    if (textBox.get_value() == "")
        ExpandRootNodes(tree);
    else {
        // locate the node
        var node = tree.findNodeByText(textBox.get_value());
        // if the entered string was not the text of a node,
        // try the CommonName attribute
        if (node == null)
            node = tree.findNodeByAttribute("CommonName", textBox.get_value());
        // if the common name was not found either, try the AltName attribute
        if (node == null)
            node = tree.findNodeByAttribute("AltName", textBox.get_value());
        if (node != null) {
            // collapse the tree
            CollapseAllNodes(tree);
        }
    }
}

```

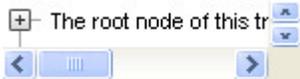
```
// open a path to the specified node
var parent = node.get_parent();
while (parent != tree) {
    parent.expand();
    parent = parent.get_parent();
}
// select the node
node.select();
}
else // couldn't find the node, just expand the root
    ExpandRootNodes(tree);
}
}
```

16.8 How To

Wrapping the text of tree nodes

The text of nodes in RadTreeView occupies a single line. This can be an issue if the node text is very long. The following walk-through illustrates the problem, and how to reconfigure the tree view so that it can wrap the text of nodes.

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Drag a RadTreeView control onto the Web Page.
3. Open the RadTreeView Item Builder to add some nodes to the tree view.
 1. Add a root node to the tree view. Set its Text property to "The root node of this tree view has a very, very, long text value."
 2. Add any additional nodes you want to fill out the tree view.
4. Using the Properties Window, set the Width property of the tree view to "150px"
5. Press CTRL-F5 to run the application. The tree view does not wrap the node text. Instead, it adds a horizontal scroll bar. Then, because the horizontal scroll bar obscures the text of the last root node, it adds a vertical scroll bar (if there is more than one root node). The result is almost impossible to use!



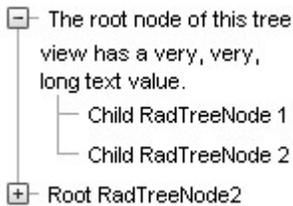
If there is only a single root node, it is impossible to use because the scroll bar obscures the expand button.

6. Close the running application, and switch to the Source view of your Web page.
7. Locate the tag for the root node with the long text values and add the `style="white-space: normal;"` attribute to the node.

[ASP.NET] Long tree node with style attribute

```
<telerik:RadTreeNode runat="server" style="white-space: normal;"
    Text="The root node of this tree view has a very, very, long text value.">
    <Nodes>
        ...
    </Nodes>
</telerik:RadTreeNode>
```

8. Press CTRL-F5 to run the application again. This time, the text of the root node wraps to fit in the allotted width.



💡 You can also add the `style="white-space: normal;"` to the `<telerik:RadTreeView>` tag to enable wrapping for all of the nodes in the tree view.

📁 You can find the complete source for this project at:
 \VS Projects\TreeView\HowToWrappingNodeText

Adding controls to nodes in the code-behind

While you can always use an item template to customize the appearance of the nodes in a tree view, another approach is to simply add controls directly to the **Controls** collection of a node in the code-behind. This gives you the power to customize nodes in the code-behind without having to implement an `ITemplate` class.

The following example illustrates how to add controls to the **Controls** collection of a node. It adds a color picker to the nodes of a tree view when the page first loads. Once controls are added to the **Controls** collection, the node no longer displays its **Text** property, but instead shows only the controls that were added.



📁 You can find the complete source for this project at:
 \VS Projects\TreeView\HowToControlsCollection

The color picker is added in the **Page_Load** event handler:
[VB] Adding controls to a node

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
    ' Go through all of the nodes in the tree view
  
```

```
For Each node As RadTreeNode In RadTreeView1.GetAllNodes()  
    'Replace nodes with text "Color" by a color picker  
    If node.Text = "Color" Then  
        Dim colors As New RadColorPicker()  
        colors.ID = node.ID + "_colors"  
        colors.Preset = ColorPreset.Opulent  
        colors.ShowEmptyColor = False \VS Projects\  
        colors.PreviewColor = False  
        'Add the color picker to the controls collection  
        node.Controls.Add(colors)  
    End If  
Next  
End Sub
```

[C#] Adding controls to a node

```
protected void Page_Load(object sender, EventArgs e)  
{  
    // Go through all of the nodes in the tree view  
    foreach (RadTreeNode node in RadTreeView1.GetAllNodes())  
    {  
        // Replace nodes with the text "Color" by a color picker  
        if (node.Text == "Color")  
        {  
            RadColorPicker colors = new RadColorPicker();  
            colors.ID = node.ID + "_colors";  
            colors.Preset = ColorPreset.Opulent;  
            colors.ShowEmptyColor = false;  
            colors.PreviewColor = false;  
            // add the color picker to the controls collection  
            node.Controls.Add(colors);  
        }  
    }  
}
```



Gotcha! Be sure to add the controls every time the page loads, not just when `IsPostBack` is false. Controls that are added this way do not persist in the View State.

16.9 Performance

`RadTreeView` supports a large number of nodes, but it can only show a limited number of nodes at a time on the client side. As the number of nodes increases, the size of the HTML file that clients must download can become significant. Not only must the tree view itself be downloaded, but the View State for state management and JavaScript that implements the behavior adds to the size of the Web page. If a tree contains more than about 200 nodes, it is not practical to load the entire tree at once.

To manage the size of downloads and improve performance, you can configure the tree view to load nodes only when they are needed. Similar to the load-on-demand mechanism you have already seen with `RadComboBox`, you can load tree view nodes on demand from either a server-side event handler or a Web service.

Load on demand

The tree view load-on-demand mechanism is designed to work on a node by node basis. That is, you can use load-on-demand to expand some nodes, while allowing others to be expanded on the client as in a smaller tree

view. To enable the load-on-demand mechanism for a RadTreeView node, set its **ExpandMode** property. **ExpandMode** can have any of four possible values:

- **ClientSide** (the default): nodes are not loaded on demand, but rather, they are downloaded with the Web page.
- **ServerSide**: When the user expands the node, it generates a postback and the **NodeExpand** event handler is called to supply child nodes.
- **ServerSideCallBack**: This is the same as **ServerSide**, except that the tree view generates an AJAX callback rather than a postback. While waiting for the results of the callback, the string specified by the tree view's **LoadingMessage** property is displayed in the position specified by the **LoadingStatusPosition** property.
- **WebService**: When the user expands the node, it generates a callback to the Web Service (and Web Method) specified by the **WebServiceSettings** property.

The following example illustrates four different expand modes. It contains a tree view that starts with four nodes, a root node and three child nodes. The root node has its **ExpandMode** property set to "ClientSide"; the first child node has its **ExpandMode** property set to "ServerSide"; the second child node has its **ExpandMode** property set to "ServerSideCallBack"; the third child node has its **ExpandMode** property set to "WebService". When the user expands a node, the **NodeExpand** callback or WebService method adds 10 child nodes, each with the same expand mode as the node that is expanding.

- ▲ Ordinary Node
 - ▲ Expand Me With a Postback
 - ▲ Child 1
 - ▷ Child 1.1
 - ▷ Child 1.2
 - ▷ Child 1.3
 - ▷ Child 1.4
 - ▷ Child 1.5
 - ▷ Child 1.6
 - ▷ Child 1.7
 - ▷ Child 1.8
 - ▷ Child 1.9
 - ▷ Child 1.10
 - ▷ Child 2
 - ▷ Child 3
 - ▷ Child 4
 - ▷ Child 5
 - ▷ Child 6
 - ▷ Child 7
 - ▷ Child 8
 - ▷ Child 9
 - ▷ Child 10
 - ▷ Expand Me with a Callback
 - ▷ Expand Me With a Web Service

You can find the complete source for this project at:
 \VS Projects\TreeView\LoadOnDemand



The **NodeExpand** event handler is called to add the children of nodes with **ExpandMode** set to "ServerSide" or "ServerSideCallBack".

[VB] Adding child nodes using a NodeExpand callback

```
Protected Sub RadTreeView1_NodeExpand(ByVal sender As Object, _
    ByVal e As Telerik.Web.UI.RadTreeNodeEventArgs) _
    Handles RadTreeView1.NodeExpand
    Dim i As Integer = 1
    While i <= 10
        ' get the naming index
        Dim nameIndex As String = e.Node.Value
        If nameIndex <> "" Then
            nameIndex += "."
        End If
        nameIndex += i.ToString()
        ' create a new node
        Dim newNode As New RadTreeNode("Child " + nameIndex)
        ' set the Value to the nameIndex
        newNode.Value = nameIndex
        ' with the same expand mode as the node currently expanding
        newNode.ExpandMode = e.Node.ExpandMode
        ' add the new node to the node that is expanding
        e.Node.Nodes.Add(newNode)
        System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
    End While
    ' now that the nodes are added, change the current node's expand mode to client-side
    e.Node.ExpandMode = TreeNodeExpandMode.ClientSide
    ' signal that the node is now expanded
    e.Node.Expanded = True
End Sub
```

[C#] Adding child nodes using a NodeExpand callback

```
protected void RadTreeView1_NodeExpand(object sender, Telerik.Web.UI.RadTreeNodeEventArgs e)
{
    for (int i = 1; i <= 10; i++)
    {
        // get the naming index
        string nameIndex = e.Node.Value;
        if (nameIndex != "")
            nameIndex += ".";
        nameIndex += i.ToString();
        // create a new node
        RadTreeNode newNode = new RadTreeNode("Child " + nameIndex);
        // set the Value to the nameIndex
        newNode.Value = nameIndex;
        // with the same expand mode as the node currently expanding
        newNode.ExpandMode = e.Node.ExpandMode;
        // add the new node to the node that is expanding
        e.Node.Nodes.Add(newNode);
    }
    // now that the nodes are added, change the current node's expand mode to client-side
    e.Node.ExpandMode = TreeNodeExpandMode.ClientSide;
}
```

```

    // signal that the node is now expanded
    e.Node.Expanded = true;
}

```

The event handler uses the properties of the node that is expanding to determine what child nodes to create (in this example, it is only to set the text and expand modes). It then adds each newly created node into the **Nodes** property collection of the expanding node. Finally, it changes the **ExpandMode** property of the expanding node to "ClientSide", since the items are now on the client, and sets its **Expanded** property to true.

The Web Service Method that supplies child nodes to any node that has an **ExpandMode** of "WebService" is very similar:

[VB] Web Service to supply child nodes

```

<System.Web.Script.Services.ScriptService> _
<System.Web.Services.WebService(Namespace:="http://tempuri.org/")> _
<System.Web.Services.WebServiceBinding(ConformsTo:=WsiProfiles.BasicProfile1_1)> _
<ToolboxItem(False)> _
Public Class TreeNodeService
    Inherits System.Web.Services.WebService
    <WebMethod()> _
    Public Function GetChildNodes(ByVal node As RadTreeNodeData, ByVal context As Object) As
RadTreeNodeData()
        Dim result As New List(Of RadTreeNodeData)()
        Dim i As Integer = 1
        While i <= 10
            ' get the naming index
            Dim nameIndex As String = node.Value
            If Not nameIndex Is Nothing Then
                nameIndex += "."
            End If
            nameIndex += i.ToString()
            ' create data for the new node
            Dim newNode As New RadTreeNodeData()
            ' assign the text
            newNode.Text = "Child " + nameIndex
            ' set the Value to the nameIndex
            newNode.Value = nameIndex
            ' with the same expand mode as the node currently expanding
            newNode.ExpandMode = TreeNodeExpandMode.WebService
            ' add the new node to the list
            result.Add(newNode)
            System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
        End While
        Return result.ToArray()
    End Function
End Class

```

[C#] Web Service to supply child nodes

```

[WebService(Namespace = "http://tempuri.org (http://tempuri.org/)")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
[System.ComponentModel.ToolboxItem(false)]
[System.Web.Script.Services.ScriptService]
public class TreeNodeService : System.Web.Services.WebService
{
    [WebMethod]
    public RadTreeNodeData[] GetChildNodes(RadTreeNodeData node, object context)

```

```
{
    List<RadTreeNodeData> result = new List<RadTreeNodeData>();
    for (int i = 1; i <= 10; i++)
    {
        // get the naming index
        string nameIndex = node.Value;
        if (nameIndex != null)
            nameIndex += ".";
        nameIndex += i.ToString();
        // create data for the new node
        RadTreeNodeData newNode = new RadTreeNodeData();
        // assign the text
        newNode.Text = "Child " + nameIndex;
        // set the Value to the nameIndex
        newNode.Value = nameIndex;
        // with the same expand mode as the node currently expanding
        newNode.ExpandMode = TreeNodeExpandMode.WebService;
        // add the new node to the list
        result.Add(newNode);
    }
    return result.ToArray();
}
```

The chief differences are

- Instead of an event arguments object that provides access to the `RadTreeNode` object for the expanding node, the Web Method has a `RadTreeNodeData` argument, which supplies information about the node properties.
- Instead of adding new nodes directly into the Nodes property collection of the expanding node, the Web Method returns an array of `RadTreeNodeData` objects for the nodes that need to be added.

 Note the second parameter of the Web Method. This is a context object (of type `IDictionary`) that can supply context information supplied by a client-side `OnClientNodeExpanding` event handler. This context object works the same way as the context object used with the `RadComboBox` load-on-demand feature.

16.10 Summary

In this chapter you looked at the `RadTreeView` control and saw how you could add the functionality of a desktop tree view to your Web applications. You created a simple application that populated one tree view with statically declared items and another with items loaded from a data source. At the same time, you looked at some properties of the tree view and tree nodes.

You looked at the design time support for the tree view and saw many of the properties and groups of properties you can use to configure the tree view and its nodes at design time. You learned about the special features of `RadTreeView`, including node editing, check boxes, drag-and-drop, and node context menus.

You learned some of the server-side properties and methods, and explored how to propagate a change to all of the ancestors or descendants of a node. You learned to build the node hierarchy dynamically in server-side code, and saw how this could be used to populate a tree view with data from multiple tables. You also learned about several of the tree view server-side events.

You explored some of the client-side methods for working with the tree node and tree view objects. You learned how to implement the 'radio button' pattern for state changes on nodes, and saw how to attach an event handler directly to the tree view's DOM object when the tree view first loads.

You learned a few "tricks" for working with the tree view, such as getting the text of nodes to wrap and how to add controls directly to tree nodes without using templates.

Finally, you learned how to use the load-on-demand feature to improve performance for large tree views, expanding nodes using either a postback, a callback, or a Web Service.

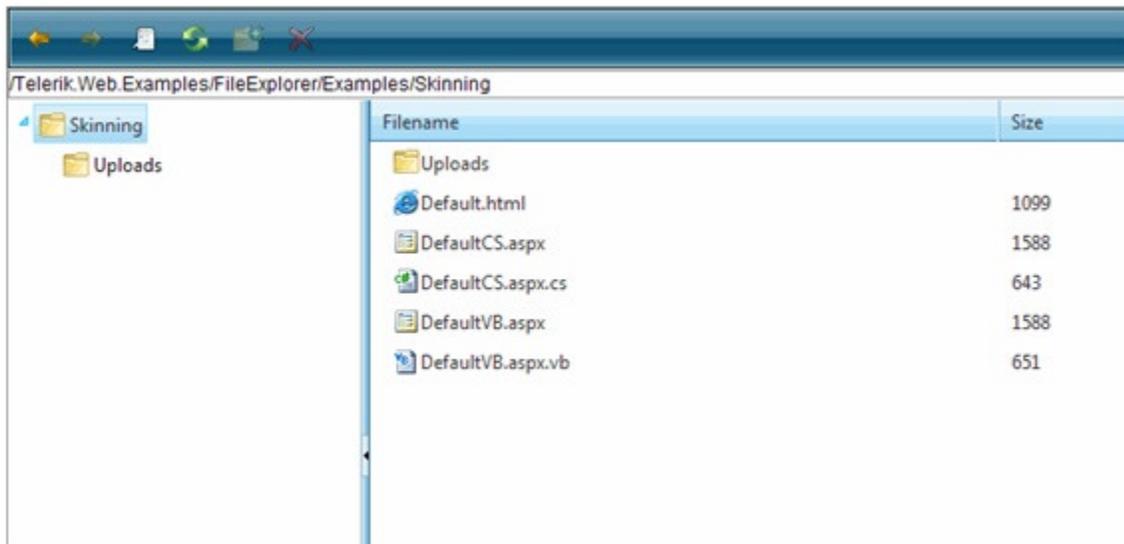
17 RadFileExplorer

17.1 Objectives

- Explore features of the FileExplorer control.
- Learn how to configure RadFileExplorer using *Property* window in Visual Studio.
- Learn how to configure RadFileExplorer on the server.
- Learn how to localize RadFileExplorer.
- Learn some advance customizations: create and register a custom FileBrowserContentProvider.
- Learn how to control RadFileExplorer's content using its client-side API.

17.2 Introduction

RadFileExplorer was officially included in the Q1 2009 release of RadControls for ASP.NET AJAX. It allows you to easily add file explorer functionality to your pages providing the users an ability to organize files and folders on the server through web interface.



Main features:

- A single control, integrated in Telerik.Web.UI - ready to drag and drop on the page
- Load on demand approach to load its content using ASP.NET AJAX Callback mechanism
- Uses a ContentProvider model which introduces an abstraction of the underlying datasource. This allows the control to be connected to any kind of datasource like OS filesystem, database, MOSS SharePoint, Amazon S3, Windows Azure, etc.
- Supports files and folders sorting
- Ability to delete and rename files and folders, create new folders, etc.
- Client and server side events for file operations like delete, create new folder, etc.
- Context menu commands for common operations

17.3 Getting Started

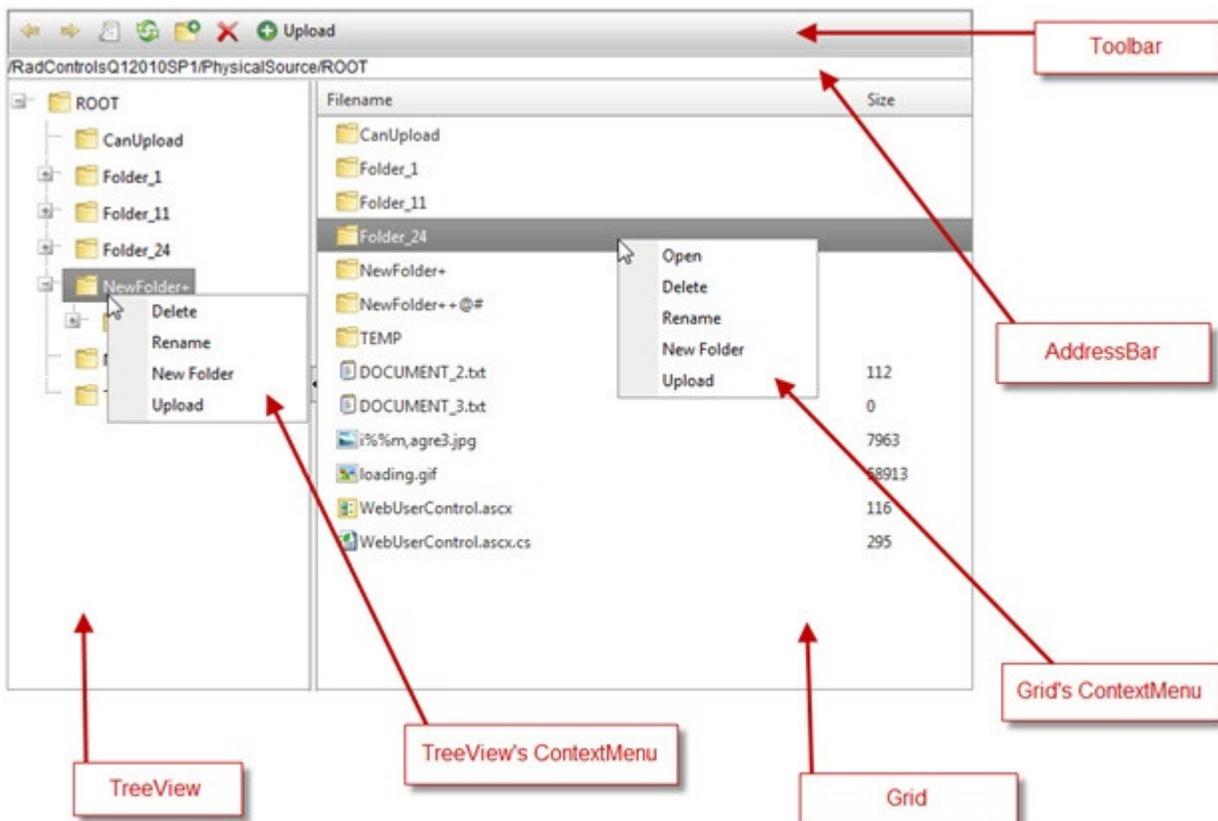
Setting up RadFileExplorer

Here are the basic steps in order to use the RadFileExplorer control in a web application.

1. Create a new ASP.NET AJAX - enabled web site.
2. Add a RadScriptManager or a standard ASP ScriptManager to the page - this step is mandatory if you are using ASP.NET AJAX controls.
3. Drag a RadFileExplorer from your VS Toolbox and drop it on the page.
4. Right-click on the inserted RadFileExplorer control and select properties.
5. Set the following properties in the Configuration section: ViewPaths, DeletePaths and UploadPaths in the following format: ~/<path> where the tilde (~) represents the root of your web application.
6. Save the page and run it in the browser.

RadFileExplorer components

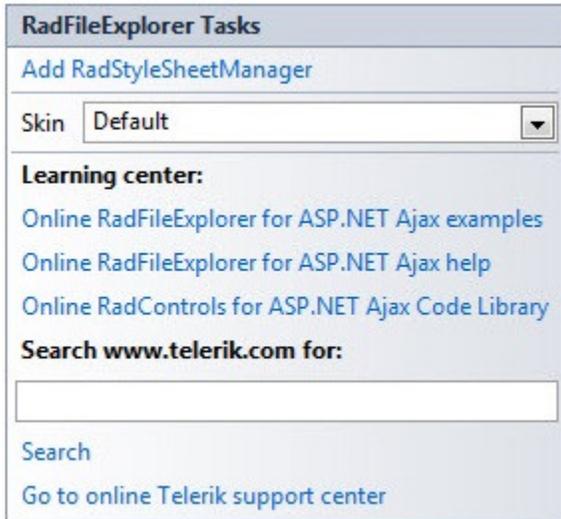
RadFileExplorer is built using these controls: RadGrid , RadTreeView, RadToolBar, RadContextMenu, RadSplitter, RadLaodingPanel, RadWindow (for previewing files) and an input element for the AddressBar.



Designer Interface

Smart tags

The RadFileExplorer Smart Tag contains only the common elements of RadControls Smart Tags: the Ajax Resources, Skin selection, and Learning center:



Property window

At design time, you can use the configuration window to configure almost every aspect of RadFileExplorer. Probably the most important properties of **RadFileExplorer** are combined in a group called *Configuration*.

Configuration	Telerik.Web.UI.FileManagerDialog
ContentProviderTypeName	
DeletePaths	~/ROOT/
MaxUploadFileSize	204800
SearchPatterns	*.*
UploadPaths	~/ROOT/
ViewPaths	~/ROOT/

These are some of the *Configuration* properties:

- **ViewPaths** - accepts an array of folder paths that will be listed in the RadFileExplorer.
- **UploadPaths / DeletePaths** - accepts an array of folder paths where the user should have Upload / Delete permissions. If you wish to restrict Upload and Delete you can simply do not provide value to these properties.
- **SearchPatterns** - this property is used in order to filter the files displayed in RadFileExplorer. The values set to this property may contain wildcards. The default value of the property is “*.*”, which means “All files”.
- **MaxUploadFileSize** - sets limit of the uploaded files.
- **ContentProviderTypeName** - sets the qualified assembly name of the custom content provider which will

be used by the RadFileExplorer control.

In the property window you can assign RadFileExplorer's client-side event handlers:

- **OnClientCreateNewFolder** is fired when the user creates a new folder.
- **OnClientDelete** is fired when the user deletes file or folder.
- **OnClientFileOpen** is fired when the user opens a file for preview (this feature have to be enabled setting the EnableFileOpen="true" property).
- **OnClientFolderChange** is fired when the user selects a folder from the TreeView.
- **OnClientFolderLoaded** is fired when the folder's content is fully loaded.
- **OnClientItemSelected** is fired when the user selects a file item (not a folder).
- **OnClientLoad** is fired when the RadFileExplorer control is fully loaded.
- **OnClientMove** is fired when the client tries to move or rename a file and /or folder.

Properties that control the RadFileExplorer's behavior:

- **EnableOpenFile** - If the value is true, by default when a file is double clicked it will be opened in a RadWindow dialog for preview. This default behavior can be easily overridden, however, in order to cover more specific scenarios. For example, if the implemented scenario requires the file to be opened in a different dialog than the default one.
- **EnableCopy** - enables or disables copy feature of the control.
- **AllowPaging** - enables paging on the Grid. This property can be combined with the **PageSize** property.
- **EnableCreateNewFolder** - allows or restricts creating a new folder.
- **VisibleControls** - controls which of the of RadFileExplorer elements (Grid, TreeView, etc.) to be visible. You can provide multiple values to this property by separating them with commas (,) in the markup or using bitwise OR (|) on the server.
- **ExplorerMode** - accepts two enum values:
 - *Default* - the default mode of the control.
 - *FileTree* - In this mode the Grid part of the control will be disabled and the files will be displayed in the TreeView.

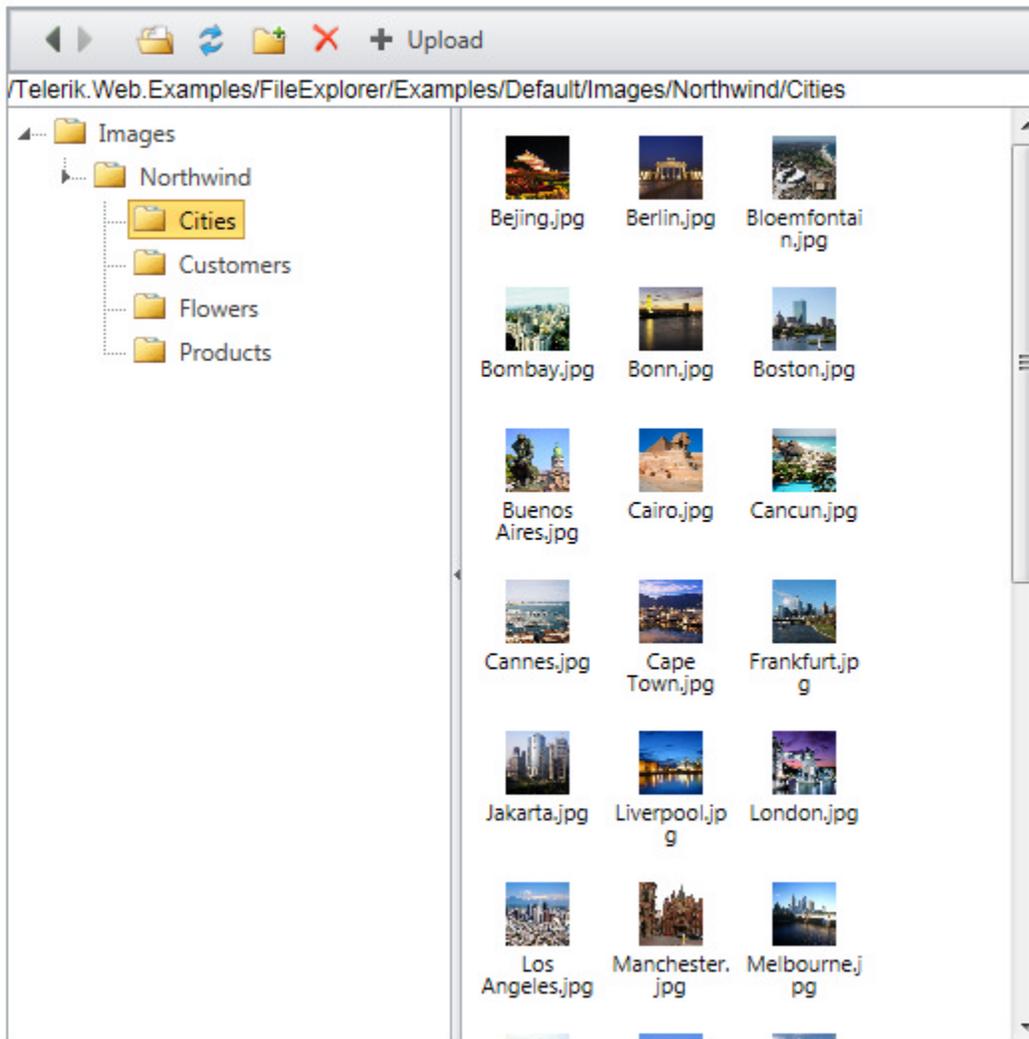
17.4 Thumbnails Mode

Since Q1 2012 the RadFileExplorer offers the Thumbnails Explorer mode. This is a new way to display the list of files, just as you would expect in Windows Explorer. The special feature is displaying images through thumbnails, not as file icons.

To enable this mode just set the **ExplorerMode** property to **Thumbnails**, e.g.

```
<telerik:RadFileExplorer runat="server" ID="FileExplorer1"
```

ExplorerMode="Thumbnails"></telerik:RadFileExplorer>



17.5 Server-Side Programming

Getting familiar with the Server-Side API

The embedded controls are exposed as RadFileExplorer's properties:

- **Grid** - returns reference to the RadGrid embedded in RadFileExplorer
- **TreeView** - returns reference to the RadTreeView embedded in RadFileExplorer
- **GridContextMenu** - returns reference to the context menu shown over the grid's items
- **ToolBar** - returns reference to the radToolBar control
- **ToolTip** - returns reference to the RadToolBarControl
- **WidnowManager** - returns reference to the RadWindowManager control embedded in RadFileExplorer
- **Splitter** - returns reference to the RadSplitter control embedded in RadFileExplorer

Server-Side events

ItemCommand

This event is called when Deleting, Uploading, Creating, Moving (move and rename are the same operations) a file/folder. The event can be canceled by setting `Cancel="true"` property of the passed argument to the *ItemCommand* event

C#

```
protected void RadFileExplorer1_ItemCommand(object sender, RadFileExplorerEventArgs e)
{
    switch (e.Command)
    {
        case "UploadFile": break;
        case "MoveDirectory": break;
        case "CreateDirectory": break;
        case "DeleteDirectory": break;
        case "DeleteFile": break;
        case "MoveFile": break;
    }
    // e.Cancel = true; // Cancel the operation
}
```

VB.NET

```
Protected Sub RadFileExplorer1_ItemCommand(ByVal sender As Object, ByVal e As RadFileExplorerEventArgs)
    Select Case e.Command
        Case "UploadFile"
            Exit Select
        Case "MoveDirectory"
            Exit Select
        Case "CreateDirectory"
            Exit Select
        Case "DeleteDirectory"
            Exit Select
        Case "DeleteFile"
            Exit Select
        Case "MoveFile"
            Exit Select
        ' e.Cancel = true // Cancel the operation
    End Select
End Sub
```

ExplorerPopulated

This event is fired twice - once when the TreeView's items are populated and second time when the Grid's items are populated. The *ControlName* property contains the name of the control, which will consume the populated data (e.List). This event can be used to sort the FileExplorer's items for example.

UI for ASP.NET AJAX

Example:

ASPX

```
<telerik:RadFileExplorer
  runat="server"
  ID="RadFileExplorer1"
  Width="575px"
  EnableCopy="true"
  Height="375px"
  OnExplorerPopulated="RadFileExplorer1_ExplorerPopulated">
  <Configuration
    ViewPaths="~/ROOT/"
    DeletePaths="~/ROOT/"
    UploadPaths="~/ROOT/" />
</telerik:RadFileExplorer>
```

C#

```
protected void RadFileExplorer1_ExplorerPopulated(object sender,
Telerik.Web.UI.RadFileExplorerPopulatedEventArgs e)
{
  switch (e.ControlName)
  {
    case "tree":
      // The TreeView control will be populated
      // Sorts the items shown in the Tree by name
      e.List.Sort(delegate(FileBrowserItem fileBrowserItem1, FileBrowserItem
fileBrowserItem2)
      {
        return fileBrowserItem1.Name.CompareTo(fileBrowserItem2.Name);
      });
      break;
    case "grid":
      // The Grid control will be populated
      // Sorts the items shown in the Grid by name
      e.List.Sort(delegate(FileBrowserItem fileBrowserItem1, FileBrowserItem
fileBrowserItem2)
      {
        return fileBrowserItem1.Name.CompareTo(fileBrowserItem2.Name);
      });
      // DESC order
      e.List.Reverse();
      break;
  }
}
```

VB.NET

```
Protected Sub RadFileExplorer1_ExplorerPopulated(sender As Object, e As
Telerik.Web.UI.RadFileExplorerPopulatedEventArgs)
  Select Case e.ControlName
  Case "tree"
    If True Then
      ' The TreeView control will be populated
```

```

        ' Sorts the items shown in the Tree by name
        e.List.Sort(Function(fileBrowserItem1 As FileBrowserItem, fileBrowserItem2 As
FileBrowserItem) fileBrowserItem1.Name.CompareTo(fileBrowserItem2.Name))
    End If
    Exit Select
Case "grid"
    If True Then
        ' The Grid control will be populated
        ' Sorts the items shown in the Grid by name
        e.List.Sort(Function(fileBrowserItem1 As FileBrowserItem, fileBrowserItem2 As
FileBrowserItem) fileBrowserItem1.Name.CompareTo(fileBrowserItem2.Name))

        ' DESC order
        e.List.Reverse()
    End If
    Exit Select
End Select
End Sub

```

Both events are shown in the `ServerSideProgramming` example project.

17.6 Client-Side Programming

As with all the RadControls you can get reference to the RadFileExplorer client-side object using the `$find()` method:

JavaScript

```
var explorer = $find("RadFileExplorer1");
```

Then you can use the rich client-side API of the control for achieving various scenarios. The example below (*ClientSideProgramming* project) shows how to create a new folder on the server using only client-side functionality of the control:

ASPX

```

<asp:Button ID="Button1" runat="server" Text="Creaste a 'TEMP' folder"
OnClick="createTempFolder();return false;" />
<telerik:RadFileExplorer ID="RadFileExplorer2" runat="server">
    <Configuration ViewPaths="~/ROOT/" DeletePaths="~/ROOT/" UploadPaths="~/ROOT/" />
</telerik:RadFileExplorer>
<script type="text/javascript">
    function createTempFolder() {
        var applicationRoot = '<%= VirtualPathUtility.ToAbsolute("~/") %>';
        var oExplorer = $find("<%= RadFileExplorer1.ClientID %>");
        oExplorer.createNewDirectory(applicationRoot + "/ROOT", "TEMP");
    }
</script>

```

createNewDirectory(path, newName) - accepts two optional parameters. Here are the possible scenarios:

- If you do not provide any of parameters a dialog will pop-up asking for name of the folder and will create it as a sub-folder to the currently selected one.
- If the function is called passing the path parameter, then the folder will be created to that path. A dialog will pop-up asking for name of the folder.

- If both of parameters are passed then the folder will be created without showing any pop-up dialog.

When the *createNewDirectory* is called, the *CreateDirectory* method of the *FileBrowserContentProvider* will be called on the server, if the event is not canceled in *RadFileExplorer*'s *ItemCommand* server-side event.

All client-side objects of the *RadFileExplorer* component (grid, *TreeView*, etc.) are exposed as properties and can be used to modify the default behavior of the control. For example, this JavaScript code shows how to get reference to the *RadGrid*'s client object:

JavaScript

```
var gridObject = radFileExplorer.get_grid();
```

17.7 How To

Set configuration properties in codebehind

The Configuration properties can be set using server-side code as well. This is an example setup:

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    // ROOT folder's content will be visible, including CanUploadDirectory and
    // CanDeleteDirectory directories
    string[] viewPaths = new string[] { "~/ROOT" };

    // Allows upload TO ~/ROOT/CanUploadDirectory
    // Delete is not allowed, so a file/folder cannot be moved FROM this folder
    // Allows copy TO this folder as well (if EnableCopy="true" is set)
    string[] uploadPaths = new string[] { "~/ROOT/CanUploadDirectory" };

    // Allows Delete FROM ~/ROOT/CanDeleteDirectory
    // A folder/file can be moved FROM this directory as well
    // Upload, copy or move TO this folder is not allowed
    string[] deletePaths = new string[] { "~/ROOT/CanDeleteDirectory" };

    RadFileExplorer1.Configuration.ViewPaths = viewPaths;
    RadFileExplorer1.Configuration.UploadPaths = uploadPaths;
    RadFileExplorer1.Configuration.DeletePaths = deletePaths;

    // Only .jpg and .gif files will be shown
    string[] searchPatterns = new string[] { "*.jpg", "*.gif" };
    RadFileExplorer1.Configuration.SearchPatterns = searchPatterns;

    // Sets the max allowed size of the uploaded files
    RadFileExplorer1.Configuration.MaxUploadFileSize = 3000;

    // Sets the AssemblyQualifiedName of a FileBrowserContentProvider class. The
    // FileSystemContentProvider is the default provider used by RadFileExplorer
    RadFileExplorer1.Configuration.ContentProviderTypeName = typeof
    (Telerik.Web.UI.Widgets.FileSystemContentProvider).AssemblyQualifiedName;
}
```

VB.NET

```

Protected Sub Page_Load(sender As Object, e As EventArgs)
    ' ROOT folder's content will be visible, including CanUploadDirectory and
    CanDeleteDirectory directories
    Dim viewPaths As String() = New String() {"~/ROOT"}

    ' Allows upload TO ~/ROOT/CanUploadDirectory
    ' Delete is not allowed, so a file/folder cannot be moved FROM this folder
    ' Allows copy TO this folder as well (if EnableCopy="true" is set)
    Dim uploadPaths As String() = New String() {"~/ROOT/CanUploadDirectory"}

    ' Allows Delete FROM ~/ROOT/CanDeleteDirectory
    ' A folder/file can be moved FROM this directory as well
    ' Upload, copy or move TO this folder is not allowed
    Dim deletePaths As String() = New String() {"~/ROOT/CanDeleteDirectory"}

    RadFileExplorer1.Configuration.ViewPaths = viewPaths
    RadFileExplorer1.Configuration.UploadPaths = uploadPaths
    RadFileExplorer1.Configuration.DeletePaths = deletePaths

    ' Only .jpg and .gif files will be shown
    Dim searchPatterns As String() = New String() {"*.jpg", "*.gif"}
    RadFileExplorer1.Configuration.SearchPatterns = searchPatterns

    ' Sets the max allowed size of the uploaded files
    RadFileExplorer1.Configuration.MaxUploadFileSize = 3000

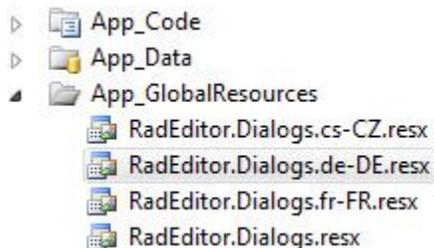
    ' Sets the AssemblyQualifiedName of a FileBrowserContentProvider class. The
    FileSystemContentProvider is the default provider used by RadFileExplorer
    RadFileExplorer1.Configuration.ContentProviderTypeName = GetType
    (Telerik.Web.UI.Widgets.FileSystemContentProvider).AssemblyQualifiedName
End Sub

```

The configuration properties should be set in Page_Load event.

Localization

The **RadFileExplorer** control uses the same localization mechanism as RadEditor (.resx files in the App_GlobalResources folder). If you want to change the localization strings, you need to copy the RadEditor.Dialogs.resx file in the App_GlobalResources folder of your application



and edit the strings inside.

For example the *Language="de-DE"* property forces the *RadEditor.Dialogs.de-DE.resx* file to be used by the **RadFileExplorer** control:

Example Title

```
<telerik:RadFileExplorer
  runat="server"
  ID="RadFileExplorer1"
  Width="575"
  Height="375"
  Language="de-DE"
  >
  <Configuration ViewPaths="~/ROOT/" DeletePaths="~/ROOT/" UploadPaths="~/ROOT/" />
</telerik:RadFileExplorer>
```

Implementing a custom provider

RadFileExplorer loads its data through a content provider. This approach frees the developers to implement their own provider which can be used in order to connect RadFileExplorer to any kind of data sources (FTP filesystem, Database, etc.). Base class for all content providers is `Telerik.Web.UI.Widgets.FileBrowserContentProvider`. To implement a custom Content Provider you need to inherit that abstract class and implement its methods:

- *ResolveRootDirectoryAsTree* - Called in order to load all sub folders of the passed as parameter folder.
- *ResolveDirectory* - called in order to load all child files of the passed as parameter folder.
- *StoreFile* - called in order to save an uploaded file.
- *DeleteFile* - called in order to delete a file.
- *DeleteDirectory* - called in order to delete a directory.
- *CreateDirectory* - called in order to create a directory.
- *CanCreateDirectory* - a readonly boolean property. This property allows explicitly restricting creating a new folder on the ContentProvider level.
- *GetFile* - used in two cases:
 - 1) To identify if a file with the same name exists in the same path when uploading a file.
 - 2) When creating a thumbnail used to get the original image content.
- *StoreBitmap* - used to save a newly created bitmap to the storage.
- *GetFileName* - used to get the file name only from the given URL.
- *GetPath* - used to get the directory path of an item (file or directory) from the given URL.

The ContentProvider model provides full server-side control over the content shown in RadFileExplorer. The model introduces a flexibility which allows customizations in order to cover unique scenarios that appear during development.

Filter TextBox

RadFileExplorer supports filtering of the files and folders in the Grid. Simply set the **EnableFilterTextBox** property to **true** and a search box will be rendered above the Grid's header. The items are filtered on every key stroke, so you don't need to press "Enter" to invoke the filtering process. Note that the FileExplorer searches for the keyword in the currently selected directory, omitting the items in the subfolders.

The text of the Label is set through the localizable **FilterTextBoxLabel** property.

It is not necessary to have the built-in filter textbox enabled in order to perform filtering. The FileExplorer's **filter(keyWord)** client-side method can be used to filter the items in the currently selected directory.

The **filter** client-side event (**OnClientFilter** property) is raised before the filtering occurs, and event argument object with the following properties and methods is passed to the event handler method:

- **get_text()** - gets the text (keyword) to search for.
- **set_text(newText)** - sets the text (keyword) to search for.
- **set_cancel(toCancel)** - sets bool value that determines whether the filtering will be cancelled - **set_cancel(true)** will cancel the filtering process.
- **get_domEvent()** - gets a reference to the current domEvent - it comes handy when you need to determine which key was pressed.

17.8 Summary

In this chapter you looked at the RadFileExplorer control and saw some of the powerful features it provides.

- We explored the client-side and server-side properties of the control.
- Learned how to configure the control server-side or using Property window.
- You used the server-side events in order to sort the items shown in the RadFileExplorer control or detect server-side activity.
- Learned how to change the localization of the control using *.resx* files.
- We explored the basic steps in order to use a custom Content Provider with the RadFileExplorer control.

18 RadSiteMap

18.1 Objectives

- Explore the features of the RadSiteMap control.
- Create a simple application to build confidence in using the site map and to see how to bind to various types of data sources, including declarative data sources.
- Explore the site map design time interface, including Smart Tag, Properties Window, Property Builder, Collection Editors and Template Design surface.
- Explore principal properties and groups of properties where most of the functionality is found.
- Learn server-side coding techniques and handling server-side events.

18.2 Introduction

With the ease of Telerik's SiteMap for ASP.NET AJAX you can organize and list the pages on your web site, customize the layout, choose from a variety of appearance options and templates. Add value to your web site by optimizing it for crawler and search engines with no extra development effort.

Products		
ASP.NET AJAX Controls	WinForms Controls	WPF Controls
Description	Description	Description
Demos	Demos	Demos
Videos	Videos	Videos
Silverlight Controls	Telerik Reporting	Telerik OpenAccess ORM
Description	Description	Description
Demos	Demos	Demos
Videos	Videos	Videos
Sitefinity ASP.NET CMS	WebUI Test Studio	
Consulting		
On-site Training	Consulting Express	Open Classes Training
Online Training	Project Consulting	Telerik Webinars

RadSiteMap combines the highly efficient rendering of RadControls for ASP.NET AJAX with a powerful set of features. You can use the familiar skinning capabilities to make the site map fit in with the look and feel of your Web site, as well adjusting the appearance using styles or item templates.

The capabilities of RadSiteMap extend beyond simply changing the appearance. You can configure the layout of RadSiteMap in a variety of modes. The nodes can be viewed in either list or flow mode. By selecting the flow property items in the group will be arranged in rows one after the other, instead of displaying them as a list. You can also alternate between single, multi-column, horizontal or vertical view. In addition you can display node lines in a fashion similar to RadTreeView.

The ASP.NET SiteMap by Telerik allows you to define a collection of dynamic templates that customize the presentation of the hierarchy and the individual nodes

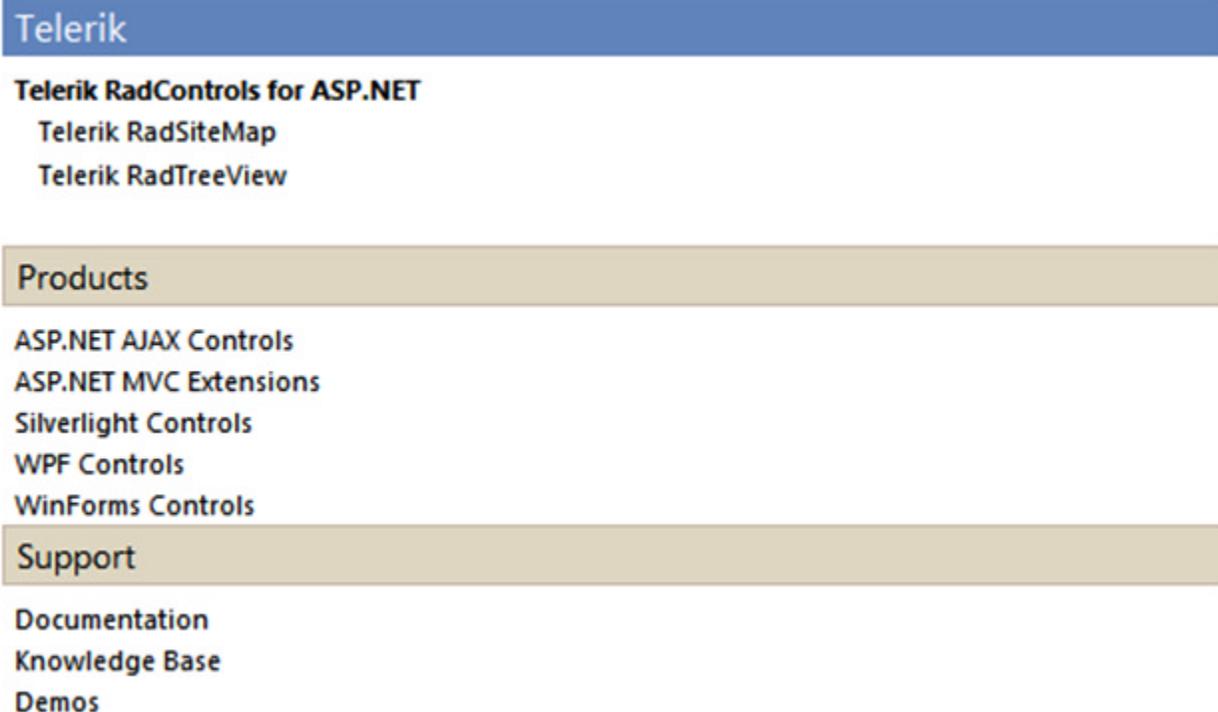
Following our long tradition of industry best cross-browser support, Telerik ASP.NET SiteMap doesn't make an exception. The component supports all major browsers, including Internet Explorer, Firefox, Safari, Opera and Google Chrome and produces identical results.



RadSiteMap completely follows the principles of Search Engine Optimization. The control renders semantic lists and standard anchor tags, which are properly recognized by search engines. As a result, all content accessible through this control will be automatically indexed and ranked with no extra effort required from the developer.

18.3 Getting started

In this walk-through you will become familiar with the RadSiteMap control. You will create two site maps: one bound to a RadSiteMapDataSource which is inherently hierarchical, and one bound to a non-hierarchical data source as SqlDataSource.





You can find the complete source for this project at:
\\VS Projects\SiteMap\GettingStarted

Prepare the project

1. Create a new ASP.NET Web Application and drag a ScriptManager from the Tool Box onto the Web page.
2. Locate the "SiteMap.mdf" file and drag it into the "App_Data" folder of your project.
3. Open the "Web.config" file of your project. Add SiteMap connection string to your project by replacing the line `<connectionStrings />` with:

web.config

```
<connectionStrings>
  <add name="SiteMapConnectionString" connectionString="Data
Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|\SiteMap.mdf;Integrated
Security=True;User Instance=True"
  providerName="System.Data.SqlClient" />
</connectionStrings>
```

Bind RadSiteMap to a RadSiteMapDataSource

1. Drag RadSiteMap control from the Toolbox to your Web page. Set its Skin property to "Web20".
2. In the Solution Explorer, choose Add New Item... In the templates dialog, select Site Map.
3. Click the Add button. Visual Studio generates the web.sitemap file with the initial code. Populate the Web.sitemap as follows:
 1. On the first siteMapNode, set the url to "http://www.telerik.com", title="Telerik" and the description to "Telerik home page".
 2. On the second, set the url to "http://www.telerik.com/radcontrols", title="Telerik RadControls for ASP.NET" and the description to "Telerik RadControls for ASP.NET".
 3. On the third, set the url to "http://www.telerik.com/products/aspnet-ajax/sitemap.aspx", title="Telerik RadSiteMap" and the description to "Telerik RadSiteMap control".
 4. On the fourth, set the url to "http://www.telerik.com/products/aspnet-ajax/treeview.aspx", title="Telerik RadTreeView" and the description to "Telerik RadTreeView".
4. Drag a RadSiteMapDataSource instance from the Toolbox to your Web page.
5. In the RadSiteMapDataSource, set the "web.sitemap" file as a value of the SiteMapFile property.
6. From the RadSiteMap Smart Tag, choose RadSiteMapDataSource1 from the "Choose Data Source" drop-down.

Bind RadSiteMap to a SqlDataSource

1. In the designer, hit the Enter key to add a line break, and then drag a second RadSiteMap control from the Tool Box onto your Web page. Set its Skin property to "Sunset".
2. In the RadSiteMap Smart Tag, select "<New data source...>" from the "Choose Data Source" drop-down.
3. In the first page of the DataSource Configuration Wizard, select "Database" as the application type, and

click OK to move to the next page.

4. On the Choose Your Data Connection page, select "SiteMapConnectionString" from the drop-down list. Then click the Next button to continue.
5. On the Configure the Select Statement page, make sure the "Specify columns from a table or view" radio button is selected, and then choose "SiteMap" from the "Name" drop-down list.
6. Check the "*" field to select all table fields. Then click the Next button to continue.
7. Test the query if you wish, and then click Finish.
8. In the Properties Window for the second site map,
 1. Set the DataFieldID property to "ID".
 2. Set the DataFieldParentID property to "ParentID".
 3. Set the DataTextField property to "Title".
 4. Set the DataNavigateUrlField property to "URL".

Run the application

1. Press Ctrl-F5 to run the application.
2. On the first site map you can click on any node and easily navigate to the page it refers to.
3. On the second site map, note that the items form a hierarchy, although they all came from the same table. This hierarchy is built using the DataFieldID and DataFieldParentID properties of items.

18.4 Designer Interface

In the Visual Studio designer, you can configure the RadSiteMap control using the Smart Tag, the Properties Window, and the RadSiteMap Item Builder. In addition, you can add data bindings using the NavigationItemBinding collection editor and add templates using the Template Design surface.

Smart Tag

The RadSiteMap Smart Tag looks like the typical Smart Tag of a RadControl that contains items which can be either statically declared or loaded from a data source:

RadSiteMap Tasks

Choose Data Source: (None) ▼

[Build RadSiteMap...](#)

Ajax Resources:

[Add RadAjaxManager...](#)

Skin: Default ▼

Learning center:

[Online RadSiteMap for ASP.NET Ajax examples](#)

[Online RadSiteMap for ASP.NET Ajax help](#)

[Online RadControls for ASP.NET Ajax Code Library](#)

Search www.telerik.com for:

[Search](#)

[Go to online Telerik support center](#)

[Edit Templates](#)

The drop-down to bind the site map and the link to bring up the Item Builder should be familiar by now. So should the standard Ajax Resources, Skin Selection, and Learning Center items. Because you can define item templates for the site map, there is also an Edit Templates link to bring up the Template Design Surface.

If you bind the site map to a data source, the Smart Tag changes to its bound version:

The bound Smart Tag lets you change the data source, reconfigure the current data source, or refresh the schema. In addition, there is a link “Edit RadSiteMap Databindings ...” to bring up the NavigationItemBinding Collection Editor. This collection should be familiar to you from the Data Binding chapter.

The bound Smart Tag still contains the Edit Templates item to bring up the Template Design Surface.

Properties Window

At design time, you can use the Properties Window to configure almost every aspect of the site map, with the exception of templates. As before, let us look at the most important properties.

Specifying Items

Probably the most important property of the site map is the one that specifies what items appear and their hierarchical relationships. What properties you choose for this task depends on whether you want to load items from a data source:

- If you want to load items from a data source, you can use the standard data-binding properties (**DataSourceID** and **DataMember**), or use the **DataSource** property and **DataBind** method in the code-behind.

When binding RadSiteMap to a data source, you can use the **DataTextField**, **DataValueField** and **DataNavigationUrlField** properties to map fields from the data source to properties of the nodes, or use the **DataBindings** property to map even more node properties.

- If you want to establish a hierarchical relationship between nodes, use the **DataFieldID** and **DataFieldParentID** properties. When setting up a hierarchy in this way, you can use the **MaxDataBindDepth** property to limit the depth of the hierarchy.
- When using an inherently hierarchical data source such as an **XmlDataSource** or **SiteMapDataSource** there is no need to use the **DataFieldID** and **DataFieldParentID** properties. The hierarchy is automatically honored

by the site map.

- If you want to use statically declared items, you can use the **Nodes** property to bring up the **RadSiteMap Item Builder** or you can switch to the Source view and define the structure directly in the mark-up.
- If you want to use both data-bound and statically-declared items, set the **AppendDataBoundItems** property to true.

Other site map properties

In addition to the **Skin** property, you can affect the look-and-feel of the site map by setting the **ShowNodeLines** property to node lines in a fashion similar to **RadTreeView**.

The following properties can be used to customize the List Layout mode: **RepeatColumns**, **RepeatDirection** and **RowAlign**.

RepeatDirection property determines the order in which the nodes in the level are rendered.

If this property is set to **RepeatDirection.Vertical**, the nodes in the level are displayed in columns loaded from top to bottom, then left to right, until all nodes are rendered.

If this property is set to **RepeatDirection.Horizontal**, the nodes in the level are displayed in rows loaded from left to right, then top to bottom, until all nodes are rendered.

RepeatDirection has no effect if **RepeatColumns** is set to 0 (default).

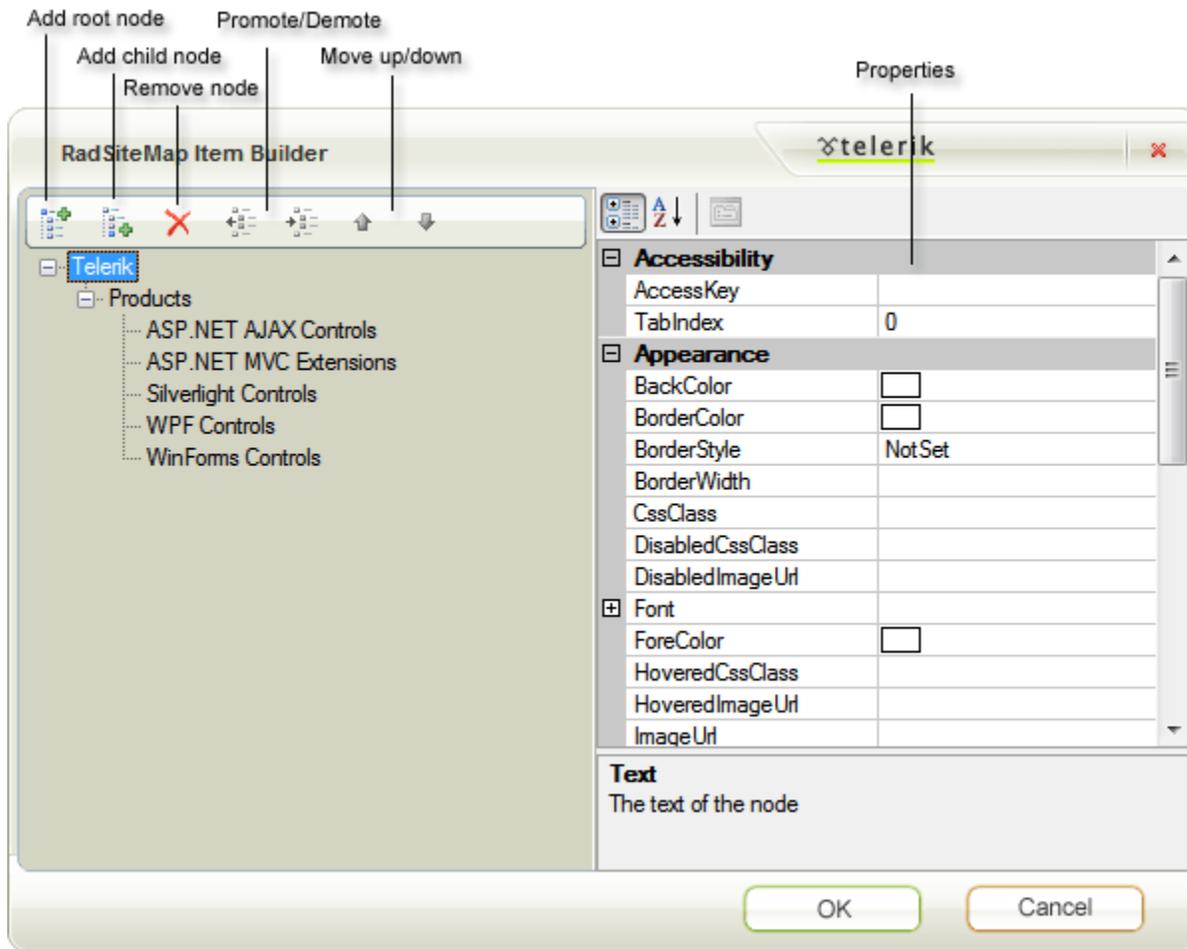
The **RepeatColumns** property specifies the number of columns for the given level.

Setting the **RowAlign** property to "true" forces the nodes in different columns to align to each other, as if they were rendered in a table.

RadSiteMap Item Builder

RadSiteMap lets you edit the list of statically defined nodes using the **RadSiteMap Item Builder**. This item builder is very similar to the hierarchical Property Builder dialogs you looked at in the chapter on Navigation controls. Display the item builder either from the Smart Tag or by clicking the ellipsis button on the **Nodes** property in the Properties Window.

Below is a screen shot of the **RadSiteMap Item Builder**. Use the buttons on the upper left to build or edit the node hierarchy. You can select any of the nodes and set its properties using the properties pane on the right of the dialog. Typically, you will set the **Text** property first.

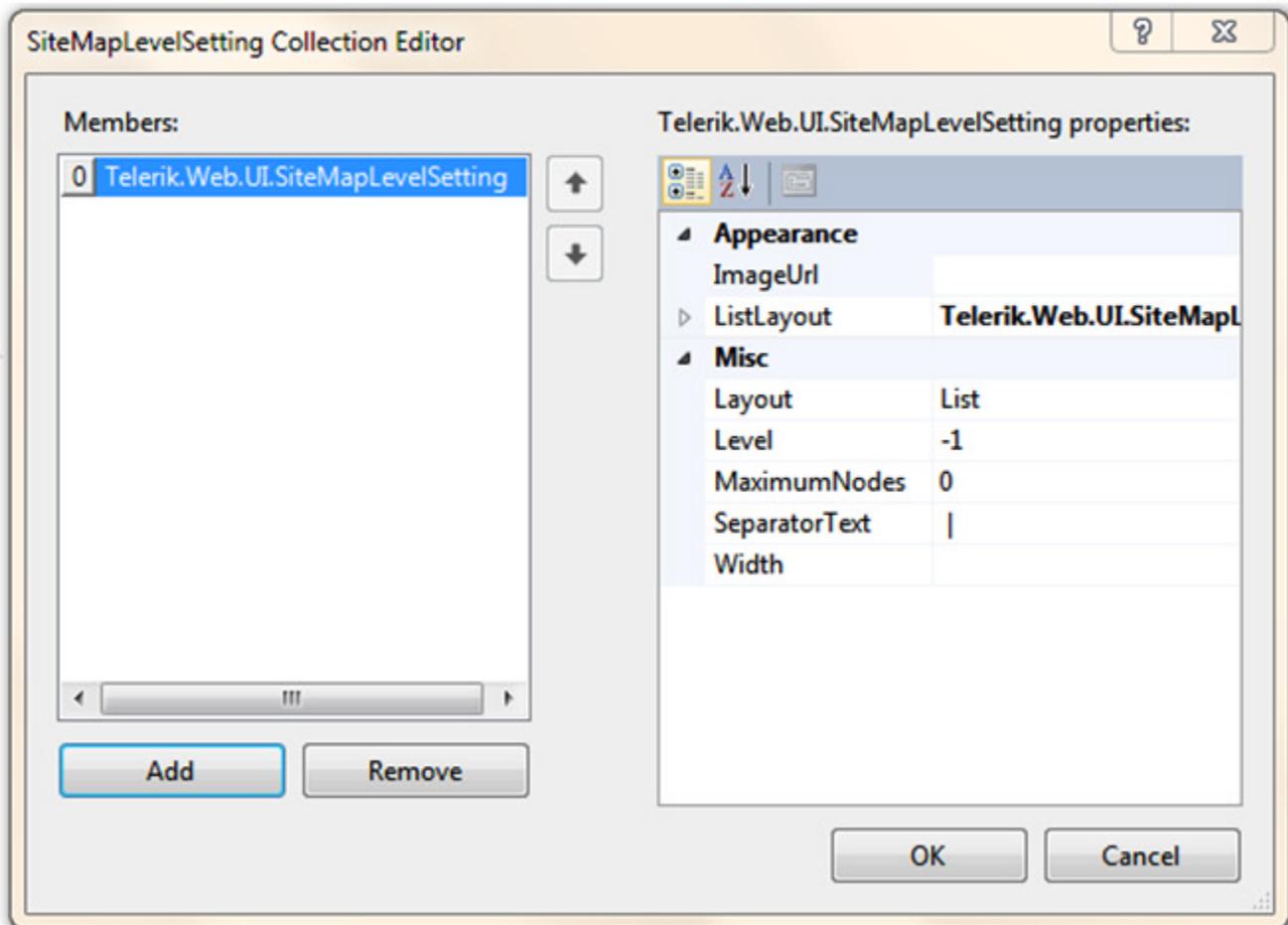


Each node has its own set of properties: **Text** is the string that represents the node, **ToolTip** is the tool tip for the node, and **Value** is a value associated with the node. You can add images to a node by setting the **ImageUrl**, **DisabledImageUrl**, **HoveredImageUrl** and **SelectedImageUrl** properties. If any of the other image properties are not set, the node uses the **ImageUrl** property as the image default. You may also want to use the **Selected** and **Enabled** properties to specify the state of the item when the Web page first loads. The **NavigateUrl** and **Target** properties let you use the node to navigate to another Web page.

Collection Editors

RadSiteMap uses two associated collection editors, the **NavigationItemBinding Collection Editor** which is used to edit the **DataBindings** property collection, and the **LevelSettings Collection Editor**, which is used to define the appearance of the nodes according to their level in the hierarchy.

You have already seen how the **NavigationItemBinding Collection Editor** works in the Data Binding chapter. Let us look briefly at the **LevelSettings Collection Editor**:



When using the **LevelSettings Collection Editor**, you can control the appearance of the associated Level in the RadSiteMap.

With the **Level** property you can choose to which Level you will perform changes in appearance. When set to -1 the following changes will be made to all levels.

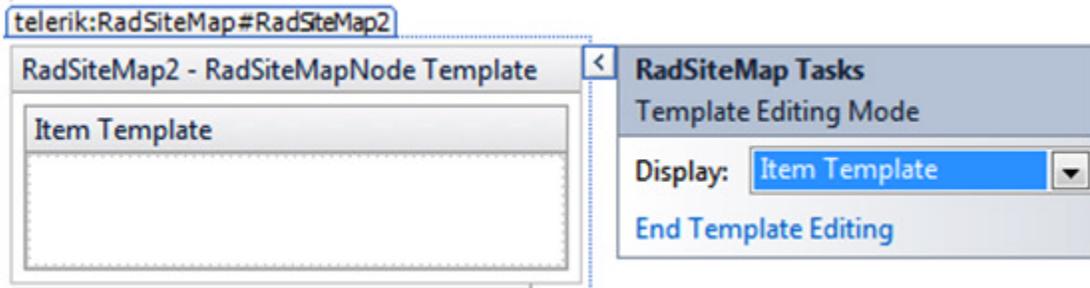
The **Layout** property has two values: **List** and **Table-Like**. It controls the way the nodes are aligned.

MaximumNodes property controls the number of nodes per parent level. When the number of nodes over reaches this property - all the nodes over the value of **MaximumNodes** are moved to other column.

SeparatorText determines the character which will separate the nodes.

Template Design Surface

You can use the site map's Smart Tag or context menu to bring up the Template Design surface, where you can create item templates. RadSiteMap supports two types of template: a global RadSiteMap template that affects all nodes in the site map, and individual item templates, which are associated with specific nodes in the Nodes collection. A drop-down control on the RadSiteMap Smart Tag (when it is in template editing mode) lets you specify which template you want to edit:



When a site map includes both a RadSiteMap template and individual item templates, the item templates have priority over the RadSiteMap template. That is, the RadSiteMap template is used for every node that does not have its own item template.

18.5 Server Side Programming

When working with RadSiteMap in the code-behind, you can leverage what you have learned already from other controls. In the chapter on Navigation controls, you were introduced to controls that have similarly hierarchical items collections, and the technique for manipulating those items is similar: the main difference is that in RadSiteMap, the items collection is called **Nodes** rather than **Items**.

Server-Side events

RadSiteMap supports a number of server-side events for responding in the code-behind when the user interacts with the site map.

NodeDataBound

The **NodeDataBound** event fires for every Node that is bound to data. The following example illustrates the use of the **NodeDataBound** event. The Web Page contains a RadSiteMap bound to a SqlDataSource. When user hovers on a node of the site map - tooltip appears referencing the text and the url of the node.

Products

ASP.NET AJAX Controls

ASP.NET MVC Extensions

Silverlight Controls

WPF Controls

WinForms Controls

Support

Documentation

Knowledge Base

Demos

ASP.NET AJAX Controls - <http://www.telerik.com/products/aspnet-ajax.aspx>



You can find the complete source for this project at:
\\VS Projects\SiteMap\ServerNodeDataBound

From the *Getting Started* chapter you are already familiar how to bind the RadSiteMap to a declarative data sources such as **SqlDataSource**. The addition here is that we are subscribing to the **NodeDataBound** event of the RadSiteMap.

[ASP.NET] Setting Tooltip when subscribing to the NodeDataBound event

```
<telerik:RadSiteMap ID="RadSiteMap1" runat="server" Skin="Sunset" DataFieldID="ID"
    DataFieldParentID="ParentID" DataSourceID="SqlDataSource1"
    DataTextField="Title" DataNavigateUrlField="URL"
    OnNodeDataBound="RadSiteMap1_NodeDataBound" >
    </telerik:RadSiteMap>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:SiteMapConnectionString %>"
        SelectCommand="SELECT * FROM [SiteMap]">
    </asp:SqlDataSource>
```

In the code-behind, the **NodeDataBound** event handler uses the **DataItem** property to access the underlying object or data row being bound to. The Tooltip of the current node is set with appending the Title and Url attributes:

[VB] NodeDataBound event handler

```
Protected Sub RadSiteMap1_NodeDataBound(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadSiteMapNodeEventArgs)
    Dim nodeData As DataRowView = TryCast(e.Node.DataItem, DataRowView)
    e.Node.ToolTip = nodeData("Title").ToString() + " - " + nodeData("URL").ToString()
End Sub
```

[CS] NodeDataBound event handler

```
protected void RadSiteMap1_NodeDataBound(object sender, RadSiteMapNodeEventArgs e)
{
    DataRowView nodeData = e.Node.DataItem as DataRowView;
    e.Node.ToolTip = nodeData["Title"].ToString() + " - " + nodeData["URL"].ToString
();
}
```

18.6 How To

Use Templates

Templates allow you to embed any content inside a RadSiteMapNode. The following walk-through illustrates how to use templates in the RadSiteMap control.

1. Create a new ASP.NET Web Site and drag a ScriptManager from the Tool Box onto the Web page.
2. Drag RadSiteMap control from the Toolbox to your Web page. Set its **Skin** property to "Black". Set Width to "300px" and **ShowNodeLines** to "true".
3. In the Solution Explorer, choose "Add New Item..." In the templates dialog, select Site Map.
4. Click the Add button. Visual Studio generates the web.sitemap file with the initial code. Populate the Web.sitemap as follows:
 - On the first siteMapNode, set only the title="Software" and the description to "Telerik home page".
 - On the second, set the url to "http://www.microsoft.com/student/en/us/software/windows-7.aspx", title="Windows 7" and the description to "Windows 7".
 - On the third, set the url to "http://www.microsoft.com/student/en/us/software/visual-studio.aspx", title="Visual Studio" and the description to "Visual Studio".
 - On the fourth, set the url to "http://www.microsoft.com/student/en/us/software/expression-studio.aspx", title="Expression Studio" and the description to "Expression Studio".
 - On the fifth, set the url to "http://www.microsoft.com/student/en/us/software/windowslive.aspx", title="Windows Live" and the description to "Windows Live".

UI for ASP.NET AJAX

- On the sixth, set the url to “http://www.microsoft.com/student/en/us/software/office-2007.aspx”, title=“Microsoft Office” and the description to “Microsoft Office”.

In the RadSiteMap Smart Tag, select “<New data source...>” from the “Choose Data Source” drop-down. 6.

1. In the first page of the DataSource Configuration Wizard, select “SiteMap” as the application type, and click OK to move to the next page.
2. Right-click on the project in Solution Explorer Window and add new folder with name “Images”. Locate the 5 needed images for the project: windows7.png, visual_studio.png, expression.png, windowslive.png and office2010.png.
3. Add the following code lines to your RadSiteMap:

[ASP.NET] How To Use Templates

```
<telerik:RadSiteMap ID="RadSiteMap1" runat="server" DataSourceID="SiteMapDataSource1"
    OnNodeDataBound="RadSiteMap1_NodeDataBound" ShowNodeLines="True"
    Skin="Black" Width="300px">
    <LevelSettings>
        <telerik:SiteMapLevelSetting Level="0" MaximumNodes="1">
            <NodeTemplate>
                <h3 class="Header">
                    <%# DataBinder.Eval(Container.DataItem, "title") %></h3>
                </NodeTemplate>
            </telerik:SiteMapLevelSetting>
            <telerik:SiteMapLevelSetting>
                <NodeTemplate>
                    <asp:Image ID="Image1" runat="server" Width="60px" Height="50px"
                        CssClass="align" />
                    <a href='<%# DataBinder.Eval(Container.DataItem, "url") %>'>
                        <%# DataBinder.Eval(Container.DataItem, "title") %>
                    </a>
                </NodeTemplate>
            </telerik:SiteMapLevelSetting>
        </LevelSettings>
    </telerik:RadSiteMap>
```

1. As you see we’ve attached to the NodeDataBound event to be able to set different images for different nodes. Here’s the code in the code-behind:

[CS] How To use Templates: NodeDataBound event handler

```
protected void RadSiteMap1_NodeDataBound(object sender, RadSiteMapNodeEventArgs e)
{
    if (!e.Node.Text.Equals("Software"))
    {
        Image img = (Image)e.Node.FindControl("Image1");
        string imageUrl = null;
        switch (e.Node.Text)
        {
            case "Windows 7": imageUrl = "Images/windows7.png";
                break;
            case "Visual Studio": imageUrl = "Images/visual_studio.png";
                break;
            case "Expression Studio": imageUrl = "Images/expression.png";
                break;
            case "Windows Live": imageUrl = "Images/windowslive.png";
                break;
        }
    }
}
```

```

        break;
    case "Microsoft Office": imageUrl = "Images/office2010.png";
        break;
    default:
        break;
    }
    img.ImageUrl = imageUrl;
}
}

```

[VB] How To use templates: NodeDataBound event handler

```

Protected Sub RadSiteMap1_NodeDataBound(ByVal sender As Object, ByVal e As
RadSiteMapNodeEventArgs)
    If Not e.Node.Text.Equals("Software") Then
        Dim img As Image = DirectCast(e.Node.FindControl("Image1"), Image)
        Dim imageUrl As String = Nothing
        Select Case e.Node.Text
            Case "Windows 7"
                imageUrl = "Images/windows7.png"
                Exit Select
            Case "Visual Studio"
                imageUrl = "Images/visual_studio.png"
                Exit Select
            Case "Expression Studio"
                imageUrl = "Images/expression.png"
                Exit Select
            Case "Windows Live"
                imageUrl = "Images/windowslive.png"
                Exit Select
            Case "Microsoft Office"
                imageUrl = "Images/office2010.png"
                Exit Select
            Case Else
                Exit Select
        End Select
        img.ImageUrl = imageUrl
    End If
End Sub

```

1. Press CTRL-F5 to run the application. You see that each node has a link and image in front of it.
2. We need to make the link to be on the line of the image - not on top. To change this add **CssClass** property to the Image control and set it **“align”**:

[ASP.NET]

```
<asp:Image ID="Image1" runat="server" Width="60px" Height="50px" CssClass="align"/>
```

1. Add this under the **<title></title>** tag:

[CSS]

```

<style type="text/css">
    .align
    {
        vertical-align: middle;
    }
</style>

```

1. Ctrl-F5 to run the application again. This time the images and links are on one level. All of the nodes of the RadSiteMap have images except the “Software” node. This is because it is in a different level and needs to differentiate from the others.



You can find the complete source for this project at:

`\VS Projects\SiteMap\HowToSiteMapTemplates`

18.7 Summary

In this chapter you’ve looked at the RadSiteMap control and saw how you could add an attractive site map to your Web applications. You’ve created a simple application that populated one site map from a **RadSiteMapDataSource** and another with items loaded from a **SqlDataSource**.

You’ve explored the design time support for the site map and understood many of the properties and groups of properties you can use to configure the site map and its nodes at design time.

You’ve learned some of the server-side properties and methods, especially the **NodeDataBound** event.

Finally, you’ve learned how to use Templates in a RadSiteMap. You’ve built a simple application that used templates learned how to find controls in server code.

19 RadGrid

19.1 Objectives

- Explore the features of the RadGrid control.
- Create a simple application that binds to some data to see the basic functionality of auto-generated columns and column manipulation.
- Explore the RadGrid design time interface, including Smart Tag, Properties View and Property Builder.
- Enable and explain the principal properties and groups of properties where the most common functionality is found.
- Learn the most commonly used server-side API events and properties.
- Learn server-side coding techniques, including manual CRUD (create, read, update and delete) operations and accessing and changing the data in a grid by replacing values with images.
- Learn the client-side API with a comprehensive reference to all of the events, methods and properties of the RadGrid and GridTableView.
- Explore some advanced techniques in client-side code, such as client-side databinding and accessing values, changing appearance and binding to events with an example of client cell selection.

19.2 Introduction

In any web application where you need to display a list of data with more than one field, you are likely going to want to use a grid. The RadGrid is an advanced control with many built in features that allow you to enable the most popular features of a grid with very little customization work.

The RadGrid...

- Allows you to enable fully functional multi-column sorting with a single setting. Sorting can be useful for arranging data in ways that are more useful to the user at any given time.
- Can enable a powerful filtering interface that can help in finding data without having to display all of it at the same time.
- Has an easily customizable paging feature that displays only small amounts of data at once to increase performance and decrease real estate usage while allowing access to a large number of records.

The diagram illustrates a RadGrid control with the following features highlighted:

- Filter interface:** Points to the search boxes above the columns.
- Sorting interface:** Points to the sort arrows on the column headers.
- Paging interface:** Points to the 'Change page' navigation controls.
- Page display:** Points to the status text 'Displaying page 1 of 3, items 1 to 5 of 15.'

VendorID	Account Number	Name	PurchasingWebServiceURL	ModifiedDate
27	CAPITAL0001	Capital Road Cycles		1/24/2002 12:00:00 AM
86	CARLSON0001	Carlson Specialties		1/24/2002 12:00:00 AM
45	CHICAGO0001	Chicago City Saddles		1/24/2002 12:00:00 AM
93	CHICAGO0002	Chicago Rent-All		1/24/2002 12:00:00 AM
64	CIRCUIT0001	Circuit Cycles		1/24/2002 12:00:00 AM

UI for ASP.NET AJAX

- Supports sub-grids to display hierarchical data.

		AccountNumber	Name	CR	Pref	Active	Purchase	
>	Select	INTERNAT0001	International	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
▼	Select	ELECTRON0002	Electronic Bike Repair & Supplies	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		
		PurchaseOrderID	RevisionNumber	OrderDate	ShipDate	TotalDue		
		32	2	2/8/2002 12:00:00 AM	2/17/2002 12:00:00 AM	31817.5358		
		111	0	4/9/2002 12:00:00 AM	4/18/2002 12:00:00 AM	21256.3601		
		190	0	7/24/2002 12:00:00 AM	8/2/2002 12:00:00 AM	53073.8959		
		269	0	11/18/2002 12:00:00 AM	11/27/2002 12:00:00 AM	53073.8959		
		348	0	6/15/2003 12:00:00 AM	6/24/2003 12:00:00 AM	53073.8959		
		Change page: < >					Displaying page 1 of 11, items 1 to 5 of 51.	
>	Select	PREMIER0001	Premier Sport, Inc.	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>		

- Can easily implement multilevel grouping of data from a single table - just drag the column header(s) to the group panel on the top, which defines the grouping order and hierarchy. You can also programmatically group the data using the group-by expressions.

ModifiedDate	AccountNumber	Name	CR	Pref
▼ ModifiedDate: 05/26/2001				
▼ CR: 1				
>	ADVANCED0001	Advanced Bicycles	1	<input checked="" type="checkbox"/>
>	LITWARE0001	Litware, Inc.	1	<input checked="" type="checkbox"/>
>	AMERICAN0001	American Bicycles and Wheels	1	<input checked="" type="checkbox"/>
▼ CR: 2				
>	ALLENSON0001	Allenson Cycles	2	<input checked="" type="checkbox"/>

- Supports all widely used column types (GridEditCommandColumn, GridBoundColumn, GridCheckBoxColumn, GridDropDownColumn, GridButtonColumn, GridHyperLinkColumn, GridClientSelectColumn, etc.), columns with other Telerik controls as column editors (GridDateTimeColumn, GridNumericColumn, GridMaskedColumn, GridHTMLEditorColumn, etc.) as well as GridTemplateColumns, which give you complete freedom over the data layout and formatting.
- Can easily export the content to Microsoft Excel/Microsoft Word/CSV/PDF.

19.3 Getting Started

This article will introduce you to the main features of the RadGrid control. You will see that in most cases they can be set up with little or no server-side coding.

Paging

RadGrid natively supports table paging, which lets you view large sets of data in small chunks for faster loading

and easier navigation. It also provides a set of events, helper methods and properties if the paging operation requires custom intervention. Set the `AllowPaging` property to `True` to have RadGrid handle paging. By default, the `AllowPaging` property is `False`. You can set the **AllowPaging** property on the entire grid, or set it for each `GridTableView` individually.

If you want to handle paging in a custom manner, set the grid's **AllowCustomPaging** property to `True` as well and pass only the data needed for the current page in the `NeedDataSource` event. Examples of custom paging are shown in this [online demo \(http://demos.telerik.com/aspnet-ajax/grid/examples/programming/custompaging/defaultcs.aspx\)](http://demos.telerik.com/aspnet-ajax/grid/examples/programming/custompaging/defaultcs.aspx).

Set the **PageSize** property on the grid or table view to specify the number of records that should appear in each chunk. When paging is enabled, RadGrid renders a pager item on the bottom and/or top of each `GridTableView` displayed when the number of records in the table view exceeds the page size. If you want to show the pager even if there is a single page of items displayed, set **GridTableView.PagerStyle.AlwaysVisible** to `true`.

Sorting

You can have RadGrid automatically sort its columns by setting the **AllowSorting** property to `True`. When sorting is enabled you can click the column headers to trigger it. Sort arrows will appear next to the header text to indicate the sort order. There are three sorting modes:

- Ascending
- Descending
- No Sort

They are toggled subsequently by clicking the column header.

You can configure the grid to allow sorting by more than one `DataField`. For this purpose set **AllowMultiColumnSorting** property to `true`.

To control the sorting of RadGrid, you can modify the `GridTableView`'s **SortExpressions** collection. You can add a `SortExpression` both declaratively and programmatically to provide a default sorting for your grid.

Scrolling

When constructing a Web page that contains a grid, there are design limitations regarding the size of the grid. In such cases, you may need to enable client-side grid scrolling, so that the RadGrid can fit in the allowed space. You can enable scrolling by setting the **ClientSettings.Scrolling.AllowScroll** property to `True` (By default its value is `False`.)



The **ClientSettings.Scrolling.ScrollHeight** property specifies the height value beyond which scrolling is turned on. The default value is `300px`.

RadGrid enhances the simple scrolling by supporting **static headers** ([grid-scroll-with-static-headers.html](#)) and **frozen columns** ([grid-frozen-columns.html](#)) - grid header and pager remain static, even when the grid is scrolled. Furthermore, there is a **virtual scrolling** ([grid-virtual-scroll-paging.html](#)) option that fetches only specified range of records to be visualized on the current page.

Filtering

RadGrid natively supports filtering of table columns. To enable or disable filtering, set the **AllowFilteringByColumn** property of the RadGrid. When filtering is enabled, a filtering item appears below the column header. The user can enter a filter value in the filter box. A menu can be toggled by clicking the button next to the filter box to allow the user to select a filter function that is applied to the column. When the user makes a selection in this menu the grid will display the records matching the filter criteria specified using the filter boxes.

You can disable filtering for specific columns by setting the column's **AllowFiltering** property to `false`. If you set **AutoPostBackOnFilter** property of a column to `True`, the user does not need to press the filter button to initiate filtering. Instead, a postback filter operation occurs when the user types a filter in the filter box and presses [Enter] from the keyboard. When `AutoPostBackOnFilter` is `True`, the column assumes a filter operation

of *Contains* for string types or *EqualTo* for numeric types. You can change this to another filter function by setting the **CurrentFilterFunction** property.

To customize filtering in RadGrid, you can manipulate the **FilterExpression** property of the respective GridTableView object. This string represents the current filter function in the same way as the **DataView.Filter** property, you can picture it as the text of a WHERE clause for filtering a table of data. If you want to customize filtering using your own custom statements, clear the FilterExpression string (to prevent the default filtering) and bind the grid to a filtered data set.

Grouping

RadGrid supports grouping of items based on the value of a particular column. You can even have multilevel grouping based on different criteria. To group the data in a grid, specify grouping criteria by setting the **GroupByExpressions** property of a table view in the grid. You can set the group-by expressions declaratively at design time, or programmatically in the code-behind.

To facilitate grouping, a special area called the **GridGroupPanel** can be displayed at the top of the grid to display grouping options. To display the group panel, set the grid's **ShowGroupPanel** property to True. When a table view is grouped, all group fields appear in this group panel as elements along with an icon that indicates the sort order. To allow users to change the grouping by dragging column headers, set the **ClientSettings.AllowDragToGroup** property to True.

You can specify whether a table view in the grid handles grouping on the client or on the server:

- **Server-side group loading:** To enable grouping on the server, set the **GroupLoadMode** property of a table view to "Server". When grouping is handled on the server, the grid performs a postback to the server every time a group is expanded.
- **Client-side group loading:** To enable loading the groups on the client, set the **GroupLoadMode** property of a table view to "Client" and the **ClientSettings.AllowGroupExpandCollapse** property to True. When grouping is handled on the client, groups are expanded client-side, without a postback. This means that the data for all groups, whether they are expanded or not, will be loaded on the client.

Each GridTableView object contains a **GroupHeaderTemplate** and **GroupFooterTemplate** properties for specifying a template that will show inside each group's header and footer rows. You can use them to provide a more customized look for the group totals display. For more information on working with group header and footer templates, check out the online documentation.

Data Editing

RadGrid exposes a number of options for editing the data that it is bound to. You could easily set up automatic operations or go to more custom scenarios with manual data editing. There are different editing modes and types of forms which you could use to deliver various ways of enabling the user to edit the grid data source.

Edit modes

RadGrid offers the following edit modes. They are switched through the **GridTableView.EditMode** property:

- **InPlace:** displays the grid column editors inline when the grid switches into edit mode - the edit controls show in place of the text values of the cells.
- **EditForms:** The grid column editors display in an auto-generated form when the grid switches into edit mode, the edit form appears immediately below the item that is being edited.
- **Popup:** The grid column editors display in an auto-generated popup form when the grid switches into edit mode.

Edit form types

RadGrid supports three types of edit forms: auto-generated, user control and form template. You can use the **GridTableView.EditFormSettings.EditFormType** property to switch between them. The default type is auto-generated. The three types function as follows:

- **AutoGenerated:** RadGrid will generate the edit form for you - when an item goes in edit mode, the built-in editors for each column, which is not marked as ReadOnly, will be displayed.
- **Template:** provides you with full control over what is shown in the grid edit form, you can define your template in the FormTemplate tag of the control (GridView-EditFormSettings-FormTemplate). Note that you can use the template edit form only with EditForms and Popup edit modes.
- **WebUserControl:** when you want to reuse the edit form, it would be best to define it in a user control and assign it to the grids which you want to use it. In this case you need to set the **EditFormType** property to **WebUserControl**. In addition in the EditFormSettings set the **UserControlName** property to the path of the custom user control.

Hierarchical RadGrid

RadGrid gives you the ability to display hierarchical data through DetailTables or by defining its NestedViewTemplate. Below are listed the different approaches you can use:

DetailTables

The RadGrid control renders one or more detail tables for each item (row) in the MasterTableView. In a multi-level hierarchy, each item of every detail table can have one or more detail tables as well. To describe the data hierarchy, each table view must have its own data source with **ParentTableRelations** or implement the **DetailTableDataBind** event handler. In the next section you can see how to build a hierarchical grid with detail tables through RadGrid designer.

NestedViewTemplate

With the NestedViewTemplate you have the ability to model the look and feel of the child table container in order to display the detail info in non table-dependant format.

In addition RadGrid exposes a property for its table view objects called **NestedViewSettings**. The NestedViewSettings allow you to specify a data source object contained on the page to which the template should be bound, as well as a relation to the parent level. These can be defined declaratively or programmatically through the **NestedViewSettings.DataSourceID** and **NestedViewSettings.ParentTableRelation** properties respectively. The ParentTableRelation is specified in the same way as the declarative relations for hierarchical tables.

With the hierarchy declarative relations, you should have a WHERE clause in the SelectCommand of the data source control for the nested view template to retrieve the record for it. The WHERE clause should include the field from the ParentTableRelation definition between the master/child table. Furthermore, the same field has to be included in the SelectParameters of the "inner" data source controls with exactly the same Name. However, no SessionField value is required.

If more than one record is fetched from the data source for the nested view template, only the first one will be used to bind the controls in the latter.

Below is a code extraction:

ASPX

```
<telerik:ScriptManager ID="ScriptManager1" runat="server" />
  <telerik:RadAjaxManager ID="RadAjaxManager1" runat="server">
    <AjaxSettings>
      <telerik:AjaxSetting AjaxControlID="RadGrid1">
        <UpdatedControls>
          <telerik:AjaxUpdatedControl ControlID="RadGrid1"
LoadingPanelID="RadAjaxLoadingPanel1" />
        </UpdatedControls>
      </telerik:AjaxSetting>
    </AjaxSettings>
  </telerik:RadAjaxManager>
  <telerik:RadAjaxLoadingPanel ID="RadAjaxLoadingPanel1" runat="server" />
```

```

<telerik:RadGrid ID="RadGrid1" DataSourceID="SqlDataSource1" runat="server"
AutoGenerateColumns="False"
    AllowSorting="True" AllowPaging="True" PageSize="5" GridLines="None"
ShowGroupPanel="True">
    <MasterTableView DataSourceID="SqlDataSource1" DataKeyNames="CustomerID"
AllowMultiColumnSorting="True"
    GroupLoadMode="Server">
        <Columns>
            <telerik:GridBoundColumn DataField="CustomerID" HeaderText="CustomerID"
ReadOnly="True"
                SortExpression="CustomerID" UniqueName="CustomerID">
            </telerik:GridBoundColumn>
            <telerik:GridBoundColumn DataField="CompanyName" HeaderText="CompanyName"
                SortExpression="CompanyName" UniqueName="CompanyName">
            </telerik:GridBoundColumn>
            <telerik:GridBoundColumn DataField="ContactName" HeaderText="ContactName"
                SortExpression="ContactName" UniqueName="ContactName">
            </telerik:GridBoundColumn>
        </Columns>
        <NestedViewSettings DataSourceID="SqlDataSource2">
            <ParentTableRelation>
                <telerik:GridRelationFields DetailKeyField="CustomerID"
MasterKeyField="CustomerID" />
            </ParentTableRelation>
        </NestedViewSettings>
        <NestedViewTemplate>
            <asp:Panel ID="NestedViewPanel" runat="server" CssClass="viewWrap">
                <div class="contactWrap">
                    <fieldset style="padding: 10px;">
                        <legend style="padding: 5px;"><b>Detail info for Customer:<%=Eval
("ContactName") %></b>
                    </legend>
                    <table>
                        .....
                        <tr>
                            <td>
                                ContactTitle:
                            </td>
                            <td>
                                <asp:Label ID="titleLabel" Text='<%=Bind("ContactTitle")
%>'
                                runat="server"></asp:Label>
                            </td>
                        </tr>
                        <tr>
                            <td>
                                Address:
                            </td>
                            <td>
                                <asp:Label ID="addressLabel" Text='<%=Bind("Address") %
>'
                                runat="server"></asp:Label>
                            </td>
                        </tr>
                        .....
                    </table>
                </div>
            </asp:Panel>
        </NestedViewTemplate>
    </MasterTableView>
</telerik:RadGrid>

```

```

        </table>
    </fieldset>
</div>
</asp:Panel>
</NestedViewTemplate>
</MasterTableView>
<PagerStyle Mode="NumericPages"></PagerStyle>
<ClientSettings AllowDragToGroup="true" />
</telerik:RadGrid>
<asp:SqlDataSource ID="SqlDataSource2" ConnectionString="<$ ConnectionStrings>"
    SelectCommand="SELECT [CustomerID],[ContactName],[ContactTitle], [Address],[City],
[PostalCode],[Country],[Phone],[Fax] FROM [Customers] where CustomerID=@CustomerID"
    runat="server">
    <SelectParameters>
        <asp:Parameter Name="CustomerID" />
    </SelectParameters>
</asp:SqlDataSource>
<asp:SqlDataSource ID="SqlDataSource1" ConnectionString="<$ ConnectionStrings>"
    SelectCommand="SELECT [CustomerID], [CompanyName], [ContactName]FROM [Customers]"
    runat="server"></asp:SqlDataSource>

```

Auto-generated hierarchy

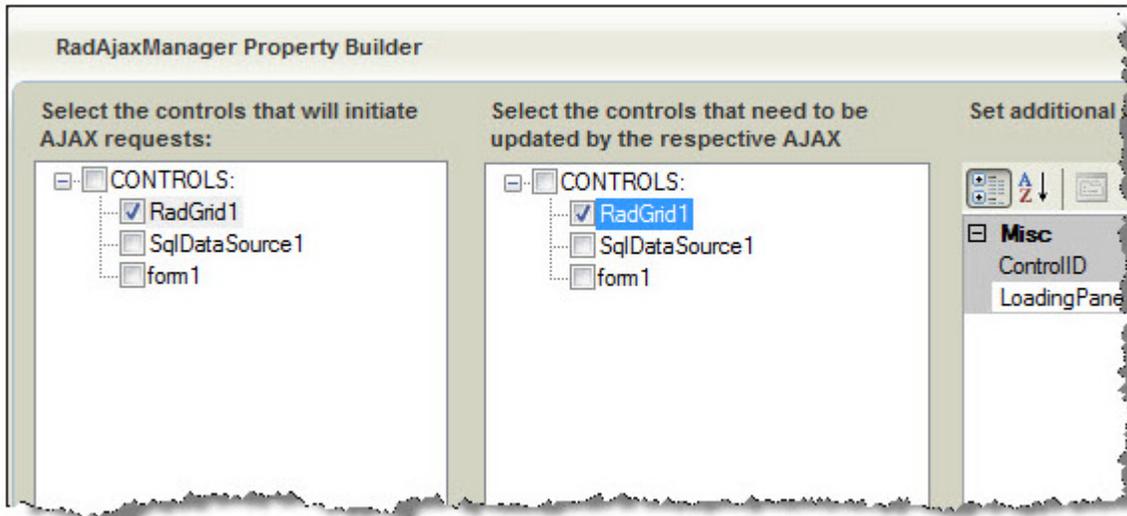
RadGrid also has the capability to auto-generate a hierarchical representation of a multi-table DataSet. Just set the **AutogenerateHierarchy** property of RadGrid to **true** and it will automatically generate the hierarchy based on the tables in the DataSet and their relations with one another.

The detail tables generation will start from the table which name has been set to the DataMember property of the DataSet. In other words, RadGrid will assume this data-table as the data source for its MasterTableView. If no table name has been specified for the DataMember of the DataSet object, the first table in the ladder will be treated as the data source for the grid's MasterTableView and the generation of the detail tables will go from there following the root data-table child relations.

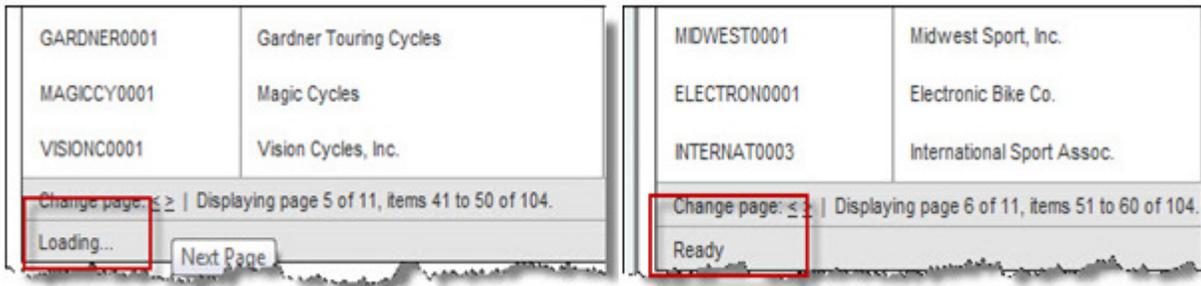
Enabling Ajax

When using the RadGrid, you will almost always want to use AJAX to prevent postbacks every time something in the RadGrid is changed.

1. If you do not already have a ScriptManager or RadScriptManager on your webpage, add a RadScriptManager using the RadGrid's Smart Tag.
2. In the Smart Tag there will be a link to register the RadScriptManager. Click that to register it in web.config.
3. Add a RadAjaxManager and configure it to have the RadGrid have an association with itself.



- Now that the association is made, enable the RadGrid's **ShowStatusBar** property which will add an indicator to your grid that shows if it is loading or not.

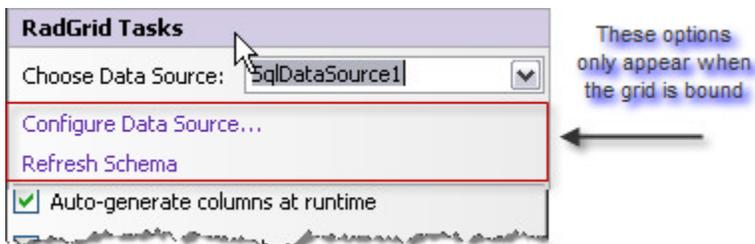


Now you have a working, databound, Ajaxified RadGrid.

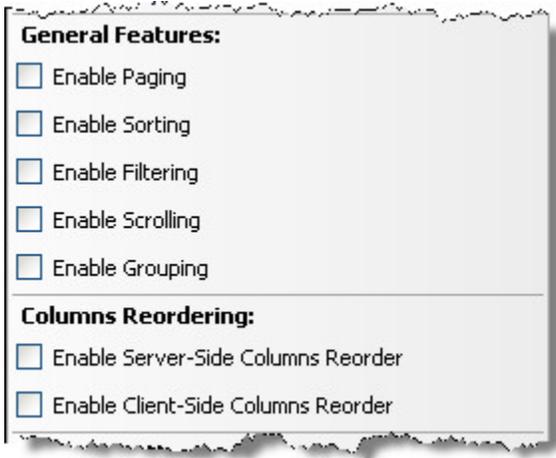
19.4 Using the Design Time Interface

Smart Tag

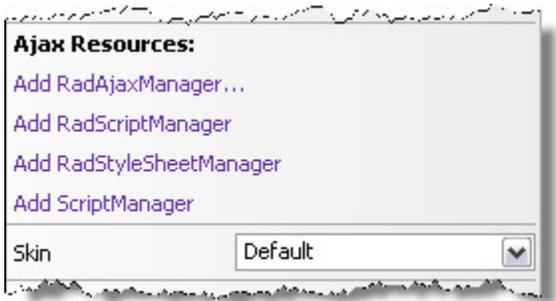
The **Smart Tag** for RadGrid contains the most popular properties so they can be easily enabled without digging through the Properties window. This includes the common databinding options.



There are three options to auto generate columns. By default, "Auto-generate columns at runtime" is turned on. The Auto-generate edit and delete column properties add ButtonColumns for deleting records and putting records into edit mode. The General Features area lets you turn on the commonly used Paging, Sorting, Filtering, Scrolling and Grouping features with their default options. Column Reordering can be enabled on the client or server side which allows you to drag columns around and change the order they appear in the grid.



It also provides useful links that allow you to easily Ajaxify the control and optimize the web page that the grid appears on.

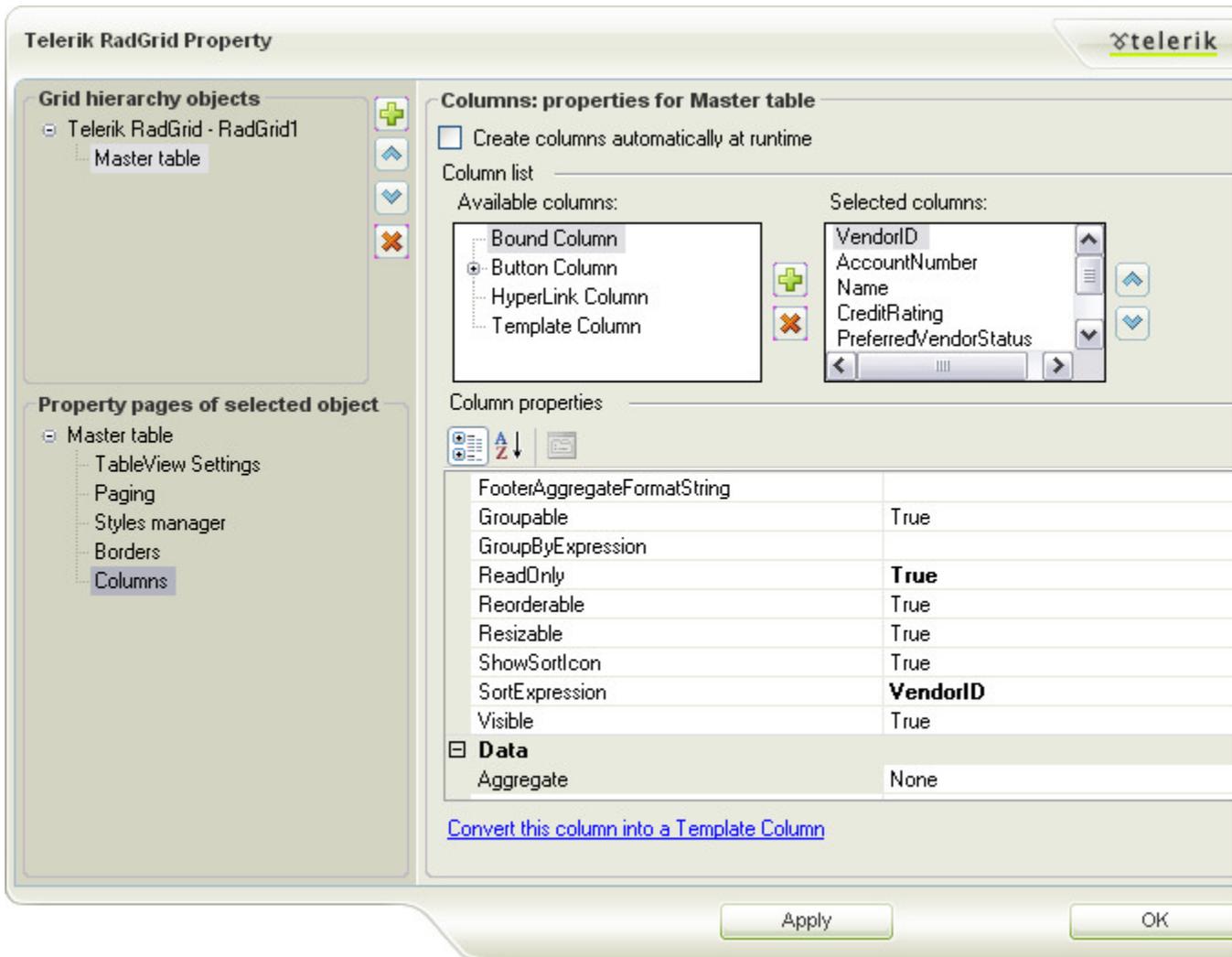


Property Builder

You can open the Property Builder in the Smart Tag.



The Property Builder is a way of accessing almost every property available to the RadGrid in a more organized format than the Properties window. It is also good at displaying the current layout of columns, allowing easy access to column layout and properties.



Properties Window

If you know your way around the control, the Properties window can be a quicker way to manipulate the RadGrid and it is the only way to access and auto-create the server-side events in the CodeBehind from the Design Time Interface.

The screenshot shows the Visual Studio Properties window for a **RadGrid1** control. The control type is identified as **Telerik.Web.UI.RadGrid**. The **Action** category is expanded, and the **NeedDataSource** event is selected. Below the event list, the **NeedDataSource** event is described as: "Fires when the grid is about to be bound and the data source must be assigned".

Category	Property/Event	Value
Action	CancelCommand	
	ColumnsReorder	
	CreateColumnEditor	
	DeleteCommand	
	DetailTableDataBind	
	EditCommand	
	GroupsChanging	
	InsertCommand	
	ItemCommand	
	ItemEvent	
	NeedDataSource	
	PageIndexChanged	
	PageSizeChanged	
	RowDrop	
	SelectedIndexChanged	
SortCommand		
UpdateCommand		
Behavior	ColumnCreated	
	ColumnCreating	
	CustomAggregate	
	ItemCreated	
	ItemDataBound	
Data	DataBinding	
	DataBound	
Data editing	ItemDeleted	
	ItemInserted	
	ItemUpdated	
Grouping	GroupPanel	RadGrid1.GroupPane
Layout	MasterTableView	RadGrid1.MasterTab
Misc	Disposed	
	ExcelMLExportRowCre.	
	ExcelMLExportStylesCr	
	Init	

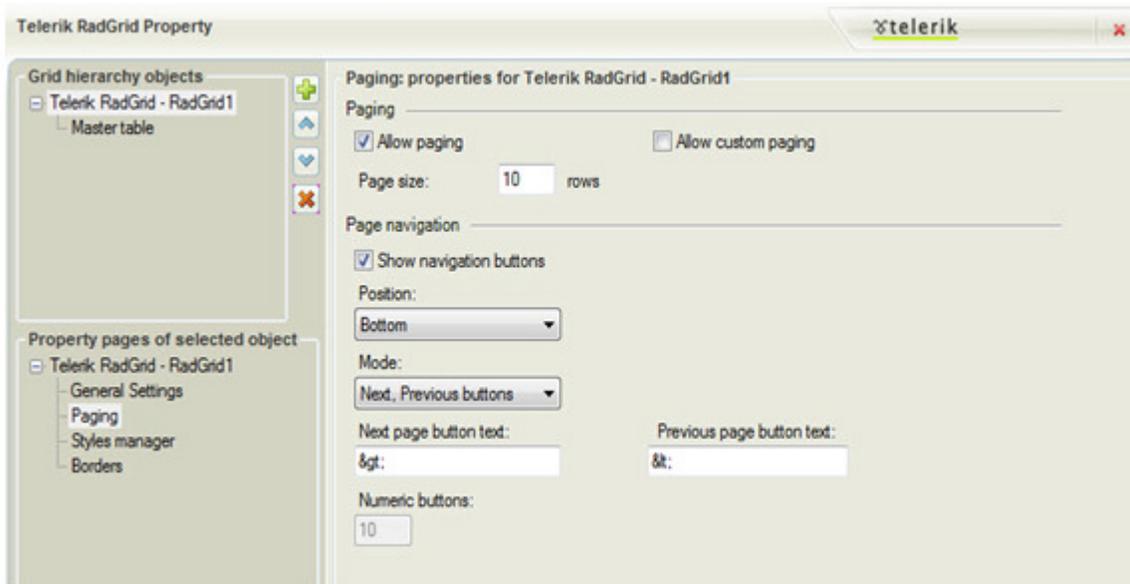
NeedDataSource
Fires when the grid is about to be bound and the data source must be assigned

Commonly Used Properties

The RadGrid has more design-time properties than perhaps any other RadControl. The most commonly used of these options are in the Smart Tag. We will also cover some of the important properties to be familiar with such as Detail Tables.

Paging

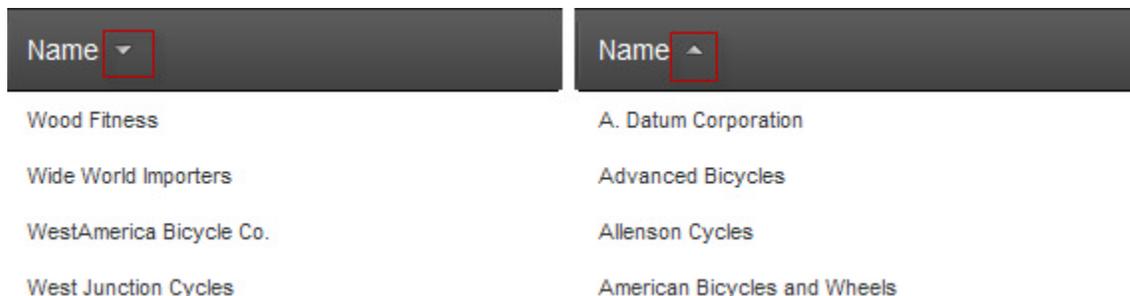
Paging allows you to limit the amount of data displayed at one time and provides controls to navigate pages of that limited data. This feature can be the single most important option for improving performance of a data-heavy web page. It also reduces the real estate usage and eliminates clumsy horizontal scrollbars. To enable this feature, simply check the **Enable Paging** checkbox in the Smart Tag. The default setting for this feature includes showing 10 records at once and some basic navigation controls. There are many options to customize this behavior, accessible through the Property Builder:



Possible settings here include the number of rows per page and navigation button settings. If you enable paging and enable AJAX, then run the application, you'll notice as you navigate the data that there are no postbacks. This slick performance is achievable with just a few mouse clicks.

Sorting

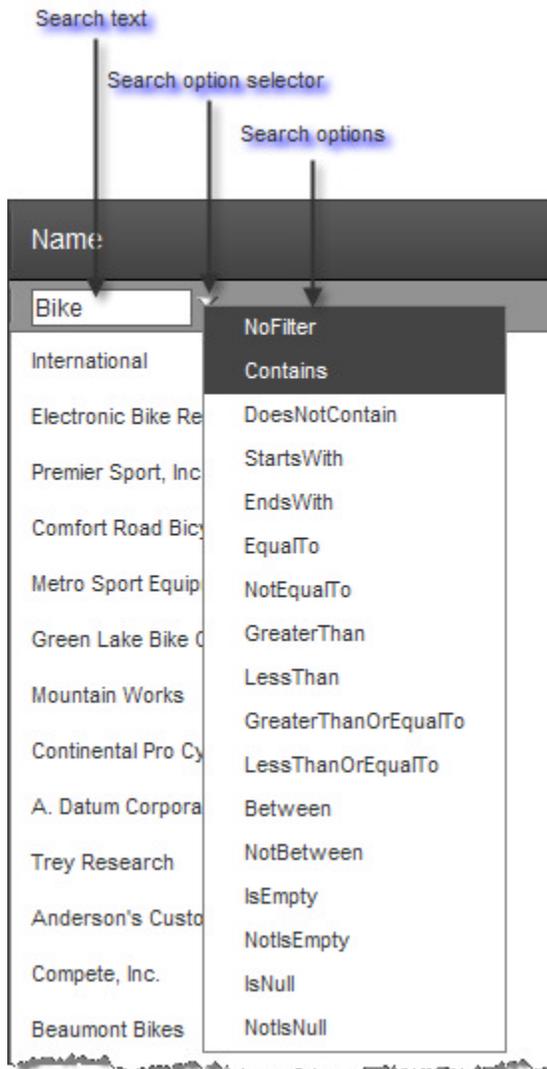
Sorting is a three-state functionality. When the contents of a sort-enabled grid are first displayed, they are sorted based on ordering from a SQL statement. When the **Enable Sorting** option is enabled, data is sorted by clicking on a column header. The first time the column header is clicked, the data will be sorted into ascending order, indicated by the triangular icon shown in the left-hand image below; the second time, it will be sorted in descending order, as shown in the right-hand image; and the third click will return the column to the original (grid-unsorted) order.



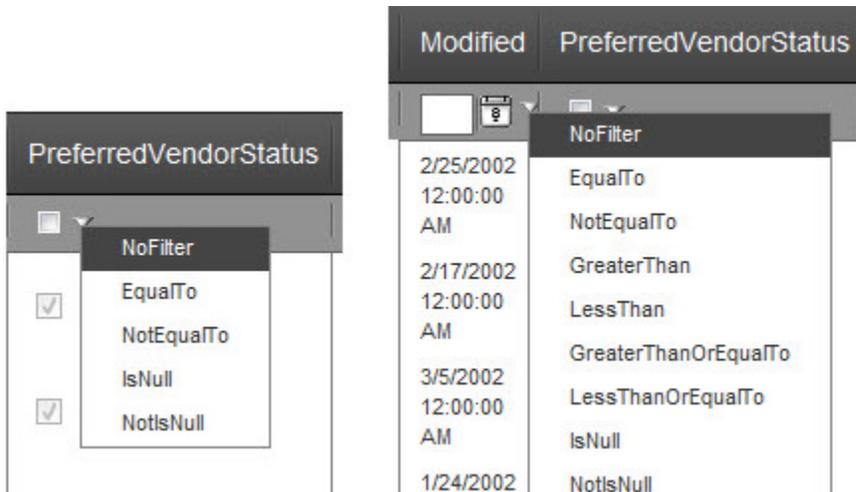
By default, only one sort specification is applied at a time; clicking on a different column header will do away with any previously selected sort order. In order to enable multiple column sorting, set the `AllowMultiColumnSorting` property.

Filtering

Filtering is really used as a search and navigation tool. If you have paging enabled, the page navigation interface can take too much work to find specific data. Enabling filtering will allow your users to easily navigate to specific data quickly while keeping the performance advantage of paging. Enabling filtering is easily done by selecting the `EnableFiltering` property in the Smart Tag.

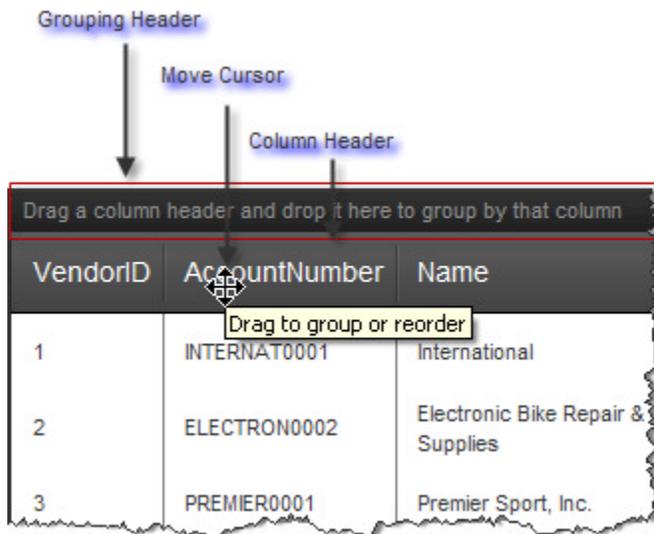


Special-type fields (such as dates and bit fields) have filter conditions conditions appropriate to the data type you're dealing with. The following screenshot shows the `PreferredVendorStatus` column showing only filter options that would apply to a boolean value, while the `Modified` column shows filter options that would only apply to dates.



Column Grouping

The next property is the Enable Grouping property. When grouping is enabled, an extra area will be displayed at the top of the grid, indicating that you can drag a column header to the grouping area to group by that column. When your cursor is hovered over a groupable column, you'll first see a tooltip indicating that the column is groupable, and the cursor will change to a "Move" cursor:



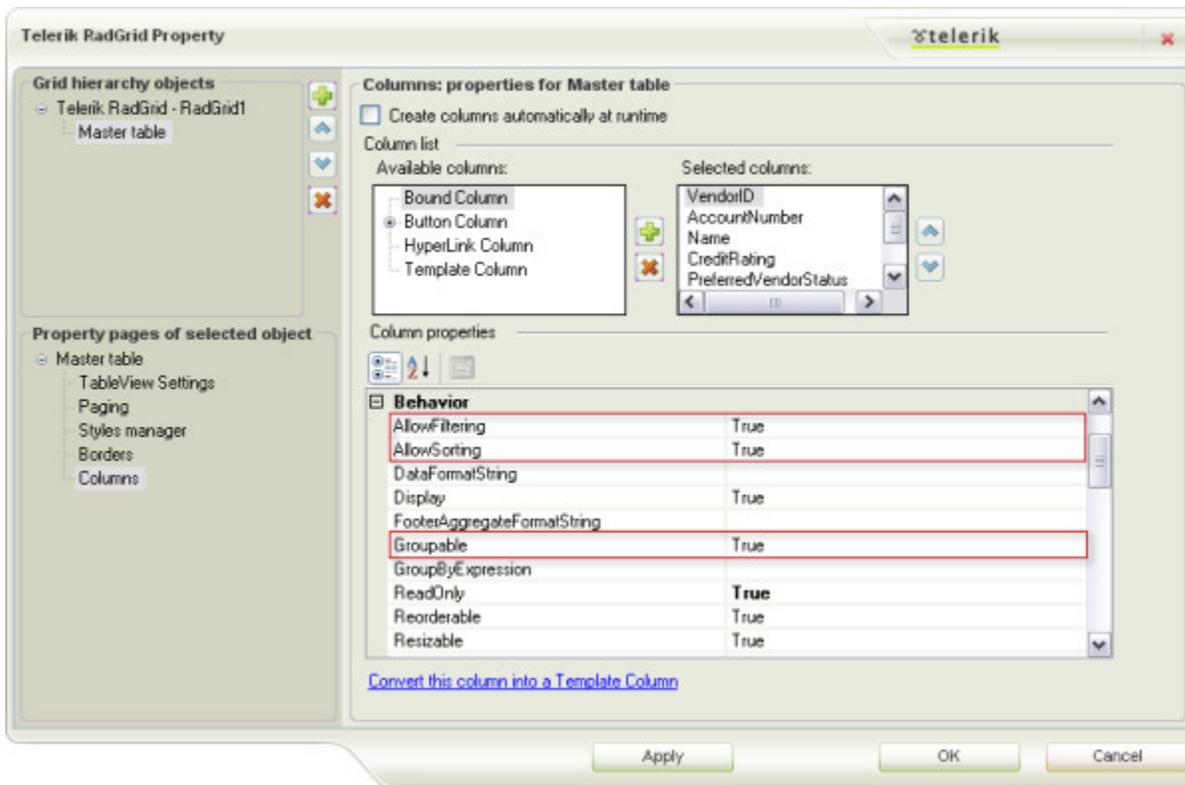
To create a column grouping, simply left-click a column header and drag it into the grouping header. As your cursor enters the grouping header, it will change back into an arrow, indicating that the group may be dropped and take effect. You can create multiple groupings that will be nested in the order that they appear in the grouping header. This order may be changed by dragging the existing groupings around in the grouping header. You may sort groupings using the arrow icon next to each grouping and each group may be expanded or collapsed using the arrow icon next to each group value.

The screenshot shows a data grid with a header row containing 'VendorID', 'AccountNumber', 'Name', and 'CreditRating'. The grid is grouped by 'ModifiedDate' and 'CreditRating'. Annotations point to the 'ModifiedDate' and 'CreditRating' headers, indicating their role in grouping and sorting. The data is organized into sections based on 'ModifiedDate' (5/26/2001 and 6/9/2001) and further grouped by 'CreditRating' (1 and 2).

VendorID	AccountNumber	Name	CreditRating
ModifiedDate: 5/26/2001 12:00:00 AM			
CreditRating: 1			
32	ADVANCED0001	Advanced Bicycles	1
83	LITWARE0001	Litware, Inc.	1
85	AMERICAN0001	American Bicycles and Wheels	1
CreditRating: 2			
38	ALLENSON0001	Allenson Cycles	2
ModifiedDate: 6/9/2001 12:00:00 AM			
CreditRating: 1			

Enabling and disabling Sorting, Filtering and Grouping

It may not make sense to have one or more of the sorting, filtering and grouping features available for any particular column. These features may be disabled on a column-by-column basis using the property builder and the appropriate column's properties.



Scrolling

When scrolling is enabled using the Enable Scrolling checkbox in the Smart Tag, you'll see a vertical scrollbar along the right-hand edge of the grid. The Enable Scrolling checkbox setting has a convenient set of defaults. The other properties associated with the scrolling sub-property look like this:

Scrolling	Telerik.We
AllowScroll	True
EnableVirtualScrollPaging	True
FrozenColumnsCount	0
SaveScrollPosition	True
ScrollHeight	300px
UseStaticHeaders	True

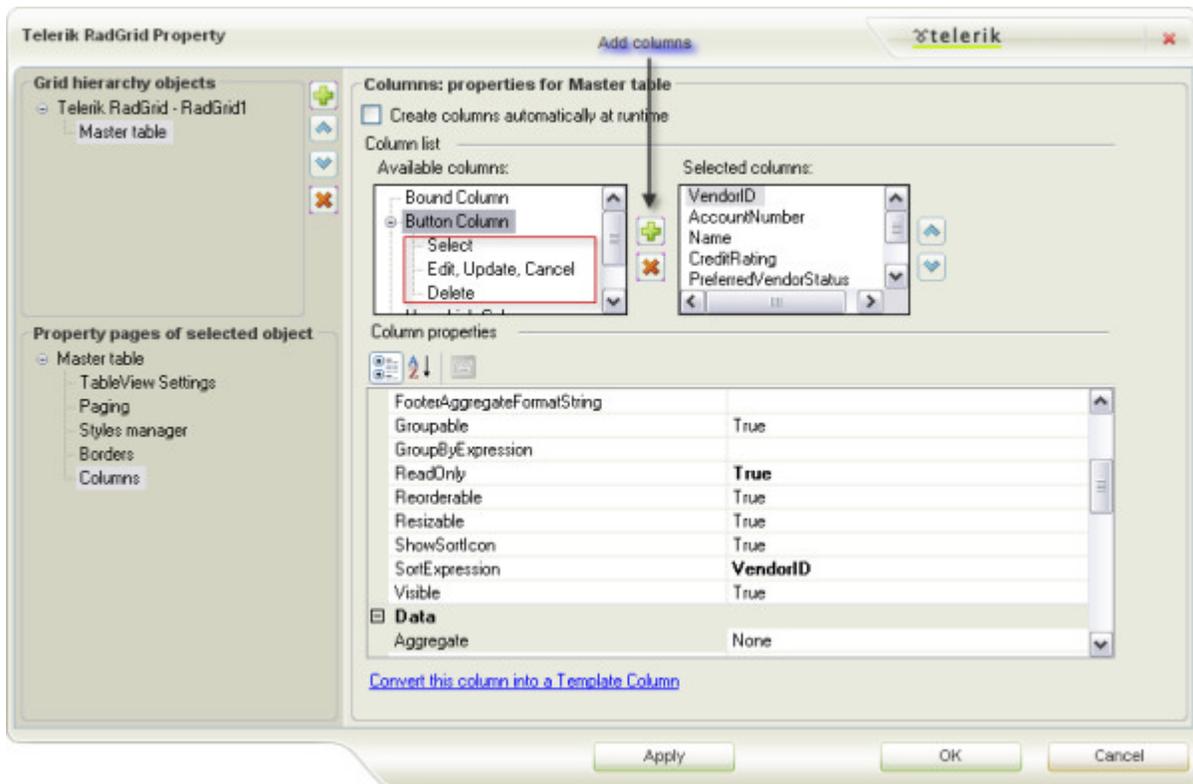
The **EnableVirtualScrolling** property is especially useful if you're dealing with large data sets. What this property does is to fill up the grid with data to the extent of the **ScrollHeight** property. Then, as you scroll down, RadGrid makes a request for additional data and displays that section of the grid. If virtual scrolling is not enabled, all data rows are returned and stored in the grid. Virtual scrolling improves initial load time when using large datasets, but will be slightly less smooth during scrolling when data is retrieved.

Using the RadGrid to Edit, Add and Delete Data

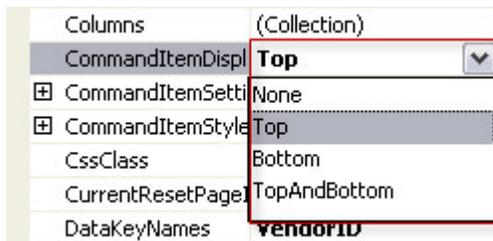
You will certainly want to edit live data in your database using the RadGrid at some point. RadGrid offers a fully functional and customizable interface to do so and most of it can be done directly from the designer. The following is the simplest path to enabling that functionality.

1. Go into the **Property Builder**, select the **Master Table**, select the **Columns** collection and then expand the **Button Column** tree in the **Available Columns** area. You will see three options there. Go ahead and add all three. These are simply button columns that perform the select action, the delete action or initiate edit mode. There are properties for each of these columns that allow you to change how they look, but for

now, concentrate on the default configuration.



2. In the Properties View of the RadGrid, go to the **MasterTableView** property and expand it, and then change the **CommandItemDisplay** property to "Top".

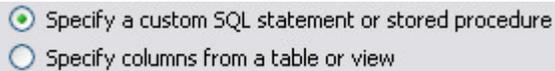


This will add a header to the top of the RadGrid that has an "Add" button and that will allow you to put the grid into an insert mode.



3. Next you need to set an edit mode for the table view. Go to the **MasterTableView** property in the Properties View, expand it and change the **EditMode** property to "InPlace". This is the simplest mode to use if you do not have any complex data entry requirements.
4. Now run the web application and play with the add, edit, select and delete buttons. You will see that the add and edit buttons create a row of entry fields to edit or change data based on the **EditMode** property "InPlace". You will also note that none of the buttons actually do anything to the data.
5. Close the web application and configure the datasource for the RadGrid.

- Click **Next** and choose "Select a custom SQL statement or stored procedure" and click **Next** again.



- In the UPDATE, INSERT and DELETE tabs, enter the following queries:

[T-SQL] Update

```
UPDATE Purchasing.[Vendor] SET [AccountNumber] = @AccountNumber, [Name] = @Name,
[CreditRating] = @CreditRating, [PreferredVendorStatus] = @PreferredVendorStatus,
[ActiveFlag] = @ActiveFlag, [PurchasingWebServiceURL] = @PurchasingWebServiceURL,
[ModifiedDate] = @ModifiedDate WHERE [VendorID] = @VendorID
```

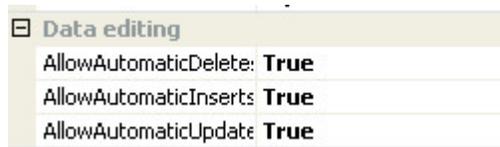
[T-SQL] Insert

```
INSERT INTO Purchasing.[Vendor] ([AccountNumber], [Name], [CreditRating],
[PreferredVendorStatus], [ActiveFlag], [PurchasingWebServiceURL], [ModifiedDate]) VALUES
(@AccountNumber, @Name, @CreditRating, @PreferredVendorStatus, @ActiveFlag,
@PurchasingWebServiceURL, @ModifiedDate)
```

[T-SQL]

```
DELETE FROM Purchasing.[Vendor] WHERE [VendorID] = @VendorID
```

- Click **Next** and **Finish**.
- Finally, in the RadGrid properties view, go to the Data Editing section and turn the following three properties to true. This will allow the grid to automatically use the provided SQL statements to perform these actions.



Now when you run the application you will be able to perform all of the edit, add and delete functions and you never had to leave the designer.

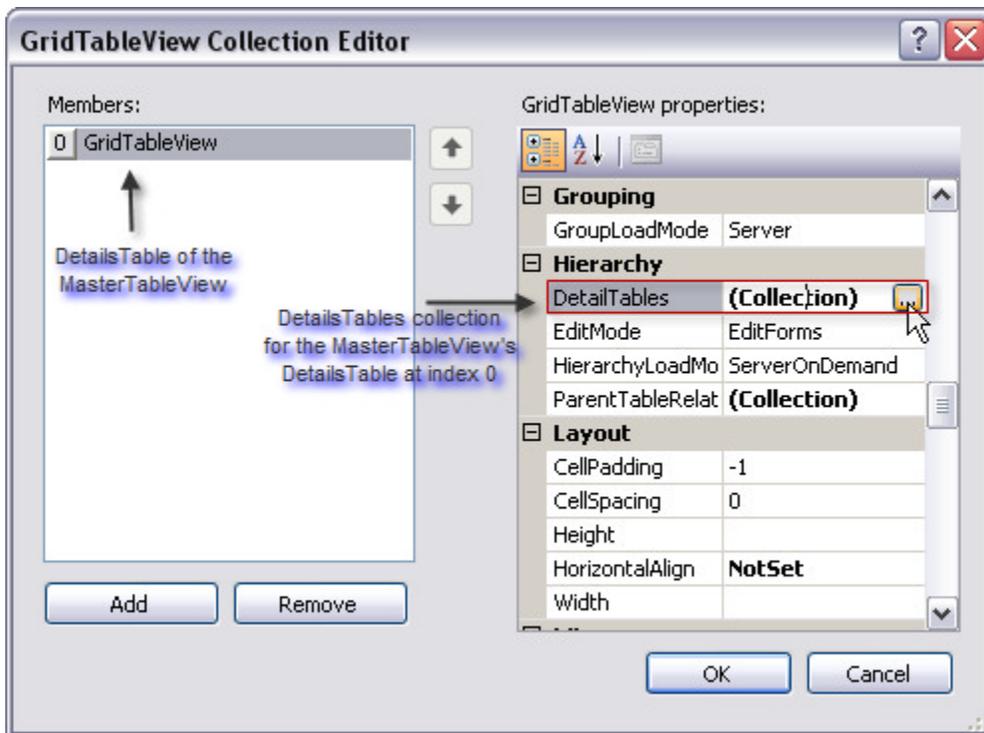
RadGrid, MasterTableView and DetailTables

The **RadGrid** control has a **MasterTableView** property that represents the top table in the grid. The instances of **RadGrid** and **MasterTableView** are almost identical, although they are of different types (**RadGrid** and **GridView**, respectively).

The main difference between **RadGrid** and **MasterTableView** is that the properties of **RadGrid** specify the defaults for every **GridView** in the grid (the **MasterTableView** and any **DetailTables**). The properties of **MasterTableView** apply only to the top-level table in the grid. In other words, property settings on **MasterTableView** are not inherited by any **DetailTables** nested inside it. The properties of **MasterTableView**, as with the properties of any **DetailTable** in the grid, act as overrides to the defaults set on the **RadGrid** object.

For example, if you set a blue border for the **RadGrid**, the **MasterTableView** and any **DetailTables** will also have blue border (assuming they do not override the property setting), while if you set blue border for the **MasterTableView**, this border will appear *only* on the top-level table, and not on any detail tables.

The **DetailTables** property of the **MasterTableView** is a collection of **GridViewTables** and therefore each **DetailTable** can have its own collection of **DetailTables**. This is how a grid hierarchy can be created. Here is a look at the **DetailTables** editor:



19.5 Server Side Code

Getting familiar with the Server Side API

Writing code for the RadGrid using the server-side API can be very useful for times when you need to manipulate the RadGrid right before or after a major event like when data is bound, altered, an item is created, a report is generated or a row drop event is invoked. We will first go over the most often used properties used in server-side code and the most useful events to use them in. We will also cover a couple examples of using server-side code to accomplish some common tasks.

Often used properties

The following properties at the RadGrid level are typically set at design-time and usually not changed in server-side code:

- **RadGrid.ClientSettings:** Provides access to the client-side events and properties.
- **RadGrid.*Style:** Allows change to the appearance of the various interfaces of the RadGrid that applies to the entire hierarchy of GridTableViews unless overridden by a non-default setting.

Style	
ActiveItemStyle	
AlternatingItemStyle	
CommandItemStyle	
EditItemStyle	
FilterItemStyle	
FooterStyle	
GroupHeaderItemStyle	
HeaderStyle	
ItemStyle	
PagerStyle	
SelectedItemStyle	

- **RadGrid.MasterTableView:** This is the base level GridTableView of the RadGrid, which is the only table you will work with if there are no details or hierarchal tables.

The next properties are important to every every GridTableView including the MasterTableView:

- **GridTableView.DetailTables:** The collection of GridTableViews that make up the sub-tables of a GridTableView.
- **GridTableView.*Style properties:** The individual settings of the GridTableView that will override any settings in the RadGrid.*Style settings.
- **GridTableView.GroupByExpressions:** Add or remove objects to this collection to set grouping programmatically.
- **GridTableView.SortExpressions:** Add or remove objects to this collection to set sorting programmatically.
- **GridTableView.Columns:** Access the Columns collection created at design-time.
- **GridTableView.AutoGeneratedColumns:** Collection of columns created at run-time.
- **GridTableView.RenderColumns:** Collection of all columns created at design-time and runtime including interface columns such as expand/collapse and group splitters.

Often used events

There are many server-side events associated with the RadGrid and the GridTableView. Here is a brief list of some of the most commonly used events and the property groups they appear in.

Action

Events in the "Action" group of properties happen in response to interface interaction:

- ***Command: ItemCommand, CancelCommand, DeleteCommand, EditCommand, InsertCommand, UpdateCommand and SortCommand.** ItemCommand is a catch-all for any clicked RadGrid button such as Edit, Delete, or Update command events. The other commands are operation specific, e.g. DeleteCommand fires when a Delete command bubbles up.

All these events except for SortCommand pass a **GridCommandEventArgs** object.

GridCommandEventArgs contains a **Canceled** property that you can set to kill the event, **CommandName** to help determine the kind of operation to expect, **CommandArgument**, **Item**, and **CommandSource**. CommandSource is a reference to the control that triggered the command. For example, the control might be a button that triggered an Expand or Collapse command. Item is a **GridItem** type that may be cast to an type appropriate for the event, i.e. **GridEditableItem** during the Update command. Here's a brief extract from a Update command event handler that shows the Item event argument in play:

[VB] Handling the UpdateCommand Event

```
Protected Sub gridQuestions_UpdateCommand(ByVal source As Object, ByVal e As
GridCommandEventArgs)
' Get the item that appears when grid is in Update Mode.
' Use the item object ExtractValues()method
' to fill a HashTable with values for the current row.
Dim item As GridEditableItem = TryCast(e.Item, GridEditableItem)
Dim ht As New Hashtable()
item.ExtractValues(ht)
'...

e.Item.OwnerTableView.Rebind()
End Sub
```

[C#] Handling the UpdateCommand Event

```
protected void gridQuestions_UpdateCommand(object source, GridCommandEventArgs e)
{
// Get the item that appears when grid is in Update Mode.
// Use the item object ExtractValues()method
// to fill a HashTable with values for the current row.
GridEditableItem item = e.Item as GridEditableItem;
Hashtable ht = new Hashtable();
item.ExtractValues(ht);
//...

e.Item.OwnerTableView.Rebind();
}
```

The SortCommand event passes a **GridSortCommandEventArgs** (descends from **GridCommandEventArgs**) and adds **NewSortOrder**, **OldSortOrder** and **SortExpression** properties. To determine the new/previous sort order on sort command, check the values for the **e.NewSortOrder** and **e.OldSortOrder** arguments.

- **PageIndexChanged:** Fired when a page selection arrow is clicked.
- **GroupsChanging:** Fired when a grouping is created or removed. You can use this event to hide or unhide columns that are being used for grouping.
- **RowDrop:** Fires when a grid row is dragged and dropped. To make this event fire, set the **ClientSettings.AllowRowsDragDrop** property and the **ClientSettings.Selecting.AllowRowSelect** property to true. The example below shows a row with product information being dropped at another location in the same grid and an alert reporting the ProductID of the dragged row. Also be aware that the arguments passed in contain a list of the dragged items, the identity of the grid being dragged onto, the DropPosition (Above, Below), and HTMLElement (the ID of an HTML element that the row was dropped on).

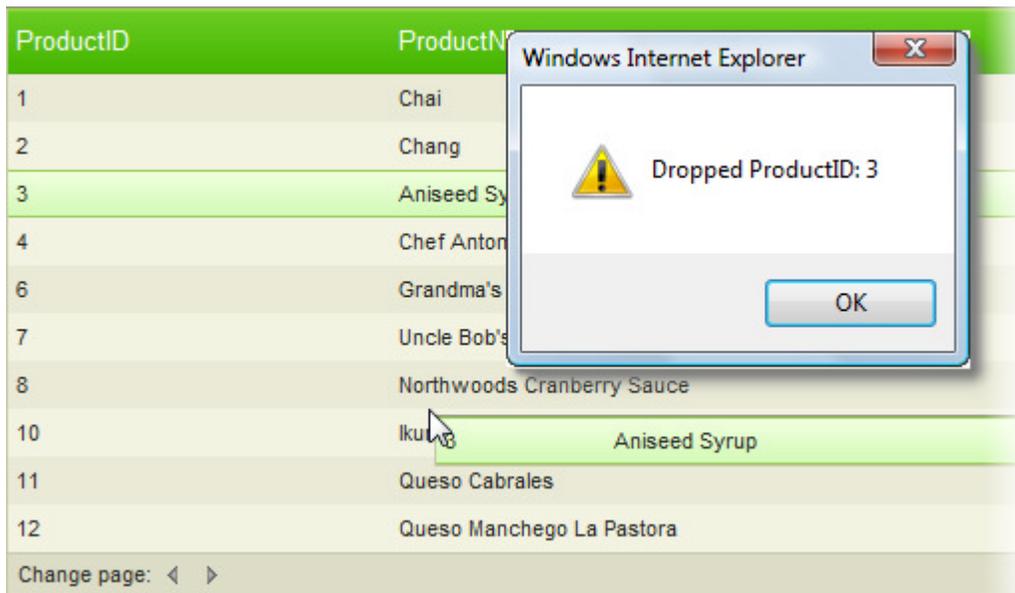
[VB] Handling the RowDrop Event

```
Protected Sub RadGrid1_RowDrop(ByVal sender As Object, ByVal e As
Telerik.Web.UI.GridDragDropEventArgs)
Dim item As GridDataItem = TryCast(e.DraggedItems(0), GridDataItem)
RadAjaxManager1.Alert("Dropped ProductID: " + item.GetDataKeyValue("ProductID"))
End Sub
```

[C#] Handling the RowDrop Event

```
protected void RadGrid1_RowDrop(object sender,
Telerik.Web.UI.GridDragDropEventArgs e)
{
GridDataItem item = e.DraggedItems[0] as GridDataItem;
```

```
RadAjaxManager1.Alert("Dropped ProductID: " + item.GetDataKeyValue("ProductID"));  
}
```



For info on the **NeedDataSource** and **DetailTableDataBind** events, see the Data section below.

Behavior

Events in the "Behavior" group of properties can be used to intercept and alter elements of the RadGrid before or after they are created.

- **ItemCreated:** Fired on the server when an item in the RadGrid control is created.
- **ItemDataBound:** Fired after an item is databound to the RadGrid control.
- **ColumnCreating:** Fired before a custom column is being created.
- **ColumnCreated:** Fired after an auto-generated column is created.

Data

Events in the "Data" group of properties respond to databinding and CRUD (Create, Remove, Update, Delete) operations.

- **DataBinding:** Fired right before the server control binds to a data source.
- **DataBound:** Fired when the server control is bound to a data source.

These next two appear in the Action group, but we're including them here along with the other data related events:

- **NeedDataSource:** Fired when RadGrid needs a data source for rebinding. This event can be useful when the data source may change at runtime or you for some reason don't want to set the data source declaratively. In this event you would set the **DataSource** property, not the **DataSourceID**.
- **DetailTableDataBind:** Fired when a DetailsTable binds to a data source.

Data Editing

These events **ItemInserted**, **ItemUpdated** and **ItemDeleted**, fire right after automatic inserts, updates and deletes. The arguments passed into these events all descend from **GridDataChangeEventArgs** and are very similar to one another. The arguments include an integer number of the **AffectedRows**, an **Exception** object, a

GridEditableItem and a **ExceptionHandled** event that can be set to true if you want to prevent the exception from propagating. The example below fires after an item is inserted. If there is an exception, the exception is considered handled, the record stays in insert mode so the user can correct it and a message displays that the product couldn't be inserted.

[VB] Handling the ItemInserted Event

```
Protected Sub RadGrid1_ItemInserted(ByVal source As Object, ByVal e As
GridInsertedEventArgs)
    If Not e.Exception Is Nothing Then
        e.ExceptionHandled = True
        e.KeepInInsertMode = True
        DisplayMessage("Product cannot be inserted. Reason: " + e.Exception.Message)
    Else
        DisplayMessage("Product inserted")
    End If
End Sub
```

[C#] Handling the ItemInserted Event

```
protected void RadGrid1_ItemInserted(object source, GridInsertedEventArgs e)
{
    if (e.Exception != null)
    {
        e.ExceptionHandled = true;
        e.KeepInInsertMode = true;
        DisplayMessage("Product cannot be inserted. Reason: " + e.Exception.Message);
    }
    else
    {
        DisplayMessage("Product inserted");
    }
}
```

Misc / Exporting

The events in this group handle exporting behavior.

- **GridExporting**: Fires when the grid exports to any output type. The event arguments include **OutputType**, an enumeration that indicates the format, e.g. Excel, MSWord, etc., and **ExportOutput**, a string that will be output to the Response.
- Excel specific events **ExcelExportingCellFormatting**, **ExcelMLExportRowCreated**, **ExcelMLExportStyesCreated**.

What are good things to do in server side code?

In cases where new data is retrieved, data is being bound or dynamic data operations must be performed, server-side code is ideal.

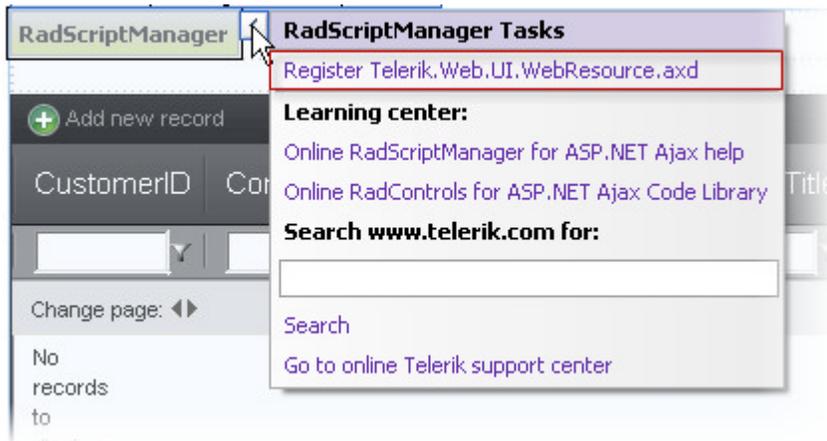
In these examples, we will show some commonly useful code using the server-side API of the RadGrid. The first will be a simple grid that binds to the datasource but does not use the **AllowAutomaticUpdates**, **AllowAutomaticDeletes** or **AllowAutomaticInserts** properties. This will require that we use the **UpdateCommand**, **InsertCommand**, and **DeleteCommand** event handlers. The second example will show how to use the **ColumnCreating** event to convert some column data to representative images before rendering.

Server-side CRUD Example

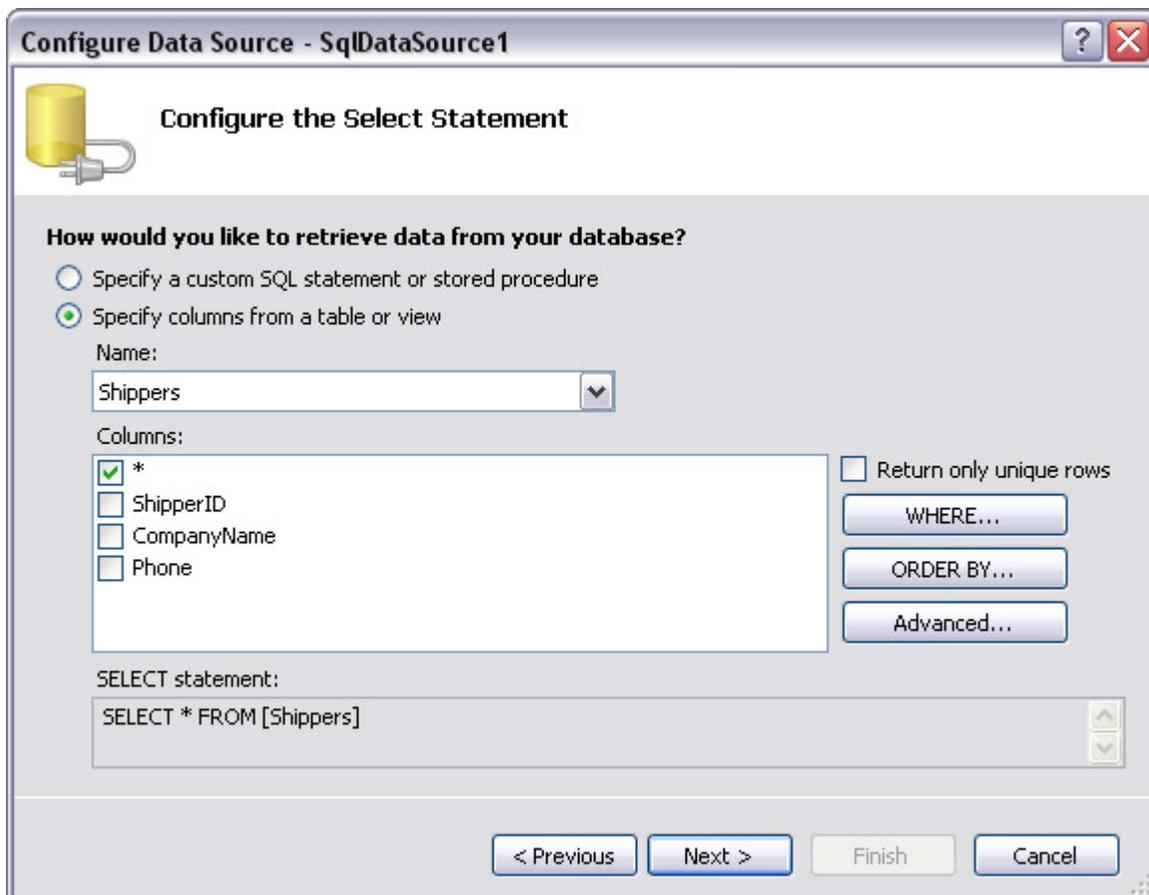
1. Create a new Web Application for ASP.NET.
2. Drop a RadGrid onto the Default.aspx page.

UI for ASP.NET AJAX

3. In the Smart Tag, click on **Add RadScriptManager**.
4. In the RadScriptManager's Smart Tag, click **Register Telerik.Web.UI.WebResource.axd**.

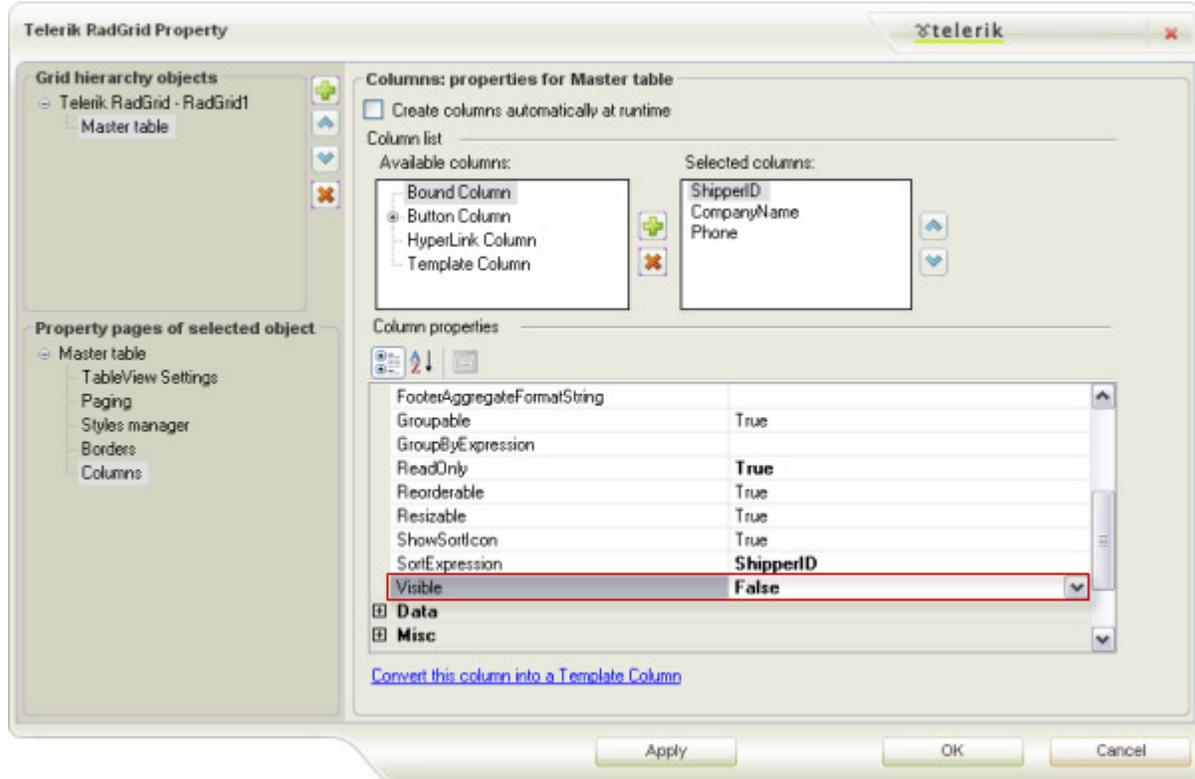


5. Configure a new datasource for the RadGrid.
6. Connect to the NorthWind database and save the connection string.
7. Choose the "Shippers" table and select all of the rows (*) for the SELECT query.



8. Click **Next | Test Query | Finish**.
9. Open the Smart Tag and select the **Auto-generate edit column at runtime** and **Auto-generate delete column at runtime** options.

10. In the Property Builder, navigate to **Master table | Columns | Selected Columns | ShipperID | Behavior | Visible**, set it to "False" and click **OK**.



11. In the Properties View, navigate to **Layout | MasterTableView | CommandItemDisplay** and set it to "Top".
 12. In the Properties View for the RadGrid go to the events tab and double click on the **DeleteCommand** event.
 13. Add the following code to the event handler:

[VB] DeleteCommand event handler

```
'Get the GridDataItem of the RadGrid
Dim item As GridDataItem = DirectCast(e.Item, GridDataItem)
'Get the primary key value using the DataKeyValue.
Dim ShipperID As String = item.OwnerTableView.DataKeyValues(item.ItemIndex)
("ShipperID").ToString()
Try
    'Delete command execute
    SqlDataSource1.DeleteCommand = "DELETE from Shippers where ShipperID='" + ShipperID +
    """"
    SqlDataSource1.Delete()
Catch ex As Exception
    RadGrid1.Controls.Add(New LiteralControl("Unable to delete Shipper. Reason: " +
ex.Message))
    e.Canceled = True
End Try
```

[C#] DeleteCommand event handler

```
//Get the GridDataItem of the RadGrid
GridDataItem item = (GridDataItem)e.Item;
//Get the primary key value using the DataKeyValue.
```

```

string ShipperID = item.OwnerTableView.DataKeyValues[item.ItemIndex]["ShipperID"].ToString();
try
{
    //Delete command execute
    SqlDataSource1.DeleteCommand = "DELETE from Shippers where ShipperID='" + ShipperID +
    "'";
    SqlDataSource1.Delete();
}
catch (Exception ex)
{
    RadGrid1.Controls.Add(new LiteralControl("Unable to delete Shipper. Reason: " +
    ex.Message));
    e.Canceled = true;
}

```

14. Create an **UpdateCommand** event handler the same way the **DeleteCommand** handler was created and insert the following code:

[VB] UpdateCommand event handler

```

'Get the GridEditableItem of the RadGrid
Dim editedItem As GridEditableItem = TryCast(e.Item, GridEditableItem)
'Get the primary key value using the DataKeyValue.
Dim ShipperID As String = editedItem.OwnerTableView.DataKeyValues(editedItem.ItemIndex)
("ShipperID").ToString()
'Access the textbox from the edit form template and store the values in string variables
Dim CompanyName As String = TryCast(editedItem("CompanyName").Controls(0), TextBox).Text
Dim Phone As String = TryCast(editedItem("Phone").Controls(0), TextBox).Text
Try
    'Update Query execution
    SqlDataSource1.UpdateCommand = "UPDATE Shippers set CompanyName='" + CompanyName +
    "',Phone='" + Phone + "'"
    SqlDataSource1.Update()
Catch ex As Exception
    RadGrid1.Controls.Add(New LiteralControl("Unable to update Shippers. Reason: " +
    ex.Message))
    e.Canceled = True
End Try

```

[C#] UpdateCommand event handler

```

//Get the GridEditableItem of the RadGrid
GridEditableItem editedItem = e.Item as GridEditableItem;
//Get the primary key value using the DataKeyValue.
string ShipperID = editedItem.OwnerTableView.DataKeyValues[editedItem.ItemIndex]
["ShipperID"].ToString();
//Access the textbox from the edit form template and store the values in string variable:
string CompanyName = (editedItem["CompanyName"].Controls[0] as TextBox).Text;
string Phone = (editedItem["Phone"].Controls[0] as TextBox).Text;
try
{
    //Update Query execution
    SqlDataSource1.UpdateCommand = "UPDATE Shippers set CompanyName='" + CompanyName +
    "',Phone='" + Phone + "'";
}

```

```

        SqlDataSource1.Update();
    }
    catch (Exception ex)
    {
        RadGrid1.Controls.Add(new LiteralControl("Unable to update Shippers. Reason: " +
ex.Message));
        e.Canceled = true;
    }
}

```

15. Create an **InsertCommand** event handler the same way the **DeleteCommand** and **UpdateCommand** handlers were created and insert the following code:

[VB] InsertCommand event handler

```

'Get the GridEditableItem of the RadGrid
Dim
newItem As GridEditableItem = TryCast(e.Item, GridEditableItem)
'Access the textbox from the edit form template and store the values in string variables
Dim CompanyName As String = TryCast(newItem("CompanyName").Controls(0), TextBox).Text
Dim Phone As String = TryCast(newItem("Phone").Controls(0), TextBox).Text
Try
    'Insert Query execution
    SqlDataSource1.InsertCommand = "INSERT INTO Shippers (CompanyName, Phone) Values ('"
CompanyName + "','" + Phone + "'"
    SqlDataSource1.Insert()
Catch ex As Exception
    RadGrid1.Controls.Add(New LiteralControl("Unable to insert Shipper. Reason: " +
ex.Message))
    e.Canceled = True
End Try

```

[C#] InsertCommand event handler

```

//Get the GridEditableItem of the RadGrid
GridEditableItem newItem = e.Item as GridEditableItem;
//Access the textbox from the edit form template and store the values in string variable:
string CompanyName = (newItem["CompanyName"].Controls[0] as TextBox).Text;
string Phone = (newItem["Phone"].Controls[0] as TextBox).Text;
try
{
    //Update Query execution
    SqlDataSource1.InsertCommand = "INSERT INTO Shippers (CompanyName, Phone) Values ('" -
CompanyName + "','" + Phone + "'"");
    SqlDataSource1.Insert();
}
catch (Exception ex)
{
    RadGrid1.Controls.Add(new LiteralControl("Unable to insert Shipper. Reason: " +
ex.Message));
    e.Canceled = true;
}

```



Gotcha! You will need to add the `Telerik.Web.UI` assembly to the code-behind `using` or `Imports` clause.

UI for ASP.NET AJAX

Now you can run the web application and try out the CRUD functionality. Note that when you try to delete you will see an error message at the foot of the RadGrid. This is due to a dependency constraint of a master/detail relationship in the database schema. To fix this you can either disallow deletions by not showing the Delete link or handle the delete command event and disable the record as suggested in the error message.

+ Add new record		Refresh	
CompanyName	Phone		
Speedy Express	(503) 555-9831	Delete	Edit
United Package	(503) 555-3199	Delete	Edit
Federal Shipping	(503) 555-9931	Delete	Edit

Unable to delete Shipper. Reason: The DELETE statement conflicted with the REFERENCE constraint "FK_Orders_Shippers". The conflict occurred in database "C:\DOCUMENTS AND SETTINGS\AARONMY DOCUMENTS\VISUAL STUDIO 2008\PROJECTS\WEBAPPLICATION5\WEBAPPLICATION5\APP_DATA\NORTHWND.MDF", table "dbo.Orders", column "ShipVia". The statement has been terminated.

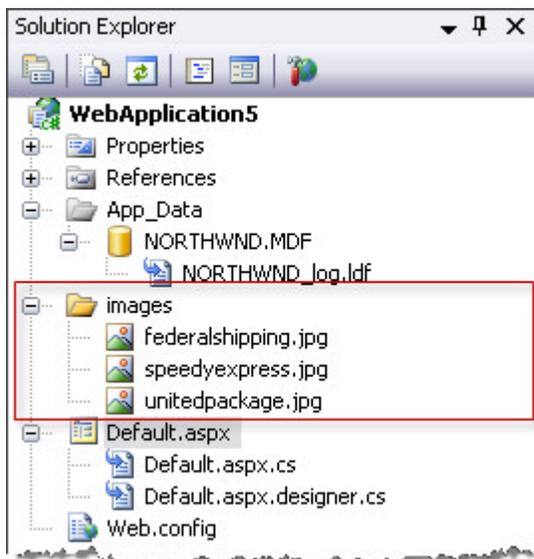
Changing a data row to images.

Images always make things look better. Where you either have a reference to an image directly, or there is a limited number of possible values that might be better displayed as images, you might want to replace the data with some nice pictures to spruce things up in your web application. This example will also demonstrate how to extract a value out of a field in a server-side event and re-assign the column text to an image path. The images can be found in the completed example project listed below:



You can find the complete source for this project at:
\\VS Projects\Grid\RadGridServerSideAPI

1. Create an \Images folder in your project. Drag the three images to represent the three shipping companies in the database into the \Images folder.



2. In the Properties View, go to the events tab for the RadGrid and double-click on **ItemDataBound**.
3. Now simply add this code to the event handler:
[VB] Converting a data field to an image

```

If TypeOf e.Item Is GridDataItem Then
    Dim item As GridDataItem = DirectCast(e.Item, GridDataItem)
    If item("CompanyName").Text = "Federal Shipping" Then
        item("CompanyName").Text = "<IMG src=""images/federalshipping.jpg""/>"
    ElseIf item("CompanyName").Text = "Speedy Express" Then

        item("CompanyName").Text = "<IMG src=""images/speedyexpress.jpg""/>"
    ElseIf item("CompanyName").Text = "United Package" Then

        item("CompanyName").Text = "<IMG src=""images/unitedpackage.jpg""/>"
    End If
End If

```

[C#] Converting a data field to an image

```

if (e.Item is GridDataItem)
{
    GridDataItem item = (GridDataItem)e.Item;
    if (item["CompanyName"].Text == "Federal Shipping")
    {
        item["CompanyName"].Text = "<IMG src=\"images/federalshipping.jpg\"/>";
    }
    else if (item["CompanyName"].Text == "Speedy Express")
    {
        item["CompanyName"].Text = "<IMG src=\"images/speedyexpress.jpg\"/>";
    }
    else if (item["CompanyName"].Text == "United Package")
    {
        item["CompanyName"].Text = "<IMG src=\"images/unitedpackage.jpg\"/>";
    }
}

```

4. Press Ctl-F5 to run the application. This code above extracts the field value for every row in the CompanyName column and uses that to replace that value with an HTML image tag. Now when we run the application it should look something like this:

CompanyName	Phone		
	(503) 555-9831	Delete	Edit
	(503) 555-3199	Delete	Edit
	(503) 555-9931	Delete	Edit

19.6 Client Side Code

When your web application is launched and the RadGrid has been populated with data, you want to avoid going back to the server to retrieve data you already have. This is where client-side code can be useful. Manipulating the RadGrid and elements of the data based on user interaction is great for performance and an immediate feedback feel of the interface. We will start this chapter by going over the properties, events and methods of the client-side API. We will also have some example projects that showcase some great usage of client-side coding.

Properties

You will need to access either column or row client objects in the GridTableView to manipulate data, appearance and behavior. The following are properties accessible in the client-side API that you would most likely use to access and alter elements of the RadGrid and that grid's GridTableViews.

To get the main RadGrid object, pass the grid's ClientID to the `$find()` method:

```
var grid = $find("<%= RadGrid1.ClientID %>");
```

To get the grid's root GridTableView object, use the `get_masterTableView()` method. Likewise you can get at the grid header, footer using `get_masterTableViewHeader()` and `get_masterTableViewFooter()`.

[JavaScript] Getting TableView, Header and Footer Client Objects

```
var tableView = $find("<%= RadGrid1.ClientID %>").get_masterTableView();
var header = $find("<%= RadGrid1.ClientID %>").get_masterTableViewHeader();
var footer = $find("<%= RadGrid1.ClientID %>").get_masterTableViewFooter();
```

Once you have the the MasterTableView object you can call its client methods. Here's an example that calls the `exportToExcel()` method:

[JavaScript] Exporting to Excel

```
function exportGrid() {
    var masterTable = $find("<%=RadGrid1.ClientID %>").get_masterTableView();
```

```

    masterTable.exportToExcel();
}

```

If you have your grid configured to show detail tables, you can get the entire collection:

[JavaScript] Getting Detail Tables

```

function pageLoad(sender, args) {
    var detailTables = $find("<%= RadGrid1.ClientID %>").get_detailTables();
    for (var i = 0; i < detailTables.length; i++) {
        alert(detailTables[i].get_name());
    }
}

```

Once you have a `GridTableView` object returned from `get_masterTableView()` or `get_detailsTables()`, you can get at these properties:

- `get_owner()`: This property is of type `RadGrid` and is the parent of the current object.
- `get_element()`: Returns an HTML table which represents the respective table for the `GridTableView` object rendered on the client.
- `get_dataItems()`: A collection which holds all data items (objects of type `GridDataItem`) in the respective table.
- `get_columns()`: A collection which holds objects of type `GridColumn` (the client-side objects) in the respective table.
- `get_name()`: String which represents the `Name` property (set on the server) for the corresponding `GridTableView` client object. Can be used to identify table in grid hierarchy client-side.
- `get_selectedItems()`: A collection which holds all selected items (objects of type `GridDataItem`) in the respective table. This collection will also include the selected items from child tables if they exist.
- `get_isItemInserted()`: Boolean value returns true if the table is in insert mode.
- `get_pageSize()`, `set_pageSize()`: The page size for the respective `GridTableView` object.
- `get_currentPageIndex()`, `set_currentPageIndex`: The current page index for the respective `GridTableView` object.
- `get_pageCount()`: Returns the page count for the respective `GridTableView` object.
- `get_clientDataKeyNames()`: One-dimensional array which holds the key fields set through the `ClientDataKeyNames` property of `GridTableView` on the server. To extract the key values you can use the `EventArgs.GetDataKeyValue` inside any row-related client event handler of `RadGrid`.
- `get_parentView()`: If called from a nested table view returns the parent table view (of type `GridTableView`) in the grid hierarchy; returns null if called from the `MasterTableView`.
- `get_parentRow()`: If called from a nested table view returns the parent row (html table row: `<tr>`) for the current nested hierarchical table view; returns null if called from the `MasterTableView`.

You can further refer to the elements of the `RadGridTable` using the functions below:

`<GridTableViewInstance>.get_columns()[n].get_element()`: the real HTML table column for the n-th column (`<th>` for header cell)

`<GridTableViewInstance>.get_dataItems()[n].get_element()`: the real HTML table row for the n-th row (`<tr>` element)

Here's an example that iterates the master table view selected items and shows the inner text of each.

[JavaScript] Iterating Selected Items

```

function showSelectedRows() {
    var dataItems = $find("<%=RadGrid1.ClientID%>").get_masterTableView().get_selectedItems();
    for (var i = 0; i < dataItems.length; i++) {

```

```
    alert(i + ": " + dataItems[i].get_element().innerText);
  }
}
```

RadGrid Events

You can follow the life-cycle of the grid using the **OnGridCreating**, **OnGridCreated** and **OnGridDestroying** client events.

[JavaScript] Handling the OnGridCreating Event

```
function gridCreating(sender, args) {
  alert("creating: " + sender.ClientID);
}
```

GridTableView Events

The following shows some of the key events for the GridTableView (either the MasterTableView or any of the detail tables). For a more complete list, consult the online help.

Creation

Like the RadGrid object, GridTableView has events that follow the life cycle of these objects **OnMasterTableViewCreating**, **OnMasterTableViewCreated**, **OnColumnCreating**, **OnColumnCreated**, **OnColumnDestroying**, **OnRowCreating**, **OnRowCreated** and **OnRowDestroying**.

Columns

You can find out when columns are resized, reordered, hidden, clicked, when the mouse hovers over a column or when a context menu appears for a column heading:

- **OnColumnResizing**, **OnColumnResized**: These events fire before and after a column is resized.
- **OnColumnSwapping**, **OnColumnSwapped**: These events fire before two columns are swapped.
- **OnColumnMovingToLeft**, **OnColumnMovedToLeft**, **OnColumnMovingToRight**, **OnColumnMovedToRight**: These events fire before and after columns are moved left or right.
- **OnColumnClick**, **OnColumnDbClick**: These events fire before and after a column is clicked.
- **OnColumnMouseOver**, **OnColumnMouseOut**: These events fire when the mouse first hovers over a column and then moves away from the column.
- **OnColumnShowing**, **OnColumnShown**: These events fire before and after a column is shown.
- **OnColumnContextMenu**: This event is fired when the context menu for a column is called.
- **OnColumnHiding**, **OnColumnHidden**, **OnColumnShowing**, **OnColumnShown**: These events fire before and after a column changes visibility.

Rows

A parallel set of events exist for grid columns:

- **OnRowResizing**, **OnRowResized**: These events fire before and after a row is resized.
- **OnRowSelecting**, **OnRowSelected**, **OnRowDeselecting**, **OnRowDeselected**: These events occur before and after the row selection is toggled..
- **OnRowClick**, **OnRowDbClick**: These events fire when a row is clicked/double-clicked.
- **OnRowMouseOver**, **OnRowMouseOut**: These events fire when the mouse first hovers over a row and again when the mouse leaves the row.
- **OnRowContextMenu**: This event is fired when the context menu for a row is called.
- **OnRowShowing**, **OnRowShown**, **OnRowHiding**, **OnRowHidden**: These events fire before and after a row's

visibility is toggled.

- **OnRowDeleting, OnRowDeleted:** These events fire before and after a row is deleted (client-side delete).

Rows have an additional set of events to handle drag and drop operations on the client side:

- **OnRowDragStarted:** This event is fired when a row is about to be dragged.
- **OnRowDropping:** This event is fired before a row is dropped.
- **OnRowDropped:** This event is fired after a row is dropped.

You can also track the active row:

- **OnActiveRowChanging:** This event is fired before the active row change.
- **OnActiveRowChanged:** This event is fired after the active row is changed.

And finally, you can be notified when a row is about to be bound on the client using **OnRowDataBound**.

Group and Hierarchy Expansion

GridView has a series of events for the group and hierarchy expanding and collapsing: **OnHierarchyExpanding, OnHierarchyExpanded, OnHierarchyCollapsing, OnHierarchyCollapsed, OnGroupExpanding, OnGroupExpanded, OnGroupCollapsing** and **OnGroupCollapsed**. As with the other RadControls client API, the "ing" events can be canceled using the `args.set_cancel(true)` method.

Command

OnCommand: This event is fired for each grid command which is about to be triggered (sorting, paging, filtering, editing, etc.) before postback/ajax request .

Methods

After you have retrieved the column or data item that you want to change, these are the methods you will most likely call to affect that change.

Data item methods

Using GridView methods you can toggle selection, visibility and the collapse/expand state of individual items:

- **selectItem(gridItem), deselectItem(gridItem):** The row passed as an argument will become selected/deselected.
- **hideItem(index), showItem(index):** Hide or show the row at the indexed position.
- **expandItem(index), collapseItem(index):** Expand or show the indexed row. If the index corresponds to nested table item, all parent items will be expanded to top. Applicable for `HierarchyLoadMode = Client` only!

[JavaScript] Collapsing an item

```
function CollapseFirstDetailTableFirstItem()
{
    $find("<%= RadGrid1.Items[0].ChildItem.NestedTableViews[0].ClientID %>").collapseItem(0);
}
```

If you have items selected in the grid, you can call methods to clear, edit, update and delete the selected items all at once:

- **clearSelectedItems():** Method which clears the selected items for the respective GridView client object. This method will clear the selected items from the table's child tables (meaningful in hierarchical grid only).
- **editSelectedItems():** Method which switches all selected items in the grid in edit mode.

- **updateSelectedItems():** Method which updates all edited items in the grid. The new data will be taken from the edit form editors.
- **deleteSelectedItems():** Method which deletes all selected items in the grid.

[JavaScript] Deleting a selected item

```
function DeleteSelectedGridItems()
{
    var masterTable = $find("<%= RadGrid1.ClientID %>").get_masterTableView();
    masterTable.deleteSelectedItems();
}
```

You can set the grid's edit mode using GridTableView methods:

- **showInsertItem():** Method which switches the grid in insert mode and displays the insertion form.
- **cancelUpdate(gridItem):** Method which cancels the update for the edited table row passed as an argument or the row corresponding to the index passed as an argument. If you have several items switched in edit mode, you can cancel the update for all of them with subsequent calls to this method.
- **cancelInsert():** Method which cancels the insert action and switches the grid in regular mode.
- **editItem(gridItem):** Method which switches the table row passed as an argument or the row corresponding to the index passed as an argument in edit mode. If you set AllowMultiRowEdit to true, you can switch multiple grid items in edit mode with subsequent calls to this method.
- **editAllItems():** Method which switches all GridDataItems in edit mode.
- **cancelAll():** Cancels the edit mode for all grid items that are switched in edit mode prior to the method call.

[JavaScript] Using cancelAll()

```
function CancelEditMode()
{
    var masterTable = $find("<%= RadGrid1.ClientID %>").get_masterTableView();
    masterTable.cancelAll();
}
```

...and you can perform CRUD operations directly on the client. The data for insertion or update is taken from the form editor's fields.

- **deleteItem(gridItem):** Method which deletes the table row passed as an argument or the row corresponding to the index passed as an argument.
- **updateItem(gridItem):** Method which updates the edited table row passed as an argument or the row corresponding to the index passed as an argument. If you have several items switched in edit mode, you can update all of them with subsequent calls to this method.
- **insertItem():** Method which inserts new table row to the grid.

[JavaScript] Using insertItem()

```
function AddNewItem()
{
    var masterTable = $find("<%= RadGrid1.ClientID %>").get_masterTableView();
    masterTable.insertItem();
}
```

Grid column methods

Handle column sizing, order, visibility and grouping using these GridTableView methods:

- **resizeColumn(columnIndex, columnWidth):** The column at the specified columnIndex will be resized to

the width specified through the `columnWidth` argument.

- **swapColumns(colUniqueName1, colUniqueName2)**: The columns with unique names `colUniqueName1` and `colUniqueName2` will be swapped.
- **reorderColumns(colUniqueName1, colUniqueName2)**: `colUniqueName1` is the "from" unique name of the table column while `colUniqueName2` is the "to" unique name of the column (i.e. the new location).
- **moveColumnToLeft(columnIndex), moveColumnToRight(columnIndex)**: The column at the specified `columnIndex` will be moved to the left or right.
- **hideColumn(columnIndex), showColumn(columnIndex)**: Hide or show the column at the specified `columnIndex` position.
- **groupColumn(colUniqueName), ungroupColumn(colUniqueName)**: Group or un-group by the column with specified `UniqueName`.

What are good things to do with Client API

When working with client-side code, the advantage is that you can use the client's readily available processing power to manipulate the web application as long as no information is needed from the server. This makes your server's job easier and frees up cycles for things like retrieving and transferring data to the world. Changing the behavior, appearance and data in the grid based on user interaction is a great use of client-side code.

Client-side databinding

RadGrid for ASP.NET AJAX supports client-side binding to web services or page methods as demonstrated in [this](http://demos.telerik.com/aspnet-ajax/grid/examples/client/declarativedatabinding/defaultcs.aspx) (<http://demos.telerik.com/aspnet-ajax/grid/examples/client/declarativedatabinding/defaultcs.aspx>) and [this](http://demos.telerik.com/aspnet-ajax/grid/examples/client/databinding/defaultcs.aspx) (<http://demos.telerik.com/aspnet-ajax/grid/examples/client/databinding/defaultcs.aspx>) online demo of the product. In order to assign data source for the grid and refresh its state on the client, utilize the **set_dataSource(dataSource)** (<http://www.telerik.com/help/aspnet-ajax/set-datasource.html>) and **dataBind()** (<http://www.telerik.com/help/aspnet-ajax/databind.html>) methods from its client-side API. Keep in mind that the data source passed as an argument to the `set_dataSource` method should have JSON signature which can be serialized by a web service or a page method.

The following example demonstrates how to find a control on a web form and load data into a grid on the client.



You can find the complete source for this project at:

`\VS Projects\Grid\RadGridClientSideAPI`

 Databinding on the client only works in RadControls for ASP.NET AJAX 2008 Q2 and later.

1. First, create a new web application.
2. You will need an XML file that contains sets of values. In this example we have an XML file that I have put in the `App_Data` folder of the project with values that look like the screenshot below. You can get this file in the complete demonstration or create your own:

```
1 | <?xml version="1.0" encoding="utf-8" ?>
2 | <Contacts>
3 |   <Contact>
4 |     <ID>1</ID>
5 |     <Name>John Doe</Name>
6 |     <Age>23</Age>
7 |     <Sex>Male</Sex>
8 |     <Email>johndoe@gmail.com</Email>
9 |     <Phone>703-865-3356</Phone>
10 |   </Contact>
11 |   <Contact>
12 |     <ID>2</ID>
13 |     <Name>Cindy Smith</Name>
14 |     <Age>32</Age>
15 |     <Sex>Female</Sex>
16 |     <Email>cindy_smith@yahoo.com</Email>
17 |     <Phone>832-934-2219</Phone>
18 |   </Contact>
19 | </Contacts>
```

3. Add the Contact class definition listed below. You can place this in a separate Contact.cs file or to the end of the default web page code-behind.

[VB] The Contact class used to define the data structure

```
Public Class Contact
    Private _ID As Integer
    Private _Name As String
    Private _Age As Integer
    Private _Sex As String
    Private _Email As String
    Private _Phone As String
    Public Property ID() As Integer
        Get
            Return Me._ID
        End Get
        Set(ByVal value As Integer)
            Me._ID = value
        End Set
    End Property
    Public Property Name() As String
        Get
            Return Me._Name
        End Get
        Set(ByVal value As String)
            Me._Name = value
        End Set
    End Property
    Public Property Age() As Integer
        Get
            Return Me._Age
        End Get
        Set(ByVal value As Integer)
            Me._Age = value
        End Set
    End Property
```

```

Public Property Sex() As String
    Get
        Return Me._Sex
    End Get
    Set(ByVal value As String)
        Me._Sex = value
    End Set
End Property
Public Property Email() As String
    Get
        Return Me._Email
    End Get
    Set(ByVal value As String)
        Me._Email = value
    End Set
End Property
Public Property Phone() As String
    Get
        Return Me._Phone
    End Get
    Set(ByVal value As String)
        Me._Phone = value
    End Set
End Property
End Class

```

[C#] The Contact class used to define the data structure

```

public class Contact
{
    public int ID { get; set; }
    public string Name { get; set; }
    public int Age { get; set; }
    public string Sex { get; set; }
    public string Email { get; set; }
    public string Phone { get; set; }
}

```

- Now create the Webmethod that parses the XML file and returns the data as a list of objects.

[VB] Parsing the XML file and returning a list of Contact objects

```

<WebMethod(> _
    Public Shared Function GetData() As List(Of Contact)
    Dim path As String = HttpContext.Current.Server.MapPath("App_Data\contacts.xml")
    Dim data As DataSet = New DataSet()
    data.ReadXml(path)
    Dim contacts As List(Of Contact) = New List(Of Contact)()
    For Each row As DataRow In data.Tables(0).Rows
        Dim contact = New Contact()
        contact.ID = CType(row.Item("ID"), Integer)
        contact.Name = CType(row.Item("Name"), String)
        contact.Age = CType(row.Item("Age"), Integer)
        contact.Sex = CType(row.Item("Sex"), String)
        contact.Email = CType(row.Item("Email"), String)
        contact.Phone = CType(row.Item("Phone"), String)
    
```

```
        contacts.Add(contact)
    Next
    Return contacts
End Function
```

[C#] Parsing the XML file and returning a list of Contact objects

```
[WebMethod]
public static List<Contact> GetData()
{
    var path = HttpContext.Current.Server.MapPath(@"App_Data\contacts.xml");
    var data = new DataSet();
    data.ReadXml(path);
    var contacts = new List<Contact>();
    foreach (DataRow row in data.Tables[0].Rows)
    {
        contacts.Add(new Contact {
            ID = Convert.ToInt32(row["ID"]),
            Name = Convert.ToString(row["Name"]),
            Age = Convert.ToInt32(row["Age"]),
            Sex = Convert.ToString(row["Sex"]),
            Email = Convert.ToString(row["Email"]),
            Phone = Convert.ToString(row["Phone"])
        });
    }
    return contacts;
}
```

5. Now drop a **RadGrid** onto your web form.
6. In the RadGrid's Properties window, change **EnableViewState** in the Behavior category to "false".
7. Add a **RadScriptManager** to the form using the RadGrid's Smart Tag.
8. Register **Telerik.Web.UI.WebResources.axd** using the RadScriptManager's Smart tag.
9. In the RadScriptManager's Properties View, change the **EnablePageMethods** property to "true".
10. We need to have the columns in this grid already created to match the fields in the Contact object. You can add them manually in the Property Builder with a DataKey name of "ID" or you can add the following code in the markup in your grid:

[ASP.NET] MasterTableView of the RadGrid including columns

```
<MasterTableView DataKeyNames="ID">
  <Columns>
    <telerik:GridBoundColumn DataField="ID"
      HeaderText="ID" />
    <telerik:GridBoundColumn DataField="Name"
      HeaderText="Name" />
    <telerik:GridBoundColumn DataField="Age"
      HeaderText="Age" />
    <telerik:GridBoundColumn DataField="Sex"
      HeaderText="Sex" />
    <telerik:GridBoundColumn DataField="Email"
      HeaderText="Email" />
    <telerik:GridBoundColumn DataField="Phone"
      HeaderText="Phone" />
  </Columns>
```

```
</MasterTableView>
```

- In the markup, add the following javascript. This will retrieve data from a web method when the page loads in the "pageLoad" event. The updateGrid() function, which is executed in response to the GetData() function call, finds the control on the page called "RadGrid1", gets its **MasterTableView**, sets the datasource to the list of Contacts and calls the RadGrid client **dataBind()** method to bind the data to the table.

 You can add static methods to an ASP.NET page and qualify them as Web methods. Then in your JavaScript you call PageMethods.<my page method name>, and the static method declared within your page gets called.

[ASP.NET] Javascript functions for databinding

```
<script type="text/javascript">
function pageLoad(sender, args)
{
    // Load data from web service
    PageMethods.GetData(updateGrid);
}

function updateGrid(result)
{
    var tableView = $find("<%= RadGrid1.ClientID %>").get_masterTableView();
    tableView.set_dataSource(result);
    tableView.dataBind();
}

function RadGrid1_Command(sender, args)
{
    // Handle the RadGrid's Command event here
}

</script>
```

- Add a reference to the OnCommand event in the markup as shown below.

[ASP.NET] OnCommand event binding

```
<ClientSettings>
    <ClientEvents OnCommand="RadGrid1_Command" />
</ClientSettings>
```

 As of this writing, handling the OnCommand event is required to avoid errors. It's expected that in later versions of the product that this limitation will be removed.

Now when you run the application you should see a fully populated grid. This code takes as much of the burden of databinding to the client as possible.

ID	Name	Age	Sex	Email	Phone
1	John Doe	23	Male	johndoe@gmail.com	703-865-3356
2	Cindy Smith	32	Female	cindy_smith@yahoo.com	832-934-2219
3	Frank Johnson	20	Male	frankyboy@suddenlink.net	719-434-2958
4	Jana Winston	30	Female	winston@msn.com	903-432-1134
5	Juan Palacios	45	Male	juanp@juno.com	210-716-3349
6	Susan Franklin	49	Female	susan@mac.com	310-222-6455
7	Alice Smith	18	Female	as1990@yahoo.com	320-245-6345
8	James Wottring	67	Male	james@gmail.com	713-522-2041
9	Renee Gonzales	21	Female	reneegonzalez@live.com	832-344-6699
10	Carl Aharoni	32	Male	carl_a_2001@aol.com	228-991-0021
11	Julia Sampson	17	Female	julez_sampson@gmail.com	979-223-4521
12	John Carrol	45	Male	johnc005@juno.com	713-444-6734
13	Ben Zeller	24	Male	zellerbm08@yahoo.com	709-991-0001
14	Meredith Jones	31	Female	mere_2001@hotmail.com	513-322-0801
15	Eric Gordon	56	Male	eric.gordon@live.com	423-601-2270

Client-side cell selection

Since Q1 2012 RadGrid provides Client-side cell selection feature.

This example demonstrates RadGrid's cell selection functionality which is controlled through the ClientSettings.Selecting.CellSelectionMode property:

- SingleCell: switches on the single cell selection functionality.
- MutliCell: allows for the selection of multiple cells.
- Column: enables single column selection by clicking on a grid column header.
- MultiColumn: turns on multi column selection for RadGrid.

RadGrid's cells can be selected with the mouse, through the keyboard or both. Regardless of the method cell selection is applied through, the following three rules always hold true: if a given cell is currently in a non-selected state and cell selection is inflicted upon it, then the cell is selected; if a given cell is currently in a selected state and cell selection is inflicted upon it, the cell will be deselected; if cell selection is applied to a certain cell or a set of cells and neither the Shift nor the Control keys are being held, then any previously selected cells will be deselected. The same rules go for column selection.

With this example we will illustrate how to access the data by collecting selected cell values and displaying them in a div elements.

1. Using the same project from the last exercise, change the grid's width to 700 pixels.
2. Add two div elements.
3. In the markup, add these javascript functions:

[ASP.NET] Javascript function that will fire for each row created

```
<script type="text/javascript">
```

```

function cellSelected(sender, args) {

    var columnName = args.get_column().get_uniqueName();
    var customer = args.get_gridDataItem().getDataKeyValue("CustomerID")

    var cellInfo = "Cell: " + columnName + " for " + customer + "
<b>selected</b><br/>";

    $get("cellSelectedEvents").innerHTML += cellInfo;
}
function cellDeselected(sender, args) {
    var columnName = args.get_column().get_uniqueName();
    var customer = args.get_gridDataItem().getDataKeyValue("CustomerID")

    var cellInfo = "Cell: " + columnName + " for " + customer + "
<b>deselected</b><br/>";

    $get("cellDeselectedEvents").innerHTML += cellInfo;
}

function selectColumn() {
    var columnName = $get("columnNameSelect").value.replace(" ", "");

    var columns = $find("<%= RadGrid1.ClientID %>").get_masterTableView
().get_columns();

    var col;

    for (var i = 0; i < columns.length; i++) {
        if (columns[i].get_uniqueName() == columnName) {
            col = columns[i];
            break;
        }
    }

    if (col != null) {
        col.set_selected(true);
    }
}

function deselectColumn() {
    var columnName = $get("columnNameDeselect").value.replace(" ", "");

    var columns = $find("<%= RadGrid1.ClientID %>").get_masterTableView
().get_columns();

    var col;

    for (var i = 0; i < columns.length; i++) {
        if (columns[i].get_uniqueName() == columnName) {
            col = columns[i];
            break;
        }
    }
}

```

UI for ASP.NET AJAX

```
        if (col != null) {  
            col.set_selected(false);  
        }  
    }</script>
```

4. Finally we need to bind the `cellSelection` and `cellDeselection` functions to the respective events of the `RadGrid`.

[ASP.NET] Binding the `OnRowCreated` event

```
<ClientEvents OnCellSelected="cellSelected" OnCellDeselected="cellDeselected" />
```

Now when you run the web application and select some cell, the cell will be highlighted with yellow color and its data will be added to the div elements above the grid.

```
Cell: CompanyName for ANTON selected  
Cell: CompanyName for BERGS selected  
Cell: CompanyName for BLONP selected  
Cell: ContactTitle for BERGS selected  
Cell: ContactName for BOLID selected
```

```
Cell: CompanyName for ANTON deselected  
Cell: CompanyName for BERGS deselected  
Cell: CompanyName for BLONP deselected  
Cell: ContactTitle for BERGS deselected
```

Customer ID	Company Name	Contact Name	Contact Title	Address
ALFKI	Alfreds Futterkiste	Maria Anders	Sales Representative	Obere Str. 57
ANATR	Ana Trujillo Emparedados y helados	Ana Trujillo	Owner	Avda. de la C
ANTON	Antonio Moreno Taquería	Antonio Moreno	Owner	Mataderos 23
AROUT	Around the Horn	Thomas Hardy	Sales Representative	120 Hanover !
BERGS	Berglunds snabbköp	Christina Berglund	Order Administrator	Berguvsvägar
BLAUS	Blauer See Delikatessen	Hanna Moos	Sales Representative	Forsterstr. 57
BLONP	Blondesddsl père et fils	Frédérique Citeaux	Marketing Manager	24, place Kléb
BOLID	Bólido Comidas preparadas	Martin Sommer	Owner	C/ Araquil, 67
BONAP	Bon app'	Laurence Lebihan	Owner	12, rue des Bc
BOTTM	Bottom-Dollar Markets	Elizabeth Lincoln	Accounting Manager	23 Tsawassen

This example is a great reference for using the `RadGrid` in client-side code, as retrieving values, setting events are very common tasks.

19.7 Summary

In this chapter you looked at the `RadGrid` control and saw some of the powerful features it provides. You created a simple application that bound the grid to live data and manipulated the auto-generated columns.

You also explored the most fundamental features of the RadGrid such as Sorting, Filtering, Grouping and Paging.

You worked with an example of implementing CRUD maintenance manually in server-side code.

You learned how to access data values and manipulated the appearance of a column in server-side code by replacing cell values with an HTML image tag.

You implemented the powerful new client-side databinding feature of the RadGrid which showed the overwhelming versatility of the RadGrid to bind to any form of data.

Finally, you learned some advanced client-side coding techniques, including accessing data values, manipulating appearance and binding to client-side events to make a responsive and flashy interface.

19.8 Columns

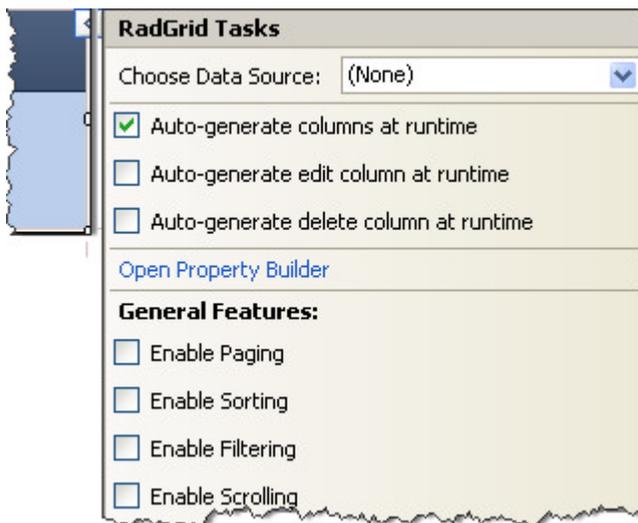
This article will introduce you to the main specifics of creating, using and customizing RadGrid columns. We will start by creating a simple web application with a single RadGrid. We will bind the grid to a datasource and take a quick look at how to manipulate the grid's columns.

Auto-generated Columns

1. Start by creating a web application.
2. With the new default.aspx page in design view, open the Toolbox and locate the RadGrid component, which looks like this:



3. Drag it onto your design surface. Immediately, the RadGrid's Smart Tag will open.



4. Configure the datasource to connect to the AdventureWorks database.

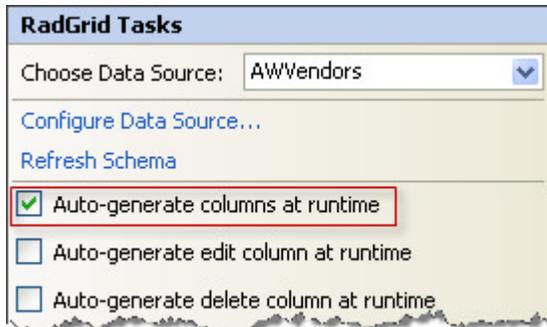
UI for ASP.NET AJAX

5. In the Configure the Select Statement step, select the "Specify a custom SQL statement" option and click **Next**. In the SELECT statement tab, enter the following query:

[SQL] Vendor select statement

```
SELECT * FROM Purchasing.[Vendor]
```

6. Click **Next**, test your query and click **Finished**.
7. You may recall that when the RadGrid's Smart Tag was first presented, the "Auto-generate columns at runtime" option is checked by default.

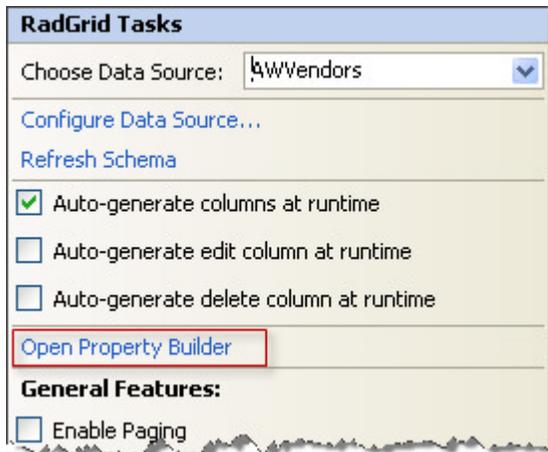


Because we did so, the columns are automatically added to the RadGrid on our design surface. When the RadGrid is displayed once again, the columns will be captioned with the corresponding columns from the database table. Automatic column generation saves a lot of manual work by automatically matching the data types with the appropriate column types, such as checkboxes for boolean fields.

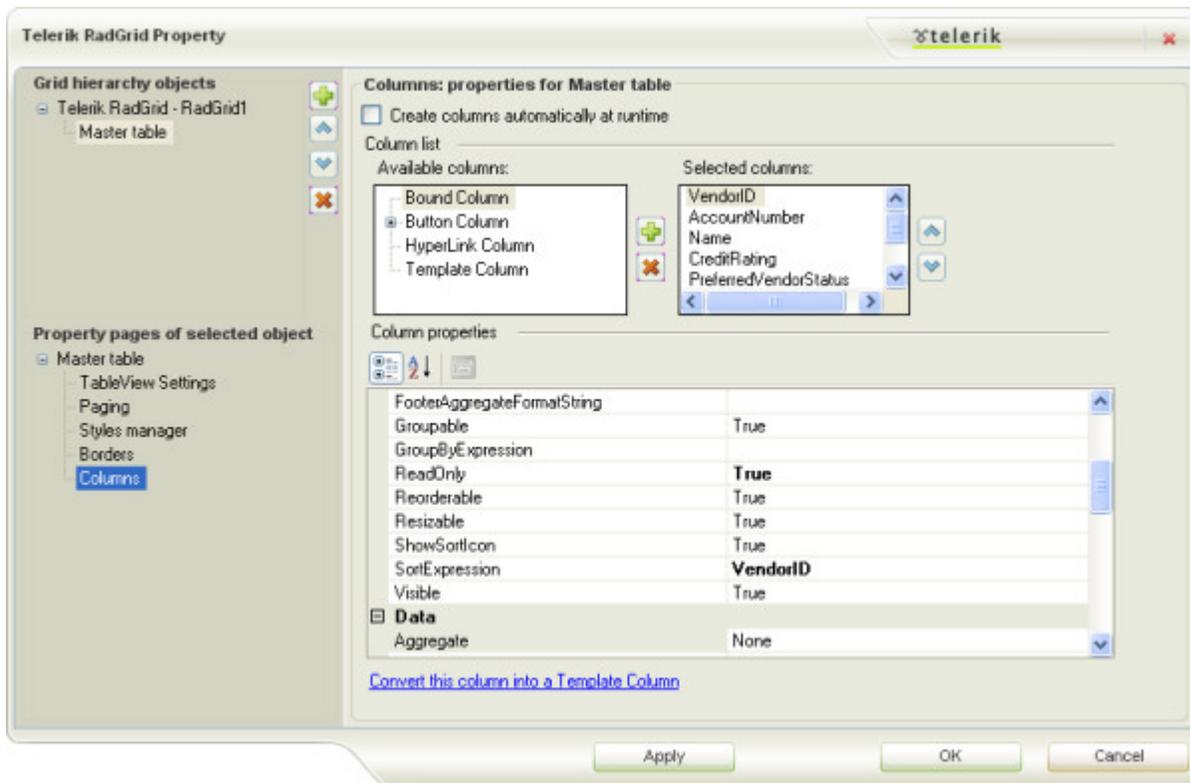
VendorID	AccountNumber	Name	CreditRating	PreferredVendorStatus	ActiveFlag	PurchasingWebServiceURL	ModifiedDate
0	abc	abc	0	<input type="checkbox"/>	<input type="checkbox"/>	abc	6/25/2008 12:00:00 AM
1	abc	abc	1	<input type="checkbox"/>	<input type="checkbox"/>	abc	6/25/2008 12:00:00 AM
2	abc	abc	0	<input type="checkbox"/>	<input type="checkbox"/>	abc	6/25/2008 12:00:00 AM
3	abc	abc	1	<input type="checkbox"/>	<input type="checkbox"/>	abc	6/25/2008 12:00:00 AM
4	abc	abc	0	<input type="checkbox"/>	<input type="checkbox"/>	abc	6/25/2008 12:00:00 AM

Manipulating Columns

1. Open the RadGrid's Smart Tag.
2. Click on the Open Property Builder link.



This displays the Telerik RadGrid Property editor dialog:



Let's consider the columns that are being displayed, and how we might present them differently:

Vendor ID: This is a data key assigned automatically by the database, so it will never be editable. Furthermore, its primary function is to relate the vendor records to data in other tables, so although it has significance within the database, it has no intrinsic meaning to the end user. Set the **Visible** property to "false" to hide this column.

CreditRating: This is a numeric score with a value between 1 and 5, so it's pretty clear that the heading is forcing the column to be much wider than it needs to be. One way to solve this problem is to shorten the column header and provide a tooltip with the full "Credit Rating" description. Shorten the header by changing the **HeaderText** property found in the Appearance category to "CR". Similarly, shorten the headings for the columns "PreferredVendorStatus" to "Stat", and "ActiveFlag" to "Act" Adding the tooltips

takes a little bit of code:

[VB] Adding tooltips to column headers

Imports Telerik.Web.UI

Protected Sub RadGrid1_ItemCreated(ByVal sender As Object, ByVal e As GridItemEventArgs)

 "Check for GridHeaderItem if you wish tooltips only for the header cells

 If TypeOf e.Item Is GridHeaderItem Then

 Dim headerItem As GridHeaderItem = TryCast(e.Item, GridHeaderItem)

 headerItem("CreditRating").ToolTip = "CreditRating"

 headerItem("PreferredVendorStatus").ToolTip = "PreferredVendorStatus"

 headerItem("ActiveFlag").ToolTip = "ActiveFlag"

 End If

End Sub

VB.NET

[C#] Adding tooltips to column headers

using Telerik.Web.UI;

protected void RadGrid1_ItemCreated(object sender, GridItemEventArgs e)

{

 //Check for GridHeaderItem if you wish tooltips only for the header cells

 if (e.Item is GridHeaderItem)

 {

 GridHeaderItem headerItem = e.Item as GridHeaderItem;

 headerItem["CreditRating"].ToolTip = "CreditRating";

 headerItem["PreferredVendorStatus"].ToolTip = "PreferredVendorStatus";

 headerItem["ActiveFlag"].ToolTip = "ActiveFlag";

 }

}

If you run the application at this point, you'll see that the columns resize themselves dynamically to accommodate the actual data. Note the tooltip above the Credit Rating column.

Name	CP	Stat CreditRating	Act	PurchasingWebServiceURL
International	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Electronic Bike Repair & Supplies	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Premier Sport, Inc.	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	
Comfort Road Bicycles	1	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	

You will also notice that the "ModifiedDate" column shows the date and time. Since the date portion is probably all you would care about in this context, let's take a look at how to set the format.

3. Close the web application and open the Property Builder again.
4. Select the **Master Table | Columns | Modified Date** entry to display the properties for the Modified Date column.
5. Locate the **DataFormatString** property in the Behavior section and enter the format string "{0:MM/dd/yyyy}" without the quote marks.

 The format string follows the Microsoft formatting conventions. The "0" at the beginning of the string indicates that the argument passed in to the formatter should be used as input.

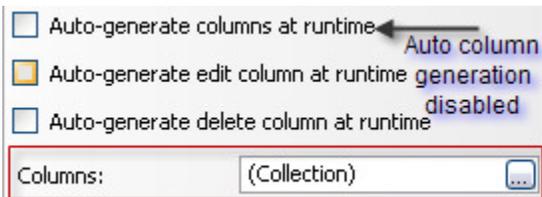
When you run the application, it will look something like this:

ModifiedDate
02/25/2002
02/17/2002
03/05/2002

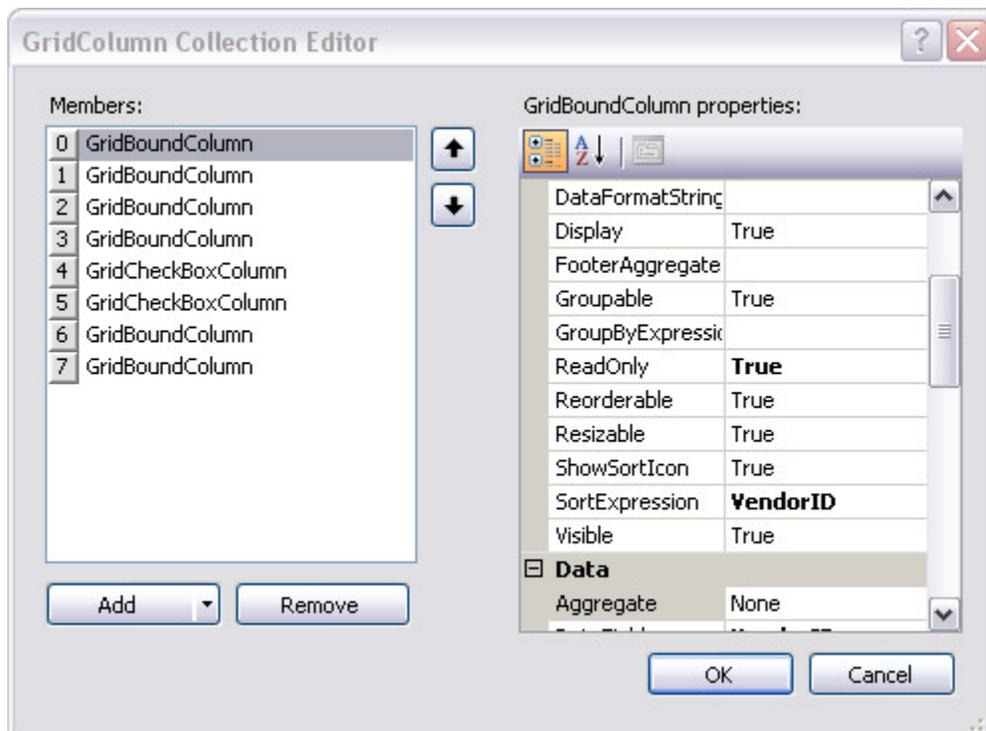
Declarative Column Creation

Whenever you connect the Grid to a data source for the first time the columns are automatically added to the **GridColumnCollection**, and more often than not you need to remove unwanted columns from the collection rather than adding them declaratively. However, if you have removed a column by mistake, it's worthwhile to know how to add the column back in to the collection.

If you have turned off automatic column generation in the smart tag, there will be a new smart tag entry available to access the columns collection editor:



Clicking the ellipses will display the Grid Column Collection Editor. The same editor can be invoked by expanding the **MasterTableView** property for the RadGrid, and clicking the ellipses on the **Columns** property there:



This editor allows access to the same properties that you'd find in the property builder. If you have removed a column that you later find you'd rather include in the display, you can add it back here. Start by clicking on the

Add button. This will expand a list of different column types you can add to your grid:

- **GridBoundColumn**: This will typically be used for unconstrained data; something other than a Boolean value or list-type data (which you would put in a `GridDropDownColumn`).
- **GridCheckBoxColumn**: Used for Boolean data.
- **GridDropDownColumn**: When the data entry choices are limited to items in a list, use this column type. When the column is in display mode, it will look just like a `GridBoundColumn`. You can also specify a data source, list text field, and list value field, so that in edit mode, the input is limited to the elements of the list.
- **GridTemplateColumn**: There's an extensive discussion of templates included in the `RadGrid` documentation. When you create a template column, you have complete control over the contents of the template. You can include textboxes and images, set the background, control the arrangement and presentation of controls using tables, include labels; in short, there's a complete web page environment contained within each cell of the grid.
- **GridEditCommandColumn**: The `EditCommandColumn` enables editing of the data contained in the row. When you add a `GridEditCommandColumn`, the `ButtonType` property can be set to one of three types: `LinkButton`, `PushButton`, and `ImageButton`.
- **GridButtonColumn**: A button column is very similar to the `GridEditCommandColumn`, except that you (not the designer) specify the name of the command that is sent to the web server for execution.
- **GridHyperLinkColumn**: As the name implies, this column has properties for the text and a URL to create a hypertext link.
- There are also `GridDateTimeColumn`, `GridMaskedColumn`, `GridNumericColumn`, `GridCalculatedColumn`, `GridClientSelectColumn`, `GridHTMLEditorColumn`, `GridImageColumn`, `GridBinaryImageColumn`, `GridRatingColumn`, `GridAttachmentColumn` - you can find more information about the rest of the columns in the `Column Types` help topic in the online documentation.

Column Resizing

If you want the columns in your grid to be resizable, set the `ClientSettings.Resizing.AllowColumnResize` property to `True`. When `AllowColumnResize` is `True`, users can resize columns by dragging the handle between column headers. The default value for this property is `false`.

The resizing feature can be adjusted using the following properties:

- When resizing is enabled (`AllowColumnResize` is `True`), you can disable column resizing for individual columns by setting the column's `Resizable` property to `False`. Setting a column's `Resizable` property has no effect if `AllowColumnResize` is `False`.
- To specify whether columns are resized using real-time resizing, set the `ClientSettings.Resizing.EnableRealTimeResize` property. The default value for this property is `False`.
- The `ClientSettings.Resizing.ResizeGridOnColumnResize` property lets you specify whether the entire grid changes size when its columns are resized. If you set `ResizeGridOnColumnResize` to `True`, the grid changes its size dynamically when the user resizes a column. All other columns retain their original sizes.
- The `ClientSettings.Resizing.ClipCellContentOnResize` property controls whether users can resize a column to the point where it can't display its entire contents. When `ClipCellContentOnResize` is `True` (the default), users can resize a column so that it is too narrow to display its entire contents. Instead, the content is clipped.
- Grid columns also support the "resize to fit" functionality. Double-clicking the resize handle or choosing "Best Fit" from the grid header context menu will automatically resize the target column to fit the widest cell's content without wrapping. To enable this feature you need to allow column resizing and set `ClientSettings.Resizing.AllowResizeToFit` to `true`.

You can see column resizing in action in the online demo available [here \(http://demos.telerik.com/aspnet-ajax/grid/examples/client/resizing/defaultcs.aspx\)](http://demos.telerik.com/aspnet-ajax/grid/examples/client/resizing/defaultcs.aspx).

Column Reordering

You can allow users to set the order of the grid columns by dragging and dropping them. Just set the `ClientSettings.AllowColumnsReorder` property to True. There are two possible modes for column reordering: client and server-side. If you want to reorder columns on client, set the `ClientSettings.ReorderColumnsOnClient` property to True.

- When columns are reordered on the client, The `ClientSettings.ColumnsReorderMethod` property determines what happens when the user drops a column in a new position. When `ColumnsReorderMethod` is "Swap" (the default), the dragged column switches places with the column that is currently in the target position. When `ColumnsReorderMethod` is "Reorder", all the columns between the dragged column's start position and its drop position shift over to make room for the dragged column. Changes do not persist on the server until after a postback.
- When columns are reordered on the server, the grid uses the "swap" method multiple times to re-order columns.

You can see column reordering in action in the online demo available [here \(http://demos.telerik.com/aspnet-ajax/grid/examples/client/resizing/defaultcs.aspx\)](http://demos.telerik.com/aspnet-ajax/grid/examples/client/resizing/defaultcs.aspx).

Column Aggregates

The `GridBoundColumn` object has an `Aggregate` property that you can set to specify a function for aggregating the values that the column displays. The `Aggregate` property can be set to any of the following values: "Sum", "Min", "Max", "Last", "First", "Count", "Avg", or "Custom". When you set the `Aggregate` property to "Custom", the grid raises an `OnCustomAggregate` event, where you can calculate the aggregate in server-side code and assign the result to the `Result` property of the event arguments object.

The grid calculates aggregated values if the `ShowFooter` property is set to True. The grid footer displays aggregates that are calculated based on all the data from the data source, except for any values that are excluded by a filter expression. When grouping is enabled in the column, you can display group aggregates by setting the `ShowGroupFooter` property to True.

The Column Aggregates feature is demonstrated in the online demo available [here \(http://demos.telerik.com/aspnet-ajax/grid/examples/generalfeatures/aggregates/defaultcs.aspx\)](http://demos.telerik.com/aspnet-ajax/grid/examples/generalfeatures/aggregates/defaultcs.aspx).

Multi-column headers

The multi-column headers of the RadGrid represent a tree-like structure where one or more columns can be grouped together by a common header. That common header in its turn can be child of another upper multi-column header which can also span both columns and other headers.

Structure rules:

- A **MultiColumn Header** can be a child of only one other multicolumn header.
- A **MultiColumn Header** must span at least one column.
- A **MultiColumn Header** should be defined only for a single row header per level. Hence a multicolumn header always has `RowSpan=1`.
- A column can have as a parent only one **MultiColumn Header**.
- Each column header can span only a single column. Hence a column header always has `ColSpan=1`.
- A column surrounded (in order of definition) by other columns with common multi header cannot have a different multicolumn header on **the same or higher row level** than the columns that surround it. This rule ensures proper rendering and avoids overlapping of multicolumn headers.

Definition:

In order to define the MultiColumn Headers in RadGrid **Column Groups** should be set.

ASPX

```
<ColumnGroups>
  <telerik:GridColumnGroup HeaderText="Product Details" Name="ProductDetails"/>
  <telerik:GridColumnGroup HeaderText="Location" Name="Location"/>
  <telerik:GridColumnGroup HeaderText="Category" Name="Category"
ParentGroupName="ProductDetails"/>
  <telerik:GridColumnGroup HeaderText="Order Details" Name="OrderDetails"
ParentGroupName="ProductDetails"/>
</ColumnGroups>
```

In order to add the needed column in the MultiColumn Header the **ColumnGroupName** property should be used:

ASPX

```
<telerik:GridBoundColumn UniqueName="Address" DataField="Address" ColumnGroupName="Location"
HeaderText="Address"/>
```

The above definition will be presented in the following output:



Location		Category		Name
Address	City	ID	Name	Name

⚠️ Frozen columns, show/hide columns on the client and resizing functionalities are officially not supported with multi-column headers.

19.9 Rows

Rows in RadGrid are presented by the **GridItem** class and its descendants. There are two types of rows:

- **Static rows** - always present in the grid structure regardless of whether they are visible or not. The number of these items is always known. To this group belong Header and Footer rows, CommandItem, Status bar item and Pager row.
- **Dynamic rows** - each dynamic row in the grid represents a record from the specified data source. Dynamic rows are represented by the **GridDataItem** class (a descendent of GridItem).

Each GridTableView has a set of rows (the **Items** collection) of type **GridDataItem**. The collection does not provide any methods to add or remove items. However, you can control the content of an item by providing a handler for the **ItemCreated** event. The number of dynamic rows depends on the number of rows (records) in the Data Source and the number of groups (if grouping is enabled). Dynamic rows consist of data items, nested-view items, group-header items and edit-form items.

Data items can come in two types:

Normal Rows - these are the odd rows of the grid. The appearance of the normal rows is controlled by the **ItemStyle** property.

Alternating Rows - these are the even rows of the grid. The appearance of the alternating rows is controlled by the **AlternatingItemStyle** property.

Below are described the main grid row features.

Row Resizing

You can allow row resizing by setting the **ClientSettings.Resizing.AllowRowResize** property to true. When you set this property, RadGrid automatically generates a column of type **GridRowIndicatorColumn**, to make it easier for users to resize rows. Rows can be resized by dragging any part of their bottom edge, so if you prefer to hide the RowIndicatorColumn, please set **ClientSettings.Resizing.ShowRowIndicatorColumn** to false.

You can see row resizing in action [here](http://demos.telerik.com/aspnet-ajax/grid/examples/client/resizing/defaultcs.aspx) (<http://demos.telerik.com/aspnet-ajax/grid/examples/client/resizing/defaultcs.aspx>).

Row Reordering

RadGrid exposes flexible event-driven mechanism to drag and drop grid records to reorder them within the same grid, move them to different grid instance or drop them over other html element on the page. In order to enable drag and drop of grid items, you need to set the two boolean grid properties to true, namely **AllowRowsDragDrop** and **AllowRowSelect**. This will make the grid data rows draggable and the end user will be able to relocate them if needed. Additionally, you can define a **GridDragDropColumn** in your GridTableView's Columns collection. This will make your grid items draggable only when grabbed by the drag handle in the GridDragDropColumn.

The event-driven model which allows you to process and complete the drag and drop operation can be separated into two phases: client-side and server-side phase.

- *Client-side phase*: there are three client grid events exposed to handle drag/drop action: **OnRowDragStarted** (cancelable), **OnRowDropping** (cancelable) and **OnRowDropped**. The **OnRowDragStarted** event can be intercepted if you want to perform some conditional check and determine whether to cancel the drag operation or not. The **OnRowDropping** event should be attached to identify the target element on which the dragged grid record is dropped. If this element does not meet your criteria for acceptable target, cancel the operation. The **OnRowDropped** event can be handled if you would like to execute some extra code logic prior to the server-side OnRowDrop event rising.
- *Server-side phase*: On the server there is a single event (named OnRowDrop). Subscribing to this event allows you to reorder the items in the source grid or remove them and append these rows to a destination grid instance. The sequence of actions you will have to undertake in order to change the source structure may vary because this depends strictly on the underlying data source and its data model. The common logic in all cases, however, is that you can use three arguments passed in the handler to accomplish the task:

1. **e.HtmlElement** - holds the html element (or grid item).
2. **e.DestDataItem** - the destination grid item object (either GridDataItem or GridNoRecordsItem).
3. **e.DraggedItems** - a collection of GridDataItems which holds the rows that are taking part in the current drop operation.
4. **e.DestinationGrid** - a reference to the grid instance to which the row has been dragged to.
5. **e.DestinationTableView** - a reference to the table to which the row has been dragged to, points to the MasterTableView or detail table in hierarchical grid.

Combining the client and server part completes the circle and separates logically each part of the drag and drop process until it is finalized.

You can see a live demo of the drag and drop functionality at [this address \(http://demos.telerik.com/aspnet-ajax/grid/examples/programming/draganddrop/defaultcs.aspx\)](http://demos.telerik.com/aspnet-ajax/grid/examples/programming/draganddrop/defaultcs.aspx).

20 RadEditor

20.1 Objectives

- Explore features of the Editor control.
- Learn how to configure RadEditor for the runtime environment.
- Explore the RadEditor design time interface including the Smart Tag and major property groups.
- Learn how to configure the tools file.
- Learn some advanced tasks including creating custom modules, content filters, buttons and drop down lists. You will also learn how to optimize for multiple editors and localize RadEditor for international use.
- Learn how to control RadEditor using the client API.

20.2 Introduction

RadEditor is a powerful but lightweight editor control you can use in your web applications where you need a full-featured editor, not a text box. It comes loaded with lots of built-in goodies like pre-defined buttons, drop down lists, File Browser dialogs and context menus that perform any tasks you are likely to need. If the built-in tools don't fit the bill, RadEditor can be extensively customized. Some of the hottest features are:

- Unmatched Loading Speed
- Minimal Script Size
- New Semantic Rendering
- Out-of-the-box XHTML-enabled Output
- Industry-best Cross-browser Support
- Single-file, Drag-and-drop Deployment (all editor resources, including the dialogs reside in the same dll)
- Multilevel Undo/Redo with Action Trails
- 7 Ways to Paste from Word
- AJAX-Based File Browser Dialogs
- Full keyboard accessibility
- Flexible Skinning mechanism
- Simplified and intuitive toolbar configuration
- Ability to have editors with different skins on the same page

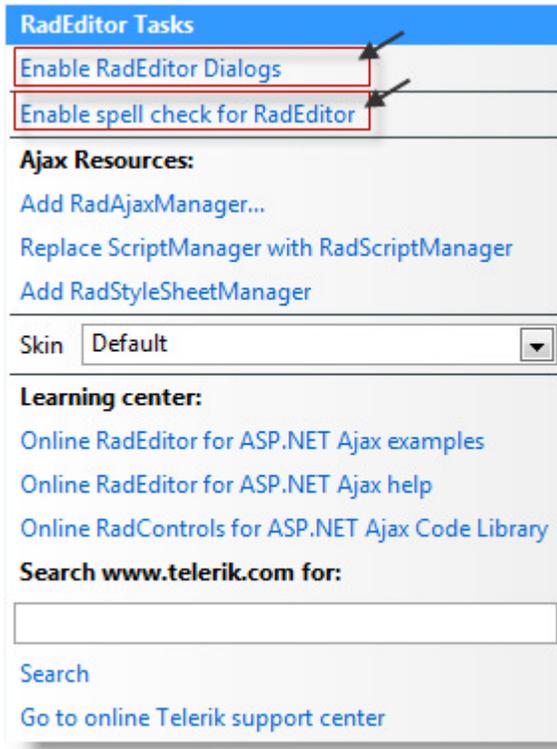


20.3 Getting Started

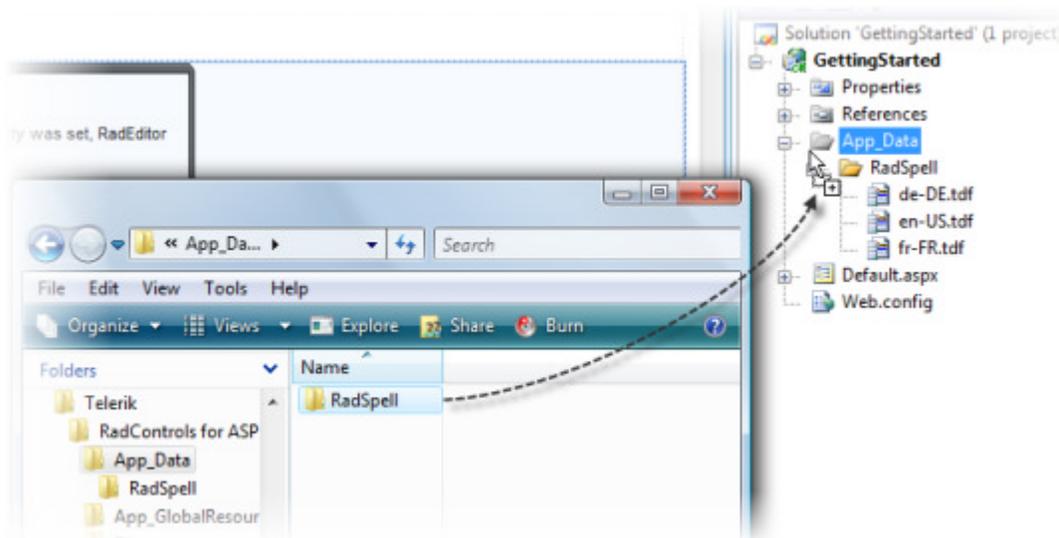
Using RadEditor with Dialogs and Spell Check

The minimum steps to getting RadEditor up and running in a browser are:

1. In a new ASP.NET Web Application, drag a **RadEditor** to the default page.
2. In the Smart tag select the following links:
 1. **Enable RadEditor Dialogs**
 2. **Enable Spell Check for RadEditor**



3. Locate the RadControls installation directory \App_Data folder and copy the contents to your project's \App_Data folder. **Note:** You really only need to copy the en-us.tdf dictionary file to allow spell checking.



4. This next part is not "minimal", but is still included here. From the Smart Tag, drop down the list of Skins and select the "Vista" skin.
5. Press Ctl-F5 to run the application. In the example below, some marketing text has been pasted to editor.

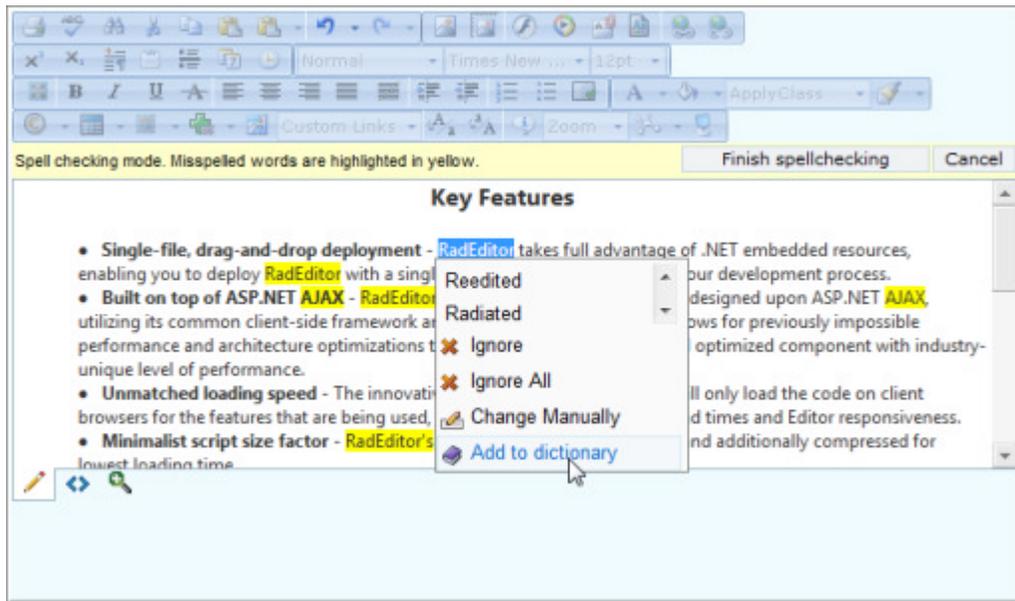


You can paste any text or HTML that is on your clipboard using the paste button.

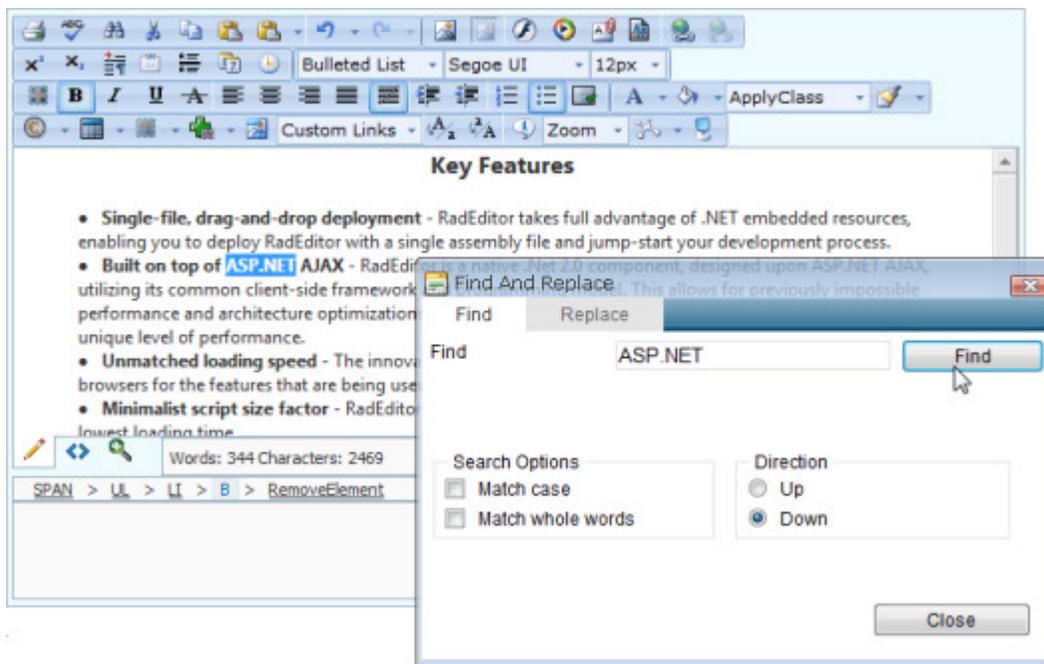


Try clicking the spell check button. The spell check occurs right in the editor content area and drops down a list of suggestions at each misspelled word with the industry standard options for Ignore, Add to

Dictionary, etc. When you're done you can click the **Finish Spellchecking** button to retain your changes or **Cancel** to abandon your changes. Spell checking was enabled by the Smart Tag "Enable Spell Check for RadEditor" link.

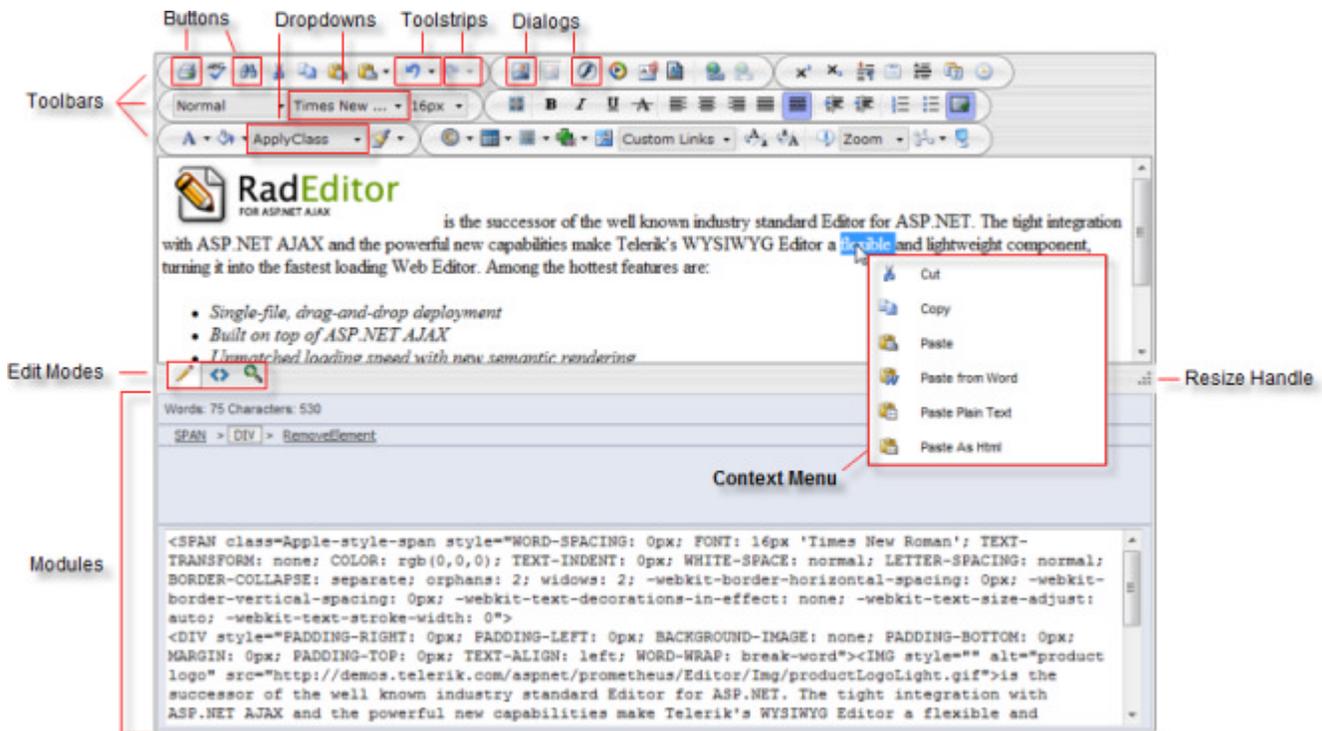


Try clicking the **Find** button and locate some string of text. The option that allows the Find dialog to appear, or any other dialog that RadEditor supports, is enabled by the Smart Tag option you took labeled "Enable RadEditor Dialogs".



RadEditor Elements

RadEditor is made up of toolbars, the content area and various modules. The toolbar in turn contains buttons, dropdown lists, toolstrips and dialogs.



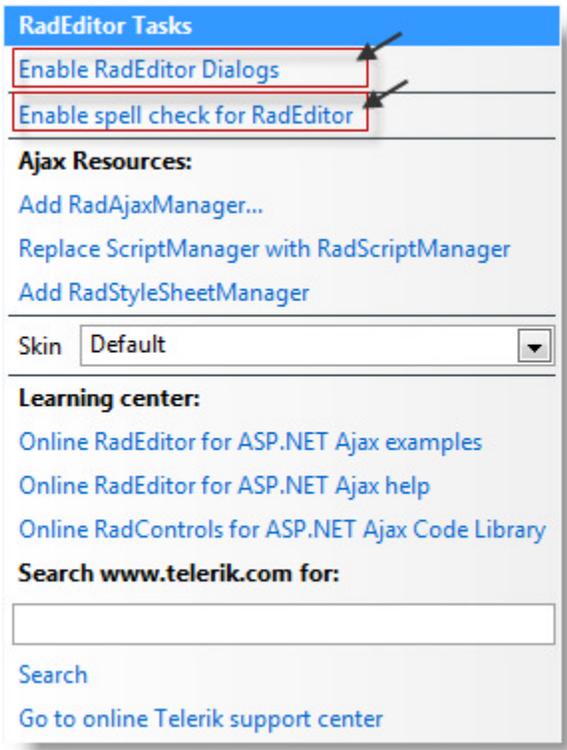
- **Toolbars:** The main elements of RadEditor are the Toolbars. They are in fact containers, which accommodate the buttons and dropdown lists of the various tools.
- **Buttons:** Used to edit content, to launch different dialogs, to Undo / Redo the content, save or cancel the changes, etc.
- **Dropdowns:** Used to format the font appearance (family, size, color, apply css class) as well as to insert objects into the content area such as html code snippets.
- **ToolStrip:** Dropdowns that contain a group of tools with related functionality and can be a very convenient means of arranging tools used in the editor.
- **Dialogs:** Used to insert objects into the content area such as images, links, media and flash files.
- **Context Menus:** Shortcut menus that include commands associated with objects on the screen. With RadEditor, you can use the default context menus as well as specify custom menus for various HTML elements (e.g. different context menus for images, tables, hyperlinks etc.) You can also disable the context menus for certain elements (e.g. for tables).
- **Modules:** Special tools used to provide extra information such as Dom Inspector, real time HTML Viewer, Statistics module
- **Editor Mode buttons:** Used to switch between the editor's viewing modes: Design, HTML and Preview.
- **Resize Handle:** Lets the user drag the editor boundaries within the browser.

20.4 Designer Interface

In the Visual Studio designer, you can configure the **RadEditor** control using the Smart Tag and the Properties Window.

Smart Tag

The RadEditor Smart Tag contains a few control-specific entries in addition to the standard Ajax Resources, Skin selection, and Learning center sections. The two entries "Enable RadEditor Dialogs" and "Enable spell check for RadEditor" add dialog and spell check handlers to your web config. After these items are clicked they no longer appear in the Smart Tag. Failing to click these two links will generate errors at runtime when you attempt to use one of the dialogs or the spell checker.



Properties Window

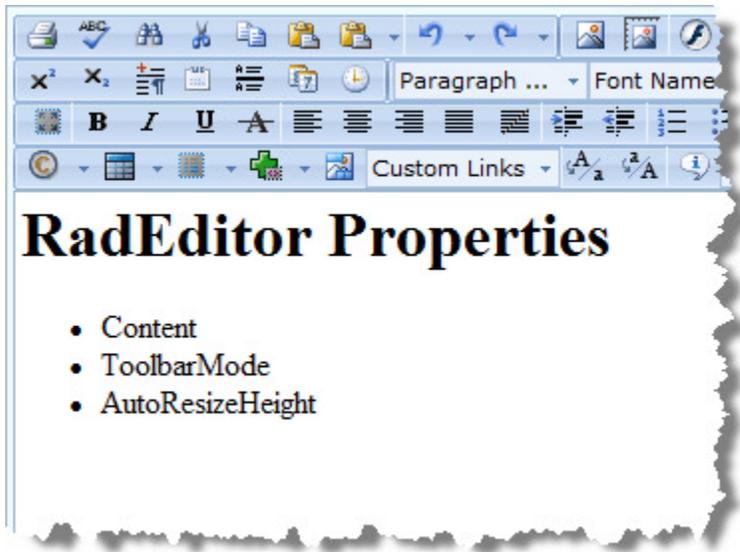
The most important property is **Content**. Content can be assigned any text or any HTML. As a user, you can copy and paste HTML directly from the clipboard to the editor. Programmatically, you can just assign to the Content property:

[VB] Assigning Content

```
RadEditor1.Content = "<H1>RadEditor Properties</H1><ul>" + "<li>Content</li>" +
"<li>ToolbarMode</li>" + "<li>AutoResizeHeight</li></ul>"
```

[C#] Assigning Content

```
RadEditor1.Content =
"<H1>RadEditor Properties</H1><ul>" +
"<li>Content</li>" +
"<li>ToolbarMode</li>" +
"<li>AutoResizeHeight</li></ul>";
```



RadEditor has an extensive set of properties, but we can group some of these to make it easier to navigate. In the Properties Window, click the Categorized button  to follow along. We won't look at every single property or group of properties, but try to get a feel for where the significant properties are found.

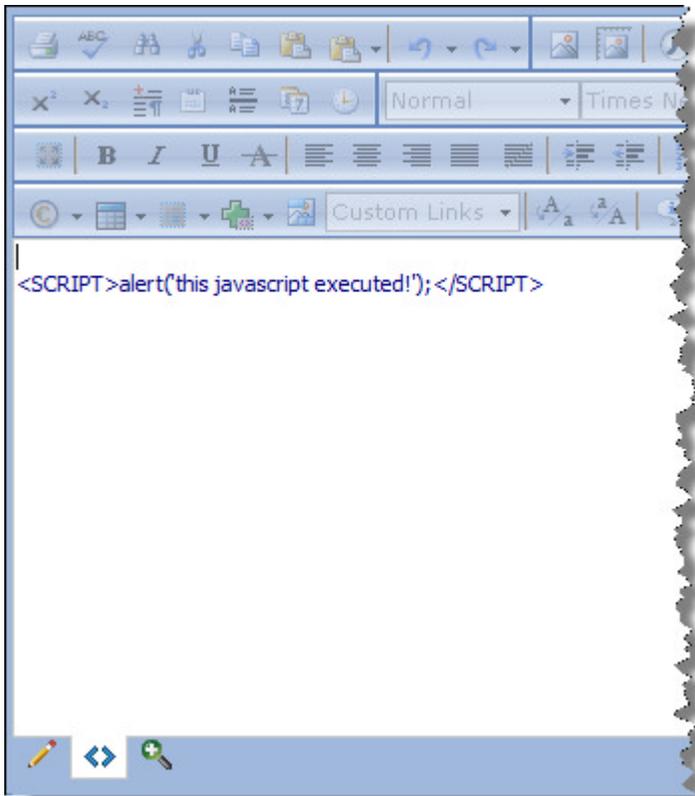
Appearance

This group of properties controls appearance on several levels:

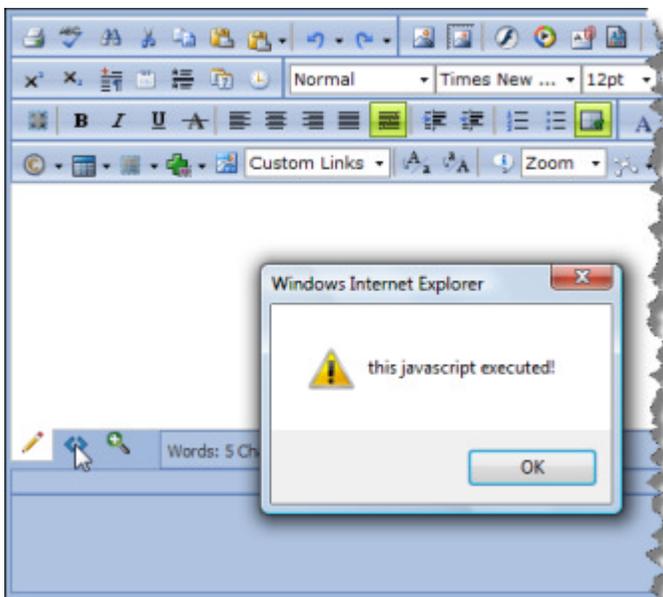
- Individual property settings such as **BorderColor**, **BorderWidth**, **ForeColor**, etc. These properties will work in limited scenarios where the styles or skins are not already at work and where you have a property that already addresses the visual change you need to make.
- CSS styles: You can set **CssClass** to assign a style to the control as a whole. You can also point **ContentAreaCssFile** to a file name that holds styles for the content area.
- Skins: You can set the **Skin** to an predefined value to get a coordinated look-and-feel. You can also customize an existing skin or build your own from scratch. Skins provide a generalized framework for changing editor appearance so that it works with the other controls in your application.

Behavior

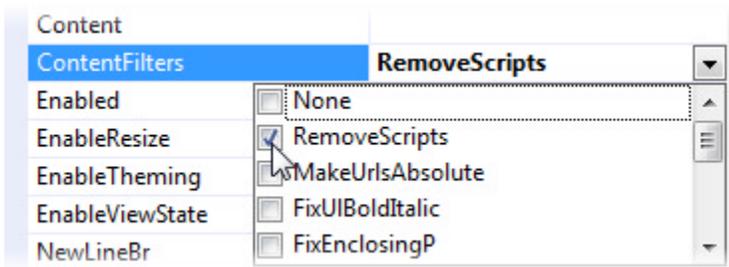
ContentFilters list a set of JavaScript objects that can be enabled to act on the content when submitting a page or when switching between Design and HTML views. For example, if no filters are activated, we can add a JavaScript tag in the editor, move between HTML and normal views and the JavaScript will actually execute. The screenshot below shows the script entered in HTML mode.



The next screenshot shows what happens when you navigate back to the Design tab and then back to the HTML tab.



When you select the RemoveScripts content filter, the script is completely removed when you move between edit modes. You can add this at design time from the Property window:



In code you can combine these flags using the bitwise OR operator:

[VB] Assign Content Filters

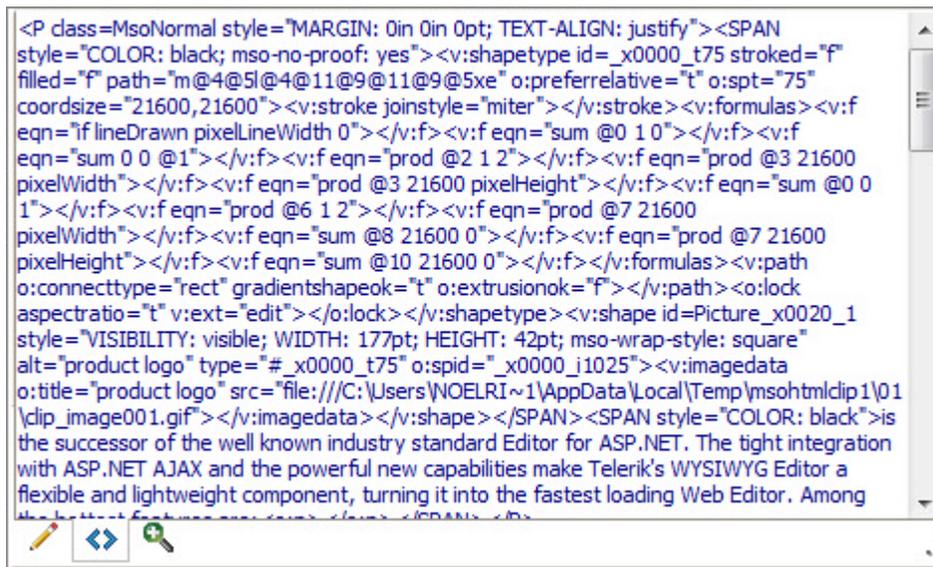
```
RadEditor1.ContentFilters = Telerik.Web.UI.EditorFilters.MakeUrlsAbsolute Or  
Telerik.Web.UI.EditorFilters.FixEnclosingP
```

[C#] Assign Content Filters

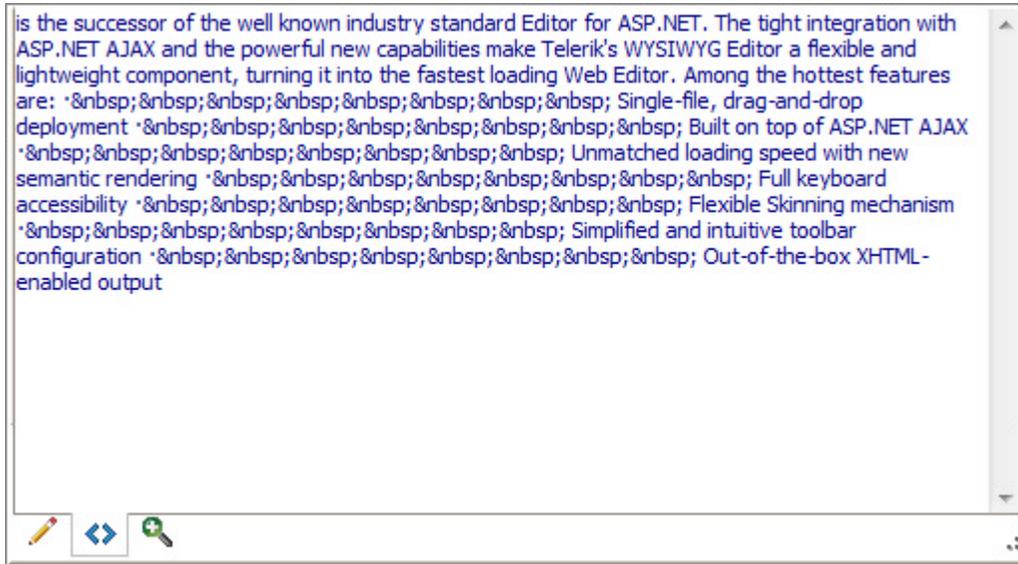
```
RadEditor1.ContentFilters =  
Telerik.Web.UI.EditorFilters.MakeUrlsAbsolute |  
Telerik.Web.UI.EditorFilters.FixEnclosingP;
```

Using the Behavior properties you can also:

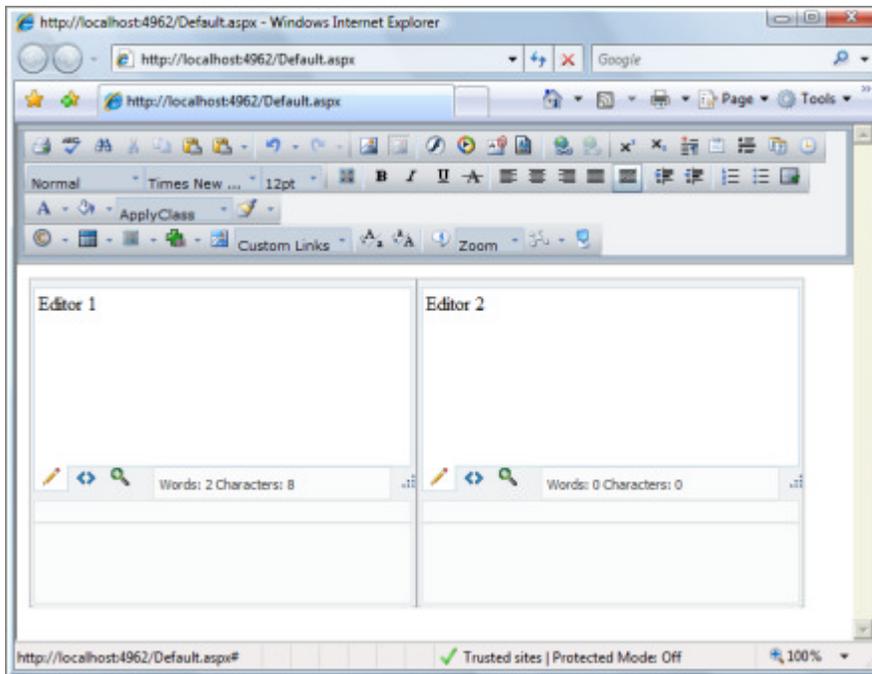
- Turn off the **EnableResize** property to prevent the user from changing the editor dimensions.
- Toggle the **NewLineBr** property. **NewLineBr** is true by default which means that every time the user hits Enter, a `
` tag is generated. If you set **NewLineBr** false, each line is surrounded with page `<p>` tags.
- Configure the **StripFormattingOptions** property to one or more of these values: **None**, **NoneSuppressCleanMessage**, **MSWord**, **MSWordNoFonts**, **MSWordRemoveAll**, **Css**, **Font**, **Span**, **AllExceptNewLines** and **All**. These can be selected from the Properties window or use the bitwise OR operator to combine these values. When you paste from MS Word with this property set to "None", you get quite the formatting circus:



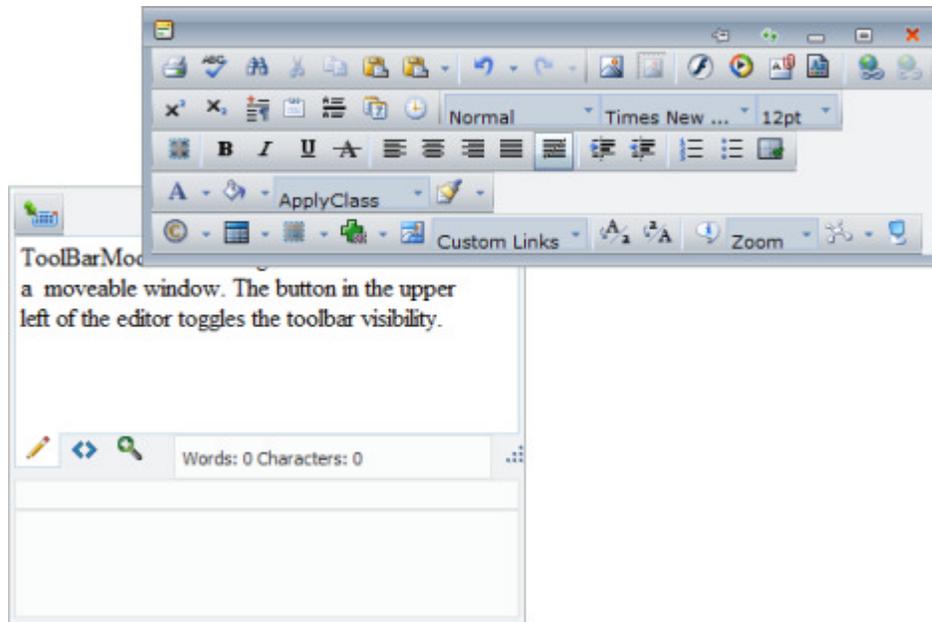
With this property set to the other extreme of "All", you get simple HTML elements only:



- Change the **ToolBarMode** property from **Default**, where the tool bar is static and positioned over the content area to **PageTop**, **ShowOnFocus** or **Floating**. If you use PageTop, the toolbar docks to the top of the entire web page, so that if you have multiple editors open, the one toolbar applies to all. This screenshot shows PageTop with two editors.



ShowOnFocus will cause the ToolBar to appear right above the editor when it gets focus. Later we will talk about how ShowOnFocus can be used together with the **ToolProviderID** for awesome performance when loading multiple editors on the same page. **Floating** will cause the toolbar to pop up in a window and will allow the user to move it over the page. This next screenshot is an example of the floating toolbar:



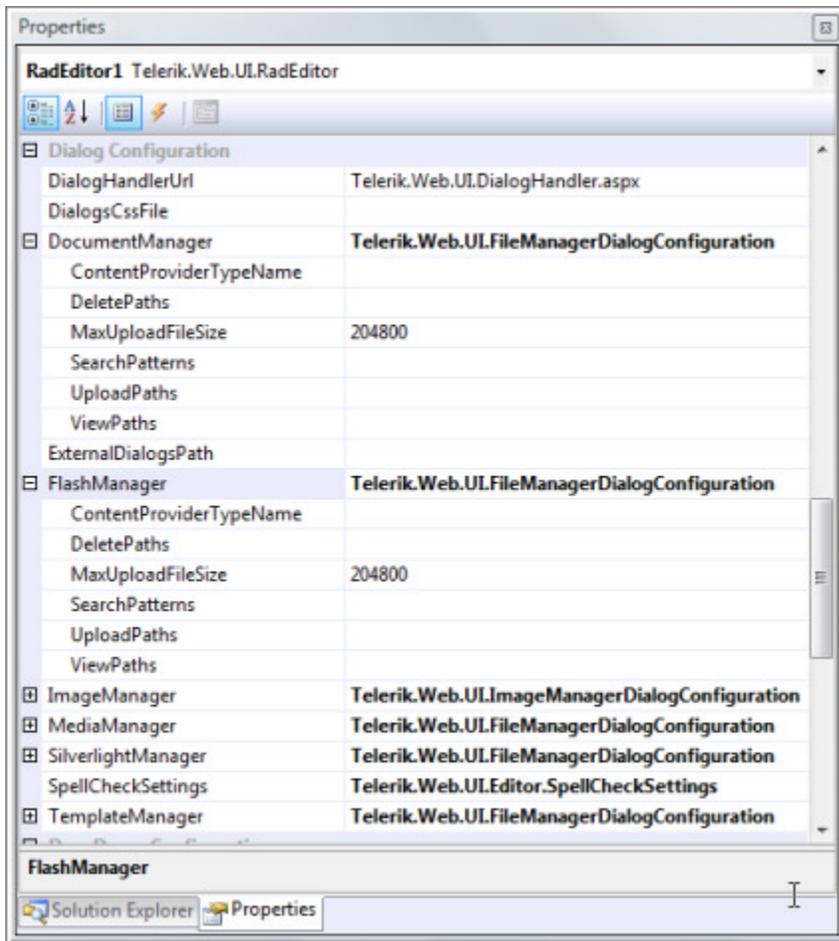
Client-Side Events

We will explore these events in the upcoming section on Client-Side Programming. For now, just know the events fire on the client when editor is first initialized and loaded, and when the user causes events to fire, i.e. when commands are executing, a paste is in process or when the text selection changes.

Dialog Configuration

RadEditor dialogs are used to insert objects into the content area. These "FileBrowser" dialogs handle images, Flash, documents, Silverlight, media and templates. The FileBrowser dialogs consist of a FileBrowser object, an object previewer/property manager and a file uploader tab. The FileBrowser provides the ability to browse directories and locate a file item. Selected file items are loaded into the previewer.

This Dialog Configuration section of the Properties Window has a number of "Manager" properties that correspond to dialogs.



The sub-properties are similar between specific managers. The `ContentProviderTypeName` allows you to create your own dialog manager by inheriting from `FileSystemContentProvider` and plugging it in to this property. The various "paths" properties determine what files can be seen by the dialog:

- **ViewPaths** specifies where files are located. The dialog will display all files found in this directory and children of this directory.
- **UploadPaths** specifies where users can upload files. To be visible, these paths need to be within "ViewPaths".
- **DeletePaths** specifies where users can delete files. Again, these paths need to be within the "ViewPaths" paths to be visible.

For example, the markup that defines the `ImageManager` paths shows the base path, that is `ViewPaths` is the "images" directory, files can be uploaded to "images/new" and files can be deleted from "images/new/articles" and "images/new/news".

[ASP.NET] Defining Dialog Paths

```
<telerik:radeditor runat="server" ID="RadEditor1" >
  <ImageManager
    ViewPaths="~/Images"
    UploadPaths="~/Images/New"
    DeletePaths="~/Images/New/Articles,~/Images/New/News"
  />
```

</telerik:radeditor>



You can set up role-based security by dynamically assigning dialog paths at runtime. For example:

[VB] Assigning Paths based on Role

```
Select Case userRole
Case "Mike"
    'Administrator
    RadEditor1.ImageManager.ViewPaths = New String() {"~/"}
    RadEditor1.ImageManager.UploadPaths = New String() {"~/"}
    RadEditor1.ImageManager.DeletePaths = New String() {"~/"}
    Exit Select
Case "John"
    'John
    RadEditor1.ImageManager.ViewPaths = New String() {"~/Common"}
    RadEditor1.ImageManager.UploadPaths = New String() {"~/Common"}
    RadEditor1.ImageManager.DeletePaths = New String() {"~/Common", "~/Marketing"}
    Exit Select
Case Else
    'all users
    RadEditor1.ImageManager.ViewPaths = New String() {"~/Common/Resources"}
    RadEditor1.ImageManager.UploadPaths = New String() {"~/Common/Resources"}
    RadEditor1.ImageManager.DeletePaths = New String() {"~/Common/Resources",
"~/Marketing/Resources"}
    Exit Select
End Select
```

[C#] Assigning Paths based on Role

```
switch (userRole)
{
    case "Mike": /*Administrator*/
        RadEditor1.ImageManager.ViewPaths = new string[] { "~/ " };
        RadEditor1.ImageManager.UploadPaths = new string[] { "~/ " };
        RadEditor1.ImageManager.DeletePaths = new string[] { "~/ " };
        break;
    case "John": /*John*/
        RadEditor1.ImageManager.ViewPaths = new string[] { "~/Common " };
        RadEditor1.ImageManager.UploadPaths = new string[] { "~/Common " };
        RadEditor1.ImageManager.DeletePaths = new string[] { "~/Common", "~/Marketing " };
        break;
    default: /*all users*/
        RadEditor1.ImageManager.ViewPaths = new string[] { "~/Common/Resources " };
        RadEditor1.ImageManager.UploadPaths = new string[] { "~/Common/Resources " };
        RadEditor1.ImageManager.DeletePaths = new string[] { "~/Common/Resources",
"~/Marketing/Resources " };
        break;
}
```

The last couple of properties for a dialog manager are **SearchPatterns**, that defines a list of extensions that can be displayed by a dialog and **MaxUploadFileSize** that controls the maximum allowed file size.

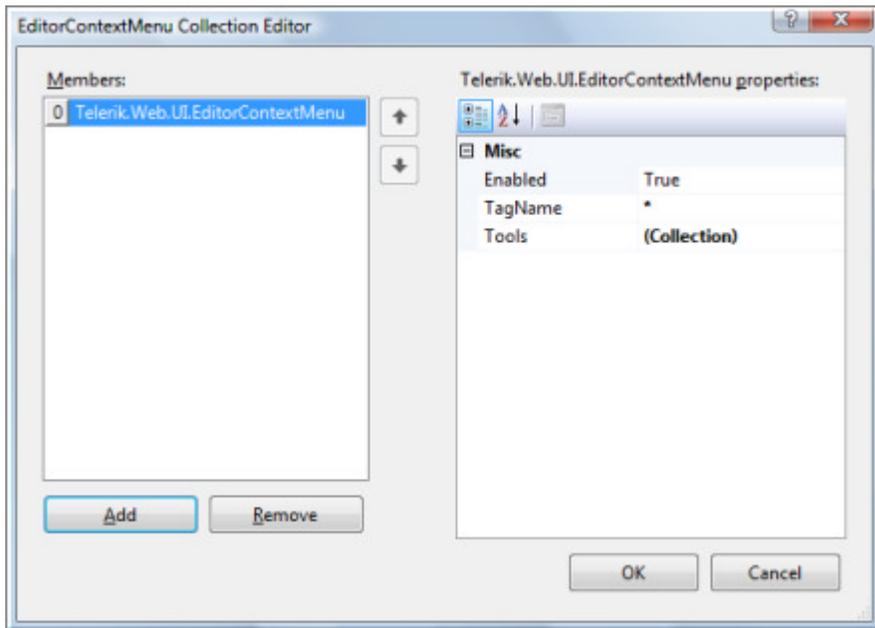
DropDown Configuration

The DropDown Configuration section of properties are collections that populate drop down lists within the

editor.

DropDown Configuration	
Colors	(Collection)
ContextMenus	(Collection)
CssClasses	(Collection)
CssFiles	(Collection)
FontNames	(Collection)
FontSizes	(Collection)
Links	(Collection)
Paragraphs	(Collection)
RealFontSizes	(Collection)
Snippets	(Collection)
Symbols	(Collection)

Clicking the ellipses for any of these properties brings up a collection editor that will allow you to define members of the collection. Properties for each member are specific to the kind of members being defined, like the context menu editor collection shown in the image below.



Tools

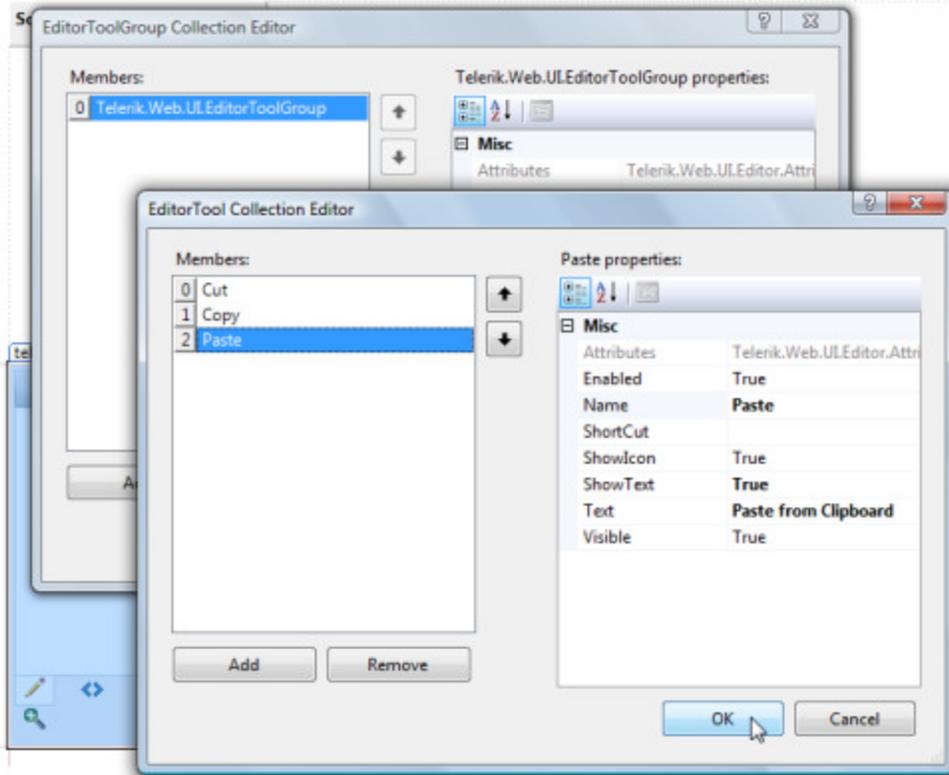
During the "Getting Started" section on "RadEditor Elements" we talked about how the editor was made up of Tools, Modules and content. The Tools category of properties controls what tools and modules you will see. The tools can be defined by either the **Tools** collection or by specifying an XML file and pointing to it with the **ToolsFile** property.

Tools	
Modules	(Collection)
Tools	(Collection)
ToolsFile	

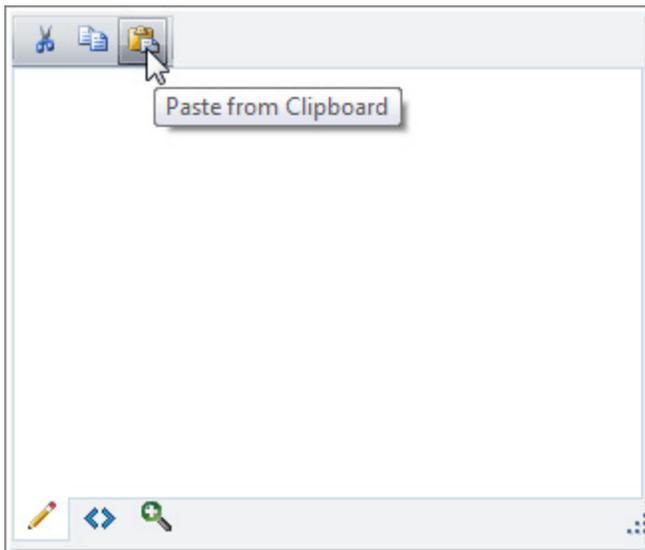
If you go the Tools collection route and click the ellipses for this property the "EditorToolGroup Collection Editor" displays. From there you can create tool groups. For each group you click the Tools collection to display

UI for ASP.NET AJAX

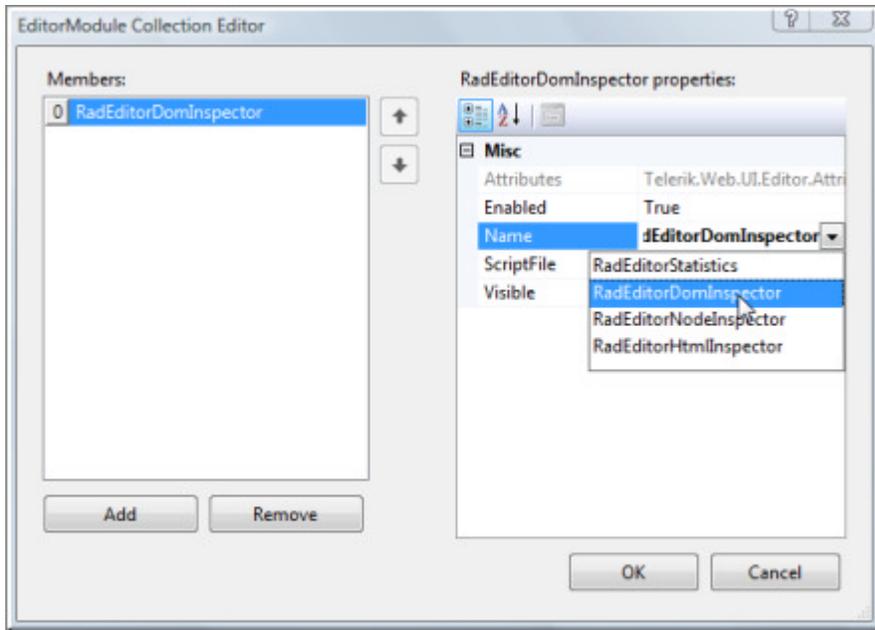
a second dialog "EditorTool Collection Editor". For each tool you select the **Name** property from a predefined drop down list of possible commands. You can also supply your own tooltip text as well as toggle the display of text and icons. The screenshot below shows the omnipresent "Cut, Copy and Paste" commands defined.



As you can see from the next screenshot, the tools defined in the Tools collection or from the ToolsFile completely replace the default tool set.



When you click the **Modules** collection ellipses, the EditorModule Collection Editor displays. You can add modules and select the **Name** property from the drop down list.



20.5 Using the NewLineMode Property

Here is some general info on how the browser works with `
` and `<p>` tags.

Carriage returns (when pressing Enter) are not significant when it comes to HTML. It doesn't matter how much whitespace is in the HTML code - it is automatically converted to a single space when your HTML document is displayed in a browser. For an example, if you put in your HTML file the following text:

Introducing Telerik RadEditor for ASP.NET AJAX. Improved user experience. Enhanced cross-browser support.

In the browser it will be displayed as a single line (empty spaces and carriage returns are being converted to a single whitespace):

Introducing Telerik RadEditor for ASP.NET AJAX. Improved user experience. Enhanced cross-browser support.

If you wish to enter line breaks, e.g. you want your text to be displayed on two or more lines, but not to start a new paragraph, you should use a `
` tag. For an example:

**Introducing Telerik RadEditor for ASP.NET AJAX. `
`Improved user experience. `
`Enhanced cross-browser support.**

is displayed in the browser as:

**Introducing Telerik RadEditor for ASP.NET AJAX.
Improved user experience.
Enhanced cross-browser support.**

Note

Note that the `
` tag is an empty tag and has no closing tag.

If you want to start a new paragraph, you should use the `<p>` (paragraph) tag. In HTML, this means automatically adding an extra blank line before and after a paragraph. e.g.:

`<p>`Introducing Telerik RadEditor for ASP.NET AJAX.`</p>``<p>`Improved user experience.`</p>``<p>`Enhanced cross-browser support.`</p>`

is displayed as:

Introducing Telerik RadEditor for ASP.NET AJAX.

Improved user experience.

Enhanced cross-browser support.

⚠ Caution

In a nutshell, if you want to end a line, but don't want to start a new paragraph, use `
` tag. If you wish to start a new paragraph - you use the `<p>` tag.

Telerik RadEditor's **NewLineMode** property specifies whether the editor should insert a **new line** (`
` tag), a **new paragraph** (`<p>` tag) or a **div** (`<div>` tag) when the **Enter** key is pressed. The default value is **"Br"** (`
` tag) in order to closely reflect the behavior of desktop word-processors. In this case you can insert paragraph tags by pressing **Ctrl+M** or the **New Paragraph** button.

If you set the **NewLineMode** property to **P**, a paragraph tag will be entered on every carriage return (pressing **Enter**). Here, pressing **Shift+Enter** will insert a `
` tag.

The last available option of the **NewLineMode** property is **Div** which will insert a **div** (`<div>` tag) when pressing **Enter**. In order to insert a `
` tag in this mode press **Shift+Enter**.

20.6 Customizing Content Area

ContentAreaMode="IFRAME" mode:

The Rich Text content area of RadEditor is an editable IFRAME element, which is a separate document that has its own CSS styles applied through the embedded in the Telerik.Web.UI.dll skin. This default content appearance in the content area can be easily overridden by setting the editor's **ContentAreaCssFile** property to point to your own **CSS file**. For example create a file named `EditorContentAreaStyles.css` and put a global body tag with the desired font and color settings in it, e.g.

```
body
{
    font-family: Verdana !important;
    font-size: 18px !important;
    color: white !important;
    background-color: #555 !important;
    text-align: left !important;
    word-wrap: break-word !important;
}
```

Since the css file is loaded first on purpose, it is possible for another global class on the page to override its settings. To prevent the overriding of the properties defined in the `EditorContentAreaStyles.css` file just set the **!important** rule after the properties values (or use the editor's **CssFiles** property to load the css file).

Save the css file and set the **ContentAreaCssFile** property to point to it:

```
<telerik:RadEditor
    ContentAreaCssFile="~/EditorContentAreaStyles.css"
    id="RadEditor1"
    runat="server">
</telerik:RadEditor>
```

To style other HTML elements in the content area you need to define global css classes for them, e.g. `p`, `div`, `span`, `table`, `td`, `ol`, `ul`, `li`, `img` etc

```

form
{
  background-color:#efefef !important;
  border:1px dashed #555555 !important;
}
table
{
  BORDER-RIGHT: #999999 1px dashed !important;
  BORDER-BOTTOM: #999999 1px dashed !important;
}
table td
{
  font-size: 12px !important;
  PADDING: 1px !important;
  BORDER-TOP: #999999 1px dashed !important;
  BORDER-LEFT: #999999 1px dashed !important;
}
div
{
  background-color: Green !important;
  color: Yellow !important;
  font-weight: bold !important;
}
img
{
  margin: 1px 1px 1px 1px !important;
  border: solid 1px blue !important;
}

```

More information on the subject is available in the following help articles:

[Setting Content Area Defaults](http://www.telerik.com/help/aspnet-ajax/editor-setting-content-area-defaults.html) (<http://www.telerik.com/help/aspnet-ajax/editor-setting-content-area-defaults.html>),

[Default Font for Editable Content](http://www.telerik.com/help/aspnet-ajax/defaultfontforeditablecontent.html) (<http://www.telerik.com/help/aspnet-ajax/defaultfontforeditablecontent.html>),

[Setting Editor Background And Color](http://www.telerik.com/help/aspnet-ajax/settingeditorbackgroundandcolor.html) (<http://www.telerik.com/help/aspnet-ajax/settingeditorbackgroundandcolor.html>),

[Content Area Appearance Problems](http://www.telerik.com/help/aspnet-ajax/editor-content-area-appearance-problems.html) (<http://www.telerik.com/help/aspnet-ajax/editor-content-area-appearance-problems.html>).

If the editor is placed in non-editable mode (**Enabled="false"**), then its content is outputted in a DIV element on the page. This DIV element will inherit the page styles or the styles of its parent elements, but not the styles of the EditorContentAreaStyles.css file and therefore the content might look different in edit and non-editable modes.

ContentAreaMode="DIV" mode:

The DIV element is part of the current page and the page CSS styles will be directly imported and applied to the content area and the contents in it. In order to customize the content area appearance of the RadEditor in Div mode, register the CSS selectors manually on the page with the appropriate cascading (**.reContentArea** <global selector: P, FORM, IMG, TABLE, TR, TD, H1-H6, etc>), e.g.

```

.reContentArea /*this selector corresponds to the body selector when RadEditor is in Iframe mode*/
{
  font-family: Verdana !important;
  font-size: 12px !important;
}

```

```
color: white;
background-color: #555 !important;
text-align: left !important;
word-wrap: break-word !important;
padding: 3px 15px 3px 15px !important;
}
```

```
.reContentArea P
{
margin: 0;
border: 1px solid #666;
color: #666;
font-size: 12px;
padding: 10px;
}
```

```
.reContentArea H1
{
margin: 0;
border: 1px solid #666;
color: #000;
padding: 20px;
}
```

```
.reContentArea OL
{
margin-top: 20px;
list-style-type: lower-roman;
border: 1px solid #666;
color: #555;
padding: 10px 10px 10px 55px;
}
```

```
.reContentArea table
{
BORDER-RIGHT: #99ff99 1px dashed;
BORDER-BOTTOM: #99ff99 1px dashed;
width:100%;
margin-top: 20px;
}
```

```
.reContentArea table td
{
font-size: 12px !important;
color: #555;
PADDING: 1px;
BORDER-TOP: #99ff99 1px dashed;
BORDER-LEFT: #99ff99 1px dashed;
text-align: center;
}
```

```
.reContentArea img
{
margin: 1px 1px 1px 1px;
border: solid 1px blue;
}
```

When the ContentAreaMode="DIV" is used the **ContentAreaCssFile** is not functional and the classes above should be registered manually using <link> and / or <style> tags.

20.7 Configuring the ToolsFile

You can configure all of your tool bars and modules at one time using an XML file with this basic structure:

[XML] ToolsFile Structure Sample

```
<root>
  <modules>
    <module />
    <module />
  </modules>
  <tools>
    <tool />
    <tool />
    ...
  </tools>
  <tools>
    <tool />
    ...
  </tools>
  ...
  <links>
    <link />
    <link />
  </links>
  <colors>
    <color />
    <color />
  </colors>
</root>
```

This next walk-through will show you how to setup the tools file and give you a few ideas on how you might use it. You can find the full reference for the possible toolsfile entries at **Using the ToolsFile.xml** (<http://www.telerik.com/help/aspnet-ajax/usintheToolsfile.html>). This next example will add several sets of toolbars with separators, a context menu that responds to <a> and <p> tags and a module that displays statistics.

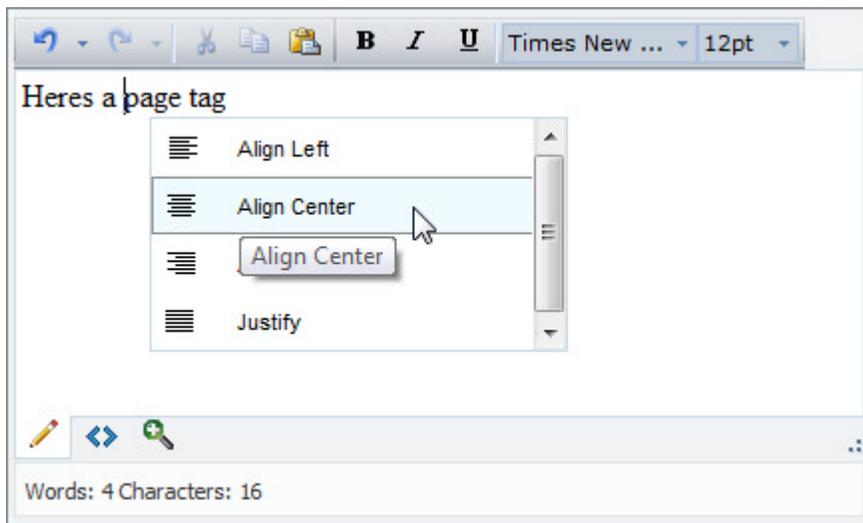
1. In a new ASP.NET Web Application, drag a **RadEditor** to the default page.
2. In the Smart tag select the following links:
 - o **Enable RadEditor Dialogs**
 - o **Enable Spell Check for RadEditor**
3. Locate the RadControls installation directory \App_Data folder and copy the contents to your project's \App_Data folder.
4. From the Solution Explorer, right-click the project and select **Add Item** from the context menu. Select the XML type and name the file "MyTools.xml".
5. Add the following to MyTools.xml:

[XML] Defining the ToolsFile

```
<root>
  <tools name="MainToolbar" enabled="true">
```

```
<tool name="Undo" />
<tool name="Redo" />
<tool separator="true"/>
<tool name="Cut" />
<tool name="Copy" />
<tool name="Paste" shortcut="CTRL+!"/>
</tools>
<tools name="Formatting" enabled="true">
  <tool name="Bold" />
  <tool name="Italic" />
  <tool name="Underline" />
  <tool separator="true"/>
  <tool name="FontName"/>
  <tool name="RealFontSize"/>
</tools>
<contextMenus>
  <contextMenu forElement="A" enabled="false">
  </contextMenu>
  <contextMenu forElement="P">
    <tool name="JustifyLeft" />
    <tool name="JustifyCenter" />
    <tool name="JustifyRight" />
    <tool name="JustifyFull" />
  </contextMenu>
</contextMenus>
<modules>
  <module name="RadEditorStatistics" visible="true" />
</modules>
</root>
```

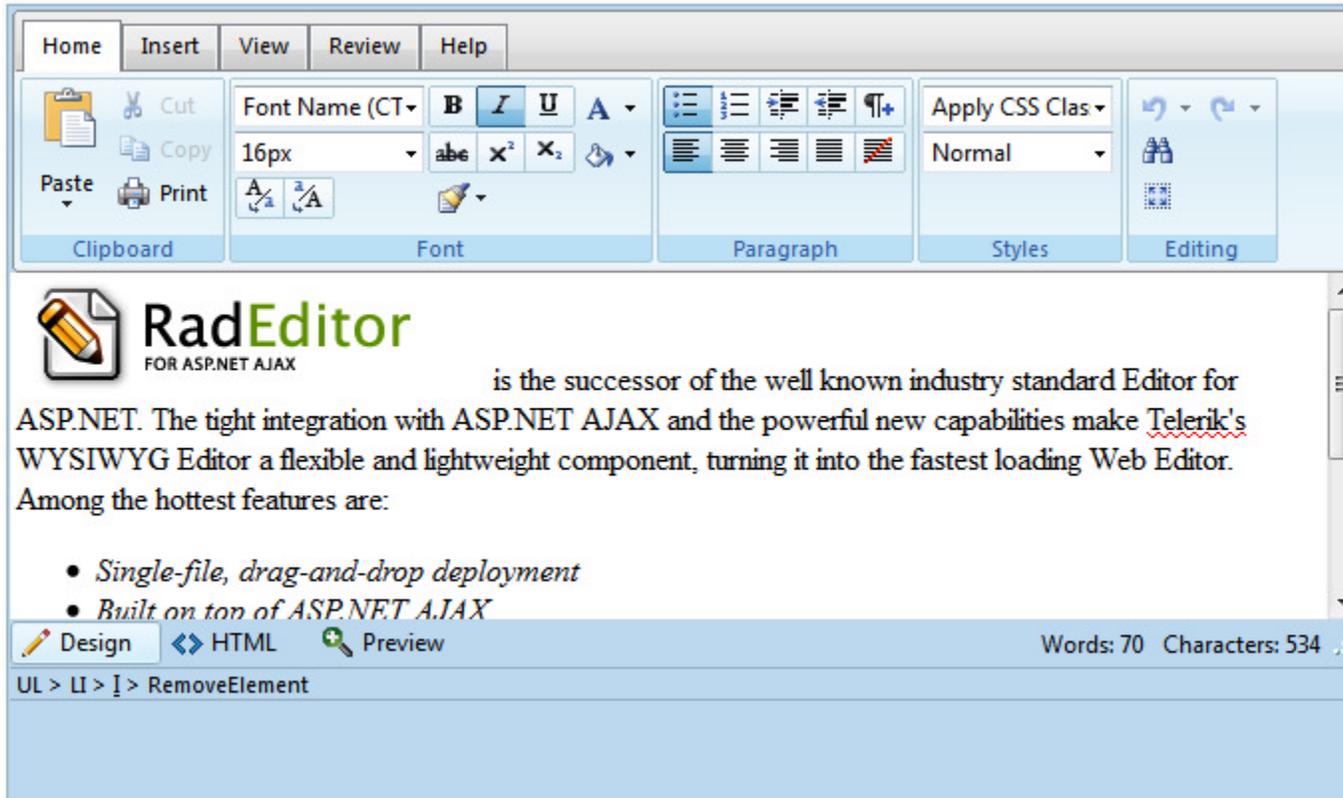
6. Press **Ctrl-F5** to run the application. Try adding a `<p>` or `<a>` tag to the HTML mode and right-click to get the custom context menu.



You can find the complete source for this project at:
\\VS Projects\\Editor\\ToolsFile

20.8 RibbonBar and Editor

The **RibbonBar** member of the **ToolBarMode** enumeration property configures RadEditor to use a **RadRibbonBar** control as a container for its tools.



There are some RibbonBar related properties which are added for the **tools** and **tool** xml elements in the editor's **ToolsFile** file (Section 20.7):

- For the **tools** elements the "name" property sets the RibbonBarGroup control, in which the tool is loaded and the "tab" property sets the RibbonBarTab control, in which the group is loaded.
- For the **tool** elements the "size" property sets the size of the buttons.

You can review the **tools.xml** file below for more information:

tools.xml configuration file

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <modules>
    <module name="RadEditorStatistics" dockingZone="Bottom"/>
    <module name="RadEditorDomInspector" />
    <module name="RadEditorNodeInspector" />
    <module name="RadEditorHtmlInspector" visible="false" />
  </modules>
  <tools name="Clipboard" tab="Home">
```

```

    <tool name="PasteStrip" size="large"/>
    <tool name="Cut" size="medium"/>
    <tool name="Copy" size="medium" shortcut="CTRL+C"/>
    <tool name="Print" size="medium" shortcut="CTRL+P"/>
</tools>
<tools name="Font" tab="Home">
    <tool name="FontName" shortcut="CTRL+SHIFT+F"/>
    <tool name="RealFontSize" shortcut="CTRL+SHIFT+P" width="80px"/>
    <tool name="ConvertToLower" strip="FontDropDowns" />
    <tool name="ConvertToUpper" strip="FontDropDowns" />
    <tool name="Bold" strip="FontBasicTools" shortcut="CTRL+B"/>
    <tool name="Italic" strip="FontBasicTools" shortcut="CTRL+I"/>
    <tool name="Underline" strip="FontBasicTools" shortcut="CTRL+U"/>
    <tool name="StrikeThrough" strip="FontBasicTools1"/>
    <tool name="Superscript" strip="FontBasicTools1"/>
    <tool name="Subscript" strip="FontBasicTools1"/>
    <tool name="FormatStripper" strip="FontDropDowns" />
    <tool name="ForeColor"/>
    <tool name="BackColor"/>
</tools>
<tools name="Paragraph" tab="Home">
    <tool name="InsertUnorderedList" strip="ListsAndIndentation"/>
    <tool name="InsertOrderedList" strip="ListsAndIndentation"/>
    <tool name="Indent" strip="ListsAndIndentation"/>
    <tool name="Outdent" strip="ListsAndIndentation"/>
    <tool name="InsertParagraph" strip="ListsAndIndentation"/>
    <tool name="JustifyLeft" strip="Align"/>
    <tool name="JustifyCenter" strip="Align"/>
    <tool name="JustifyRight" strip="Align"/>
    <tool name="JustifyFull" strip="Align"/>
    <tool name="JustifyNone" strip="Align"/>
</tools>
<tools name="Styles" tab="Home">
    <tool name="ApplyClass" />
    <tool name="FormatBlock" />
</tools>
<tools name="Editing" tab="Home">
    <tool name="Undo" shortcut="CTRL+Z"/>
    <tool name="FindAndReplace" shortcut="CTRL+F"/>
    <tool name="SelectAll" shortcut="CTRL+A"/>
    <tool name="Redo" shortcut="CTRL+Y"/>
</tools>
<tools name="Tables" tab="Insert">
    <tool name="InsertTableLight" size="large"/>
    <tool name="InsertTable" />
</tools>
<tools name="Media" tab="Insert">
    <tool name="ImageManager" size="large" shortcut="CTRL+G"/>
    <tool name="MediaManager" size="medium" />
    <tool name="FlashManager" size="medium"/>
    <tool name="SilverlightManager" size="medium"/>
</tools>
<tools name="Links" tab="Insert">
    <tool name="LinkManager" size="large" shortcut="CTRL+K"/>
    <tool name="Unlink" strip="LinksStrip" shortcut="CTRL+SHIFT+K"/>

```

```

    <tool name="DocumentManager" strip="LinksStrip"/>
    <tool name="ImageMapDialog" strip="LinksStrip"/>
    <tool name="InsertCustomLink" shortcut="CTRL+ALT+K"/>
</tools>
<tools name="Content" tab="Insert">
    <tool name="InsertSymbol"/>
    <tool name="InsertFormElement" />
    <tool name="InsertSnippet"/>
    <tool name="TemplateManager" />
    <tool name="FormatCodeBlock" />
    <tool name="InsertGroupbox" />
    <tool name="InsertHorizontalRule" />
    <tool name="InsertDate" />
    <tool name="InsertTime" />
    <tool name="FindAndReplace" shortcut="CTRL+F"/>
</tools>
<tools name="Zoom" tab="View">
    <tool name="Zoom"/>
</tools>
<tools name="Preferences" tab="View">
    <tool name="ToggleTableBorder" size="medium"/>
    <tool name="ToggleScreenMode" size="medium" shortcut="F11"/>
    <tool name="ModuleManager"/>
</tools>
<tools name="Proofing" tab="Review">
    <tool name="AjaxSpellCheck" size="large"/>
    <tool name="XhtmlValidator" size="large"/>
</tools>
<tools name="Help" tab="Help">
    <tool name="Help" size="medium"/>
    <tool name="AboutDialog" size="medium"/>
</tools>
</root>

```

20.9 Server-Side Programming

From the server you can set editor content or work with any of the editor collections, such as modules, tools, fonts, snippets, etc. For example you could add a link in the content when the page first loads, add some edit/insert tools and add a text "snippet" for the "InsertSnippet" tool.

[VB] Working with Content and Collections

```

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        ' Set the content visible in the editor when the editor first displays:
        RadEditor1.Content = "<a href='http://www.telerik.com' title='Telerik home
page'>Telerik</a>"
        ' Add the Node Inspector module
        Dim [module] As New EditorModule()
        [module].Name = "RadEditorNodeInspector"
        RadEditor1.Modules.Add([module])
        ' Add Tool groups
        Dim editGroup As New EditorToolGroup()
        editGroup.Tools.Add(New EditorTool("Cut"))
        editGroup.Tools.Add(New EditorTool("Copy"))
    End If
End Sub

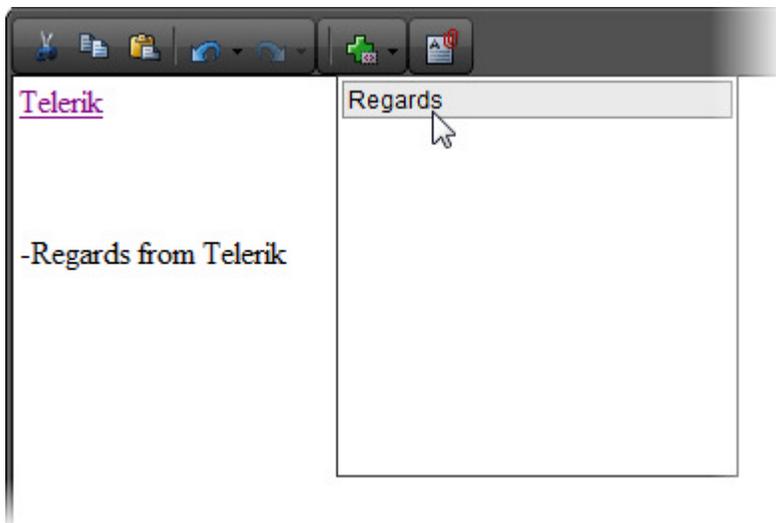
```

```
editGroup.Tools.Add(New EditorTool("Paste"))
editGroup.Tools.Add(New EditorSeparator())
editGroup.Tools.Add(New EditorTool("Undo"))
editGroup.Tools.Add(New EditorTool("Redo"))
RadEditor1.Tools.Add(editGroup)
Dim insertGroup As New EditorToolGroup()
insertGroup.Tools.Add(New EditorSeparator())
insertGroup.Tools.Add(New EditorTool("InsertSnippet"))
RadEditor1.Tools.Add(insertGroup)
Dim dialogGroup As New EditorToolGroup()
dialogGroup.Tools.Add(New EditorTool("DocumentManager"))
RadEditor1.Tools.Add(dialogGroup)
' Add a Snippet
RadEditor1.Snippets.Add(New EditorSnippet("Regards", "<br>-Regards from Telerik"))
End If
End Sub
```

[C#] Working with Content and Collections

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // Set the content visible in the editor when the editor first displays:
        RadEditor1.Content =
            "<a href='http://www.telerik.com (http://www.telerik.com/)' title='Telerik home page'>Telerik</a>";
        // Add the Node Inspector module
        EditorModule module = new EditorModule();
        module.Name = "RadEditorNodeInspector";
        RadEditor1.Modules.Add(module);
        // Add Tool groups
        EditorToolGroup editGroup = new EditorToolGroup();
        editGroup.Tools.Add(new EditorTool("Cut"));
        editGroup.Tools.Add(new EditorTool("Copy"));
        editGroup.Tools.Add(new EditorTool("Paste"));
        editGroup.Tools.Add(new EditorSeparator());
        editGroup.Tools.Add(new EditorTool("Undo"));
        editGroup.Tools.Add(new EditorTool("Redo"));
        RadEditor1.Tools.Add(editGroup);
        EditorToolGroup insertGroup = new EditorToolGroup();
        insertGroup.Tools.Add(new EditorSeparator());
        insertGroup.Tools.Add(new EditorTool("InsertSnippet"));
        RadEditor1.Tools.Add(insertGroup);
        EditorToolGroup dialogGroup = new EditorToolGroup();
        dialogGroup.Tools.Add(new EditorTool("DocumentManager"));
        RadEditor1.Tools.Add(dialogGroup);
        // Add a Snippet
        RadEditor1.Snippets.Add(new EditorSnippet("Regards", "<br>-Regards from Telerik"));
    }
}
```

After loading the page, the editor will look something like the screenshot below:



Events

FileDelete and FileUpload

If you have one of the file browser dialog managers configured so that files can be viewed, uploaded or deleted, the FileDelete and FileUpload events can fire. You can cancel the delete or uploaded based on the parameters passed to the event. "Sender" is the dialog object instance and can be used to know which dialog initiated the event, for example:

```
if (sender is Telerik.Web.UI.Editor.DialogControls.DocumentManagerDialog) { ... }
```

The "fileName" is the path of the file as it will be used by the editor. In the example below "MyDocuments" is a directory within the web application project, not the local drive being uploaded from.

If the user is uploading or deleting within a "Document Manager" dialog, then we process some special logic. The FileUpload event handler allows the upload only if the file doesn't already exist. The FileDelete event handler allows the file to be deleted only if the path ends with a ".sav" extension. If this is dialog other than the "Document Manager", always perform the upload or delete operation.

```

public bool RadEditor1_FileDelete(object sender, string fileName)
{
    if (sender is
        Telerik.Web.UI.Editor.DialogControls.DocumentManagerDialog)
    {
        return fileName.EndsWith(".sav");
    }
    else
        return true;
}

public bool RadEditor1_FileUpload(object sender, string fileName)
{
    if (sender is
        Telerik.Web.UI.Editor.DialogControls.DocumentManagerDialog)
    {
        return !(File.Exists(fileName));
    }
    else
        return true;
}

```

20.10 Client-Side Programming

Getting Client Object References

As with all the RadControls, you can get a client reference to the editor using the `$find()` method:

```
var editor = $find("<%=RadEditor1.ClientID%>");
```

From there you can use `get_document()` get references to the editor document:

```
var document = editor.get_document();
```

Use the document object to execute browser commands. `execCommand()` has three parameters:

- The command name.
- An optional property that if true, displays a user interface (if the command supports one).
- An optional property for passing a value.

In the example below we call the "Bold" command with no user interface or values being passed. Any selected text will be bolded

[JavaScript] Using execCommand()

```

function ApplyBold() {
    // return a reference to RadEditor
    var editor = $find("<%=RadEditor1.ClientID%>");
    // get a reference to the editor's document
    var document = editor.get_document();
    // execute a document command
    document.execCommand("Bold", false, null);
}

```

Because these are actually browser commands surfaced by the RadEditor control, you can find information for **execCommand syntax** (<http://msdn.microsoft.com/en-us/library/ms536419.aspx>) and **Command Identifiers** (<http://msdn.microsoft.com/en-us/library/ms533049.aspx>) on MSDN and also see **Rich Text Editing in Mozilla** (http://developer.mozilla.org/en/Rich-Text_Editing_in_Mozilla) for other browsers.

You can also get a reference to a selection. Use the selection object to get the selected text or HTML. You can also get the document *element* (an object that represents the HTML tag that contains the selection). The example below gets a reference to the selection, then gets a reference to the tag that the selection is contained within. If the tag is a link, the tag's font size is changed. If the tag is an image it is decorated with a grooved border.



You can find the complete source for this project at:
 \VS Projects\Editor\ClientSide

Responding to Client Events

You can respond to the **OnClientInit** event when the editor is first initialized but hasn't been loaded. For example, the material within the Content tag will not be available here and won't show up until the **OnClientLoad** event that fires later. In the example below an event listener is hooked up to detect when the edit mode changes.

[JavaScript] Responding to the OnClientInit Event

```
function ClientInit(sender, args) {
    sender.add_modeChange(modeChange);
}
function modeChange(sender, args) {
    alert('Mode Changed');
}
```

OnClientInit may be too early to access some properties and methods. For example, calling `attachEventHandler()` during **OnClientInit** will fail. Instead use the **OnClientLoad** event to access all the client properties and methods. All the content and other markup will be available at this point. In this next example an "onkeydown" event is hooked up to display each key press in the editor as it occurs.

[JavaScript] Responding to the OnClientLoad Event

```
function ClientLoad(sender, args) {
    sender.attachEventHandler("onkeydown", function(e) {
        alert("Content area key down " + e.keyCode);
    });
}
```

Earlier you saw how the selection object could be accessed on the client. The **OnClientSelectionChange** event lets you know when the user has selected an item, de-selected or moved the cursor.

You can also respond to commands before and after they are executed. **OnClientCommandExecuting** provides argument methods to get the command name, the tool that initiated the command and a value if the user selected something in a drop down list. The example below retrieves all three bits of information but only uses the command name.

The "args" passed in also make a `set_cancel()` method available so you can prevent the command from running. In the example below a confirmation dialog asks the user if they want to execute a particular command and prevents execution if the user cancels.

[JavaScript] Responding to the OnClientCommandExecuting Event

```
function ClientCommandExecuting(sender, args) {
    // The command name
```

```
var commandName = args.get_commandName();
// The tool that initiated the command
var tool = args.get_tool();
// The selected value [if command is coming from a dropdown]
var value = args.get_value();
// Perform some action
var answer =
    confirm("OnClientCommandExecuting \nExecute command " + commandName + "?");
// Cancel the command execution by calling args.set_cancel(true);
args.set_cancel(!answer);
}
```

The **OnClientCommandExecuted** event fires next after **OnClientCommandExecuting** if the command isn't canceled.



You can find the complete source for this project at:
\\VS Projects\\Editor\\ClientSide

Implementing Custom Buttons and Drop Downs

Implementing Custom Buttons

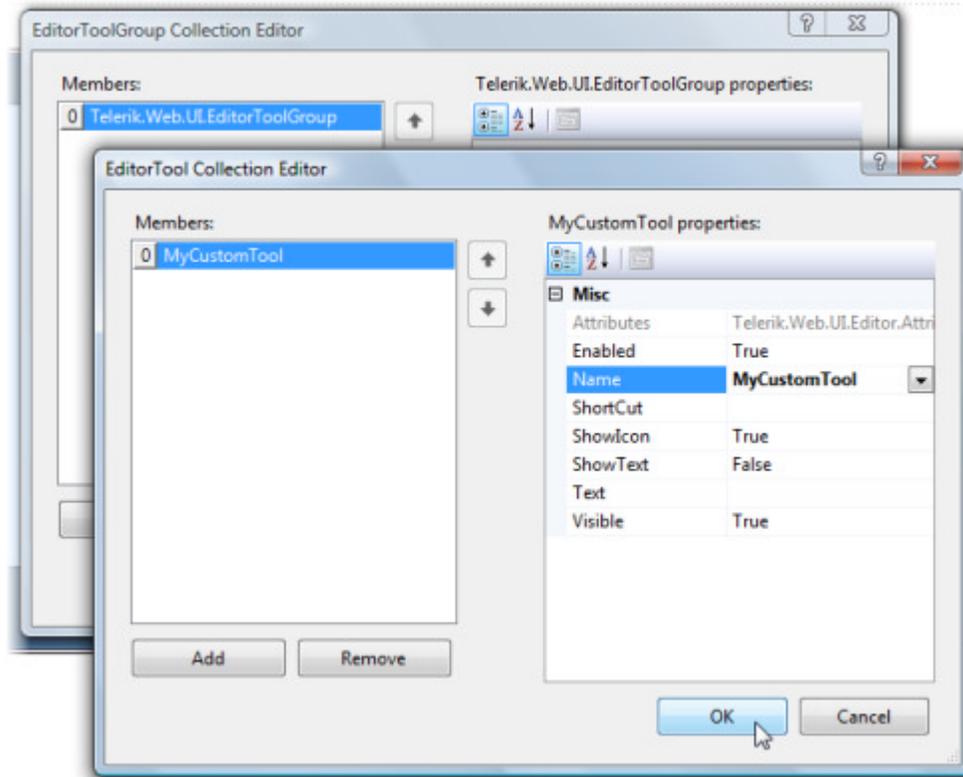
If the built-in tools don't serve your particular purposes you can add your own. Buttons and drop downs can be added in the same way that built-in tools are added, i.e. through the Tools file, declaratively or in server-side code.

Implementing a custom button is a three step process:

1. Define the custom button in the tools file, in the markup or programmatically.
2. Define a style that assigns an image to the button.
3. Add script that defines the command functionality.

Try this walk-through that demonstrates creating a custom button that adds a horizontal rule element.

1. Create a new web application, add a RadEditor, a ScriptManager and configure it (see "Getting Started" if you are unsure how to configure RadEditor).
2. Copy the image file "arrow.png" from \\VS Projects\\images to your project.
3. In the Properties window for the RadEditor, locate the **Tools** property and click the ellipses.
4. In the EditorToolGroup Collection Editor, click the **Add** button.
5. Locate the **Tools** property for the new EditorToolGroup and click the ellipses. This will open the EditorTool Collection Editor.
6. Click the **Add** button to create a new tool. Set the Name property of the new tool to "MyCustomTool".



- Click OK twice to close both collection editors.
- In the ASP.NET markup for the page, add the style below inside the <head> tag.

[ASP.NET] Adding the Custom Tool Button Image

```
<style type="text/css">
.rade_toolbar.Default .MyCustomTool
{
    background-image: url("arrow.png");
}
</style>
```



Gotcha! Be careful to keep the space between ".rade_toolbar.Default" and ".MyCustomTool". They are two different CSS selectors, not one long string.

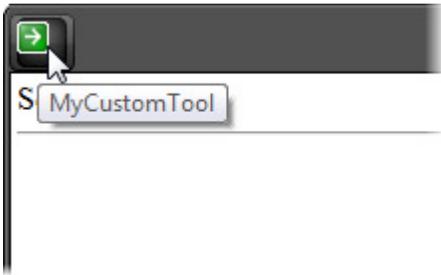
- Add the script below.

*This bit of script actually assigns the function to the editor's **CommandList** but does not run it. The **MyCustomTool** command runs when the button is clicked. The command executes a `pasteHtml()` method that in turn drops in a horizontal rule tag.*

[JavaScript] Define the Button Command Functionality

```
<script type="text/javascript">
Telerik.Web.UI.Editor.CommandList["MyCustomTool"] = function(commandName, editor, args)
    editor.pasteHtml("<hr>");
};
</script>
```

10. Press Ctl-F5 to run the application.



Implementing Custom Drop Downs

You can also add custom drop down lists. These can be defined in your tools file, the markup or you can create them programmatically. Programmatically works particularly well if you need to populate the list from a database.

Instead of defining a `CommandList` item as in the button example, you can also hook into the `OnClientCommandExecuting` event, do whatever logic you want and then cancel the command. You cancel the command because the command will not actually be defined.

Try extending the Custom Button example above:

1. Add a `OnClientCommandExecuting` function to be defined later. Also define a new "Salutations" tool within the Tools collection. You can do this directly in the ASP.NET markup or in the Tools collection editor. The "Salutations" drop down should have a `ItemsPerRow` attribute of "3" so that the three items "Mr.", "Mrs." and "Ms." all appear on one row.

[ASP.NET]

```
<telerik:RadEditor ID="RadEditor1" runat="server"
  OnClientCommandExecuting="ClientCommandExecuting">
  <Tools>
    <telerik:EditorToolGroup>
      <telerik:EditorTool Name="MyCustomTool" />
      <telerik:EditorSeparator />
      <telerik:EditorDropDown Name="Salutations" PopupHeight="30" ItemsPerRow="3">
        <telerik:EditorDropDownItem Name="Mr" Value="Mr." />
        <telerik:EditorDropDownItem Name="Mrs" Value="Mrs." />
        <telerik:EditorDropDownItem Name="Ms" Value="Ms." />
      </telerik:EditorDropDown>
    </telerik:EditorToolGroup>
  </Tools>
</Content>
</Content>
</telerik:RadEditor>
```

2. Add the following JavaScript inside the `<body>` tag.

[JavaScript] Responding to the Custom Drop Down

```
<script type="text/javascript">
function ClientCommandExecuting(sender, args) {
  var name = args.get_name();
  var value = args.get_value();
  if (name == "Salutations") {
    sender.pasteHtml("Dear&nbsp;" + value + "&nbsp;");
    //Cancel the further execution of the command --
    // does not exist in the editor command list
  }
}
```

```
        args.set_cancel(true);  
    }  
}  
</script>
```

3. Press **Ctrl-F5** to run the application.

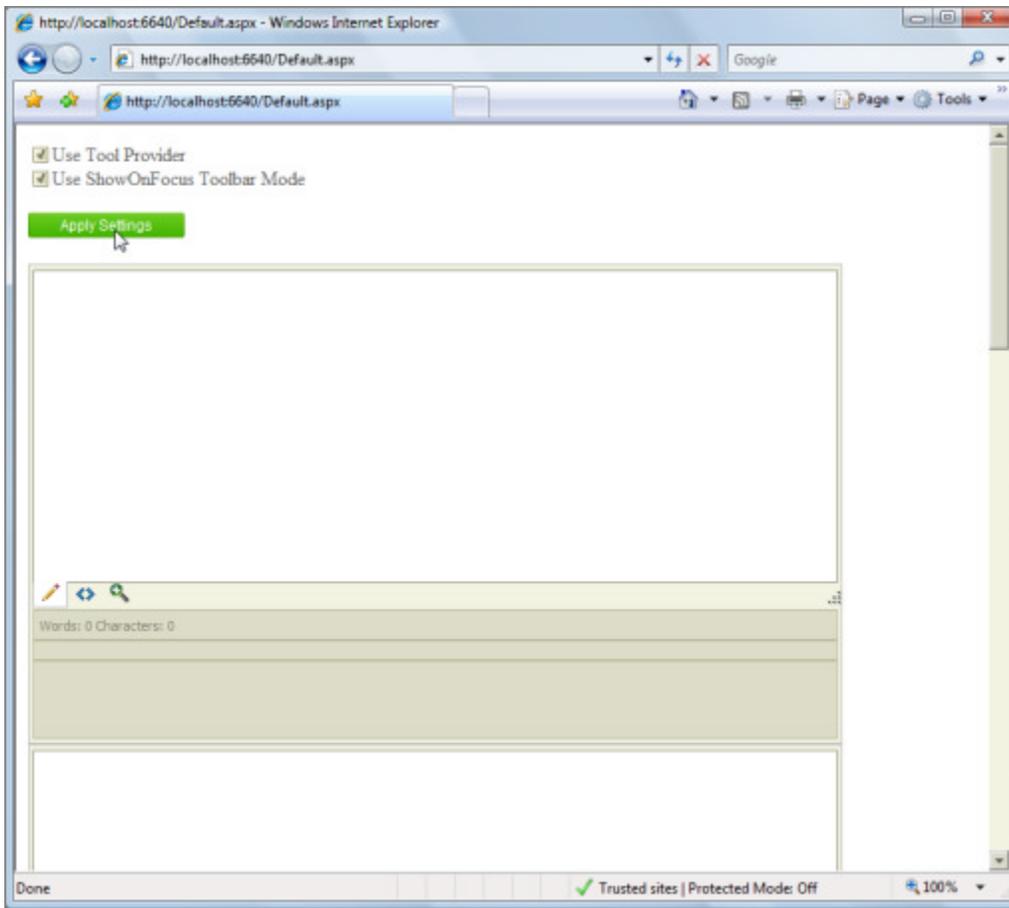


20.11 How To

Optimization for Multiple RadEditors

Starting in version 2008.1.619, RadEditor has substantial tools loading performance improvements. The big benefit comes when you have multiple RadEdit controls on the page. In the past there would be markup defined for each toolbar so that the more edit controls on the page, the longer it would take to load from the server and render in the browser. Now, thanks to a new property **ToolsProviderID**, the toolbar is defined for one editor and the other editors use only that toolbar. Use **ToolsProviderID** together with the **ToolBarMode ShowOnFocus** or **PageTop** to get even more performance improvement.

This next example should give you a gut feel for how these properties work together by toggling them and resubmitting the page. When using the default **ToolBar Mode** and not supplying a **ToolsProviderID**, notice how even after the editors are loaded on the page, the toolbars still take extra time to finish loading.



1. Create a ASP.NET Web Application.
2. Add the following markup to define the checkboxes and submit button.

[ASP.NET] Adding Checkboxes and Submit Button

```
<ul style="padding: 0; margin: 0; list-style: none;">
  <li>
    <asp:CheckBox ID="UseToolProvider" runat="server" Checked="false"
      Text="Use Tool Provider" Title="Use Tool Provider" /></li>
  <li>
    <asp:CheckBox ID="UseShowOnFocusToolbarMode" Checked="false"
      Text="Use ShowOnFocus Toolbar Mode" runat="server"
      Title="Use ShowOnFocus Toolbar Mode" /></li>
</ul>
<p>
<asp:Button ID="SubmitButton" runat="server" Text="Apply Settings"
  title="Apply Settings" />
</p>
```

3. Add five RadEditor controls below the submit button.
4. In the code-behind, add a Form_Load event handler:

[VB] Configuring the Editor Controls

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  ' track the first editor
  Dim firstEditor As RadEditor = Nothing
End Sub
```

```

' iterate the form looking for editors
For Each control As Control In Me.Form.Controls
If TypeOf control Is RadEditor Then
Dim editor As RadEditor = TryCast(control, RadEditor)
' if we're using ToolProviderID, either
' save the first editor found
' or assign the first editor's ID to the
' ToolProviderID.
If Me.UseToolProvider.Checked Then
If [Object].Equals(firstEditor, Nothing) Then
firstEditor = editor
Else
editor.ToolProviderID = firstEditor.ID
End If
Else
editor.ToolProviderID = [String].Empty
End If
' Use the ShowOnFocus toolbar mode if the checkbox is selected.
editor.ToolbarMode = IIf
(Me.UseShowOnFocusToolbarMode.Checked, EditorToolbarMode.ShowOnFocus, EditorToolbarMode.
[Default])
End If
Next
End Sub

```

[C#] Configuring the Editor Controls

```

protected void Page_Load(object sender, EventArgs e)
{
// track the first editor
RadEditor firstEditor = null;
// iterate the form looking for editors
foreach (Control control in this.Form.Controls)
{
if (control is RadEditor)
{
RadEditor editor = control as RadEditor;
// if we're using ToolProviderID, either
// save the first editor found
// or assign the first editor's ID to the
// ToolProviderID.
if (this.UseToolProvider.Checked)
{
if (Object.Equals(firstEditor, null))
{
firstEditor = editor;
}
else
{
editor.ToolProviderID = firstEditor.ID;
}
}
else
editor.ToolProviderID = String.Empty;
// Use the ShowOnFocus toolbar mode if the checkbox is selected.
editor.ToolbarMode = this.UseShowOnFocusToolbarMode.Checked ?

```

UI for ASP.NET AJAX

```
        EditorToolBarMode.ShowOnFocus : EditorToolBarMode.Default;  
    }  
}
```

5. Press **Ctrl-F5** to run the application. Try all combinations of these two settings. Look at the HTML source and find "rade_toolbarWrapper", a style that occurs just before the toolbars are defined. With ToolProviderID, the toolbar is defined just once, with the ToolProviderID blank, the toolbar is defined multiple times.



You can find the complete source for this project at:

\\VS Projects\Editor\ToolProviderID

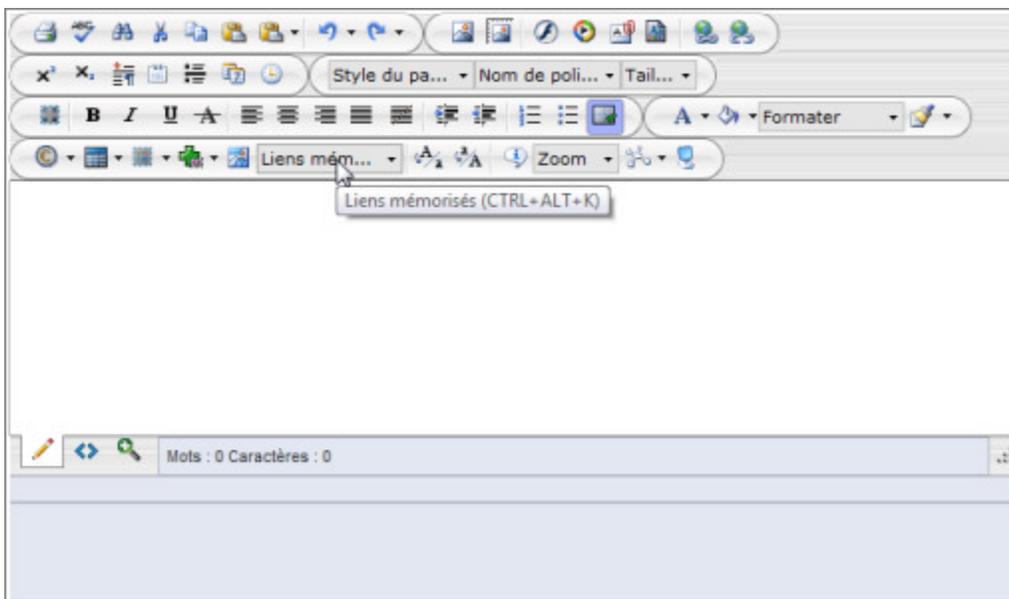
- ☞ You can also find this project with additional JavaScript code that provides specific timings at the **ToolProviderID demo** (<http://demos.telerik.com/ASPNET/Prometheus/Editor/Examples/ToolProvider/DefaultCS.aspx>).

Localization

All the language for the editor user interface can be automatically translated to either a built-in language or a translation you provide. Localization not only takes care of the text for each UI element, but also supplies a translated tool tip. Localization can be specified...

- Using resource files that come with the installation found in "\\RadControls for ASPNET AJAX <version>\App_GlobalResources" or using your own resource files. The "built-in" resource files are translations for English, French and German that come with the RadControls installation.
- Using the **Localization** property to set specific strings for Dialogs, Main, Modules and Tools programmatically. Localization settings override the resource file settings.
- If you have a tools file defined, you can also call the RadEditor **FindTool()** method and set the text for a specific tool.

Here's an example of the editor localized to French:



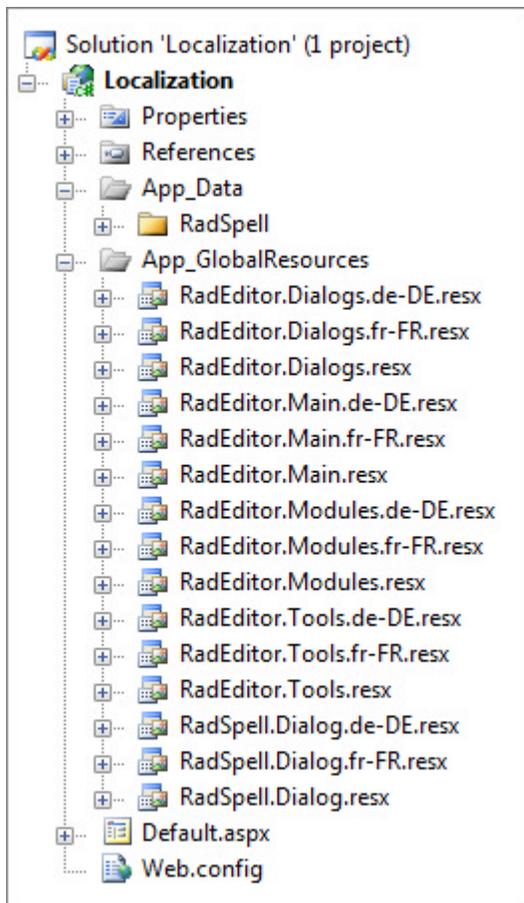
- ☞ The order of precedence for localization is generally that **Text** assigned at runtime (in the Page_Load for

example) is considered first, then in-line declaration and lastly the resource file strings are the default.

Localizing Using Resource Files

1. Create a new web application.
2. Add a RadEditor to the default page.
3. From the Smart Tag select **Add ScriptManager**.
4. Drag the \RadSpell directory from the \App_Data folder in your RadControls installation directory to the \App_Data folder in the Solution Explorer.
5. In the Solution Explorer, right-click the project and from the context menu select **Add | Add ASP.NET Folder | App_GlobalResources**.
6. From the RadControls installation directory, drag the contents of the \App_GlobalResources directory to the \App_GlobalResources directory in your selection.

Your project structure should look something like the example Solution Explorer screenshot shown below:



7. Set the RadEditor **Language** property to "Fr-fr"
8. Press **Ctrl-F5** to run the application. Notice that both the text and tool tips display in French.



You can find the complete source for this project at:
 \VS Projects\Localization\Localization

Creating New Translations

To create a new set of resource files, copy the default resource files to your App_GlobalResources directory and rename with a language code for the specific culture:

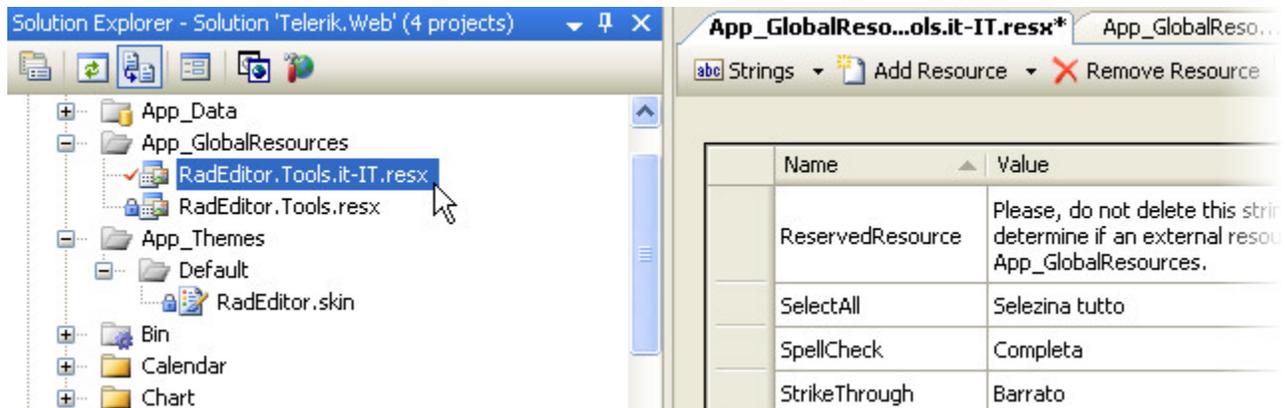
- RadEditor.Dialogs.<Your_Language>.resx
- RadEditor.Main.<Your_Language>.resx
- RadEditor.Modules.<Your_Language>.resx
- RadEditor.Tools.<Your_Language>.resx

So, for Italian, the resource files will be named:

- RadEditor.Dialogs.it-IT.resx
- RadEditor.Main.it-IT.resx
- RadEditor.Modules.it-IT.resx
- RadEditor.Tools.it-IT.resx

✍ The culture name has to follow the RFC 1766 standard in the format [Language Code]-[County/Region Code]. In our example, it-IT stands for Italian - Italy.

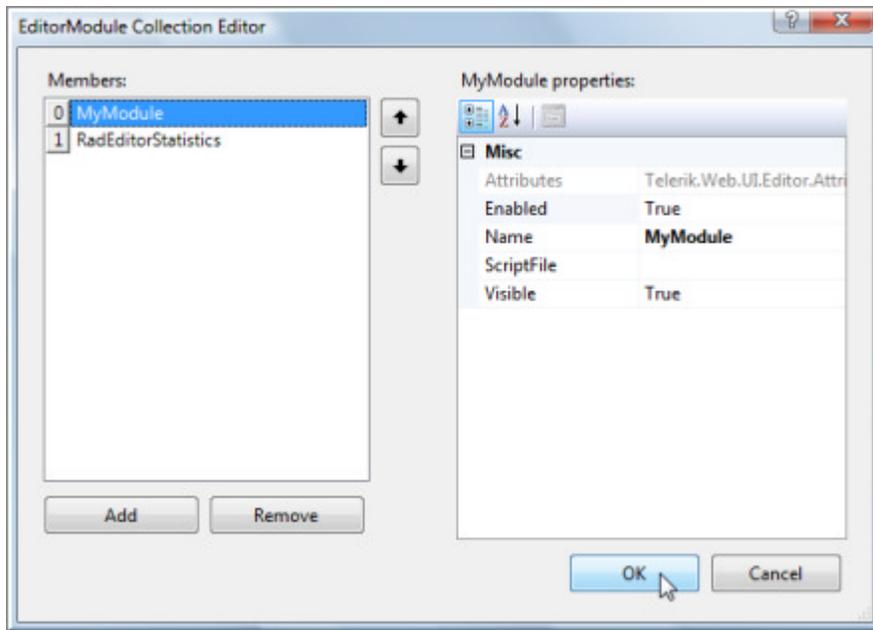
All resx file contain two columns **Name** and **Value** as shown in the screenshot below. Leave the Name as-is and change the Value for each of these to the target language.



Creating a Custom Module

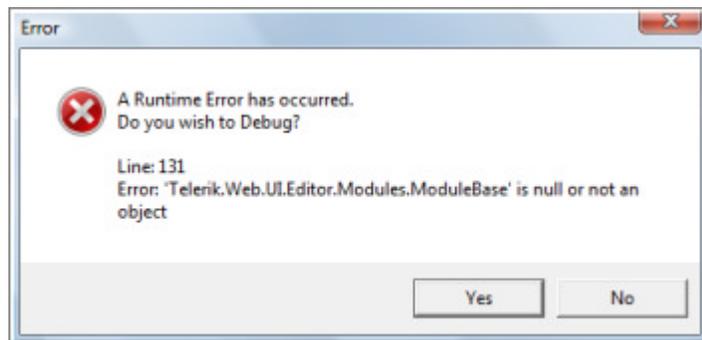
To build a custom module you need to define a JavaScript "class" that inherits from Telerik.Web.UI.Editor.Modules.ModuleBase. That class needs to define a method and hook the method up to one of the editors events, usually the SelectionChange event. We can create a minimal custom module by first doing the setup work, i.e. create a new web application with a RadEditor, adding the HTTP handlers, the ScriptManager and the finally adding the dictionary files to the App_data folder. Then...

1. In the Properties window for the RadEditor, click the **Modules** property ellipses and add two modules. In the first module enter the **Name** property as "MyModule". For the second Name property, select RadEditorStatistics (or any other predefined module) from the drop down list.



Gotcha!

You must add at least one of the built-in modules when you create a custom module. Due to optimization, the editor will not register the custom modules javascript code if a built-in module is not declared. When the JavaScript tries to register your custom class it will fail with an error that ModuleBase was null.



2. Add a JavaScript class with the same name as the custom module right below the RadEditor markup.

The constructor takes a single parameter "element". Within the constructor, call the ModuleBase initializeBase() method passing a reference to the module and the element. In the prototype, add two methods. The first, initialize(), extends an inherited method from ModuleBase. In this initialize() method, call the inherited method, then hook up the SelectionChange event of the editor to a method to be defined next called doSomething(). Also call doSomething() once explicitly to cause the module to display immediately.

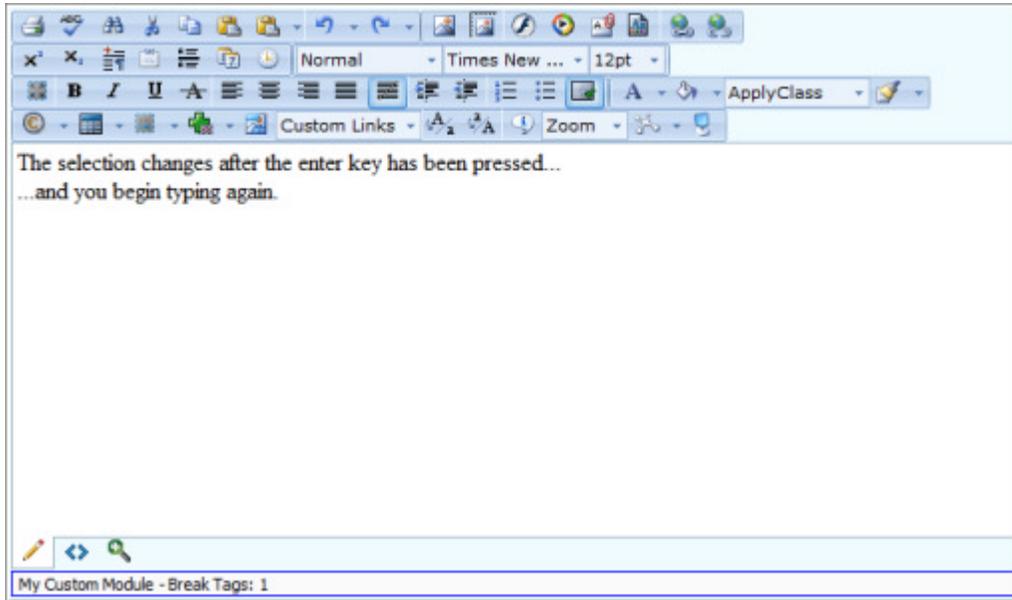
In the doSomething() method, create a SPAN element and load it with the contents of the RadEditor's html. Using the span getElementsByTagName() method, get a count of all the break "
" tags in the html. Finally display and format an informational method within the modules element that shows the number of breaks encountered so far.

[JavaScript] Adding the Custom Module Script

```
<script type="text/javascript">
```

```
// Create a new class with the same name as the custom module
MyModule = function(element) {
    // call the inherited initializeBase() method
    MyModule.initializeBase(this, [element]);
}
MyModule.prototype =
{
    // override the initialize() method
    initialize: function() {
        // call the inherited method
        MyModule.callBaseMethod(this, 'initialize');
        var selfPointer = this;
        // hook up to the SelectionChanged event to
        // call doSomething()
        this.get_editor().add_selectionChange(
            function() { selfPointer.doSomething(); });
        // call doSomething once explicitly
        this.doSomething();
    },
    doSomething: function() {
        // create a new "SPAN" element so we have access to its
        // getElementsByTagName() method.
        var span = document.createElement("SPAN");
        // get whatever html is available in the editor
        // when this is called.
        span.innerHTML = this.get_editor().get_html();
        // get a count of all the break tags
        var BrCount = span.getElementsByTagName("BR").length;
        // get the element for this module. Place the
        // information about the breaks there and format it
        var element = this.get_element();
        element.innerHTML = "My Custom Module - Break Tags: " + BrCount;
        element.style.border = "1px solid blue";
        element.style.color = "black";
    }
};
// register the class as being inherited from ModuleBase
MyModule.registerClass('MyModule', Telerik.Web.UI.Editor.Modules.ModuleBase);
</script>
```

3. Press Ctl-F5 to run the application. Notice that the custom module appears right away in the lower right of the editor. Try typing some text and hitting the enter key. Notice that the SelectionChange event fires when you begin typing on the next line, click away from what you're currently typing or otherwise move the selection.



Creating a Custom Content Filter

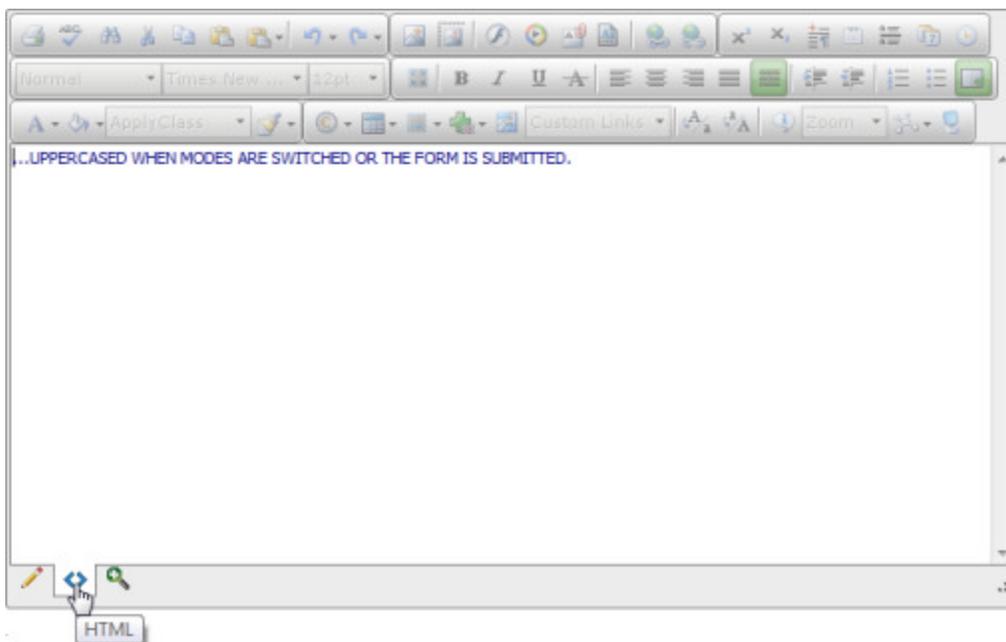
You can create your own custom content filter in a manner similar to creating the custom module. That is, you create a JavaScript object that inherits from a base type and implement certain expected methods. In this case the base type is `Telerik.Web.UI.Editor.Filter` and the methods are `getHtmlContent()` and `getDesignContent()`. Hookup the filter during the RadEditor `OnClientLoad` event handler:

[JavaScript] Implementing a Content Filter.

```
<telerik:RadEditor runat="server" ID="RadEditor2" OnClientLoad="OnClientLoad"
  Skin="Telerik">
  <Content>
  </Content>
</telerik:RadEditor>
<script type="text/javascript">
  // when the editor finishes loading,
  // hook up the new custom filter
  function OnClientLoad(editor, args) {
    editor.get_filtersManager().add(new MyFilter());
  }
  // in the constructor, set the
  // properties of the filter
  MyFilter = function() {
    MyFilter.initializeBase(this);
    this.set_isDom(false);
    this.set_enabled(true);
    this.set_name("RadEditor filter");
    this.set_description("RadEditor filter description");
  }
  // You must implement getHtmlContent and
  // getDesignContent.
  MyFilter.prototype =
  {
    getHtmlContent: function(content) {
      var newContent = content;
      newContent = newContent.toUpperCase();
    }
  }
</script>
```

```
        return newContent;
    },
    getDesignContent: function(content) {
        var newContent = content;
        newContent = newContent.toUpperCase();
        return newContent;
    }
}
MyFilter.registerClass('MyFilter', Telerik.Web.UI.Editor.Filter);
</script>
```

In the running application, all text content is uppercased when the editing mode changes as this screenshot shows:



If filtering were initiated during Design mode, then entering or pasting large content would be in danger of slowing down or crashing the browser. For best typing, formatting and editing performance, the editor's content filters act only when switching to Html mode or when submitting the content.



You can find the complete source for this project at:
\\VS Projects\\Editor\\ContentFilter

20.12 Summary

In this chapter we explored the RadEditor's rich feature set, learned how to configure RadEditor for the runtime environment and looked at the editor's design-time interface. You also configured the Toolsfile to create a custom tool layout. You learned how to manipulate RadEditor using client-side code including how to reference the editor, the document and the current selection, as well as responding to editor client events. Finally, you learned some of the editor's customization possibilities, how to optimize RadEditor for multiple instances and how to localize RadEditor for a specific language.

21 RadBarcode

21.1 Objectives

- Explore features of the RadBarcode control.
- Explore the most important properties of RadBarcode.
- Specifying RadBarcode types and their characteristics.

21.2 Introduction

The **RadBarcode** control can be used for automatic Barcode generation directly from a numeric or character data. It supports several standards that can be used when creating the image.

Properties

After a barcode is added on the page, it is necessary to adjust the basic properties of the barcode, and to specify its type, text and appearance.

- **Type** - Specifies the type of the standard used for encoding the text.
- **Text** - Text that will be encoded with Barcode and rendered in SVG file.
- **Width** - Use for specifying the width of the SVG file and the HTML Span element in which it is wrapped.
- **Height** - Use for specifying the height of the SVG file and the HTML Span element in which it is wrapped.
- **RenderChecksum** - Specifies if the checksum will be rendered in the barcode.
- **ShowChecksum** - Specifies if the checksum should be written under the barcode.
- **ShowText** - Specifies if the text will be shown under the barcode.
- **ShortLinesLengthPercentage** - Specifies the ration between long and short lines in the rendered barcode and adjust the Height and Width of the bars in percentage of the barcode's wrapper. Expects value varies from 0.00 to 100.00 (90 by default).



If the value in the **Text** property is invalid for the selected type, the Barcode will not be visible.

21.3 Barcode types

- **Codabar** - is a discrete, self-checking symbology that may encode 16 different characters, plus an additional 4 start/stop characters.
- **Code11** - is a barcode symbology which is discrete and is able to encode the numbers 0 through to 9, the dash symbol (-), and start/stop characters
- **Code128** - is a barcode symbology which encodes alphanumeric characters into a series of bars. It is of variable length, and accepts numbers, upper and lower case characters. It also includes an obligatory MOD 103 checksum. Code128 is divided into three subsets A, B, and C.
- **Code 25 Interleaved** - Interleaved 2 of 5 is a higher-density numeric symbology based upon the Standard 2 of 5 symbology. Interleaved 2 of 5 encodes any even number of numeric characters in the widths of the bars and spaces of the bar code. Unlike Standard 2 of 5, which only encodes information in the width of the bars, Interleaved 2 of 5 encodes data in the width of both the bars and spaces. This allows Interleaved 2 of 5 to achieve a somewhat higher density. The symbology is called "interleaved" because the first numeric data is encoded in the first 5 bars while the second numeric data is encoded in the first 5 spaces that separate the first 5 bars. Thus the first 5 bars and spaces actually encode two characters. This is also why the bar code can only encode an even number of data elements.

- **Code 25 Standard** - Standard 2 of 5 is a low-density numeric symbology. The spaces in the barcode exist only to separate the bars themselves. Additionally, a bar may either be wide or narrow, a wide bar generally being 3 times as wide as a narrow bar. The exact size of the spaces is not critical, but is generally the same width as a narrow bar.
- **Code39** - is a barcode symbology which encodes alphanumeric characters into a series of bars. It is of variable length and accepts uppercase letters, as well as numbers. It includes an optional Mod 43 checksum. **Code39Extended** is an extended version of code 39, which includes a bigger character set. If there is a requirement to use the Code39 barcode with characters other than numbers and uppercase alphabets, then this is the recommended barcode.
- **Code93** - was designed to complement and improve upon Code 39. Code 93 is similar in that it, like Code 39, can represent the full ASCII character set by using combinations of 2 characters. It differs in that Code 93 is a continuous symbology and produces denser code. It also encodes 47 characters compared to Code 39's 43 characters. **Code93Extended** is an extended version of code 93, which includes a bigger character set. Code93Extended can encode full 128 character ASCII using the four additional characters: (\$) (%) (/) (+).
- **EAN13** - is a barcode symbology which encodes numbers into a series of bars. It is of fixed length, of 13 digit (12 data and 1 check), and accepts numbers. First digit is always placed outside the symbol; additionally a right quiet zone indicator (>) is used to indicate Quiet Zones that are necessary for barcode scanners to work properly. It includes a checksum.
- **EAN8** - is a barcode symbology which encodes numbers into a series of bars. It is of fixed length, of 7 digits, and accepts numbers only. It includes a checksum.
- **MSI**(also known as Modified Plessey) - is a barcode symbology is a continuous, non-self-checking symbology. It is used primarily for inventory control, marking storage containers and shelves in warehouse environments. The length of this barcode type is variable.
- **Postnet**(Postal Numeric Encoding Technique) - is a barcode symbology which encodes numbers into a series of bars. It is of variable length and accepts numbers only. It includes a checksum.
- **UPCA** - is a barcode symbology, which consists of 12 digits, one of which is a checksum. This barcode identifies the manufacturer and specific product, so point-of-sale cash register systems can automatically look up the price.
- **UPCE** - is a variation of the UPCA symbol that is used for number system 0. By suppressing zeroes, UPCE codes can be printed in a very small space and are used for labeling small items.
- **UPC Supplement 2** - A two digit UPC supplementary code. This barcode should only be used with magazines, newspapers and other such periodicals.
- **UPC Supplement 5** - A five digit UPC supplementary code. This barcode is used on books to indicate a suggested retail price.

22 RadButton

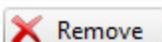
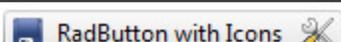
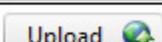
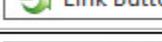
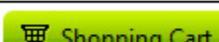
22.1 Objectives

- Explore features of the RadButton control.
- Getting Started.
- Use RadButton with external or embedded icons.
- RadButton as an Image Button.
- RadButton as a toggle button. Learn how to implement radios, checkboxes and a three state checkbox.
- Explore the most important properties of RadButton.
- Creating a single click button.
- Creating bigger icons and buttons.
- Confirm postback with RadButton.
- Specifying the content of a RadButton.

22.2 Introduction

The RadButton control provides access to the features of the ASP.NET Button, ImageButton, LinkButton, RadioButton, and CheckBox controls. The control can be easily styled by changing the **Skin** property, and alternatively setting properties that change the look of the control. This will eliminate the need to use the RadFormDecorator just to style a single button. Developers can easily migrate their applications from using the standard ASP.NET (button) controls to the new RadButton control, because most of their functionality is provided by our control and is controlled by the same or similar(intuitive) properties.

RadButton can be also configured to behave as a toggle button and rendered as a check box, a radio button or a completely customized toggle button with multiple states. You can make the control even more intuitive by placing an Icon right next to the text by choosing from the predefined icons or specifying your own.

	Standard RadButton
	Disabled Standard RadButton
	Standard RadButton with Primary Icon
	Standard RadButton with Primary and Secondary Icons
	Standard RadButton with Secondary Icon
	Standard Link button styled with RadButton Control
	Disabled Standard Link button
	Standard Link button with Primary Icon
	Standard Link button with Primary and Secondary Icons
	Standard Link button with Secondary Icon
	Image Button
	Image Button and text
	Image Button, text and Primary Icon
	Image Button, text with Primary and Secondary Icons
	Image Button, text and Secondary Icons
<input type="checkbox"/> ToggleButton Checkbox	ToggleButton styled as checkbox
<input type="checkbox"/> ToggleButton Checkbox	Disabled ToggleButton styled as checkbox
<input type="radio"/> ToggleButton RadioButton	ToggleButton styled as radio button
<input type="radio"/> ToggleButton RadioButton	Disabled ToggleButton styled as radio button

22.3 Getting Started

The following tutorial demonstrates how to set up a page with a **RadButton** control and attach its OnClick server event:

1. In the default page of a new ASP.NET AJAX-enabled Web Application add a RadButton control:

RadButtons' declarations

```
<telerik:RadButton ID="RadButton1" runat="server" Text="My Button"></telerik:RadButton>
```

The **Text** property specifies the text displayed in the RadButton control.

- To hook to the **OnClick** server-side event of RadButton switch to Design view of Visual Studio and double click on the button. This operation will insert the following function in the codebehind file:

C# Click Event Handler Function

```
protected void RadButton1_Click(object sender, EventArgs e)
{
}
```

VB.NET Click Event Handler Function

```
Protected Sub RadButton1_Click(sender As Object, e As EventArgs)
End Sub
```

as well as add **onclick="RadButton1_Click"** to the RadButton's declaration.

In the Click event handler add code that you want to be executed when the RadButton controls is clicked.

Here is more information about the different RadButton types and specific properties:

RadButton offers a special **ButtonType** property, which controls how the button is rendered on the client as a: **StandardButton** (default), **LinkButton** or **ToggleButton**.

- StandardButton:**

The control is rendered as `<input/>` of type="submit" or type="button". The **UseSubmitBehavior** (default value "true") property determines whether the `<input/>` type will be "submit" (when set to true) or "button" (when set to false). The user can disable the built-in styles and CSS of the button, and let the client browser apply its default styling for `<input type="submit|button" />` elements, by setting the **EnableBrowserButtonStyle** property to true.

StandardButton specific properties:

- UseSubmitBehavior** - gets or sets a bool value indicating whether the RadButton control uses the client browser's submit mechanism or the ASP.NET postback mechanism.
- EnableBrowserButtonStyle**

- LinkButton:**

The control is rendered as `<a/>` (anchor) element with child `` element used to specify the text. The purpose of this button type is to provide a "LinkButton" look of the control, and enable the user to specify URL to navigate to without requiring a page post back to the server. Target window or frame can be specified, in which the Web page content will be displayed, when the control is clicked, using the Target property.

LinkButton specific properties:

- NavigateUrl**
- Target**

- ToggleButton**

The control is rendered in the same way as the LinkButton; the difference is in the styles applied. The **ToggleButton** looks like a check box or radio button, depending on the value specified for the

ToggleType property. It can also look like a simple text (label) button [clickable text], if **ToggleType="None"** or **"CustomToggle"** is used.

This button type should be used in scenarios when richly styled check boxes or radio buttons are needed.

More information about the important properties of RadButton can be found in the **Important Properties (Section 22.7)** article.

22.4 Specifying RadButton Icons

You can make your button more intuitive by showing an icon or two on the left or right side of the control. All the Icon-related properties are controlled through the RadButton.Icon inner property. To display an icon on the button, the user needs to set either the Icon.PrimaryIconCssClass (SecondaryIconCssClass) property, or the Icon.PrimaryIconUrl (SecondaryIconUrl) property.

RadButtons with Icons

```
<telerik:RadButton ID="RadButton1" runat="server" Text="Shopping Cart">
  <Icon PrimaryIconUrl="~/img/Cart.png" PrimaryIconCssClass="classCart" />
</telerik:RadButton>
<telerik:RadButton ID="RadButton2" runat="server" Text="Standard Button With Two Icons">
<Icon PrimaryIconUrl="~/img/right_arrow.png" PrimaryIconTop="5px" PrimaryIconLeft="7px"
  SecondaryIconUrl="~/img/left_arrow.png" SecondaryIconTop="5px" SecondaryIconRight="7px"
 />
</telerik:RadButton>
```



RadButton provides an easy way to show different icon when the mouse is hovering over the control or the button is pressed. This is achieved through the **Icon.PrimaryHoveredIconUrl(SecondaryHoveredIconUrl)** and **Icon.PrimaryPressedIconUrl(SecondaryPressedIconUrl)**.

At first the icons might not be positioned the way we want, but this can be easily fixed by directly setting the properties that control the top, bottom, left or right edge of the respective icon. These are:

- PrimaryIconTop (SecondaryIconTop)
- PrimaryIconBottom (SecondaryIconBottom)
- PrimaryIconLeft (SecondaryIconLeft)
- PrimaryIconRight (SecondaryIconRight)

Additionally, a CSS class can be set to the icon, and the position configured using CSS.

1. Properties:

```
<telerik:RadButton ID="RadButton2" runat="server" Text="Shopping Cart">
  <Icon PrimaryIconUrl="~/img/Cart.png" PrimaryIconTop="4px" PrimaryIconLeft="5px"
  SecondaryIconUrl="~/img/Add.png" SecondaryIconTop="4px" SecondaryIconRight="5px">
</telerik:RadButton>
```

2. Or the same configuration using CSS classes:

```
<style type="text/css">
.classCart
{
    top: 4px !important;
    left: 5px !important;
}
.classAdd
{
    top: 4px !important;
    right: 5px !important;
}
</style>

<telerik:RadButton ID="RadButton3" runat="server" Text="Shopping Cart">
    <Icon PrimaryIconUrl="-/img/Cart.png" PrimaryIconCssClass="classCart"
    SecondaryIconUrl="-/img/Add.png" SecondaryIconCssClass="classAdd">
</telerik:RadButton>
```

To make the control even easier to use we offer a predefined set of built-in RadButton Icons. The developer needs to set the **PrimaryIconCssClass** or **SecondaryIconCssClass** property to the predefined CSS class, and the respective icon will be shown on the control. Some of the CSS classes include the following:

- rbOk
- rbCancel
- rbAdd
- rbRemove
- rbAttach

Button Icon and Class	Primary Icon	Secondary Icon
 rbAdd	 Add	Add 
 rbRemove	 Remove	Remove 
 rbOk	 OK	OK 
 rbCancel	 Cancel	Cancel 

The full list of the classes can be found on our **online demos** (<http://demos.telerik.com/aspnet-ajax/button/examples/embeddedicons/defaultcs.aspx>) site:

Note: The CssClass are composed in the following way: [r]ad[b]utton[Iconname] == rbAdd

22.5 RadButton as an Image Button

RadButton provides an easy way to show a custom image on the control. The image can be used as a background, or can represent the button itself (Image Button). When using the **RadButton** control as Image Button, the user must set **Width** and **Height**, because we don't use an actual `` tag, but the image is set as background to the Button's wrapper element (`<a/>`). All the Image-related properties are controlled through the **RadButton.Image** property.



RadButton used as ImageButton (the image represents the button)



RadButton with background image, icons and text.

There are two ways to display an image on the control:

1. The first and the easiest way is to set the **Image.ImageUrl** property to the location of the image used. Setting the **IsBackgroundImage** to true enables the developer to use the image as background, and set text and icons to his/her button.

ASPX

```
<telerik:RadButton ID="ImageButton1" runat="server" Width="37px" Height="36px"
Text="Download">
  <Image ImageUrl="~/img/cb_download.png" />
</telerik:RadButton>
```

2. The second way to set the image using **RadButton's CssClass** property. Basically we set the background-image in the **CssClass**, and enable the image button functionality by setting **Image.EnableImageButton=true**.

This approach is preferred when you want to use an image sprite for the button (see sample below). You set the background-image and background-position in the **CssClass**, and then in the **HoveredCssClass** and **PressedCssClass**, only the background-position of the hovered and pressed image.

If the user wants she/he can display a different image when the mouse is hovering over the control, or the button is pressed using the **HoveredImageUrl** and **PressedImageUrl** properties respectively.

Sample for p.2 (RadButton and Image Sprites):

CSS

```

<style type="text/css">
    .classImage
    {
        background: url(img/rbPredefinedIcons.png);
        background-position: 0 0;
        width: 16px;
        height: 16px;
    }
    .classHoveredImage
    {
        background-position: -24px 0;
    }
    .classPressedImage
    {
        background-position: -48px 0;
    }
</style>

```

ASPX

```

<telerik:RadButton ID="ImageButton2" runat="server" Text="Image Button"
CssClass="classImage"
    HoveredCssClass="classHoveredImage" PressedCssClass="classPressedImage">
    <Image EnableImageButton="true" />
</telerik:RadButton>

```



Note: It is always good to set the `Text` property, no matter if the control is used solely as image button (no text and icons shown), because this way the accessibility of the control is improved.

22.6 RadButton as a Toggle Button

The `RadButton` control can be easily configured to behave as a toggle button. Simply set the `ToggleType` property to a value different than `ButtonType.None`, and the button is transformed into a check box, a radio button or a completely customized toggle button. Since the `ButtonType` property controls how the component looks, the user can have his/her buttons look like standard buttons or even `<input type="submit|button" />` elements, and behave like check boxes or radio buttons. Here are some code samples showing how this is achieved:

CheckBoxes:

ASPX

```

<telerik:RadButton ID="btnToggle1" runat="server" Text="Checkbox 1" ToggleType="CheckBox"
ButtonType="StandardButton"></telerik:RadButton>
<telerik:RadButton ID="btnToggle2" runat="server" Text="Checkbox 2" ToggleType="CheckBox"

```

UI for ASP.NET AJAX

```
ButtonType="LinkButton"></telerik:RadButton>  
<telerik:RadButton ID="btnToggle3" runat="server" Text="Checkbox 3" ToggleType="CheckBox"  
ButtonType="ToggleButton"></telerik:RadButton>
```

ToggleType="CheckBox", ButtonType="StandardButton"	ToggleType="CheckBox", ButtonType="LinkButton"	ToggleType="Chec ButtonType="Togg
<input type="checkbox"/> Checkbox 1	<input type="checkbox"/> Checkbox 2	<input type="checkbox"/> Checkbox 3

Checked state:

ToggleType="CheckBox", ButtonType="StandardButton", Checked="true"	ToggleType="CheckBox", ButtonType="LinkButton", Checked="true"	ToggleType="Chec ButtonType="Togg
<input checked="" type="checkbox"/> Checkbox 1	<input checked="" type="checkbox"/> Checkbox 2	<input checked="" type="checkbox"/> Checkbox 3

Radios:

ASPX

```
<telerik:RadButton ID="btnToggle4" runat="server" Text="Radio Button 1" ToggleType="Radio"  
ButtonType="StandardButton"></telerik:RadButton>  
<telerik:RadButton ID="btnToggle5" runat="server" Text="Radio Button 2" ToggleType="Radio"  
ButtonType="LinkButton"></telerik:RadButton>  
<telerik:RadButton ID="btnToggle6" runat="server" Text="Radio BUtton 3" ToggleType="Radio"  
ButtonType="ToggleButton"></telerik:RadButton>
```

ToggleType="Radio", ButtonType="StandardButton"	ToggleType="Radio", ButtonType="LinkButton"	ToggleType="Radi
<input type="radio"/> Radio Button 1	<input type="radio"/> Radio Button 2	<input type="radio"/> Radio BUtton 3

Checked state:

ToggleType="Radio", ButtonType="StandardButton", Checked="true"	ToggleType="Radio", ButtonType="LinkButton", Checked="true"	ToggleType="Radi Checked="true"
<input checked="" type="radio"/> Radio Button 1	<input checked="" type="radio"/> Radio Button 2	<input type="radio"/> Radio BUtton 3

If a **Radio ToggleType** mode is chosen, the developer could also set the **GroupName** property, which specifies the name of the group that the radio button belongs to. Use this property to specify a grouping of radio buttons to create a mutually exclusive set of controls.

CustomToggle buttons:

ASPX

```

<telerik:RadButton ID="btnToggle7" runat="server" ToggleType="CustomToggle"
ButtonType="StandardButton">
<ToggleStates>
  <telerik:RadButtonToggleState Text="Unchecked" PrimaryIconCssClass="rbToggleCheckbox" />
  <telerik:RadButtonToggleState Text="Filled" PrimaryIconCssClass="rbToggleCheckboxFilled" />
  <telerik:RadButtonToggleState Text="Checked"
PrimaryIconCssClass="rbToggleCheckboxChecked" />
</ToggleStates>
</telerik:RadButton>
<telerik:RadButton ID="btnToggle8" runat="server" ToggleType="CustomToggle"
ButtonType="LinkButton">
<ToggleStates>
  <telerik:RadButtonToggleState Text="Unchecked" PrimaryIconCssClass="rbToggleCheckbox" />
  <telerik:RadButtonToggleState Text="Filled" PrimaryIconCssClass="rbToggleCheckboxFilled" />
  <telerik:RadButtonToggleState Text="Checked"
PrimaryIconCssClass="rbToggleCheckboxChecked" />
</ToggleStates>
</telerik:RadButton>
<telerik:RadButton ID="btnToggle9" runat="server" ToggleType="CustomToggle"
ButtonType="ToggleButton">
<ToggleStates>
  <telerik:RadButtonToggleState Text="Unchecked" PrimaryIconCssClass="rbToggleCheckbox" />
  <telerik:RadButtonToggleState Text="Filled" PrimaryIconCssClass="rbToggleCheckboxFilled" />
  <telerik:RadButtonToggleState Text="Checked"
PrimaryIconCssClass="rbToggleCheckboxChecked" />
</ToggleStates>
</telerik:RadButton>

```

Three State CheckBox Button: ToggleType= "CustomToggle" ButtonType= "StandardButton" PrimaryIconCssClass= "rbToggleCheckbox" PrimaryIconCssClass= "rbToggleCheckboxFilled" PrimaryIconCssClass= "rbToggleCheckboxChecked"	Three State CheckBox LinkButton: ToggleType= "CustomToggle" ButtonType= "LinkButton" PrimaryIconCssClass= "rbToggleCheckbox" PrimaryIconCssClass= "rbToggleCheckboxFilled" PrimaryIconCssClass= "rbToggleCheckboxChecked"	Three State CheckBox ToggleType= "Cust ButtonType= "Togg PrimaryIconCssCla: PrimaryIconCssCla: PrimaryIconCssCla:
<input type="checkbox"/> UnChecked	<input type="checkbox"/> UnChecked	<input type="checkbox"/> UnChecked

Filled State:

Three State CheckBox Button: ToggleType= "CustomToggle", ButtonType= "StandardButton" PrimaryIconCssClass= "rbToggleCheckbox" PrimaryIconCssClass= "rbToggleCheckboxFilled", Selected= "true" PrimaryIconCssClass= "rbToggleCheckboxChecked"	Three State CheckBox LinkButton: ToggleType= "CustomToggle" ButtonType= "LinkButton" PrimaryIconCssClass= "rbToggleCheckbox" PrimaryIconCssClass= "rbToggleCheckboxFilled", Selected= "true" PrimaryIconCssClass= "rbToggleCheckboxChecked"	Three State CheckBox ToggleType= "Cust ButtonType= "Togg PrimaryIconCssCla: PrimaryIconCssCla: Selected= "true" PrimaryIconCssCla:
<input checked="" type="checkbox"/> Filled	<input checked="" type="checkbox"/> Filled	<input checked="" type="checkbox"/> Filled

Checked State:

<p>Three State CheckBox Button: ToggleType="CustomToggle", ButtonType="StandardButton" PrimaryIconCssClass="rbToggleCheckbox" PrimaryIconCssClass="rbToggleCheckboxFilled" PrimaryIconCssClass="rbToggleCheckboxChecked", Selected="true"</p>	<p>Three State CheckBox LinkButton: ToggleType="CustomToggle" ButtonType="LinkButton" PrimaryIconCssClass="rbToggleCheckbox" PrimaryIconCssClass="rbToggleCheckboxFilled" PrimaryIconCssClass="rbToggleCheckboxChecked", Selected="true"</p>	<p>Three State Check ToggleType="Cus ButtonType="Tog PrimaryIconCssCl PrimaryIconCssCl PrimaryIconCssCl Selected="true"</p>
<input checked="" type="checkbox"/> Checked	<input checked="" type="checkbox"/> Checked	<input checked="" type="checkbox"/> Checked

The user is free to specify as many toggle states as needed, and can completely change the look of the control using the different **RadButtonToggleState** properties. In the code above, the **PrimaryIconCssClass** property is used to specify a three-state (3-state) checkbox and the **Text** property to have different text depending on the currently selected state.

To take a closer look at RadButton's "toggle button" functionality please visit our [online demos](http://demos.telerik.com/aspnet-ajax/button/examples/togglebutton/defaultcs.aspx) (<http://demos.telerik.com/aspnet-ajax/button/examples/togglebutton/defaultcs.aspx>).

22.7 Important Properties

The most important properties of the RadButton control are presented below:

Common properties:

- **Text** - specifies the text displayed in the RadButton control.
- **AutoPostBack** - specifies a bool value indicating whether the control will automatically post the page back to the server.
- **CausesValidation** - specifies a bool value indicating whether validation is performed when the RadButton is clicked.
- **ButtonType** - specifies the type of the button. The following types exist:
 - StandardButton (default)
 - LinkButton
 - ToggleButton
Each ButtonType provides certain functionality that is unique. More information on the features of different button types can be found in each button category.
- **CommandName** - specifies the group of controls for which the RadButton control causes validation when it posts back to the server.

Inner <Icon> tag specific properties:

- **PrimaryIconUrl** - specifies the URL to the image used as Primary Icon.
- **PrimaryIconCssClass** - specifies the CSS class applied to the Primary icon.
- **SecondaryIconUrl** - specifies the URL to the image used as Secondary Icon.
- **SecondaryIconCssClass** - specifies the CSS class applied to the Secondary icon.

Inner <Image> tag specific properties:

- **IsBackgroundImage** - specifies a bool value indicating how the image is used - i.e. as a background image

or as an Image Button.

- **ImageUrl** - specifies the URL to the image used as button.
- **HoveredImageUrl** - specifies the URL to the image showed when the RadButton is hovered.
- **PressedImageUrl** - specifies the URL to the image showed when the RadButton is pressed.
- **EnableImageButton** - specifies a bool value indicating whether the RadButton is rendered as Image Button.

SplitButton specific properties:

- **EnableSplitButton** - specifies a bool value that indicates whether the SplitButton functionality will be enabled
- **SplitButtonPosition** - specifies the position where the SplitButton will appear relative to the main button (Left or Right). Position:
 - Right (default)
 - Left
- **SplitButtonCssClass** - specifies the CSS class applied to the SplitButton

Type:Button specific properties:

- **UseSubmitBehavior** - gets or sets a bool value indicating whether the RadButton control uses the client browser's submit mechanism or the ASP.NET postback mechanism.

Type:LinkButton specific properties:

- **NavigateUrl** - specifies the URL of the page to navigate to, without posting the page back to the server. When this property is sets, the button is rendered as an <a/> (anchor) element.
- **Target** - specifies the target window or frame in which to display the Web page content linked to when the RadButton control is clicked.

Type: ToggleButton specific properties:

- **ToggleType** - specifies the type of the Toggle Button. There are three toggle types:
 - None (default)
 - Radio
 - CheckBox
 - CustomToggle
- **GroupName** - Valid when ToggleType: Radio. Gets or sets the name of the group that the radio button belongs to.
- **Checked** - specifies a bool value indicating whether the RadButton control is checked. When the ToggleButton has more than two states, the control is not checked if the current state of the RadButton is the First state. Otherwise, it is Checked.
- **Direction** - specifies the direction in which the states will be switched, when more than two ToggleStates are specified. Directions:
 - Standard (default)
 - Reversed

- **SelectedToggleState** - specifies the current state of the RadButton.
- **SelectedToggleStateIndex** - specifies the index of the currently selected ToggleState of the RadButton control, when used as a custom toggle button.
- **ToggleStates** - Collection of **RadButtonToggleState**. The different states are controlled through a collection of states. The collection can contain maximum of four states. The order of switching the states is determined by the 0-based position index at which the state occurs in the collection. So, the first item in the ToggleStates is the first state, the second item is the second state, and so on. When the ToggleType is Radio or CheckBox, the first item (state) of the ToggleStates is used as the alternate state of the RadButton.

RadButtonToggleState specific properties:

- **Text** - specifies the text displayed in the RadButton control.
- **Height - Selected-** Gets or sets a bool value indicating whether the ToggleState is selected or not.
CssClass-

22.8 Creating a single click button

The single click button is used to avoid multiple postbacks/callbacks to the server. This feature is useful in database and/or e-mail send scenarios when the developer should prevent submitting of identical content multiple times to the server

The example below demonstrates how to disable RadButton when clicked and change the button's text:

Default.aspx

```
<script type="text/javascript">
function OnClientClicked(sender, eventArgs) {
    //disable the button
    sender.set_enabled(false);
    //update the button text
    if (sender.get_text() == "Submit") sender.set_text("Submitting...");
}
function pageLoad(sender, eventArgs) {
    //Set the initial button's text and enable it
    var btnStandard = $find("<%=btnStandard.ClientID%>");
    btnStandard.set_text("Submit");
    btnStandard.set_enabled(true);
}
</script>
<telerik:RadButton ID="btnStandard" runat="server" Text="Submit" OnClick="btnStandard_Click"
UseSubmitBehavior="false" OnClientClicked="OnClientClicked" Style="clear: both;
float: left; margin: 10px 0;">
</telerik:RadButton>
<asp:Label ID="lblText" runat="server"></asp:Label>
```

RadButton Click Server-side handler C#

```
protected void btnStandard_Click(object sender, EventArgs e)
{
    lblText.Text = System.DateTime.Now.ToString();
    if (Page.IsPostBack) System.Threading.Thread.Sleep(3000);
}
```

```
}

```

RadButton Click Server-side handler VB.NET

```
Protected Sub btnStandard_Click(sender As Object, e As EventArgs)
    lblText.Text = System.DateTime.Now.ToString()
    If Page.IsPostBack Then
        System.Threading.Thread.Sleep(3000)
    End If
End Sub

```

Note: The **disabled="disabled"** attribute is applied to the control's HTML element, when it is disabled. This causes the client browser to not submit the page correctly (i.e. the values of the input fields are not submitted), and as a result RadButton's server-side events are not fired. In some browsers (IE6,7 and 8) the page is not submitted at all. That's why the ASP.NET postback mechanism should be used to submit the page. This is achieved by setting **UseSubmitBehavior="false"** [UseSubmitBehavior - Gets or sets a value indicating whether the RadButton control uses the client browser's submit mechanism or the ASP.NET postback mechanism].

22.9 Bigger Icons and Buttons

The StandardButton (RadButton with **ButtonType="StandardButton"**) has a fixed height which by default is 22px.

In Q1, 2011 we introduced a way to have a StandardButton with a height of 65px. This allows you to use bigger icons (24x24 pixels), and to place the content (icons and text) horizontally or vertically, e.g

Setting Bigger Height

```
<telerik:RadButton ID="RadButton1" runat="server" Text="Standard
Button" Height="65px"></telerik:RadButton>

```

The predefined icons set for RadButton also offer a bigger [icon for every existing one](http://demos.telerik.com/aspnet-ajax/button/examples/embeddedicons/defaultcs.aspx) (http://demos.telerik.com/aspnet-ajax/button/examples/embeddedicons/defaultcs.aspx). The following code will render the "Add" icon with greater dimensions:

Setting Bigger Icon and Button Height

```
<telerik:RadButton ID="RadButton3" runat="server" Text="Add" Height="65px"
AutoPostBack="false"
    Font-Size="18px">
    <Icon PrimaryIconCssClass="rbAdd24" PrimaryIconLeft="8" PrimaryIconTop="20" />
</telerik:RadButton>

```

Sometimes it is also necessary to create button with a Custom height. The following help article provides guidance how to implement buttons with custom height:

RadButton Custom Height Tutorial (<http://www.telerik.com/help/aspnet-ajax/button-custom-height.html>)

22.10 Confirm postback with RadButton

This example shows different ways to confirm the submission of the page to the server when using RadButton control:

RadButton with browser's `window.confirm`:

Browser's `window.confirm`

```
<script type="text/javascript">
  function StandardConfirm(sender, args)
  {
    args.set_cancel(!window.confirm("Are you sure you want to submit the page?"));
  }
</script>
<telerik:RadButton ID="btnStandardConfirm" runat="server" Text="Standard window.confirm"
OnClientClicking="StandardConfirm">
</telerik:RadButton>
```

RadButton with `radconfirm` (requires `RadWindowManager` on the page):

RadButton with `radconfirm`

```
<script type="text/javascript">
function RadConfirm(sender, args)
{
  var callBackFunction = Function.createDelegate(sender, function (shouldSubmit)
  {
    if (shouldSubmit)
    {
      this.click();
    }
  });

  var text = "Are you sure you want to submit the page?";
  radconfirm(text, callBackFunction, 300, 100, null, "RadConfirm");
  args.set_cancel(true);
}
</script>
<telerik:RadButton ID="btnRadConfirm" runat="server" Text="RadConfirm"
OnClientClicking="RadConfirm">
</telerik:RadButton>
<telerik:RadWindowManager ID="windowManager1" runat="server">
</telerik:RadWindowManager>
```

RadButton with `RadWindow`. Using `RadWindow` as the confirmation window gives the developer the ability to fully customize the look and feel of the dialog.

RadButton with `RadWindow`

```
<script type="text/javascript">
function CustomRadWindowConfirm(sender, args)
{
  $find("<%=confirmWindow.ClientID %>").show();
  $find("<%=btnYes.ClientID %>").focus();
  args.set_cancel(true);
}
function YesOrNoClicked(sender, args)
```

```

{
    var oWnd = $find("<%=confirmWindow.ClientID %>");
    oWnd.close();
    if (sender.get_text() == "Yes")
    {
        $find("<%=btnCustomRadWindowConfirm.ClientID %>").click();
    }
}
</script>
<telerik:RadButton ID="btnCustomRadWindowConfirm" runat="server" Text="Confirm"
    OnClientClicking="CustomRadWindowConfirm">
</telerik:RadButton>
<telerik:RadWindow ID="confirmWindow" runat="server" VisibleTitlebar="false"
VisibleStatusBar="false"
    Modal="true" Behaviors="None" Height="150px" Width="300px">
<ContentTemplate>
<asp:Label ID="lblConfirm" Text="Are you sure you want to submit the page?"
runat="server" />
    <br />
    <telerik:RadButton ID="btnYes" runat="server" Text="Yes" AutoPostBack="false"
OnClientClicked="YesOrNoClicked">
        <Icon PrimaryIconCssClass="rbOk" />
    </telerik:RadButton>
    <telerik:RadButton ID="btnNo" runat="server" Text="No" AutoPostBack="false"
OnClientClicked="YesOrNoClicked">
        <Icon PrimaryIconCssClass="rbCancel" />
    </telerik:RadButton>
</ContentTemplate>
</telerik:RadWindow>

```

22.11 Specifying the content of a RadButton

It is now possible to define the appearance of a RadButton control by adding ASP.NET controls and HTML elements in its content. There are a few ways to achieve this:

- Set the **ContentTemplate** property:
 - In the markup of your page you can add controls to the **ContentTemplate** inner property as shown below:

ASPX

```

<telerik:RadButton runat="server" ID="RadButton1" Width="90"
    Height="90">
    <ContentTemplate>
        <span>RadButton Content</span>
    </ContentTemplate>
</telerik:RadButton>

```

- In code-behind you can set the **ContentTemplate** property with an instance of a class that implements the **ITemplate** interface:

C#

```

protected void Page_Load(object sender, EventArgs e)

```

```
{
    CustomRadButton.ContentTemplate = new ButtonContentTemplate();
}
public class ButtonContentTemplate : ITemplate
{
    void ITemplate.InstantiateIn(Control container)
    {
        Label contentLabel = new Label();
        contentLabel.ID = "contentLabel";
        contentLabel.Text = "Label";
        container.Controls.Add(contentLabel);
    }
}
```

VB.NET

```
Protected Sub Page_Load(sender As Object, e As EventArgs)
    CustomRadButton.ContentTemplate = New ButtonContentTemplate()
End Sub
Public Class ButtonContentTemplate
    Implements ITemplate
    Sub InstantiateIn(container As Control) Implements ITemplate.InstantiateI
        Dim contentLabel As New Label()
        contentLabel.ID = "contentLabel"
        contentLabel.Text = "Label"
        container.Controls.Add(contentLabel)
    End Sub
End Class
```

- Add controls to the **Controls** collection from the code-behind:

C#

```
Label contentLabel = new Label();
contentLabel.ID = "contentLabel";
contentLabel.Text = "Label";
CustomRadButton.Controls.Add(contentLabel);
```

VB.NET

```
Dim contentLabel As New Literal()
contentLabel.ID = "contentLabel"
contentLabel.Text = "Label"
CustomRadButton.Controls.Add(contentLabel)
```

Note that the RadButton is rendered as an anchor HTML element, wrapping the inserted content, so you should add in the template only inline elements in order to maintain XHTML compliance.

23 RadBinaryImage

23.1 Objectives

- Learn what Binary Image is.
- Where it can be used and how to bound data to it.
- Review which are the most important properties of the RadBinaryImage control.

23.2 Introduction

The Binary Image control is used for showing an image stored as binary data in a database. The control can be used in any data bound control (RadGrid, Repeater, DataList, GridView, etc.). The control uses an internal http handler which streams the image from the binary source to the page in which it has to be visualized.

The storage of the binary stream when transferred between the control itself and the handler is the `HttpContext.Current.Cache` object and the image is cached in the browser. Its default expiration time is 2 hours (unless the control in which the RadBinaryImage is nested is rebound or recreated). In case the browser cache is disabled, the image will be persisted for 2 minutes on the server before it is streamed to the page from the data source.

You need to register the http handler of the **RadBinaryImage** control either using its **Smart Tag** or manually in the **web.config** file to ensure that it will be served as expected when the page is rendered.

[Web.config] Classic mode

```
<httpHandlers>
  <remove path="*.asmx" verb="*" />
  ...
  <add path="Telerik.Web.UI.WebResource.axd" type="Telerik.Web.UI.WebResource" verb="*"
  validate="false" />
</httpHandlers>
```

[Web.config] Integrated mode

```
<system.webServer>
  ...
  <handlers>
    <add name="Telerik_Web_UI_WebResource_axd" verb="*" preCondition="integratedMode"
    path="Telerik.Web.UI.WebResource.axd" type="Telerik.Web.UI.WebResource" />
  </handlers>
</system.webServer>
```

The most important properties of the RadBinaryImage control are presented below:

- **DataValue**: Property which specifies the source field from which the data will be passed as a byte array.
- **Height**: Specifies the height of the binary image.
- **Width**: Specifies the width of the binary image.
- **AlternateText**: The text that will replace the image when it is not available/cannot be streamed.
- **ToolTip**: The text that will be displayed in a browser tooltip when you hover the image.
- **AutoAdjustImageControlSize**: Scales the image based on explicitly set width/height dimensions to avoid stretch or blur effect when its original dimensions do not fit. The default value is true.
- **HandlerUrl**: Can be used to specify the location of a custom http handler which extends the default

RadBinaryImage http handler. When not set, RadBinaryImage has its own handler which is invoked through the common Telerik.WebResource.axd handler.

- **ImageUrl:** Applicable when no DataValue is specified to gracefully degrade to regular ASP.NET Image mode. When null value is returned from the source, the ImageUrl property can be used to specify default image for RadBinaryImage.
- **ImageAlign:** Specifies the image alignment inside its container.
- **ResizeMode:** Specifies whether the image should be sized automatically if width and height of the image are set in pixels. Possible values are:

Crop (the image will be trimmed)

Fit (the image will be sized to fit the given dimensions)

None (default)

- **SavedImageName:** Sets image's filename which will appear inside SaveAs browser dialog if image is saved.

23.3 Getting Started

RadBinaryImage - Thinking Inside the Box

Let's start with a simple RadBinaryImage demonstration used in RadGrid TemplateColumn.



You can find the complete source for this project at:
\\VS Projects\RadBinaryImage\RadBinaryImageSample

1. Create a new web application and add a ScriptManager component to the page
2. Add a **RadGrid** control to the page.
3. Configure the datasource to connect to the Telerik database. .
4. In the Configure the Select Statement step in the RadGrid SmartTag, select the "Specify a custom SQL statement" option and click **Next**. In the SELECT statement tab, enter the following query:

[SQL] Image select statement

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%=  
ConnectionStrings:TelerikConnectionString %%"  
SelectCommand="SELECT * FROM [Images]" >  
</asp:SqlDataSource>
```

5. Add GridTemplateColumn with RadBinaryImage in it.

6. Set the RadBinaryImage main properties: **ID**, **AlternateText**, **DataValue**, **Height**, **Width**, **ResizeMode**, **ToolTip**.

Adding BinaryImage to GridTemplateColumn

```
<telerik:RadGrid runat="server" ID="RadGrid1" AllowPaging="True" AllowSorting="True"  
AutoGenerateColumns="False" Width="97%" DataSourceID="SqlDataSource1"  
GridLines="None"  
PageSize="3">  
  <MasterTableView Width="100%" DataKeyNames="ID" DataSourceID="SqlDataSource1">  
    <Columns>
```

```
        <telerik:GridTemplateColumn HeaderText="Image Name"
UniqueName="ImageName" SortExpression="Name">
            <ItemTemplate>
                <telerik:RadBinaryImage ID="RadBinaryImage1" runat="server"
AlternateText='<## "Image of " + Eval("Description") %>' DataValue='<## Eval("Data") %>'
Height="80px" Width="80px" ResizeMode="Fit" ToolTip='<## "Image of " + Eval("Description") %
>' />
            </ItemTemplate>
            <HeaderStyle Width="30%" />
        </Columns>
    </MasterTableView>
</telerik:RadGrid>
```

24 RadFilter

24.1 Objectives

- Explore the features of **RadFilter** control
- Learn how to define filter expressions manually
- Learn how to use **RadFilter** in conjunction with **RadGrid** and **RadListView**
- Explore the client-side and server-side events

24.2 Introduction

The purpose of **RadFilter** control is to allow the developer to supply an interface for constructing strongly typed filter expressions. These expressions can be used to:

- query data from a data source control
- supply filter expressions to controls that support such expressions, for example **RadListView** and **RadGrid**

Most of the **RadFilter** control functionalities require that the ViewState is enabled - both for the control, and any container controls, in order for the operations to be handled properly and the filter expressions persisted.



24.3 Getting Started

Using RadFilter with RadGrid and RadListView

One of the most common uses of **RadFilter** is to be paired with other controls that support filter expressions. This enables the users to construct complex expressions that can not be created using the built-in filter controls.

Coupling the **RadFilter** and **RadGrid** (or **RadListView**) is really simple - the **FilterContainerID** property should point to the relevant target control ID.

[ASP.NET]

```
<telerik:RadFilter ID="RadFilter1" runat="server" FilterContainerID="RadGrid1" />
<telerik:RadGrid ID="RadGrid1" runat="server" AllowFilteringByColumn="true"
DataSourceID="SqlDataSource1">
</telerik:RadGrid>
```

Constructing filter expressions

It is possible to manually construct filter expressions and add them to **RadFilter** afterwards. This is demonstrated in the code-snippet below:

[C#]

```
if (!IsPostBack)
{
```

```
RadFilterEqualToFilterExpression<string> expression = new
RadFilterEqualToFilterExpression<string>("CategoryName");
RadFilter1.RootGroup.AddExpression(expression);
expression.Value = "Beverages";
}
```

[VB.NET]

```
If Not IsPostBack Then
    Dim expression As New RadFilterEqualToFilterExpression(Of String)("CategoryName")
    RadFilter1.RootGroup.AddExpression(expression)
    expression.Value = "Beverages"
End If
```

Note that when **RadFilter** is paired to other control and this control is bound to datasource control you should add the expression of the PreRender event of the relevant control. Furthermore, if you want this expression to take immediate effect you have to invoke the FireApplyCommand method.

24.4 Events

Server-side events

OnApplyExpressions - this event is raised when the user presses the "Apply" button or when the FireApplyCommand method has been invoked. Essentially, this is the moment when the actual filter expression is constructed. Note that pressing the "Apply" button does not trigger the OnItemCommand event handler.

OnItemCommand - This event is raised when a command is issued by the control - for example, when the end user adds a new filtering group

OnLoad - Occurs when the control loads

OnPreRender - This event is raised when the control is about to be rendered on the page

OnFilterEditorCreating - fires only for custom editors that inherit the built-in **RadFilter** editors. One could handle this event to replace or modify the editor instance that should be created and added into the relevant collection.

OnFilterEditorCreated - fires when **RadFilter** is paired with container control or when used to filter datasource control. This event can be used to change the DisplayName of **RadFilter** editor programmatically.

Client-side events

OnFilterCreated - fires when the filter control is created

OnFilterCreating - **RadFilter** rises this event when the control is being created but the process is not yet completed

OnFilterDestroying - this event fires when the filter control is being destroyed on the client

24.5 Summary

So far we have discussed the most important features of **RadFilter**. We learned how to use the control in conjunction with **RadGrid** and **RadListView**. This topic also covered the custom expressions as well as the events (both client-side and server-side). You can examine the **RadControls for ASP.NET AJAX** section on our website for more information (and demos) about the control.

25 RadImageEditor

25.1 Objectives

- Getting familiar with **RadImageEditor** control and its features
- Explore the design time interface - **Smart Tag** and **Properties Window**
- Learn how to utilize the **RadImageEditor**'s rich client-side API
- Learn how to use a **ToolsFile** or the **Tools** inner tag to configure the toolbar
- Learn how to use the server-side events to manipulate images from alternative source (database)

25.2 Introduction

RadImageEditor is a powerful and flexible graphics editing component that allows the users to modify their images directly in the browser, without installing any third party plugins. The actions on the image are performed on the client or, through a light callback, on the server, giving you the ability to perform the editing quickly and see the changes on the fly. The control is fully customizable, intuitive to work with and provides many features. Here are some of the key ones:

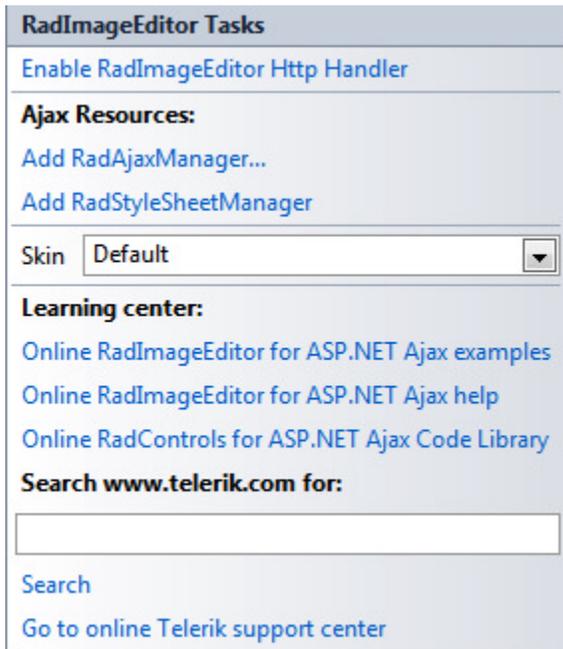
- Rich Client-side functionality - **RadImageEditor** exposes many of the methods used to perform the operations on the image, so that the image can be easily modified programmatically.
- Customizable Tools - The developer has full control over the ImageEditor's set of tools. The **ToolsFile** property or the **Tools** collection can be used to specify the desired buttons that will appear in the ToolBar.
- Intuitive Dialogs - **RadImageEditor** provides a set of built-in dialog controls that are easy to use and offer the user a quick and intuitive way of modifying the images in the desired fashion.
- Integrated RadControls - We have embedded some of our ASP.NET Ajax RadControls into the **ImageEditor** to benefit from their rich functionality. This not only makes the new component feature rich, but a perfect example of how the RadControls work together in complex scenarios. The integrated controls are loaded on demand thus ensuring optimal loading speed and scalability.
- Undo/Redo Actions - **RadImageEditor** saves all the operations, so each change performed on the image can be reverted or re-applied again. The operations stack is cleared on postbacks, because we assume the user won't initiate a postback unless she is finished with the editing.
- Image Operations - A variety of operations can be performed on a given image. This includes rotation, resizing, changing the transparency, cropping and many more.

25.3 Smart Tag

RadImageEditor offers rich functionality in its design-time Smart Tag. You can display it by right clicking on the control and choosing "Show Smart Tag", or clicking the small rightward-pointing arrow located in the upper right corner of the control. You can also use it to:

- add the Telerik WebResource handler - if it is not present in the web.config you are presented with an option to add it
- add the needed AJAX resources to the web application - ScriptManager or RadScriptManager, RadAjaxManager, RadStyleSheetManager
- change the **Skin** of the control via the dedicated dropdown
- quickly get help via our online learning center - online help, examples, knowledge base and code library, web search and our support center

Below is an example how the smart tag will look if you do not have the Telerik Http Handler in the web.config. Notice the first option:



25.4 Getting Started

This tutorial demonstrates how to add a **RadImageEditor** in a page, load an image in it so that you can start modifying it:

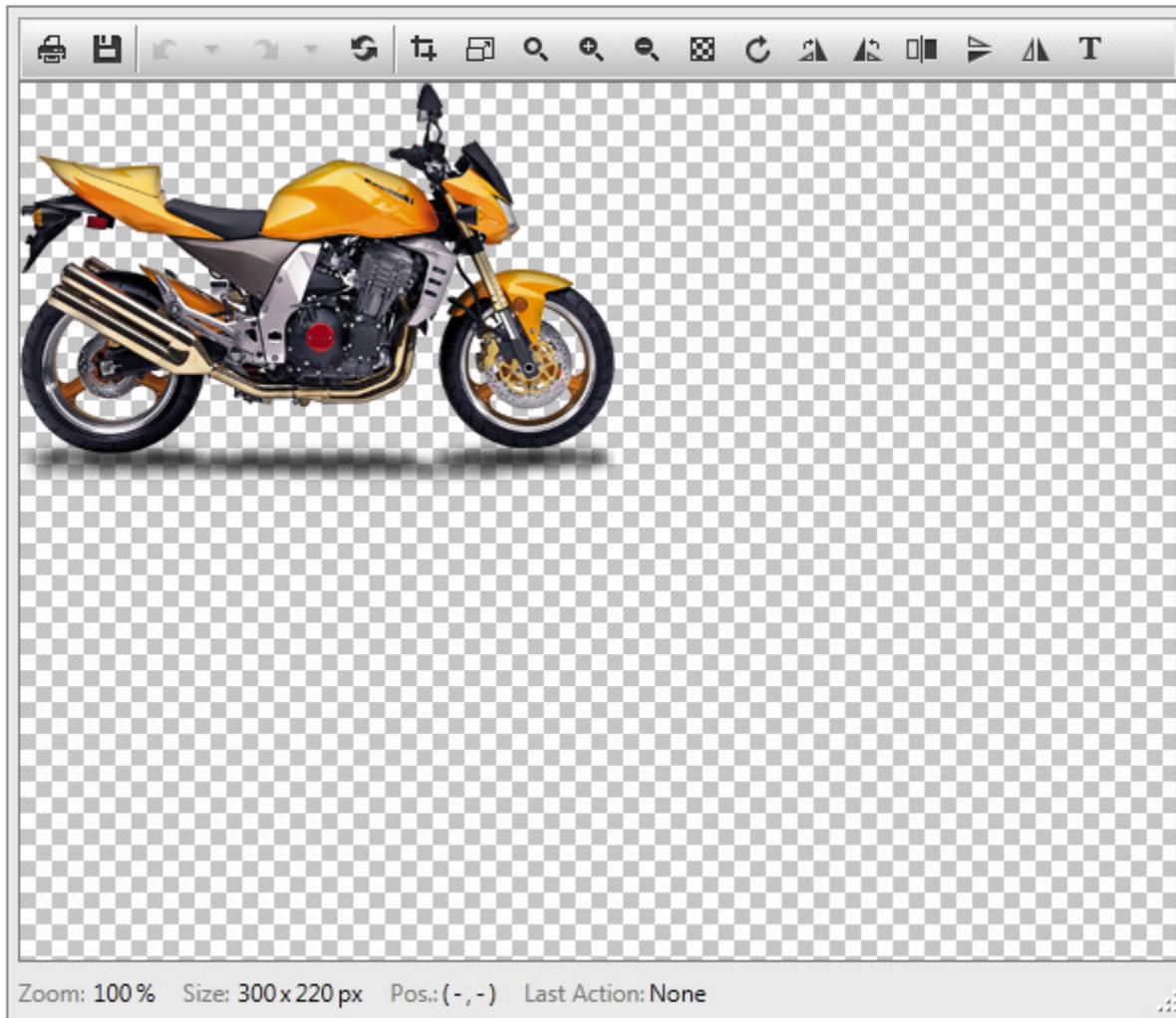
1. Create a new ASP.NET AJAX - enabled web site
2. Add a RadScriptManager or the standard ASP ScriptManager to the page - this step is mandatory if you are using ASP.NET AJAX controls
3. Drag a **RadImageEditor** from your VS Toolbox and drop it on the page
4. Right-click on the inserted **RadImageEditor** control and select properties
5. Set the **ImageUrl** property from the Properties tab to point to an image to be loaded. Please note that the image must be located in a folder accessible from the application
6. If you are not creating a RadControls WebSite you should add the **RadImageEditor** HttpHandler from the smart tag
7. You can further modify the control by setting these properties:
 1. **ImageCacheStorageLocation** - where the control should store its cache
 2. **StatusBarMode** - controls the position of the status bar
 3. **Width** - sets explicit width for the control in pixels
 4. **Height** - sets explicit height for the control, also in pixels
8. Save the page and run it in the browser.

The resulting markup should look similar to this if no additional properties are set:

RadImageEditor declaration

```
<telerik:RadImageEditor runat="server" ID="RadImageEditor1"
    ImageUrl="image1.png"></telerik:RadImageEditor>
```

If you do not set explicit dimensions the resulting **ImageEditor** will revert to the default values and will look similar to this:



25.5 Configuring the Toolbar

The **RadImageEditor** comes with a predefined toolbar, containing the default set of tools. This, however, may be a bit too many options for some scenarios, or you would like the order of the buttons to be different, or the tooltips, etc. This is the reason why we offer two ways to modify the toolbar - via the **Tools inner tag** and via the **ToolsFile** property.

The **ToolsFile** property can take the path to an xml file containing the custom toolbar definition. This allows for a single template in the entire application - i.e. an easy way of providing a custom, yet consistent functionality. Here follows an example declaration:

Setting the ToolsFile property

```
<telerik:RadImageEditor ID="RadImageEditor1" runat="server"
ImageUrl="logo.png" ToolsFile="ToolsFile.xml" />
```

ToolsFile.xml

```
<?xml version="1.0" encoding="utf-8" ?>
<root>
  <tools name="MainToolbar">
    <tool name="Print" togglebutton="true" />
    <tool name="Save" togglebutton="true" />
    <tool separator="true"/>
    <tool name="Undo" toolstrip="true" />
    <tool name="Redo" toolstrip="true" />
    <tool name="Reset" />
    <tool separator="true"/>
    <tool name="Crop" togglebutton="true" />
    <tool name="Resize" togglebutton="true" />
    <tool name="Zoom" togglebutton="true" />
    <tool name="Opacity" togglebutton="true" />
    <tool name="Rotate" togglebutton="true" />
    <tool name="Flip" togglebutton="true" />
    <tool name="AddText" togglebutton="true" />
  </tools>
</root>
```

This will result in the following toolbar:



The **Tools** inner tag allows you to set the tools you wish to use in the markup of the control. This is more difficult to maintain than an xml source, yet allows for a nice exception of the common feel if you need one:

Setting the inner Tools tag

```
<telerik:RadImageEditor ID="RadImageEditor1" runat="server" ImageUrl="">
  <Tools>
    <telerik:ImageEditorToolGroup>
      <telerik:ImageEditorTool CommandName="Save" />
      <telerik:ImageEditorToolSeparator />
      <telerik:ImageEditorToolStrip CommandName="Undo" Text="Undo" />
      <telerik:ImageEditorToolStrip CommandName="Redo" Text="Redo" />
      <telerik:ImageEditorToolSeparator />
      <telerik:ImageEditorTool CommandName="Crop" />
      <telerik:ImageEditorTool CommandName="Opacity" />
    </telerik:ImageEditorToolGroup>
  </Tools>
</telerik:RadImageEditor>
```

This declaration will give the following:



If either one of these options is used the default toolbar is overridden by the new declaration. If both are used - i.e. you declare **Tools** in the markup and point the **ToolsFile** property to a valid source you will get both toolbars in the rendered **RadImageEditor** - first the ones from the xml, then the ones from the markup:



25.6 Localization

RadImageEditor is fully localized by using Global resources. The control comes with three built-in language packs, English, German, and French, and you can easily switch by setting the **Language** property of the control or setting **UICulture** property to the **@Page**. The **Language** property is with higher priority than the global **UICulture** setting. There are two options to localize the **RadImageEditor** - by using resource files and by using the **Localization** property of the control.

You can use *.resx files to localize (or customize) the control's localization strings with minimum efforts. The **RadControls'** installation wizard copies the built-in resource files in the **App_GlobalResources** folder in your local installation. You can either create your own language pack or use an existing one (if available for your language).

The following steps demonstrate how to create a new language pack for **RadImageEditor**.

1. Add **App_GlobalResources** folder to the application folder (if it does not already exist)
2. Copy **RadImageEditor.Main.resx** and **RadImageEditor.Dialogs.resx** files to **App_GlobalResources** folder
3. Create new copies of the above files and name them according to the new language's culture. **RadImageEditor.Main.[your_language].resx** and **RadImageEditorDialogs.[your_language].resx**. Please note that you need to keep the original files in the folder as well
4. Open the newly copied language specific resource file and modify the keys' values, but you should not modify/remove the **ReservedResource** key
5. Set the **RadImageEditor's Language** property to the corresponding language
6. When you run the application, the new resources will be recognized and the corresponding hints or other UI elements will display in the new language

The **Localization** property can be set in the code-behind and allows for easy minor adjustments on a small scale, for example:

Localization Property C#

```
RadImageEditor1.Localization.Main.AddText = "Добавитекст";  
RadImageEditor1.Localization.Main.Opacity = "Прозрачность";
```

Localization Property VB.NET

```
RadImageEditor1.Localization.Main.AddText = "Добави текст"
```

```
RadImageEditor1.Localization.Main.Opacity = "Прозрачность"
```

If the **Localization** property is set it will override any other settings defined in the selected language's resource file, for example setting the **Language** property to de-DE and using the above localization strings will result in the following tooltips:



While the other ones will be in German:



25.7 Creating a Custom Tool

The **RadImageEditor** exposes many methods and properties on the client, giving you the ability to perform any action on the image programmatically. You could easily change the transparency, decrease the dimensions and save the changes of the image by calling the correct method. This enables you to create your own custom tools if you need some functionality you cannot find built-in the control. Here follows an example of a custom command that applies a predefined string in the bottom right corner of the image:

Creating a custom tool

```
<telerik:RadImageEditor ID="RadImageEditor1" runat="server" ImageUrl="~/images/11.jpg">
  <Tools>
    <telerik:ImageEditorToolGroup>
      <telerik:ImageEditorTool Text="Custom Text" CommandName="CustomText" />
    </telerik:ImageEditorToolGroup>
  </Tools>
</telerik:RadImageEditor>
<script type="text/javascript">
  //define new custom command CustomText
  Telerik.Web.UI.ImageEditor.CommandList["CustomText"] = function (imageEditor,
commandName, args)
  {
    var editedImage = imageEditor.getImage();
    var imageBounds =
      {
        "width": editedImage.clientWidth,
        "height": editedImage.clientHeight
      };
    var textObj = createTextObject("Telerik", "Verdana", "18pt",
"#33ff00");//Telerik.Web.UI.ImageEditor.ImageText

    var textSize = getTextBounds(textObj);//this function precalculates the size
```

of the text that will be drawn

```
//calculate text starting position
var x = imageBounds.width - textSize.width;
var y = imageBounds.height - textSize.height;
imageEditor.addToImage(x, y, textObj);
}
//create a DIV element, with the predefined styling and text as a content to
calculate the size of the text that will be drawn
function getTextBounds(textObj)
{
    var tempDiv = document.createElement("DIV");
    tempDiv.style.cssText = "position:absolute;top:-9999;left:-9999;padding-
right:5pt;font-family:" + textObj.get_fontFamily() + ";font-size:" + textObj.get_fontSize()
+ "pt;";

    tempDiv.innerHTML = textObj.get_text();
    document.body.appendChild(tempDiv);
    var elementSize =
        {
            "width": tempDiv.clientWidth,
            "height": tempDiv.clientHeight
        }
    document.body.removeChild(tempDiv);
    return elementSize;
}
//Telerik.Web.UI.ImageEditor.ImageText
function createTextObject(text, fontFamily, fontSize, color)
{
    var textObject = new Telerik.Web.UI.ImageEditor.ImageText();
    textObject.set_fontFamily(fontFamily);
    textObject.set_fontSize(parseInt(fontSize));
    textObject.set_color(color);
    textObject.set_text(text);
    return textObject;
}
</script>
```

25.8 Save a Thumbnail

RadImageEditor offers a variety of server-side methods and events that enable you to manipulate an image on the server, not just on the client. You can use the **OnImageSaving** event to perform additional processing on the image before it is saved and the **OnImageLoading** event to change the image that will be loaded according to some custom logic. The example below saves a thumbnail of the edited image (in the thumbs folder) along with the image the user works with in the **OnImageSaving** event. The **Cancel** property must be set to true if custom actions are performed so that the default ones are cancelled. This is done in when a new image is loaded, but we do not do it in this case when saving the thumbnail, as we also want the original to be saved.

The **RadFileExplorer** control is used to allow the user to navigate through the available images and when one is selected it is populated in the **RadImageEditor**. Upon saving of the image a new copy is created that is 90 by 90 pixels large. Also, after the image is saved, the **FileExplorer** is refreshed.

Save a Thumbnail demo

```

<asp:ScriptManager runat="server" ID="ScriptManager1"></asp:ScriptManager>
<telerik:RadAjaxManager ID="RadAjaxManager1" runat="server"
OnAjaxRequest="RadAjaxManager1_AjaxRequest"
RequestQueueSize="3">
  <AjaxSettings>
    <telerik:AjaxSetting AjaxControlID="RadAjaxManager1">
      <UpdatedControls>
        <telerik:AjaxUpdatedControl ControlID="RadImageEditor1"
LoadingPanelID="RadAjaxLoadingPanel1" />
      </UpdatedControls>
    </telerik:AjaxSetting>
  </AjaxSettings>
</telerik:RadAjaxManager>
<telerik:RadAjaxLoadingPanel ID="RadAjaxLoadingPanel1" runat="server">
</telerik:RadAjaxLoadingPanel>
<div style="width: 980px;">
  <telerik:RadFileExplorer ID="RadFileExplorer1" runat="server"
DisplayUpFolderItem="true"
Width="260px" Height="442px" Style="float: left;"
VisibleControls="Grid,Toolbar,AddressBox"
EnableCreateNewFolder="false"
OnClientItemSelected="FileExplorer_OnClientItemSelected"
OnClientLoad="FileExplorer_OnClientLoad">
  </telerik:RadFileExplorer>
  <div style="width: 700px; float: left;">
    <telerik:RadImageEditor ID="RadImageEditor1" runat="server" Width="700px"
Height="430px"
OnClientSaved="ImageEditor_OnClientSaved" ToolsLoadPanelType="XmlHttpRequest"
EnableResize="false"
OnImageSaving="RadImageEditor1_ImageSaving"
OnImageLoading="RadImageEditor1_ImageLoading">
  </telerik:RadImageEditor>
  </div>
  <br style="clear: both;" />
</div>
<telerik:RadCodeBlock ID="RadCodeBlock1" runat="server">
  <script type="text/javascript">
    var ajaxFlag = false;
    function FileExplorer_OnClientLoad(sender, args)
    {
      ajaxFlag = true;
    }
    function FileExplorer_OnClientItemSelected(explorer, args)
    {
      var item = args.get_item();
      if (!item.isDirectory() && ajaxFlag)
      {
        var ajaxManager = $find("<%= RadAjaxManager1.ClientID %>");
        ajaxManager.ajaxRequest(item.get_path());
      }
    }
    function ImageEditor_OnClientSaved(imgEditor, args)
    {
      var fileExplorer = $find("<%= RadFileExplorer1.ClientID %>");
      fileExplorer.refresh();
    }
  </script>
</telerik:RadCodeBlock>

```

```
    }  
    </script>  
</telerik:RadCodeBlock>
```

C#

```
private string pathToImage = "~/Files/Images/logo.png";  
private string pathToThumbs = "~/Files/thumbs/";  
protected void Page_Load(object sender, EventArgs args)  
{  
    if (!IsPostBack)  
    {  
        string[] paths = new string[] { "~/Files/Images" };  
        RadFileExplorer1.Configuration.ViewPaths = paths;  
        RadFileExplorer1.Configuration.DeletePaths = paths;  
        RadFileExplorer1.Configuration.UploadPaths = paths;  
        RadFileExplorer1.Configuration.MaxUploadFileSize = 4 * 1024 * 1024;  
        RadFileExplorer1.Configuration.SearchPatterns = new string[] { "*.jpg",  
        "*.jpeg", "*.gif", "*.png", "*.bmp" };  
        RadFileExplorer1.EnableOpenFile = false;  
        string initialPath = Page.ResolveUrl(pathToImage);  
        RadFileExplorer1.InitialPath = initialPath;  
        RadImageEditor1.ImageUrl = initialPath;  
        foreach (RadToolBarButton item in RadFileExplorer1.ToolBar.Items)  
        {  
            if (item.Value != "Upload" && item.Value != "Delete")  
                item.Visible = false;  
        }  
    }  
}  
protected void RadImageEditor1_ImageSaving(object sender, ImageEditorSavingEventArgs  
args)  
{  
    var thumbImage = args.Image.Clone();  
    thumbImage.Resize(90, 90);  
    var ms = new MemoryStream();  
    thumbImage.Image.Save(ms, thumbImage.RawFormat);  
    File.WriteAllBytes(String.Format("{0}{1}.{2}", MapPath(pathToThumbs),  
args.FileName, thumbImage.Format), (byte[])ms.ToArray());  
}  
protected void RadImageEditor1_ImageLoading(object sender,  
ImageEditorLoadingEventArgs args)  
{  
    args.Image = new EditableImage(MapPathSecure(pathToImage));  
    args.Cancel = true;  
}  
protected void RadAjaxManager1_AjaxRequest(object sender, AjaxRequestEventArgs e)  
{  
    pathToImage = e.Argument;  
    RadImageEditor1.ImageUrl = pathToImage;  
}
```

VB.NET

```

Private pathToImage As String = "~/Files/Images/logo.png"
Private pathToThumbs As String = "~/Files/thumbs/"
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles
Me.Load
    If Not IsPostBack Then
        Dim paths As String() = New String() {"~/Files/Images"}
        RadFileExplorer1.Configuration.ViewPaths = paths
        RadFileExplorer1.Configuration.DeletePaths = paths
        RadFileExplorer1.Configuration.UploadPaths = paths
        RadFileExplorer1.Configuration.MaxUploadFileSize = 4 * 1024 * 1024
        RadFileExplorer1.Configuration.SearchPatterns = New String() {"*.jpg", "*.jpeg",
"*.gif", "*.png", "*.bmp"}
        RadFileExplorer1.EnableOpenFile = False
        Dim initialPath As String = Page.ResolveUrl(pathToImage)
        RadFileExplorer1.InitialPath = initialPath
        RadImageEditor1.ImageUrl = initialPath
        For Each item As RadToolBarButton In RadFileExplorer1.ToolBar.Items
            If item.Value <> "Upload" AndAlso item.Value <> "Delete" Then
                item.Visible = False
            End If
        Next
    End If
End Sub
Private Sub RadImageEditor1_ImageLoading(ByVal sender As Object, ByVal args As
Telerik.Web.UI.ImageEditorLoadingEventArgs) Handles RadImageEditor1.ImageLoading
    args.Image = New EditableImage(MapPathSecure(pathToImage))
    args.Cancel = True
End Sub
Private Sub RadImageEditor1_ImageSaving(ByVal sender As Object, ByVal args As
Telerik.Web.UI.ImageEditorSavingEventArgs) Handles RadImageEditor1.ImageSaving
    Dim thumbImage = args.Image.Clone()
    thumbImage.Resize(90, 90)
    Dim ms = New MemoryStream()
    thumbImage.Image.Save(ms, thumbImage.RawFormat)
    File.WriteAllBytes([String].Format("{0}{1}.{2}", MapPath(pathToThumbs),
args.FileName, thumbImage.Format), DirectCast(ms.ToArray(), Byte()))
End Sub
Private Sub RadAjaxManager1_AjaxRequest(ByVal sender As Object, ByVal e As
Telerik.Web.UI.AjaxRequestEventArgs) Handles RadAjaxManager1.AjaxRequest
    pathToImage = e.Argument
    RadImageEditor1.ImageUrl = pathToImage
End Sub

```

26 RadListView

26.1 Objectives

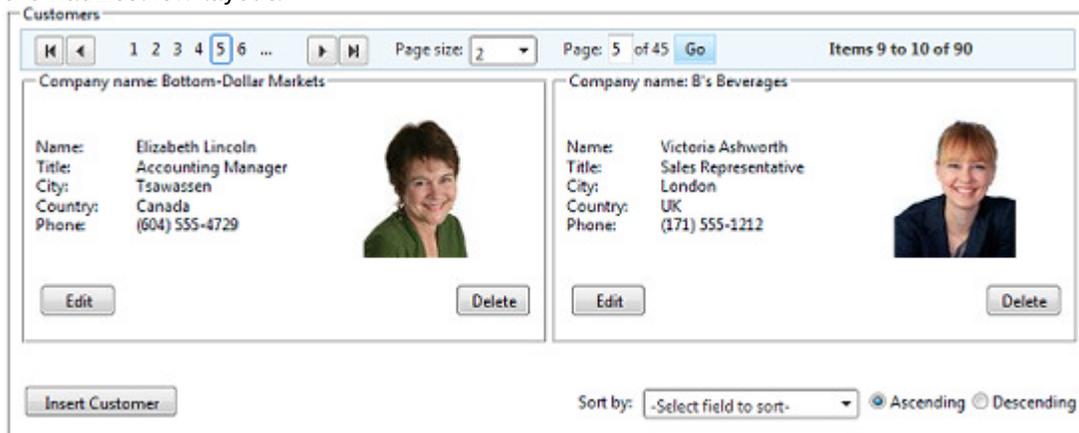
- Explore the features of the RadListView control.
- Explore the RadListView design time interface, including Smart Tag and Properties View.
- Create simple application for binding data using the RadListView predefined layouts and the most common features.
- Learn how to perform manual CRUD (create, read, update and delete) operations through the RadListView server-side API.
- Explore the RadDataPager control and see how you can use it for paging navigation.

26.2 Introduction

RadListView is data-bound control and you can use it in any web application where you want to display data in a custom manner with a unique look and feel.

RadListView:

- Provided the following templates: LayoutTemplate, ItemTemplate, AlternatingItemTemplate, EditItemTemplate, InsertItemTemplate, EmptyDataTemplate, ItemSeparatorTemplate, SelectedItemTemplate, GroupTemplate, GroupSeparatorTemplate, EmptyItemTemplate for customizing the RadListView layout.



- Supports paging navigation either using built-in commands or RadDataPager/DataPager control.
- Has rich server-side API for filtering items.
- Supports sorting and items selection.
- ListView-like grouping can be achieved with RadListView. For that purpose the GroupTemplate, GroupSeparatorTemplate and EmptyItemTemplate should be defined.
- RadListView offers the Items Drag & Drop capability, allowing you to easily implement scenarios that require dragging and moving data items on the page.

26.3 Getting Started

Here we will describe the main features of the RadListView and the properties/methods you should know to enable them.

RadListView Templates

- **LayoutTemplate** - It helps you define the overall appearance of the control, the outer wrapper that will be used for the listview rendering as well as the holder of its content. Note that you have to specify **ItemPlaceholderID** property value for RadListView that matches the id of an ASP.NET server control (with id and runat="server" properties set) which will be used as a holder of the actual listview data content.

ASPX

```
<telerik:RadListView ID="RadListView1" runat="server" ItemPlaceholderID="itemPlaceholder"
  <LayoutTemplate>
    <fieldset>
      <legend>Items</legend>
      <asp:PlaceHolder ID="itemPlaceholder" runat="server" />
    </fieldset>
  </LayoutTemplate>
</telerik:RadListView>
```

- **ItemTemplate** and **AlternatingItemTemplate** - These templates mark out how the data that is bound to the listview will be visualized in its odd/even items respectively. Since those are templates, you are free to customize their layout according to your custom conventions.
- **EmptyDataTemplate** - The content of the EmptyDataTemplate is displayed when no data is available in the RadListView data source.
- **ItemSeparatorTemplate** - Define the ItemSeparatorTemplate by adding there the html content which would display between the different items in the RadListView control.
- **EditItemTemplate** - Determines how and what controls will be rendered when an item is in edit mode.
- **InsertItemTemplate** - As the above one, determines how and what controls will be rendered in the RadListView insert form.
- **SelectedItemTemplate** - Defines here the contents that represents the selected item in a RadListView.
- **GroupTemplate** - Create the group structure and look by defining the **GroupTemplate**.
- **GroupSeparatorTemplate** - Define this template to separate the different groups in a RadListView.
- **EmptyItemTemplate** - The EmptyItemTemplate content is displayed in place of the missing items for a group. For instance if the a group should contain 8 items but only 7 are available in database, the EmptyItemTemplate will be used for the eighth item of the group.

Paging

RadListView has native paging support. To enable paging, set the **AllowPaging** property to **true**.

If you choose to use the integrated paging, you can add command controls (Button, LinkButton, ImageButton) with a **CommandName** value of **Page**. In this case you also need to set the **CommandArgument** property of the command buttons. The possible values for it are: Next, Prev, First and Last. Thus your pager buttons would allow the user to navigate to the next, previous, first and last page.

Another page command you might want to use is the **ChangePageSize** command. It is should be set as **CommandName** of a button which will change the page of the RadListView. Here the **CommandArgument** of the button should be the new page size value.

A simple pager might look as below:

ASPX

```
<telerik:RadListView ID="RadListView1" runat="server" ItemPlaceholderID="itemPlaceholder"
  AllowPaging="true">
  <LayoutTemplate>
    <fieldset>
      <legend>Items</legend>
      <asp:PlaceHolder ID="itemPlaceholder" runat="server" />
    </fieldset>
  </LayoutTemplate>
</telerik:RadListView>
```

```
<div>
  <div style="float: left; margin-left: 30%;">
    <asp:Button runat="server" ID="btnFirst" CommandName="Page"
CommandArgument="First"
      Text="First" Enabled="<%=Container.CurrentPageIndex > 0 %>" />
    <asp:Button runat="server" ID="btnPrev" CommandName="Page"
CommandArgument="Prev"
      Text="Prev" Enabled="<%=Container.CurrentPageIndex > 0 %>" />
    <span style="vertical-align: top; position: relative; top: 4px">Page
      <%=Container.CurrentPageIndex + 1 %> of <%=Container.PageCount %
></span>
    <asp:Button runat="server" ID="btnNext" CommandName="Page"
CommandArgument="Next"
      Text="Next" Enabled="<%=Container.CurrentPageIndex + 1 <
Container.PageCount %>" />
    <asp:Button runat="server" ID="btnLast" CommandName="Page"
CommandArgument="Last"
      Text="Last" Enabled="<%=Container.CurrentPageIndex + 1 <
Container.PageCount %>" />
  </div>
  <div>
    <span style="vertical-align: middle; font-weight: bold; padding-left:
5px;">Change Page Size to 20:</span>
    <asp:Button runat="server" ID="btnPrev" CommandName="ChangePageSize"
CommandArgument="20"
      Text="Go" />
  </div>
</div>
</fieldset>
</LayoutTemplate>
</telerik:RadListView>
```

Additionally, you can choose to add a **RadDataPager** control for paging navigation which provided all the desired functionalities in one. We will dive into it later in this chapter.

Sorting

Implementing sorting is quite easy with RadListView. All you need to do is to add **SortExpression** to its **SortExpressions** collection. You can do it either declaratively, so the RadListView data is sorted all the time, or do it dynamically on particular user action. Let's say we have a sort button which, after clicked, will sort the listview by a field descending. Then the Click event handler of the button would look like this:

C#

```
protected void SortButton_Click(object sender, EventArgs e)
{
    RadListViewSortExpression expression = new RadListViewSortExpression();
    RadListView1.SortExpressions.Clear();
    expression.FieldName = "myFieldName";
    expression.SortOrder = RadListViewSortOrder.Descending;
    RadListView1.SortExpressions.AddSortExpression(expression);
    RadListView1.Rebind();
}
```

VB.NET

```
Protected Sub SortButton_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim expression As New RadListViewSortExpression()
```

```
RadListView1.SortExpressions.Clear()
expression.FieldName = "myFieldName"
expression.SortOrder = RadListViewSortOrder.Descending
RadListView1.SortExpressions.AddSortExpression(expression)
RadListView1.Rebind()
End Sub
```

Furthermore, if you want to implement sorting in a custom manner, you can do it by setting the **AllowCustomSorting** property to **true**. Then add a command button to the RadListView LayoutTemplate with a CommandName value of **Sort** and the desired CommandArgument. Thus the Sorting event of the RadListView will fire for you to handle the custom sorting and for provide sorted data to the RadListView directly. In this case you need to rebind the RadListView control so sorting is applied.

Filtering

RadListView provided rich server-side API for creating and applying filter expressions. It gives you the ability to filter the data displayed in a RadListView control without creating complex database queries. For more information on how to operate with it, you can refer to the documentation of the RadListView for ASP.NET AJAX control.

Grouping

RadListView supports ListView-like grouping for its items. You can easily achieve displaying of data in groups with RadListView by setting the properties: **GroupItemCount**, **GroupPlaceholderID**, **ItemPlaceholderID**. In addition you need to define the following Templates: **LayoutTemplate**, **GroupTemplate**, **GroupSeparatorTemplate**, **EmptyItemTemplate**. Thus the skeleton for a RadListView which displays its data in groups will be:

ASPX

```
<telerik:RadListView ID="RadListView1" runat="server" ItemPlaceholderID="itemPlaceholder"
GroupPlaceholderID="groupsPlaceholder"
GroupItemCount="4">
<LayoutTemplate>
  <asp:Placeholder ID="groupsPlaceholder" runat="server" />
</LayoutTemplate>
<GroupTemplate>
  <fieldset style="float: left; width: 330px;">
    <legend>Items group</legend>
    <table>
      <tr>
        <td>
          <asp:Placeholder ID="itemPlaceholder" runat="server" />
        </td>
      </tr>
    </table>
  </fieldset>
</GroupTemplate>
<GroupSeparatorTemplate>
  <hr />
</GroupSeparatorTemplate>
<EmptyItemTemplate>
  <div style="float: left; width: 160px; height: 120px">
    
  </div>
</EmptyItemTemplate>
```

```
<ItemTemplate>  
...  
</ItemTemplate>  
</telerik:RadListView>
```

Selecting

The selected items are accessible through the **SelectedItems** collection that consist of **RadListViewDataItem** objects. By default you can select only one item at a time. Multiple selection is possible if enabled via the **AllowMultiItemSelection** property. There are several ways to select/deselect an item in **RadListView**:

- use the **Selected** property of **RadListViewDataItem**
- fire **Select/Deselect** command
- add/remove item's index to the **SelectedIndexes** collection

The selected items can be cleared using the **ClearSelectedItems** method.

For detailed information on RadListView items selection feature, see the control documentation and online demos.

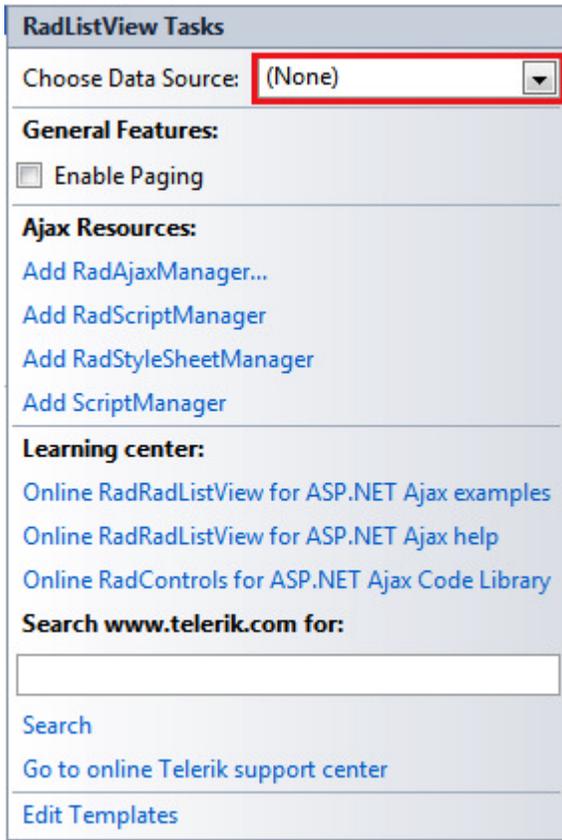
Items Drag & Drop

RadListView Items Drag & Drop capability is useful and is easy to implement for scenarios that require dragging and moving data items on the page. To enable Items Drag & Drop in RadListView first set the **RadListView.ClientSettings.AllowItemsDragDrop** property to **true**. You also need to add a **RadListViewItemDragHandle** control to your **ItemTemplate** / **AlternatingItemTemplate** and add a CSS marker on a data item container element (**.rlvI** for **ItemTemplate**, **.rlvA** for **AlternatingItemTemplate**). When item is dropped RadListView fires the **ItemDrop** event on the server for you to handle the items drop.

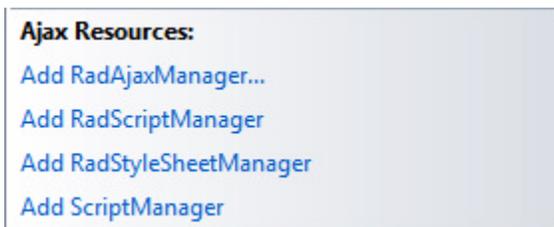
26.4 Using the design Time Interface

Smart Tag

The **Smart Tag** for RadListView allows to configure the control's layout with ease. Even if you don't have any knowledge, the integrated **Layout editor** will help you choose predefined layout in codeless manner. There you can choose the Data Source for the control:



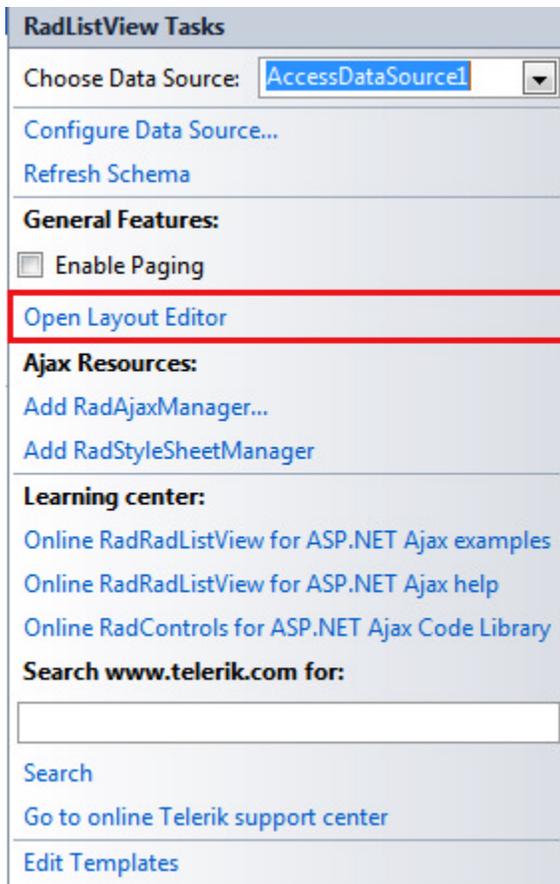
It also provides useful links that allow you to easily Ajaxify the control and optimize the web page that the grid appears on.



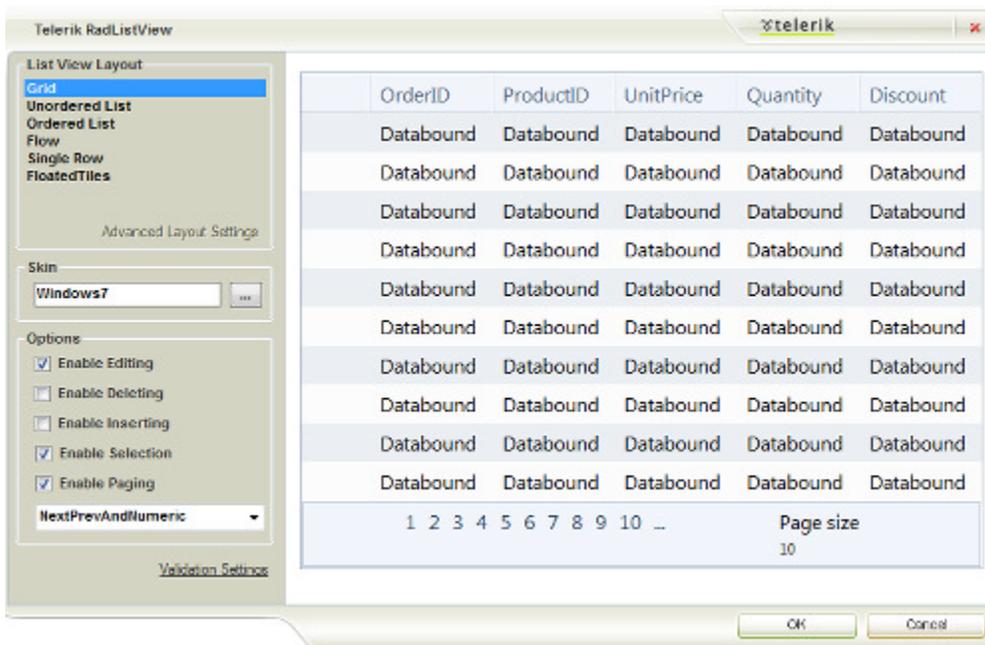
Layout Editor

Once you have selected Data Source for the RadListView the Open Layout Editor link appears in the Smart Tag.

UI for ASP.NET AJAX



Use to build the RadListView by choosing any of the predefined layouts, select Skin for the control or enable the most commonly used features.



Properties Window

Use the Properties window to manipulate the RadListView properties or to auto-generate server-side event

handlers:

RadListView1 Telerik.Web.UI.RadListView

(Expressions)	
(ID)	RadListView1
AccessKey	
AllowCustomPaging	False
AllowCustomSorting	False
AllowMultiFieldSorting	False
AllowMultiItemEdit	False
AllowMultiItemSelectio	False
AllowNaturalSort	False
AllowPaging	False
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CanRetrieveAllData	True
ClientDataKeyNames	
ClientSettings	Telerik.Web.UI.RadListV
ConvertEmptyStringTo	True
CssClass	
DataKeyNames	OrderID,ProductID
DataMember	
DataSourceID	AccessDataSource1
EnableAjaxSkinRenderi	True
Enabled	True
EnableEmbeddedBaseS	True
EnableEmbeddedScript	True
EnableEmbeddedSkins	True
EnableTheming	True
EnableViewState	True
Font	
ForeColor	
GroupItemCount	1
GroupPlaceholderID	groupPlaceholder
Height	
InsertItemPosition	None
ItemPlaceholderID	itemPlaceholder
OverrideDataSourceCo	False

(Expressions)
The expressions that are bound to properties of this control.

Properties Solution Explorer

RadListView1 Telerik.Web.UI.RadListView

DataBinding	
DataBound	
Disposed	
Init	
ItemCanceling	
ItemCommand	
ItemCreated	
ItemDataBound	
ItemDeleted	
ItemDeleting	
ItemDrop	
ItemEditing	
ItemInserted	
ItemInserting	
ItemUpdated	
ItemUpdating	
LayoutCreated	
Load	
NeedDataSource	
PageIndexChanged	
PageSizeChanged	
PreRender	
SelectedIndexChanged	
Sorting	
Unload	

NeedDataSource
Raised when the listView is about to be bound and the data source must be assigned

Properties Solution Explorer

26.5 Server Side Code

RadListView - Manual CRUD Operations sample

Here we will illustrate how you can configure RadListView and use its server-side API to perform manual CRUD (create, read, update delete) operations. For that purpose:

1. Create new project in Visual Studio and add a RadScriptManager on top of the form. You can also add RadSkinManager and set its Skin property, and a RadFormDecorator control for page styling.
2. Then add the RadListView control itself using the below code for it:

ASPX

```
<telerik:RadListView ID="RadListView1" runat="server"
OnNeedDataSource="RadListView1_NeedDataSource"
ItemPlaceholderID="ProductItemContainer" DataKeyNames="ProductID" AllowPaging="true"
OnItemCommand="RadListView1_ItemCommand">
<LayoutTemplate>
<fieldset style="width: 760px;" id="FieldSet1">
<legend>Products</legend>
<table cellpadding="0" cellspacing="0">
<tr>
<td>
<asp:Button ID="Button1" runat="server" Text="Add new product"
CommandName="InitInsert"
Visible="<%=Container.InsertItemPosition ==
RadListView.InsertItemPosition.None %>"
CausesValidation="false" />
</td>
</tr>
<tr>
<td>
<asp:Panel ID="ProductItemContainer" runat="server" />
</td>
</tr>
<tr>
<td>
<telerik:RadDataPager ID="RadDataPager1" runat="server"
PagedControlID="RadListView1"
PageSize="6">
<Fields>
<telerik:RadDataPagerButtonField FieldType="FirstPrev" />
<telerik:RadDataPagerButtonField FieldType="Numeric" />
<telerik:RadDataPagerButtonField FieldType="NextLast" />
</Fields>
</telerik:RadDataPager>
</td>
</tr>
</table>
</fieldset>
</LayoutTemplate>
<ItemTemplate>
<fieldset style="float: left; width: 223px; height: 160px;">
<table cellpadding="2" cellspacing="0" style="height: 100%;">
<tr>
<td style="width: 20%;">
Name:
```

```

        </td>
        <td style="width: 80%; padding-left: 5px;">
            <%=# Eval("ProductName") %>
        </td>
    </tr>
    <tr>
        <td>
            Quantity:
        </td>
        <td style="width: 80%; padding-left: 5px;">
            <%=# Eval("QuantityPerUnit") %>
        </td>
    </tr>
    <tr>
        <td>
            Price:
        </td>
        <td style="width: 80%; padding-left: 5px;">
            <%=# DataBinder.Eval(Container.DataItem, "UnitPrice", "{0:C}") %>
        </td>
    </tr>
    <tr>
        <td>
            Units:
        </td>
        <td style="width: 80%; padding-left: 5px;">
            <%=# Eval("UnitsInStock") %>
        </td>
    </tr>
    <tr>
        <td>
            Discontinued:
        </td>
        <td style="width: 80%; padding-left: 5px;">
            <asp:CheckBox ID="CheckBox1" runat="server" Checked='<%=# Eval
("Discontinued") %>'
                Enabled="false" />
        </td>
    </tr>
    <tr>
        <td colspan="2">
            <asp:Button ID="Button1" runat="server" CommandName="Edit"
Text="Edit" CausesValidation="false" />
            &nbsp;
            <asp:Button ID="Button2" runat="server" CommandName="Delete"
OnClick="ConfirmDelete(this);
                return false;" Text="Delete" />
        </td>
    </tr>
</table>
</fieldset>
</ItemTemplate>
<EditItemTemplate>
    <fieldset style="float: left; width: 223px; height: 160px;">
        <table cellpadding="0" cellspacing="2" style="height: 100%">

```

```

        <tr>
            <td style="width: 20%;">
                Name:
            </td>
            <td style="width: 80%; padding-left: 5px;">
                <asp:TextBox ID="TextBox1" runat="server" Text='<# Bind
("ProductName") %>' Width="120px">
                </asp:TextBox>
                <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server" ErrorMessage="*"
                    ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
            </td>
        </tr>
        <tr>
            <td>
                Quantity:
            </td>
            <td style="width: 80%; padding-left: 5px;">
                <asp:TextBox ID="TextBox2" runat="server" Text='<# Bind
("QuantityPerUnit") %>' Width="120px">
                </asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                Price:
            </td>
            <td style="width: 80%; padding-left: 5px;">
                <asp:TextBox ID="TextBox3" runat="server" Text='<# Bind("UnitPric
%>' Width="65px"></asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                Units:
            </td>
            <td style="width: 80%; padding-left: 5px;">
                <asp:TextBox ID="TextBox4" runat="server" Text='<# Bind
("UnitsInStock") %>' Width="65px">
                </asp:TextBox>
            </td>
        </tr>
        <tr>
            <td>
                Discontinued:
            </td>
            <td style="width: 80%; padding-left: 5px;">
                <asp:CheckBox ID="CheckBox1" runat="server" Checked='<# Bind
("Discontinued") %>' />
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <asp:Button ID="Button1" runat="server" CommandName="Update"
Text="Update" />
            </td>
        </tr>
    </table>

```

```

        <asp:Button ID="Button2" runat="server" CommandName="Cancel"
Text="Cancel" CausesValidation="false" />
    </td>
</tr>
</table>
</fieldset>
</EditItemTemplate>
<InsertItemTemplate>
    <fieldset style="float: left; width: 223px; height: 160px;">
        <table cellpadding="0" cellspacing="2" style="height: 100%">
            <tr>
                <td style="width: 20%;">
                    Name:
                </td>
                <td style="width: 80%; padding-left: 5px;">
                    <asp:TextBox ID="TextBox1" runat="server" Width="120px" Text='<#
Bind("ProductName") %>'>
                    </asp:TextBox>
                    <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server" ErrorMessage="*"
                    ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
                </td>
            </tr>
            <tr>
                <td>
                    Quantity:
                </td>
                <td style="width: 80%; padding-left: 5px;">
                    <asp:TextBox ID="TextBox2" runat="server" Width="120px" Text='<#
Bind("QuantityPerUnit") %>'>
                    </asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                    Price:
                </td>
                <td style="width: 80%; padding-left: 5px;">
                    <asp:TextBox ID="TextBox3" runat="server" Width="65px" Text='<#
("UnitPrice") %>'></asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>
                    Units:
                </td>
                <td style="width: 80%; padding-left: 5px;">
                    <asp:TextBox ID="TextBox4" runat="server" Width="65px" Text='<#
("UnitsInStock") %>'>
                    </asp:TextBox>
                </td>
            </tr>
            <tr>
                <td>

```

```

                Discontinued:
            </td>
            <td style="width: 80%; padding-left: 5px;">
                <asp:CheckBox ID="CheckBox1" runat="server" Checked='<%= Bind
("Discontinued") %>' />
            </td>
        </tr>
        <tr>
            <td colspan="2">
                <asp:Button ID="Button1" runat="server" CommandName="PerformInsert"
Text="Insert" />
                <asp:Button ID="Button2" runat="server" CommandName="Cancel"
Text="Cancel" CausesValidation="false" />
            </td>
        </tr>
    </table>
</fieldset>
</InsertItemTemplate>
</telerik:RadListView>

```

3. Wrap the RadListView into RadAjaxPanel to omit the page flickering when RadListView items are changing their modes.
4. Add RadWindowManager controls at the bottom of the page. We will use it to show RadConfirm when deleting an item.
5. To bind the grid we will handle the NeedDataSource event of the RadListView:

C#

```

protected void RadListView1_NeedDataSource(object sender, RadListViewNeedDataSourceEvent/
e)
{
    DataTable listViewDataSource;
    using (SqlConnection sqlConnection1 = new SqlConnection
(ConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString))
    {
        listViewDataSource = new DataTable();
        //Select Query to populate the RadGrid with data from table Customers.
        const string selectQuery = "SELECT * FROM [Products]";
        using (SqlDataAdapter sqlDataAdapter = new SqlDataAdapter())
        {
            sqlDataAdapter.SelectCommand = new SqlCommand(selectQuery, sqlConnection1);
            sqlDataAdapter.Fill(listViewDataSource);
        }
    }
    RadListView1.DataSource = listViewDataSource;
}

```

VB.NET

```

Protected Sub RadListView1_NeedDataSource(ByVal sender As Object, ByVal e As
RadListViewNeedDataSourceEventArgs)
    Dim listViewDataSource As DataTable
    Using sqlConnection1 As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
        listViewDataSource = New DataTable()
        'Select Query to populate the RadGrid with data from table Customers.

```

```

Const selectQuery As String = "SELECT * FROM [Products]"
Using sqlDataAdapter As New SqlDataAdapter()
    sqlDataAdapter.SelectCommand = New SqlCommand(selectQuery, sqlConnection1)
    sqlDataAdapter.Fill(listViewDataSource)
End Using
End Using
RadListView1.DataSource = listViewDataSource
End Sub

```

6. And to perform the database operations, we will implement the ItemCommand event handler:

C#

```

protected void RadListView1_ItemCommand(object sender, RadListViewCommandEventArgs e)
{
    if (e.CommandName == RadListView.UpdateCommandName)
    {
        RadListViewDataItem editedItem = e.ListViewItem as RadListViewDataItem;
        string productID = editedItem.GetDataKeyValue("ProductID").ToString();
        Hashtable newValues = new Hashtable();
        editedItem.ExtractValues(newValues);
        SqlConnection sqlConnection1 = new SqlConnection
(ConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString);
        try
        {
            const string updateQuery = "UPDATE [Products] SET [ProductName] = @ProductName
[QuantityPerUnit] = @QuantityPerUnit"
+ ",[UnitPrice] = @UnitPrice, [UnitsInStock] = @UnitsInStock, [Discontinued]
= @Discontinued WHERE [ProductID]=@ProductID";
            SqlCommand updateCommand = new SqlCommand(updateQuery, sqlConnection1);
            updateCommand.Parameters.AddWithValue("ProductID", productID);
            updateCommand.Parameters.AddWithValue("ProductName", newValues["ProductName"]);
            updateCommand.Parameters.AddWithValue("QuantityPerUnit", newValues
["QuantityPerUnit"]);
            updateCommand.Parameters.AddWithValue("UnitPrice", newValues["UnitPrice"]);
            updateCommand.Parameters.AddWithValue("UnitsInStock", newValues["UnitsInStock"]);
            updateCommand.Parameters.AddWithValue("Discontinued", newValues["Discontinued"]);
            sqlConnection1.Open();
            updateCommand.ExecuteNonQuery();
        }
        catch
        {
            e.Canceled = true;
        }
        finally
        {
            sqlConnection1.Close();
        }
    }
    if (e.CommandName == RadListView.PerformInsertCommandName)
    {
        RadListViewEditableItem insertedItem = (RadListViewEditableItem)e.ListViewItem;
        Hashtable newValues = new Hashtable();
        insertedItem.ExtractValues(newValues);
        SqlConnection sqlConnection1 = new SqlConnection
(ConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString);
        try

```

```

        {
            const string insertQuery = "INSERT INTO [Products] ([ProductName],
[QuantityPerUnit], [UnitPrice], [UnitsInStock], [Discontinued])"
                + "VALUES (@ProductName, @QuantityPerUnit, @UnitPrice, @UnitsInStock,
@Discontinued)";
            SqlCommand insertCommand = new SqlCommand(insertQuery, sqlConnection1);
            insertCommand.Parameters.AddWithValue("ProductName", newValues["ProductName"]);
            insertCommand.Parameters.AddWithValue("QuantityPerUnit", newValues
["QuantityPerUnit"]);
            insertCommand.Parameters.AddWithValue("UnitPrice", newValues["UnitPrice"]);
            insertCommand.Parameters.AddWithValue("UnitsInStock", newValues["UnitsInStock"]);
            insertCommand.Parameters.AddWithValue("Discontinued", newValues["Discontinued"]);
            sqlConnection1.Open();
            insertCommand.ExecuteNonQuery();
            RadListView1.InsertItemPosition = RadListViewInsertItemPosition.None;
        }
        catch
        {
            e.Canceled = true;
        }
        finally
        {
            sqlConnection1.Close();
        }
    }
    if (e.CommandName == RadListView.DeleteCommandName)
    {
        RadListViewDataItem item = e.ListViewItem as RadListViewDataItem;
        string productID = item.GetDataKeyValue("ProductID").ToString();
        SqlConnection sqlConnection1 = new SqlConnection
(ConfigurationManager.ConnectionStrings["NorthwindConnectionString"].ConnectionString);
        try
        {
            const string deleteQuery = "DELETE FROM [Products] WHERE [ProductID]=@ProductID";
            SqlCommand deleteCommand = new SqlCommand(deleteQuery, sqlConnection1);
            deleteCommand.Parameters.AddWithValue("ProductID", productID);
            sqlConnection1.Open();
            deleteCommand.ExecuteNonQuery();
            sqlConnection1.Close();
        }
        finally
        {
            sqlConnection1.Close();
        }
    }
}
}
}

```

VB.NET

```

Protected Sub RadListView1_ItemCommand(ByVal sender As Object, ByVal e As
RadListViewCommandEventArgs)
    If e.CommandName = RadListView.UpdateCommandName Then
        Dim editedItem As RadListViewDataItem = TryCast(e.ListViewItem, RadListViewDataItem)
        Dim productID As String = editedItem.GetDataKeyValue("ProductID").ToString()
        Dim newValues As New Hashtable()
    End If
End Sub

```

```

        editedItem.ExtractValues(newValues)
        Dim sqlConnection1 As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
        Try
            Const updateQuery As String = "UPDATE [Products] SET [ProductName] =
@ProductName, [QuantityPerUnit] = @QuantityPerUnit" + ",[UnitPrice] = @UnitPrice,
[UnitsInStock] = @UnitsInStock, [Discontinued] = @Discontinued WHERE [ProductID]=@ProductID"
            Dim updateCommand As New SqlCommand(updateQuery, sqlConnection1)
            updateCommand.Parameters.AddWithValue("ProductID", productID)
            updateCommand.Parameters.AddWithValue("ProductName", newValues("ProductName");
            updateCommand.Parameters.AddWithValue("QuantityPerUnit", newValues
("QuantityPerUnit"))
            updateCommand.Parameters.AddWithValue("UnitPrice", newValues("UnitPrice"))
            updateCommand.Parameters.AddWithValue("UnitsInStock", newValues("UnitsInStock"))
            updateCommand.Parameters.AddWithValue("Discontinued", newValues("Discontinued"))
            sqlConnection1.Open()
            updateCommand.ExecuteNonQuery()
        Catch
            e.Canceled = True
        Finally
            sqlConnection1.Close()
        End Try
    End If
    If e.CommandName = RadListView.PerformInsertCommandName Then
        Dim insertedItem As RadListViewEditableItem = DirectCast(e.ListViewItem,
RadListViewEditableItem)
        Dim newValues As New Hashtable()
        insertedItem.ExtractValues(newValues)
        Dim sqlConnection1 As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)
        Try
            Const insertQuery As String = "INSERT INTO [Products] ([ProductName],
[QuantityPerUnit], [UnitPrice], [UnitsInStock], [Discontinued])" + "VALUES (@ProductName,
@QuantityPerUnit, @UnitPrice, @UnitsInStock, @Discontinued)"
            Dim insertCommand As New SqlCommand(insertQuery, sqlConnection1)
            insertCommand.Parameters.AddWithValue("ProductName", newValues("ProductName");
            insertCommand.Parameters.AddWithValue("QuantityPerUnit", newValues
("QuantityPerUnit"))
            insertCommand.Parameters.AddWithValue("UnitPrice", newValues("UnitPrice"))
            insertCommand.Parameters.AddWithValue("UnitsInStock", newValues("UnitsInStock"))
            insertCommand.Parameters.AddWithValue("Discontinued", newValues("Discontinued"))
            sqlConnection1.Open()
            insertCommand.ExecuteNonQuery()
            RadListView1.InsertItemPosition = RadListViewInsertItemPosition.None
        Catch
            e.Canceled = True
        Finally
            sqlConnection1.Close()
        End Try
    End If
    If e.CommandName = RadListView.DeleteCommandName Then
        Dim item As RadListViewDataItem = TryCast(e.ListViewItem, RadListViewDataItem)
        Dim productID As String = item.GetDataKeyValue("ProductID").ToString()
        Dim sqlConnection1 As New SqlConnection(ConfigurationManager.ConnectionStrings
("NorthwindConnectionString").ConnectionString)

```

```
Try
  Const deleteQuery As String = "DELETE FROM [Products] WHERE [ProductID]
=@ProductID"
  Dim deleteCommand As New SqlCommand(deleteQuery, sqlConnection1)
  deleteCommand.Parameters.AddWithValue("ProductID", productID)
  sqlConnection1.Open()
  deleteCommand.ExecuteNonQuery()
  sqlConnection1.Close()
Finally
  sqlConnection1.Close()
End Try
End If
End Sub
```



You can find the complete source for this project at:
\\VS Projects\ListView\ManualCRUDOperations

26.6 RadDataPager

Use **RadDataPager** to display paging navigation controls for other data-bound controls that implement the **IPageableItemContainer** or **IRadPageableItemContainer** interface (like the **RadListView** and **MS ListView**). You can easily add the **RadDataPager** control to a Web Form within Visual Studio. The paging interface appears wherever you place the **RadDataPager** control on the page. You may place it before or after the **RadListView** control, as well as within its **LayoutTemplate** element.



To use the **RadDataPager** control in its default state you can refer to the following properties:

- **PagedControlID** is the ID of the control that implements one of the following interfaces - **IPageableItemContainer** or **IRadPageableItemContainer**. This is the control that will be paged by **RadDataPager** control. If **RadDataPager** is placed in **Controls** collection of **IPageableItemContainer** / **IRadPageableItemContainer** setting this property is optional. In case **PagedControlID** is not set **RadDataPager** will attempt to find its container automatically.
- **PageSize** is the number of items and rows to display on each page.
- **StartRowIndex** gets the index of the first record that is displayed on a page of data.
- **TotalRowCount** gets the total number of records that are displayed in the underlying data source.
- **MaximumRows** gets the maximum number of records that are displayed for each page of data.

The **RadDataPager** fields lets you choose the controls that will appear in the pager field to help users navigate through the pages. To specify the pager fields, list the desired field elements between the opening and closing **<Fields>** tag inside the **RadDataPager** control.

The **RadDataPager** built-in fields are: **RadDataPagerButtonField**, **RadDataPagerPageSizeField**, **RadDataPagerSliderField**, **RadDataPagerGoToPageField** and **RadDataPagerTemplatePageField**. You can use one or more pager field objects in a single **RadDataPager** control.

For example when **RadDataPager** contains a **RadDataPagerButtonField** you have the ability to add arrow buttons for navigation to **Next/Previous/First/Last** page, link buttons with page numbers or both.

ASPX

```
<telerik:RadDataPager ID="RadDataPager1" PagedControlID="RadListView1" PageSize="2"
```

```

runat="server">
  <Fields>
    <telerik:RadDataPagerButtonField FieldType="FirstPrev" FirstButtonText="First"
PrevButtonText="Prev" />
    <telerik:RadDataPagerButtonField FieldType="Numeric" PageButtonCount="5" />
    <telerik:RadDataPagerButtonField FieldType="NextLast" NextButtonText="Next"
LastButtonText="Last" />
  </Fields>
</telerik:RadDataPager>

```

You can also create custom paging UI by using the `RadDataPagerTemplatePageField` object. You can use it to display custom navigation controls and show information about the underlying data source, such as total number of records and the current page number. The `RadDataPagerTemplatePageField` has no built-in layout. Therefore, you must explicitly create the layout in its `PagerTemplate`. You can format the content by using cascading style sheets (CSS) classes or inline style elements. You can reference the `RadDataPager` control that contains the `RadDataPagerTemplatePageField` object by using the `Container.Owner` property.

The following example shows how to add a `RadDataPagerTemplatePageField`, which contains `RadComboBox` and custom button for changing the current page size in a `RadDataPager` control.

ASPX

```

<telerik:RadDataPager runat="server" ID="RadDataPager1" PagedControlID="ListView1">
  <Fields>
    <telerik:RadDataPagerTemplatePageField>
      <PagerTemplate>
        <asp:Button runat="server" ID="CustomButton" Text="My custom button"
CommandName="Custom Command Name" />
        <telerik:RadComboBox runat="server" ID="RadComboBox1" AutoPostBack="true"
SelectedValue='<%#Container.Owner.PageSize %>'
OnSelectedIndexChanged="RadComboBox1_SelectedIndexChanged">
          <Items>
            <telerik:RadComboBoxItem Text="15" Value="15" />
            <telerik:RadComboBoxItem Text="30" Value="30" />
            <telerik:RadComboBoxItem Text="60" Value="60" />
          </Items>
        </telerik:RadComboBox>
      </PagerTemplate>
    </telerik:RadDataPagerTemplatePageField>
  </Fields>
</telerik:RadDataPager>

```

C#

```

protected void RadComboBox1_SelectedIndexChanged(object o,
RadComboBoxSelectedIndexChangedEventArgs e)
{
    var combo = o as RadComboBox;
    (combo.NamingContainer as RadDataPagerFieldItem).Owner.PageSize = int.Parse(e.Value);
}

```

VB.NET

```

Protected Sub RadComboBox1_SelectedIndexChanged(o As Object, e As
RadComboBoxSelectedIndexChangedEventArgs)
    Dim combo As var = TryCast(o, RadComboBox)
    (TryCast(combo.NamingContainer, RadDataPagerFieldItem)).Owner.PageSize = Integer.Parse
(e.Value)
End Sub

```

In addition, you can use SEO paging of the RadDataPager control. To use this pager functionality you need to set AllowSEOPaging property to True. When it is False, the RadDataPager does not use SEO paging. To specify the query page key for the grid that is used as part of the page query you can set the SEOPagingQueryPageKey property. This is useful when the data pager resides in several containers and its id becomes too long and not very readable. Using SEOPagingQueryPageKey property you get many more search engine optimized links to the other pages.

26.7 Summary

In this chapter we looked at the RadListView control and explored its most commonly used features like paging, sorting, filtering, grouping, items selection and drag and drop.

Learned how to use RadListView in Design Time and build its layout with ease.

You saw how to implement a sample project on how to manipulate the data with RadListView.

Finally, we described the RadDataPager control and how to use it for paging navigation in data-bound controls.

27 RadNotification

27.1 Objectives

- Explore the main features of the RadNotification control
- Getting started by running a simple example
- Review the most important properties and how they are used together to control the RadNotification's behavior
- Review the basic ways to populate content in the notification
- Examine the extra built-in functionality - context menus and icons
- See the use of a lightweight callback to update the content
- Examine a real-life scenario where the user is updated about a server-side event occurring

27.2 Introduction

The RadNotification is a very light control which can be used to display a notification message from both the server and the client. The notification is completely customizable, can be loaded on demand through a callback or WebService, can be automatically displayed and/or updated at specific intervals and supports different animation effects and at different positions.

Its most notable features are:

- Semantic rendering - no HTML tables used
- Load on Demand through callback or WebService
- Built-in, fully customizable context menu
- Can contain simple text, HTML content and ASP.NET controls
- Automatically calculates position relative to the screen
- Automatic updates at a specified interval
- Automatic show at a specified interval
- Keep on mouse over
- AutoClose Delay
- Animation effects
- Content scrolling
- Advanced Skinning

27.3 Getting Started

The following tutorial demonstrates how to add a **RadNotification** to the page and have it show once the page is loaded:

1. In a new AJAX-Enabled Web Application drop a **RadNotification** from the ToolBox to the default web page.
2. Use either the Properties pane or by writing directly in the markup to set the following properties:
 1. Set the **VisibleOnPageLoad** property to true.

2. Set the **Position** property to Center.
 3. Set the **Text** property to Sample notification text.
 4. Set the **Width** property to 250px.
 5. Set the **Height** property to 100px.
3. Optionally you may also:
1. Set the **Title** property to Title.
 2. Set the **EnableRoundedCorners** property to true.
 - c. Set the **EnableShadow** property to true.
4. Press F5 to run the application. You will see a simple popup in the center of the browser.

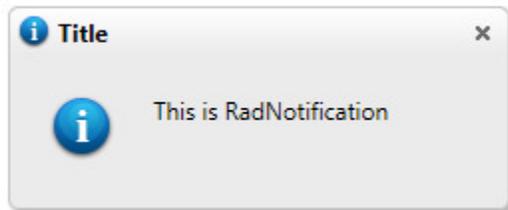
Your markup should look similar to this:

RadNotification's declaration

```
<telerik:RadNotification runat="server" ID="RadNotification1" VisibleOnPageLoad="true"
Position="Center" Text="Sample notification text"
Width="250px" Height="100px" Title="Title" EnableShadow="true"
EnableRoundedCorners="true">
</telerik:RadNotification>
```

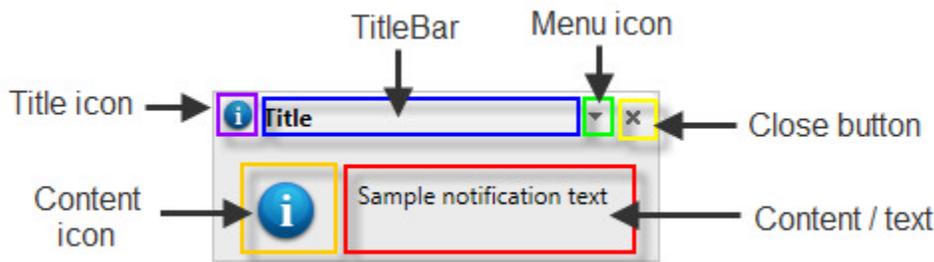
Please note that the **EnableRoundedCorners** and **EnableShadow** properties use CSS3 to create these effects, thus if you are using an old browser (most notably IE versions prior to IE9) these properties will have no effect.

On your screen you should see a message box similar to the following image flash when the page is opened:

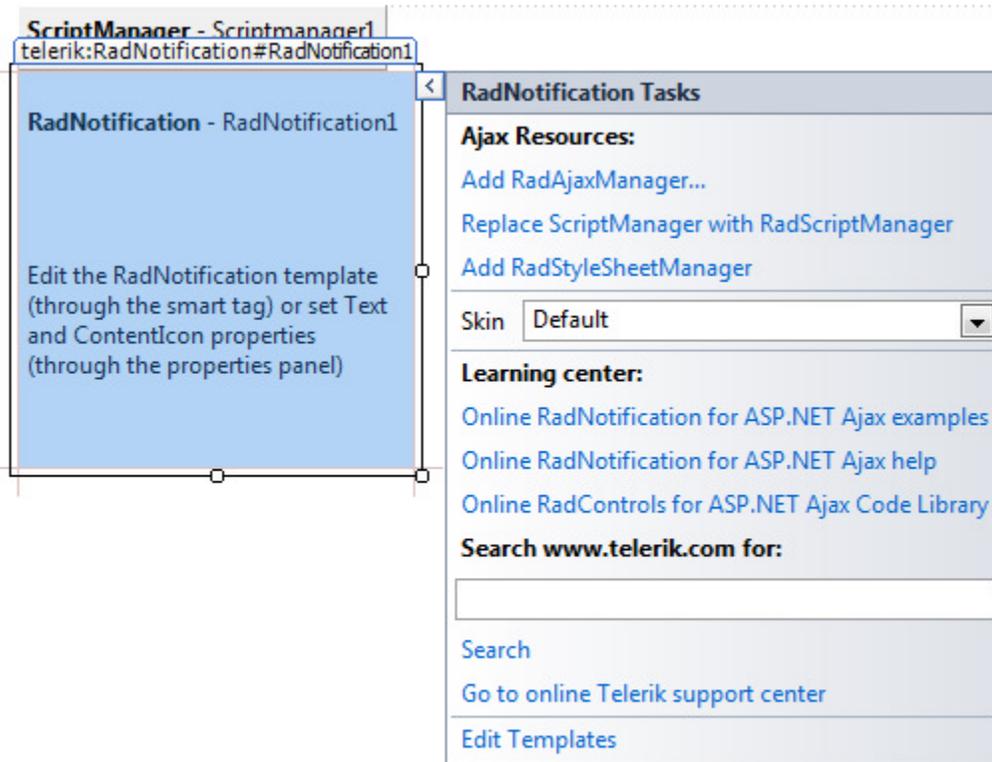


The RadNotification's visual appearance is defined by several elements:

- **TitleBar:** This is the title of the notification.
- **TitleIcon:** The small image (16x16 pixels) shown in the titlebar.
- **Menu Icon:** The button used to show the title menu. It appears only if the ShowTitleMenu property is set to true.
- **Close Button:** The button used to close the notification. It appears only if the ShowCloseButton property is set to true.
- **ContentIcon:** The image (32x32 pixels) shown in the content area of the notification.
- **Content or Text:** This is the main part of the control. It can be customized using Text property or by declaring content between the RadNotification's ContentTemplate tags in ASP.NET.



You can use the **RadNotification's SmartTag** to add the needed AJAX resources to the web application (ScriptManager or RadScriptManager, RadAjaxManager, RadStyleSheetManager) or edit the content template of the RadNotification in the Design-time mode of the Visual Studio:



27.4 Notification Menu

RadNotification offers a built-in context menu which can be used to extend the control and to attach some custom functionality. You can use it like the regular **RadContextMenu** - it supports multiple targets, fully customizable layout, many client-side events and more - essentially the full functionality of the **RadContextMenu** is employed in **RadNotification**.

If the **ShowTitleMenu** property is set to true the menu icon will appear next to the close button in the titlebar. You can use both the left and the right mouse buttons to invoke this menu.

To activate the Notification Menu you need to declare some items in it and also a target if you are not going to use the `ShowTitleMenu` property. In this case the `ShowTitleMenu` property is used:

Adding options to the context menu of RadNotification

```
<telerik:RadNotification runat="server" ID="RadNotification1" VisibleOnPageLoad="true"
Position="Center" Width="300px" Height="120px"
  ShowTitleMenu="true" Text="Sample notification text. Use the menu for more options"
AutoCloseDelay="0" ShowCloseButton="false">
  <NotificationMenu OnClientItemClicked="OnClientItemClicked">
    <Items>
      <telerik:RadMenuItem Text="Close the notification" Value="1">
      </telerik:RadMenuItem>
      <telerik:RadMenuItem Text="Get more information" value="2">
      </telerik:RadMenuItem>
    </Items>
  </NotificationMenu>
</telerik:RadNotification>
<script type="text/javascript">
  function OnClientItemClicked(sender, args)
  {
    var itemValue = args.get_item().get_value();
    switch (itemValue)
    {
      case "1": $find("RadNotification1").hide(); break;
      case "2": window.open("http://google.com (http://google.com/)"); break;
      default: break;
    }
  }
</script>
```

This small example shows how to extend the functionality of the **RadNotification** by adding options to the menu and executing different actions according to the item that has been clicked.

27.5 Embedded Icons

The **RadNotification** control provides two icons in its UI to help convey the message - the **TitleIcon** and **ContentIcon**. The **TitleIcon**'s size is 16 by 16 pixels and the **ContentIcon**'s size is 32 by 32 pixels. You can set these properties to an URL that points to the icon that is to be shown in the respective place. It also has a number of built-in icons that are accessed by typing a simple word instead of an URL.

By default the info icon is used if nothing else is specified. If you do not want to use any icons set these properties to an empty string. The built-in icons are designed to match the skin of the control and are therefore different for different skins. The following list shows the available built-in icons:

- Info
- Delete
- Deny
- Edit
- Ok
- Warning
- None

27.6 Different Ways to Show A Notification

The **RadNotification** provides both a server-side and a client-side method **show()** that can be used to make it pop up for the user. This is designed to allow it to cover more scenarios, including ones that determine if it should be shown on the server, as well as on the client. The client-side **show()** method will force the notification regardless of the **ShowInterval** property and the server-side **Show()** method can be invoked even from inside an AJAX request that does not update the notification control.

The following simple example uses JavaScript to show the notification on a click of a button:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" Text="You just clicked the
button." Width="250px" Height="110px"
    Position="Center">
</telerik:RadNotification>
<asp:Button ID="Button1" Text="show a notification" runat="server"
OnClick="showNotification();" />
<script type="text/javascript">
    function showNotification()
    {
        $find("RadNotification1").show();
    }
</script>
```

This example simulates some server-side logic that will determine if a notification should be shown, and if yes, calls its **Show()** method:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" Text="You just clicked the
button." Width="250px" Height="110px"
    Position="Center">
</telerik:RadNotification>
<asp:Button ID="Button1" Text="show a notification" runat="server"
OnClick="Button1_Click" />
```

[C# codebehind]

```
protected void Button1_Click(object sender, EventArgs e)
{
    bool toShowNotification = true;
    if(toShowNotification)
    {
        RadNotification1.Show();
    }
}
```

[VB.NET codebehind]

```
Protected Sub Button1_Click(sender As Object, e As System.EventArgs) Handles
Button1.Click
    Dim toShowNotification As Boolean = True
```

```
If toShowNotification Then
    RadNotification1.Show()
End If
End Sub
```

27.7 Populating Plain Text And Rich Content

There are several ways to set content in the **RadNotification** control. The easiest of them is by using its **Title** and **Text** properties. You can also use the **ContentIcon** and **TitleIcon** (Section 27.5) properties to add a visual emphasis to the message.

You can also add rich content (i.e. server controls for example) in its **ContentTemplate** - both declaratively in the markup and dynamically in the code-behind.

When the **Title** property is set the string is placed in the **TitleBar** of the **RadNotification** and is always visible if the **TitleBar** is visible. If some content is set in the **ContentTemplate** (or added in the code-behind) it has greater priority than the **Text** and therefore the text is not shown. If rich content is added in the **RadNotification** the **ContentIcon** is not shown as well.

The below example shows the effect of these properties:

In this example there will be text, title and two icons in the notification. It also shows that you can set them dynamically in the code-behind:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" Width="250px"
Height="110px" VisibleOnPageLoad="true">
</telerik:RadNotification>
```

[C# codebehind]

```
protected void Page_Load(object sender, EventArgs e)
{
    RadNotification1.Text = "Sample Notification text";
    RadNotification1.TitleIcon = "info";
    RadNotification1.ContentIcon = "info";
}
```

[VB.NET codebehind]

```
Protected Sub Page_Load(sender As Object, e As System.EventArgs) Handles Me.Load
    RadNotification1.Text = "Sample Notification text"
    RadNotification1.TitleIcon = "info"
    RadNotification1.ContentIcon = "info"
End Sub
```

The following example shows how to add more complex content in the **RadNotification**:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification2" VisibleOnPageLoad="true"
  TitleIcon="info" ContentIcon="info" Width="250px" Height="100px">
  <ContentTemplate>
    Rich content:<br />
    <asp:Button ID="Button1" Text="Button in a notification" runat="server" />
  </ContentTemplate>
</telerik:RadNotification>
```

You can do this in the code-behind as well, but then you would need to have an empty **ContentTemplate** declared in the markup, so that the control may know that this template will be needed in order to create it. This example also shows that rich content hides the **ContentIcon**:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" VisibleOnPageLoad="true"
  Width="250px" Height="100px">
  <ContentTemplate>
  </ContentTemplate>
</telerik:RadNotification>
```

[C# codebehind]

```
protected void Page_Load(object sender, EventArgs e)
{
    RadNotification1.ContentContainer.Controls.Add(new LiteralControl("Rich
content:<br />"));
    Button button = new Button();
    button.ID = "Button1";
    button.Text = "Button in a notification";
    RadNotification1.ContentContainer.Controls.Add(button);
    RadNotification1.TitleIcon = "info";
    RadNotification1.ContentIcon = "info";
}
```

[VB.NET codebehind]

```
Protected Sub Page_Load(sender As Object, e As System.EventArgs) Handles Me.Load
    RadNotification1.ContentContainer.Controls.Add(New LiteralControl("Rich
content:<br />"))
    Dim button As New Button()
    button.ID = "Button1"
    button.Text = "Button in a notification"
    RadNotification1.ContentContainer.Controls.Add(button)
    RadNotification1.TitleIcon = "info"
    RadNotification1.ContentIcon = "info"
End Sub
```

This last example shows that the **Text** property is overridden by the other content:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" VisibleOnPageLoad="true"
    TitleIcon="info" ContentIcon="info" Width="250px" Height="100px" Text="Sample
notification text">
    <ContentTemplate>
        Rich content:<br />
        <asp:Button ID="Button1" Text="Button in a notification" runat="server" />
    </ContentTemplate>
</telerik:RadNotification>
```

27.8 Callback Support

You can use the built-in **OnCallbackUpdate** event of the **RadNotification** to load its content from the server via a callback. It is especially useful in combination with the **LoadContentOn** property set to **EveryShow**. The key advantage of using a callback is that the server Page does not go through its whole lifecycle, but only a small part of it. The client state is not updated, and it is not sent back to the client-side.

You can use this callback to set the notification's **Value** property, because it will be passed to the client. You can use it, for example, as a flag or some small piece of necessary data in your application's logic. In the following example the **Value** property is used as a flag to determine if the notification should be shown, according to the current time:

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" LoadContentOn="TimeInterval"
UpdateInterval="5400"
    AutoCloseDelay="2400" Position="BottomRight" Width="250px" Height="120px"
    OnCallbackUpdate="OnCallbackUpdate" OnClientUpdated="OnClientUpdated">
</telerik:RadNotification>
<script type="text/javascript">
    function OnClientUpdated(sender, args)
    {
        var theValue = sender.get_value();
        if (theValue != "0")
        {
            sender.show();
        }
    }
</script>
```

[C# codebehind]

```
Random rnd = new Random();
protected void OnCallbackUpdate(object sender, RadNotificationEventArgs e)
{
    int newMsgs = rnd.Next(0, 11);
    if (newMsgs == 5 || newMsgs == 7 || newMsgs == 8 || newMsgs == 9)
    {
        newMsgs = 0;
    }
    RadNotification1.Value = newMsgs.ToString();
    RadNotification1.Text = "You have " + newMsgs + " new messages!";
}
```

[VB.NET codebehind]

```

Dim rnd As New Random()
    Protected Sub RadNotification1_CallbackUpdate(sender As Object, e As
Telerik.Web.UI.RadNotificationEventArgs) Handles RadNotification1.CallbackUpdate
        Dim newMsgs As Integer = rnd.[Next](0, 11)
        If newMsgs = 5 OrElse newMsgs = 7 OrElse newMsgs = 8 OrElse newMsgs = 9 Then
            newMsgs = 0
        End If
        RadNotification1.Value = newMsgs.ToString()
        RadNotification1.Text = "You have " & newMsgs & " new messages!"
    End Sub

```

27.9 How To Combine Properties

The following sets of properties are best used together to configure some aspects of the RadNotification's appearance and behavior:

- Use the **Position**, **OffsetX**, **OffsetY** properties to control the position in which the notification is shown
- Use the **Animation**, **AnimationDuration** properties to control the way in which the notification is initially shown
- Use the **ContentScrolling** property to customize the availability of scrollbars. Possible values are Auto, None X, Y and Both. This requires the content to have explicit dimensions set in pixels to work correctly
- Use the **ShowCloseButton**, **ShowTitleMenu**, **VisibleTitlebar**, **TitleIcon** properties to control the behavior of the titlebar
- Use the **AutoCloseDelay**, **KeepOnMouseOver**, **ShowInterval** properties to control the time, after which the notification will automatically show/hide
- Use the **LoadContentOn**, **UpdateInterval**, **ShowInterval** properties to control when new content is loaded. They are most often used in a more advanced scenario when a Callback is used to fetch the content
- Use the **Text**, **Title**, **ContentIcon**, **WebMethodName**, **WebMethodpath**, **OnCallbackUpdate** properties to control how the content is loaded. For more information refer to the **Populating Plain Text and Rich Content (Section 27.7)** section
- You can declare a custom menu between the **NotificationMenu** tags
- You can change the overall look and feel of the notification by changing its skin via the **Skin**, **EnableShadow**, **EnableRoundedCorners** properties.

The **Position** property gets/sets the top/left position of the notification relative to the browser. Its value is an enumerator with the following options: TopLeft, TopCenter, TopRight, MiddleLeft, Center, MiddleRight, BottomLeft, BottomCenter, BottomRight. The default is BottomRight.

OffsetX and **OffsetY** are used for fine-tuning the horizontal and vertical offset from the designated Position. Their values are set in pixels.

The **Animation** property is used to get/set the animation effect of the notification. It is also an enumerator with the following options: None, Resize, Fade, Slide, FlyIn. The default is None.

The **TitleIcon** and **ContentIcon** are used to set the icons that are shown in the notification. Set them to an empty string to disable the icon. For more information see the **Embedded Icons (Section 27.5)** article.

The **NotificationMenu** and the **ContentTemplate** inner tags are used to declare the built in context menu and rich content respectively. For more information see the **Notification Menu (Section 27.4)** and **Populating Plain**

Text and Rich Content (Section 27.7) sections.

The `EnableRoundedCorners` and `EnableShadow` properties are quite self-explanatory, yet it should be noted that they use CSS3 to create these effects; if you are using an old browser (most notably IE versions prior to IE9) these properties will have no effect.

`AutoCloseDelay`, `KeepOnMouseOver`, `ShowInterval` are used to control how long the notification stays opened on the screen (the `AutoCloseDelay` property) and how long it takes before it is shown again (the `ShowInterval` property). The `KeepOnMouseOver` is set to true by default and it stops the `AutoClose` timer while the mouse is over the notification. The `ShowInterval` should be longer than the time it takes for the notification to show and hide again, including the animation's duration.

27.10 Auto Save RadEditor's content and notify the user

This example shows a real-life scenario in which we notify the user of an event that has happened on the server. Here we are going to save the content of a `RadEditor` every 20 seconds automatically and show a `RadNotification` to the user when this happens. The `OnClientLoad` event of the `RadEditor` is used to stop the autosaving when the user is performing a spell check.

ASPX/ASCX

```
<telerik:RadNotification runat="server" ID="RadNotification1" Width="250px" Height="120px"
Text="The RadEditor's content has been saved"
    KeepOnMouseOver="false" Position="BottomCenter"></telerik:RadNotification>
<telerik:RadEditor runat="server" ID="RadEditor1" OnClientLoad="OnClientLoad">
    <Content>
        Some example content that you can modify.
    </Content>
</telerik:RadEditor>
<asp:Timer ID="Timer1" runat="server" Interval="20000" OnTick="Timer1_Tick"></asp:Timer>
<asp:UpdatePanel ID="UpdatePanel1" UpdateMode="Conditional" runat="server">
    <Triggers>
        <asp:AsyncPostBackTrigger ControlID="Timer1" />
    </Triggers>
    <ContentTemplate>
        <span style="font-weight:bold;font-size: 15px;margin-bottom: 10px;">The
Saved content will be displayed below:</span>
        <div style="border: solid 3px #d6eefd; padding:5px; ">
            <div style="border: solid 1px black; line-height: 22px; padding: 0
4px">
                <asp:Label runat="server" ID="lb11"/>
            </div>
        </div>
    </ContentTemplate>
</asp:UpdatePanel>
<script type="text/javascript">
function OnClientLoad(sender, args)
{
    var timer = $find("<%=Timer1.ClientID %>");
    //Attach to the spellCheckLoaded event as the spell itself is loaded with AJAX
    sender.add_spellCheckLoaded(function()
    {
        var spell = sender.get_ajaxSpellCheck();
        spell.add_spellCheckStart(function(sender, args)

```

```
        {
            //stop the timer while the spell check is in progress
            timer._stopTimer();
        });

        spell.add_spellCheckEnd(function(sender, args)
        {
            //Restart the timer;
            timer._startTimer();
        });
    }
);
}
</script>
```

[C# codebehind]

```
protected void Timer1_Tick(object sender, EventArgs e)
{
    lbl1.Text = RadEditor1.Content;
    RadNotification1.Show();
}
```

[VB.NET codebehind]

```
Protected Sub Timer1_Tick(sender As Object, e As EventArgs)
    lbl1.Text = RadEditor1.Content
    RadNotification1.Show()
End Sub
```

28 RadCompression

28.1 Objectives

- Learn about the benefits of **RadCompression**
- Learn how to configure the **RadCompression** module

28.2 Introduction

RadCompression is a `HttpModule` that is designed to *automatically* compress your AJAX and Web Service responses. It will intercept the bits that your server is sending back to a browser (or Silverlight-client, for that matter) and compress them. Once the compressed response reaches the browser, standard browser technology takes over and decompresses the response. The compression process is completely transparent to your client-side (JavaScript or Silverlight) and server-side code. It simply reduces the number of bits that must be sent from your server to your client and thus it improves your page performance.

RadCompression is *not* designed to be a complete replacement for the other HTTP compression tools, such as the built-in HTTP Compression in IIS 7. Instead, it is designed to work with those existing tools to cover scenarios they usually miss - namely the compression of the responses of AJAX requests. If you have HTTP Compression enabled in IIS7, you'll discover that it *does not* compress your AJAX and Web Service responses; it only compresses the initial bits sent to the browser when the page is requested. By adding **RadCompression** to your project, you cover those gaps and start compressing your XHR (`XmlHttpRequest`).

So, if **RadCompression** does not cover all HTTP traffic, what does it cover? **RadCompression** will automatically detect and compress requests that expect these content response types (as found in the HTTP request's "Content-Type" header or "AcceptsTypes" header):

- `application/x-www-form-urlencoded`
- `application/json`
- `application/xml`
- `application/atom+xml`
- `text/xml`

28.3 Using RadCompression

Enabling RadCompression

In order to enable **RadCompression** you need to register it as `HttpModule` in your `web.config`.

[web.config]

```
<httpModules>
...
<add name="RadCompression" type="Telerik.Web.UI.RadCompression" />
</httpModules>
<!-- If you're using IIS7, then add this, too-->
<system.webServer>
  <modules>
    ...
    <add name="RadCompression" type="Telerik.Web.UI.RadCompression" />
  </modules>
...

```

ViewState compression

In addition to the default compression mechanism of **RadCompression**, you can specify whether you would like to compress the page ViewState and store it either in a hidden field or in the Session (to pass and retrieve it from there on form submits). For this purpose you can use additional page adapters which override the default page adapter for ViewState storage.

To enable ViewState compression you can register these control adapters in the BrowserFile.browser file under the App_Browsers folder in your web site/project.

[BrowserFile.browser] Storing the compressed ViewState in a hidden field

```
<browsers>
  <browser refID="Default">
    <controlAdapters>
      <adapter controlType="System.Web.UI.Page"
adapterType="Telerik.Web.UI.RadHiddenFieldPageStateCompression" />
    </controlAdapters>
  </browser>
</browsers>
```

[BrowserFile.browser] Storing the compressed ViewState in the Session

```
<browsers>
  <browser refID="Default">
    <controlAdapters>
      <adapter controlType="System.Web.UI.Page"
adapterType="Telerik.Web.UI.RadSessionPageStateCompression" />
    </controlAdapters>
  </browser>
</browsers>
```

Postback compression

You can enable the postback compression by setting the *enablePostbackCompression* property of the **RadCompression** module to true (the default value is *false*). This can be done at application level in the following manner:

[web.config]

```
<configSections>
  ...
  <sectionGroup name="telerik.web.ui">
    <section name="radCompression"
type="Telerik.Web.UI.RadCompressionConfigurationSection, Telerik.Web.UI,
PublicKeyToken=121fae78165ba3d4" allowDefinition="MachineToApplication"
requirePermission="false"/>
  </sectionGroup>
  ...
</configSections>
<telerik.web.ui>
  <radCompression enablePostbackCompression="true"/>
</telerik.web.ui>
```

28.4 Summary

The impact that **RadCompression** has on your site depends on where your users are located. If you have a site that is deployed over the web, where latency and connection speeds are unpredictable, reducing the bytes you send over the wire is an easy way to improve your site's performance. And since **RadCompression** can literally be implemented with a single change to your config file, you really don't have much to lose.

29 RadCaptcha

29.1 Objectives

- Explore features of the RadCaptcha control.
- Learn how to configure RadCaptcha for the runtime environment.
- Explore the RadCaptcha design time interface including the Smart Tag and major property groups.
- Learn how to configuring RadCaptcha for maximum security.
- Configure RadCaptcha audio.

29.2 Introduction



Telerik RadCaptcha is UI control that provides two major strategies for protection against automated form submissions:

- **Image with Modified Symbols (Captcha Image)** - They are displayed in a form, and the user is required to input the symbols in a textbox. The Image is generated with an HttpHandler.
- **Automatic Robots Discovery** - this strategy uses predefined rules which decide whether the input comes from a robot or not. At this point, there are two implemented rules that could be applied either separately or simultaneously.
 - **Minimum form submission time** - the presumption is that a human cannot input the fields in a form correctly for a time less than 3 seconds (this is set by default, and could be modified). If the submission is executed faster than the predefined value, it is assumed that the executor is a robot.
 - **Invisible textbox in the form (the so-called "honeypot")** - this rule requires the insertion of a textbox which is not visible when the form is styled. Still, it will be detected by a robot, and therefore if any data is entered, the executor is considered to be a robot.

Key features:

- **Three Modes for Protection** - you can easily define which strategies to be used for spam protection. These

are: `Captcha`, `InvisibleTextBox` and `MinimumTimeout`.

- **Set Custom Error Message** - the error message that is displayed when the condition being validated fails. Simply set the `ErrorMessage` property of the `RadCaptcha` and the value will be displayed if the page is not valid.
- **Background and Line Noise Level of the Captcha Image** - you can easily control the background and the line noise of the Image by setting the respective value (None, Low, Medium, High or Extreme). The default value of the background and line noise level is Low.
- **Font Family and Font Warp of the Captcha Image** - you can easily choose which font family to be used for the Image text. Courier New is used as a default value for the font family. Furthermore, the amount of random font warping to apply to the rendered text can be changed by setting the `FontWarp` property of the `CaptchaImage`. The default amount of font warping is Low.
- **Text Length and Possible Characters of the Captcha Image** - the default length of the text is 5 characters, and the characters could be either letters or either numeric characters. Alternatively, you can choose what kind of characters to be used (only letters or only numeric characters), and change the length of the text.
- **Maximum Time Interval of the Captcha Image** - the maximum number of minutes the Captcha Image will be cached and valid.
- **Minimum Timeout** - minimum number of seconds the form must be displayed before it is valid. If you're too fast, you must be a robot. This is set when "Minimum form submission time" mode is used for Spam Protection.

29.3 Getting Started

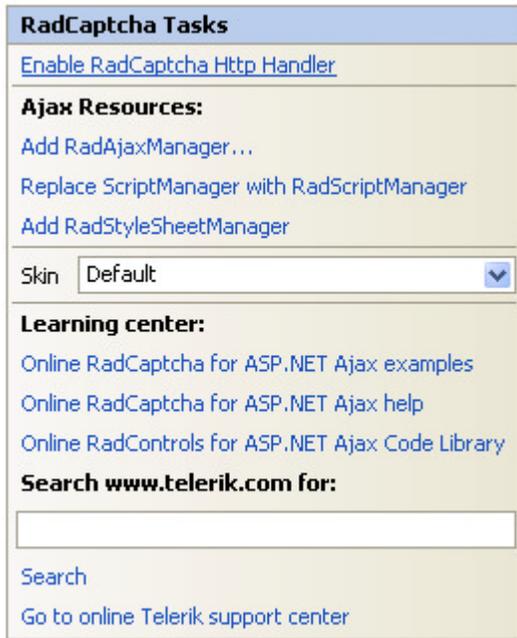
The following tutorial demonstrates using **RadCaptcha** to validate page submission. The walk-through will also show how to display the Error Message in a `ValidationSummary`.

1. In the default page of a new ASP.NET AJAX-enabled Web Application add a **RadCaptcha** control, a `Button` control that causes post back on a click and a `ValidationSummary` control.



Type the code from the image

2. Open the **RadCaptcha** Smart Tag and select the **Enable RadCaptcha** `httpHandler` link. Click OK to close the confirmation dialog for the `RadCaptcha` handler.



The httpHandler can be also enabled by placing the following lines in the web.config file:

web.config

```
<httpHandlers>
  <add path="Telerik.Web.UI.WebResource.axd" type="Telerik.Web.UI.WebResource" verb='
validate="false" />
</httpHandlers>

<handlers>
  <add name="Telerik_Web_UI_WebResource_axd" verb="*" preCondition="integratedMode"
path="Telerik.Web.UI.WebResource.axd" type="Telerik.Web.UI.WebResource" />
</handlers>
```

3. In the Properties Window for the **RadCaptcha** control set the following properties:
4. **ErrorMessage** = You have entered an invalid code.
5. **ValidationGroup** = SubmitGroup
6. In the Properties Window for the ValidationSummary control set the **ValidationGroup** property to the same value as in RadCaptcha (ValidationGroup="SubmitGroup").
Do the same for the Button control.

Here is how the controls' declarations look after setting the above mentioned properties:

Setting Properties

```
<telerik:RadCaptcha ID="RadCaptcha1" runat="server" ErrorMessage="You have entered an
invalid code" ValidationGroup="SubmitGroup"></telerik:RadCaptcha>
<asp:Button ID="Button1" runat="server" Text="Button" ValidationGroup="SubmitGroup" />
<asp:ValidationSummary ID="ValidationSummary1" ValidationGroup="SubmitGroup"
runat="server" />
```

Press F5 to run the Application. **RadCaptcha** validates the input on a post back.

You have entered invalid code.



Type the code from the image

Verify Code

- You have entered invalid code.

29.4 Important Properties

The most important properties of the RadCaptcha control are presented below:

Common properties:

- **ErrorMessage** - The error message text generated when the condition being validated fails.
- **Display** - Gets or sets display behavior of error message. The available modes are:
 - **None** (Validator content never displayed inline)
 - **Static** (Validator content physically part of the page layout)
 - **Dynamic** (Validator content dynamically added to the page when validation fails)
- **ValidatedTextBoxID** - Gets or sets the ID of the textbox to be validated, when only the RadCaptcha image is rendered on the page. To render only the Captchalmage and use Custom TextBox for user input, the Captchalmage-RenderImageOnly property has to be set to true. See the description of the **RenderImageOnly** property below.
- **ValidatedTextBox** - Read-only. Gets the TextBox that is being validated by the RadCaptcha
- **ValidationGroup** - specifies which group of controls is validated on validation

Inner <Captchalmage> tag specific:

- **EnableCaptchaAudio** - Gets or sets the bool value indicating whether the CaptchaAudio will be enabled. When set to true a LinkButton is rendered that, when clicked, retrieves the audio code. Use the **'.rcCaptchaAudioLink'** selector to apply custom skinning to the LinkButton.
- **UseAudioFiles** - Gets or sets a bool value indicating whether the audio code will be generated by concatenation of the audio files from a given folder.
- **AudioFilesPath** - Gets or sets the path to the directory where the audio (.wav) files are located. The default path is **~/App_Data/RadCaptcha** where tilde (~) represents the root of the web application.
- **RenderImageOnly** - Gets or sets bool value that indicates whether the RadCaptcha image will only be rendered on the page (without the CaptchaTextBox and Label). When set to true only the image is

rendered on the page. By setting the “ValidatedTextBoxID” property of the RadCaptcha, the user can choose a custom TextBox where the Captcha code will be entered and validated.

- **ImageStorageLocation** - Gets or sets the storage location for the Captchalmage (see note below):
 - Cache
 - Session

AutoBot Discovery specific Properties

- **InvisibleTextBoxLabel** - Gets or sets the hidden textbox strategy label text.
- **MinTimeout** - Gets or sets the minimum number of seconds form must be displayed before it is valid. If you're too fast, you must be a robot.

Here is a sample declaration for RadCaptcha using some of the properties above:

RadCaptcha declaration

```
<telerik:RadCaptcha ID="RadCaptcha1" runat="server" ErrorMessage="You have entered an
invalid code" ValidationGroup="SubmitGroup">
  <CaptchaImage EnableCaptchaAudio="true" UseAudioFiles="true" />
</telerik:RadCaptcha>
```

29.5 Optimize for Maximum Security

There are some easy takeaways for configuring RadCaptcha for maximum bot blocking:

1. **Don't rely on visual CAPTCHA protection only**
Bots often give away their identity by trying to submit forms too quickly or by trying to submit a form too many times. Take advantage of RadCaptcha's *non-visual protections* (<http://demos.telerik.com/aspnet-ajax/captcha/examples/default/defaultcs.aspx>) to maximize bot prevention.
2. **Maximize Line Noise Level, Eliminate Background Noise Level**
Research says background noise is first thing a CAPTCHA bot throws-out, so it offers little value to your image. Instead, maximize your CAPTCHA image line noise and font warp factor to make segmentation hard for bots. Set properly, RadCaptcha can produce very secure CAPTCHA images like this:



3. **Use a Custom Character Set**
Many bots rely on encountering a predictable set of characters or words to accurately parse a website's CAPTCHA image. By using a *custom character set* (<http://demos.telerik.com/aspnet-ajax/captcha/examples/characterset/defaultcs.aspx>) with RadCaptcha that includes non-alphanumeric characters (like @, !, #, \$), you can increase your odds of beating the bots.

No visual CAPTCHA image is perfect, and with the modern trend of *employing humans to beat CAPTCHAs* (http://www.theregister.co.uk/2008/04/10/web_mail_throttled/), a CAPTCHA is a road bump at best. Still, they prevent the casual spam bot from infiltrating your site and protect your forms from the script kiddies.

Telerik will continue to add improved security features to RadCaptcha in future releases, but by following these simple guidelines, you can confidently get the most value out of a CAPTCHA today that a CAPTCHA can provide.

29.6 Configure RadCaptcha audio

In order for **RadCaptcha** to be accessible by visually impaired users, the control can generate an audio code. To enable this functionality you need to simply set the **CaptchaImage-EnableCaptchaAudio** property to true. This will cause a link button, that retrieves the audio code, to be rendered below the **CaptchaImage**. To control the visual appearance of the link button, the user should use the **.rcCaptchaAudioLink** CSS class.

Tutorial - How to configure RadCaptcha to generate audio code?

The following tutorial demonstrates how to configure RadCaptcha to generate audio code.

1. Follow the steps from the "Getting Started (Section 29.3)" tutorial to create a web-site with RadCaptcha control.
2. In the Solution Explorer, right-click the project and select *Add | Add ASP.NET Folder | App_Data*.
3. Locate the **App_Data** folders in your RadControls installation.
4. Copy the **App_Data\RadCaptcha** to the project's **\App_Data** folder.
5. The project structure should now look like the screenshot below.



6. Enable the CaptchaAudio feature by setting the **EnableCaptchaAudio** property of the inner **<CaptchaImage>** tag to true, e.g.

Enable RadCaptcha Audio

```
<telerik:RadCaptcha ID="RadCaptcha1" runat="server" ErrorMessage="Page not valid. The code you entered is not valid."
    ValidationGroup="SubmitGroup" >
    <CaptchaImage EnableCaptchaAudio="true" BackgroundColor="#609f0a" TextColor="White"
    BackgroundNoise="None" />
</telerik:RadCaptcha>
```

Press F5 to run the Application. RadCaptcha validates the input on a post back.



[Get Audio Code](#)

Type the code from the image

Verify Code

30 RadXmlHttpPanel

30.1 Objectives

- Learn about the Callback and WebService update mechanisms and how to configure RadXmlHttpPanel.
- Supported Scenarios: Learn when and how to use RadXmlHttpPanel.
- Known issues with RadScriptManager and RadStyleSheetManager.

30.2 Introduction and Overview

Telerik **RadXmlHttpPanel** is a panel that can load content on demand. Unlike UpdatePanel, or RadAjaxPanel, it is not universal, and cannot be used in all scenarios. However, in scenarios where it is possible to use it, it will deliver much better performance compared to its AJAX counterparts. This is due to the fact that the XmlHttpPanel uses callbacks, web services and WCF services to update its content.

During partial page updates with AJAX, the page goes through its full lifecycle. The whole control tree is created, all event handlers are executed, the ViewState is processed and updated, and sent back to the client. The callbacks, web services, and WCF services, on the other hand, carry a much smaller (or even no additional) overhead, and this results in increased performance and responsiveness of the page.

There are two ways for loading data on the RadXmlHttpPanel - by using the ASP.NET Callback mechanisms and WebService and a WCF Service Ajax call.

- **Callback** - When a client callback is used, the server Page does not go through its whole lifecycle, but only a small part of it. The client state is not updated, and it is not sent back to the client-side. When Callbacks are used, a POST request is made from the client to the server, and the values of all FORM fields, such as hidden fields (including the view state field) are sent to the server. When the view state is large, this could mean increased overhead. On the other hand, no extra files are needed to use this mode (unlike when using a WebService).
- **WebService** - can be used to handle the data request of the RadXmlHttpPanel. The *WebMethodPath* and the *WebMethodName* properties should be set and the RadXmlHttpPanel automatically retrieves and loads the data. Similarly as in the Client Callback the client state is not affected. A web service requires a couple of extra files to set up, but it is the most efficient approach, as no data, other than the **Value** string is sent over from the client to the server.
- **WCF Service** - can be used to handle the data request of the RadXmlHttpPanel. The *WcfRequestMethod*, *WcfMethodPath* and the *WcfMethodName* properties should be set and the RadXmlHttpPanel automatically retrieves and loads the data. Similarly as in the Client Callback the client state is not affected. A WCF Service requires a couple of extra files to set up, but it is an efficient approach, as no data, other than the Value string, is sent over from the client to the server.

30.3 Supported Scenarios

How does the XmlHttpPanel work?

Imagine you have a `<div/>` element, and you want to paste some HTML content within, using JavaScript. One would use either `div.innerHTML` or `div.appendChild` for this purpose.

Well that's the underlying principle of the XmlHttpPanel. It pastes the HTML content received from the web service or the callback, within the panel's HTML element (`` or `<div/>`). The HTML content can be created

by adding controls to the panel during the callback, or returning an HTML string by the web method (of the web service). You will find more information on how to configure the control, to use both update mechanisms, in the *Configuring the XmlHttpRequest* article.

Supported scenarios

Since the page does not go through its standard lifecycle during ASP.NET callbacks and web services, any changes that are made to the content within the XmlHttpRequest will be lost if a postback (or AJAX call that affects the content) occurs. This poses a limitation on the controls residing in the panel to not perform any postbacks and to not execute server-side events.

This being said, the XmlHttpRequest is primarily intended to be used for loading presentation data. If any modifications or updates are to be performed on the content they should be done on the client-side or by the panel itself.

30.4 Configuring the XmlHttpRequest

How to initiate a partial page update

Partial updates are initiated from the client-side, using RadXmlHttpRequest's `set_value("string_value")` client-side method. Values can be passed to the server by providing a single parameter when calling the method.

Callback configuration:

1. Add RadXmlHttpRequest ASP.NET AJAX to the page
2. Set the `EnableClientScriptEvaluation` property to `true`, to enable the evaluation of scripts loaded by the controls within the XmlHttpRequest
3. Place a Label control inside the RadXmlHttpRequest
4. Handle the `ServiceRequest` server-side event of RadXmlHttpRequest
5. In the handler method add the following code:

```
protected void RadXmlHttpRequest1_ServiceRequest(object sender,
Telerik.Web.UI.RadXmlHttpRequestEventArgs e)
{
    Label1.Text = "Label updated by XmlHttpRequest callback at: " + DateTime.Now.ToString();
}
```

6. Create an `<input/>` of type button that will call the `set_value` client method of the XmlHttpRequest on a button click.
You can also access the callback value from the client on the server using the `e.Value` property in the `ServiceRequest` event.

Here is how the page and its codebehind should look after completing the steps above:

Default.aspx

```
<asp:ScriptManager ID="ScriptManager1" runat="server" />
<telerik:RadXmlHttpRequest runat="server" ID="RadXmlHttpRequest1"
    EnableClientScriptEvaluation="true"
    onservicerequest="RadXmlHttpRequest1_ServiceRequest">
    <asp:Label ID="Label1" runat="server"></asp:Label>
</telerik:RadXmlHttpRequest>
<br /><br /><br />
<input type="button" value="Set Value" onclick="SetValue();return false;" />
```

```
<script type="text/javascript">
    function SetValue() {
        var panel = $find("<%=RadXmlHttpPanel1.ClientID %>");
        panel.set_value("string_value");
    }
</script>
```

Default.aspx.cs

```
protected void RadXmlHttpPanel1_ServiceRequest(object sender,
Telerik.Web.UI.RadXmlHttpPanelEventArgs e)
{
    Label1.Text = "Label updated by XmlHttpPanel callback at: " + DateTime.Now.ToStrin
();
    //access the callback value from the client on the server using the e.Value proper
    Label1.Text += "<br/> The returned value from the client's set_value() function is:
<strong>" + e.Value + "</strong>";
}
```

Default.aspx.vb

```
Protected Sub RadXmlHttpPanel1_ServiceRequest(sender As Object, e As
Telerik.Web.UI.RadXmlHttpPanelEventArgs)
    Label1.Text = "Label updated by XmlHttpPanel callback at: " + DateTime.Now.ToString()
    'access the callback value from the client on the server using the e.Value property
    Label1.Text += "<br/> The returned value from the client's set_value() function is:
<strong>" + e.Value + "</strong>"
End Sub
```

WebService Configuration:

1. Add XmlHttpPanel and set **EnableClientScriptEvaluation** to true.
2. Right click on the WebSite to **Add New Item** and in the window opened choose "Web Service". Make sure the check-box "Place code in separate file" is checked. If you work in a Web Application scenario a WebService codebehind file will be created.
3. Open the newly created "Web Service" class in the App_Code folder of your application.
4. Uncomment the [System.Web.Script.Services.ScriptService] just above the class to enable the Web Service to be called from the XmlHttpPanel.
5. Create a method that returns a string and accepts a single parameter of type object. Mark the method as [WebMethod] i.e.

```
[WebMethod]
public string GetHTML(object context)
{
    return "Content updated by XmlHttpPanel using WebService at: " +
DateTime.Now.ToString();
}
```

The string returned from this method is the actual HTML content that will be pasted within the XmlHttpPanel.

6. Set the **WebMethodPath** property to the “Web Service” (usually the .asmx file), and the **WebMethodName** to the method that will be called by the XmlHttpPanel (i.e. GetHTML).
7. Create an <input/> that will call set_value() method of the XmlHttpPanel

Here is how the page with the XMLHttpPanel and the WebService codebehind file should look when accomplishing the steps above:

Default.aspx

```
<asp:ScriptManager ID="ScriptManager1" runat="server"></asp:ScriptManager>
<telerik:RadXmlHttpPanel runat="server" ID="RadXmlHttpPanel1"
    EnableClientScriptEvaluation="true"
    WebMethodPath="WebService1.asmx"
    WebMethodName="GetHTML">
</telerik:RadXmlHttpPanel>
<br /><br /><br />
<input type="button" value="Set Value" onclick="SetValue();return false;" />
<script type="text/javascript">
    function SetValue() {
        var panel = $find("<%=RadXmlHttpPanel1.ClientID %>");
        var array = [];
        array[0] = "string0";
        array[1] = "string1";
        //you can pass any kind of object to the GetHTML method
        //right now we will pass an array
        panel.set_value(array);
    }
</script>
```

App_Code\WebService.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.Services;

/// <summary>
/// Summary description for WebService
/// </summary>
[WebService(Namespace = "http://tempuri.org (http://tempuri.org/)")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script, using ASP.NET AJAX, uncomment the
following line.
[System.Web.Script.Services.ScriptService]
public class WebService : System.Web.Services.WebService {

    public WebService () {
        //Uncomment the following line if using designed components
        //InitializeComponent();
    }

    [WebMethod]
    public string GetHTML(object context)
```

```

{
    Dictionary<string, object> dictionary = context as Dictionary<string, object>;

    //The value passed to the XmlHttpPanel can be of type object
    object value = dictionary["Value"];
    string value1 = ((object[])(value))[0].ToString();
    string value2 = ((object[])(value))[1].ToString();
    return "Content updated by XmlHttpPanel using WebService at: "
        + DateTime.Now.ToString()
        + "<br/>These are the passed values to the XmlHttpPanel: <strong>" + value1 +
</strong> and <strong>" + value2 + " </strong>";
}
}

```

WCF Service configuration:

- In the properties pane for the RadXmlHttpPanel component, set the *WcfRequestMethod*, *WcfMethodPath* and the *WcfMethodName* properties to identify the Web service:
 - WcfRequestMethod** - Gets or sets the request method for WCF Service used to populate content GET, POST, PUT, DELETE
 - WcfServicePath** - Gets or sets a string value that indicates the virtual path of the WCF Service used by the RadXmlHttpPanel
 - WcfServiceMethod** - Gets or sets a string value that indicates the WCF Service method used by the RadXmlHttpPanel.
- Setting the **Value** property of the panel depends on the *WcfRequestMethod* property. In both cases country is the name of the parameter in the *WcfRequestMethod* method:
 - If *WcfRequestMethod* = "POST" the Value property should be set to "{country: "value"}" or "{country: "value"}".
 - If *WcfRequestMethod* = "GET" the Value property should be set to "country=value".

Inline declaration

```

<telerik:RadXmlHttpPanel runat=server" ID="XmlHttpPanelWCF"
    Value="{country: "Argentina"}"
    WcfServicePath="XmlHttpPanelWcfService.svc"
    WcfServiceMethod="GetCustomersByCountry"
    WcfRequestMethod="POST">
</telerik:RadXmlHttpPanel>

```

- Define the Contracts of the WCF Service in an interface:

C#

```

[ServiceContract]
public interface IXmlHttpPanelWcfService
{
    [OperationContract]
    [WebInvoke(Method = "POST", BodyStyle = WebMessageBodyStyle.Wrapped, ResponseFormat =
WebMessageFormat.Json)]
    string GetCustomersByCountry(string country);
}

```

VB.NET

```

<ServiceContract(> _
Public Interface IXmlHttpPanelWcfService
    <OperationContract(> _
        <WebInvoke(Method:="POST", BodyStyle:=WebMessageBodyStyle.Wrapped,
ResponseFormat:=WebMessageFormat.Json)> _
        Function GetCustomersByCountry(ByVal country As String) As String
    End Interface

```

4. Implement the contract in the WCF Service class:

C#

```

[AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
public class XmlHttpPanelWcfService : IXmlHttpPanelWcfService
{
    public string GetCustomersByCountry(string country)
    {
        return "The content of XmlHttpPanel";
    }
}

```

VB.NET

```

<AspNetCompatibilityRequirements
(RequirementsMode:=AspNetCompatibilityRequirementsMode.Allowed)> _
Public Class XmlHttpPanelWcfService
    Implements IXmlHttpPanelWcfService
    Public Function GetCustomersByCountry(ByVal country As String) As String
        Return "The content of XmlHttpPanel"
    End Function
End Class

```

5. Define the configuration in web.config:

web.config

```

<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="XmlHttpPanelWcfBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="XmlHttpPanelWcfBehavior">
          <webHttp />
        </behavior>
      </endpointBehaviors>
    </behaviors>
    <services>
      <service behaviorConfiguration="XmlHttpPanelWcfBehavior"
name="XmlHttpPanelWcfService">

```

```
        <endpoint address="" binding="webHttpBinding"
contract="IXmlHttpPanelWcfService" behaviorConfiguration="XmlHttpPanelWcfBehavior"/>
    </service>
</services>
</system.serviceModel>
</configuration>
```

6. Optionally, set the `OnClientResponseEnding` property to a client-side event handler that handles the response of the WCF Service.

ASPX

```
function OnClientResponseEnding (sender, args)
{
    //The actual result data is in the [WcfServiceMethod]Result property of the content
    object.
    var data = args.get_content().GetCustomersByCountryResult,
    args.set_cancel(true);
}
```

7. Optionally, set the `OnClientResponseEnded` and `OnClientResponseError` properties to client-side event handlers that respond when the WCF Service has successfully updated the panel's content or when the WCF Service has generated an error while trying to service the item request, respectively.

ASPX

```
function OnClientResponseEnded (sender, args)
{
    //...
}

//Fired when the request for the items fails.
function OnClientResponseError (sender, args)
{
    // Disable the notifying error alert.
    args.set_cancelErrorAlert(true);
    //...
}
```

30.5 Client-Side Programming

Use the ASP.NET AJAX's native `$find()` method to get a reference to the `RadXmlHttpPanel` control:

JavaScript

```
var panel = $find("<%=RadXmlHttpPanel.ClientID %>");
```

The following table lists the most important methods of the client-side `RadXmlHttpPanel` object:

Method	Description
<code>get_value()</code>	Returns the value that is passed to the <code>RadXmlHttpPanel</code>
<code>set_value(value)</code>	passes a value to the <code>RadXmlHttpPanel</code> depending on which content is loaded inside the <code>RadXmlHttpPanel</code> and initiates a partial page request.
<code>get_element()</code>	returns the DOM element for this control
<code>set_html(content)</code>	sets a custom HTML content to the <code>RadXmlHttpPanel</code> .

Client - side events

OnClientResponseEnded

The **OnClientResponseEnded** occurs immediately after the data (content) is pasted into the **RadXmlHttpPanel**. This client-side event is subsequent to the **OnClientResponseEnding** event. The event handler receives a single parameter: the instance of the **RadXmlHttpPanel** control firing the event.

OnClientResponseEnding

The **OnClientResponseEnding** client-side event replaces the existing **OnClientResponseEnd** client-side event. Please note that although the **OnClientResponseEnding** should be used from now on, the **OnClientResponseEnd** is still present in the control's API so that any existing applications are not broken after an upgrade to a newer version of the control.

The **OnClientResponseEnding** occurs before the data (content) is pasted into the **RadXmlHttpPanel**, after a partial update request has been initiated by the **RadXmlHttpPanel** `set_value` method.

The event handler receives two parameters:

- The instance of the **RadXmlHttpPanel** control firing the event.
- An **EventArgs** parameter containing the following properties and methods:
 - **set_cancel** lets you prevent from loading the content inside the **RadXmlHttpPanel** and raising the **OnClientResponseEnded** client-side event.
 - **get_cancel** returns a boolean value indicating whether the **RadXmlHttpPanel**'s content update was canceled.
 - **get_content()** gets the HTML content rendered inside the **RadXmlHttpPanel**.

The following example demonstrates how the user can cancel the loading of the content inside the **RadXmlHttpPanel**. An event handler should be provided for the **OnClientResponseEnding** client-side event where the action can be canceled by using the `cancel` property of the **EventArgs** passed to the handler.

ASPX

```
<script type="text/javascript">
    function SetValue()
    {
        var panel = $find("<%= RadXmlHttpPanel1.ClientID %>");

        var value = "some_value";
        panel.set_value(value);
    }
    function OnClientResponseEnding(panel, args)
    {
        var result = confirm("Do not load the content in the XmlPanel?");
        args.set_cancel(result);
    }
</script>
<input type="button" value="Refresh RadXmlHttpPanel1" onclick="SetValue()" />
<telerik:RadXmlHttpPanel ID="RadXmlHttpPanel1" runat="server" OnServiceRequest="
RadXmlHttpPanel1_ServiceRequest"
    OnClientResponseEnding="OnClientResponseEnding">
    <telerik:RadGrid RegisterWithScriptManager="false" ID="RadGridTeamPlayer"
```

```
runat="server"
AllowSorting="False" AutoGenerateColumns="True" GridLines="Both"
Height="100%"
ShowFooter="false" Style="border: solid 1px black; outline: 0">
  <MasterTableView>
    <Columns>
      <telerik:GridBoundColumn DataField="username" HeaderText="Player"
SortExpression="username">
        <ItemStyle HorizontalAlign="center" />
        <HeaderStyle HorizontalAlign="center" />
      </telerik:GridBoundColumn>
    </Columns>
  </MasterTableView>
</telerik:RadGrid>
</telerik:RadXmlHttpPanel>
```

C#

```
protected void RadXmlHttpPanel1_ServiceRequest(object sender,
Telerik.Web.UI.RadXmlHttpPanelEventArgs e)
{
  string val = e.Value;
  BindGrid(val);
}
void BindGrid(string parametervalue)
{
  RadGrid1.DataSource = new string[] { "1", "2", DateTime.Now.ToLongTimeString() };
  RadGrid1.DataBind();
}
```

VB.NET

```
Protected Sub RadXmlHttpPanel1_ServiceRequest(sender As Object, e As
Telerik.Web.UI.RadXmlHttpPanelEventArgs)
  Dim val As String = e.Value
  BindGrid(val)
End Sub
Sub BindGrid(parametervalue As String)
  RadGrid1.DataSource = New String() {"1", "2", DateTime.Now.ToLongTimeString()}
  RadGrid1.DataBind()
End Sub
```

OnClientResponseError

The **OnClientResponseError** occurs in the cases when an error (WebService or Callback error) occurs when the **RadXmlHttpPanel** tries to load certain content.

The event handler receives two parameters:

1. The instance of the **RadXmlHttpPanel** control in which the error occurred.
2. An **EventArgs** parameter containing the following properties and methods:

- `set_cancelErrorAlert` lets you prevent from displaying the built-in error alert that notifies the user that an error has occurred, and gives the possibility to display a custom error message.
- `get_cancelErrorAlert` returns a boolean value indicating whether the `RadXmlHttpRequest`'s displaying of the built-in error alert has been canceled.

The following example demonstrates how the user can display: a custom content inside the `RadXmlHttpRequest` or a custom error message (alert), if an error has occurred while loading content inside the panel. The panel tries to load `RadCalendar` control, but an error will occur because the control's `RegisterWithScriptManager` property has not been set to false.

ASPX

```
<script type="text/javascript">
function LoadCalendar()
{
    var panel = $find("RadXmlHttpRequest1");
    panel.set_value(value);
}

function OnClientResponseError(panel, args)
{
    alert("OnClientResponseError fired because an error occurred");
    args.set_cancelErrorAlert(true);
    var content = "<label style='color: Red;'>The Control could not be loaded because of an callback error!</label>";
    panel.set_html(content);
}
</script>
<input type="button" value="LoadRadCalendar" onclick="LoadCalendar()" />
<telerik:RadXmlHttpRequest ID="RadXmlHttpRequest1" runat="server"
OnServiceRequest="RadXmlHttpRequest1_ServiceRequest"
OnClientResponseError="OnClientResponseError">
</telerik:RadXmlHttpRequest>
```

C#

```
protected void RadXmlHttpRequest1_ServiceRequest(object sender, RadXmlHttpRequestEventArgs e)
{
    RadCalendar calendar = new RadCalendar();
    calendar.ID = "RadCalendar1";
    //calendar.RegisterWithScriptManager = false
    RadXmlHttpRequest1.Controls.Add(calendar);
}
```

VB.NET

```
Protected Sub RadXmlHttpRequest1_ServiceRequest(ByVal sender As Object, ByVal e As RadXmlHttpRequestEventArgs)
    Dim calendar As New RadCalendar()
    calendar.ID = "RadCalendar1"
    'calendar.RegisterWithScriptManager = false
    RadXmlHttpRequest2.Controls.Add(calendar)
End Sub
```

30.6 Server-Side Programming

RadXmlHttpPanel provides the following server-side properties:

Property	Description
LoadingPanelID	Gets or sets the ID of the RadAjaxLoadingPanel control that will be displayed over the control during the partial page update.
Value	Gets or sets a string value depending on which a certain content is loaded in the RadXmlHttpPanel.
WebMethodPath	Gets or sets a string value that indicates the virtual path of the WebService used by the RadXmlHttpPanel.
WebMethodName	Gets or sets a string value that indicates the WebService method used by the RadXmlHttpPanel.
EnableClientScriptEvaluation	Gets or sets a boolean value indicating whether or not the client scripts loaded by the RadControls hosted inside the RadXmlHttpPanel should be executed.

30.7 Known Issues

Compatibility issues with RadScriptManager and RadStyleSheetManager

The **RadXmlHttpPanel** has known compatibility issues with **RadScriptManager** and **RadStyleSheetManager**. Both managers combine all the requests (**RadScriptManager** combines the requests to the javascript assembly resource files and **RadStyleSheetManager** combines the ones to stylesheets resource files of all **RadControls** present on the page), into a single request.

Because the page does not go through its normal life cycle, after the **RadControls** have been updated by the **RadXmlHttpPanel**, the controls' scripts and stylesheets need to be evaluated and applied, respectively. This however, cannot be done if the scripts (and the stylesheets) are combined into a single file - the **RadXmlHttpPanel** cannot find the right scripts and styles for the respective **RadControl**. That is why there might be client-script errors and the styles will not be applied correctly if the **RadXmlHttpPanel** is used together with the **RadStyleSheetManager** and the **RadScriptManager**.

There are 2 ways to solve this problem:

1. use the Microsoft AJAX ScriptManager control
OR
2. set `EnableScriptCombine="false"` for **RadScriptManager** and `EnableStyleSheetCombine="false"` to **RadStyleSheetManager**.

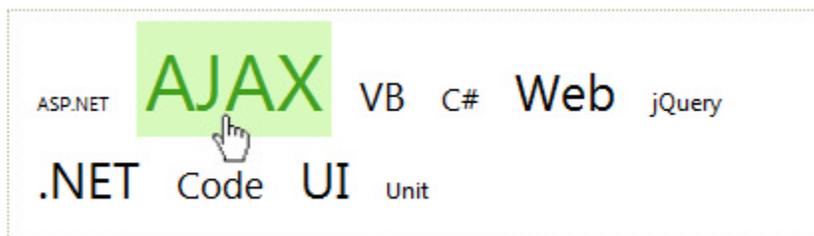
31 RadTagCloud

31.1 Objectives

- Introduction
- Getting Started
- Learn how to bind RadTagCloud to DataSource
- Configuring RadTagCloud items
- Generating TagCloud from External Sources

31.2 Introduction

Telerik RadTagCloud is a flexible UI component for categorization and weighted visualization of user-generated tags or related keywords. The user can easily customize the appearance of the control, choose the items that will appear in the cloud, sort the tags alphabetically or by weight, in ascending or descending order, and use various other configuration options.



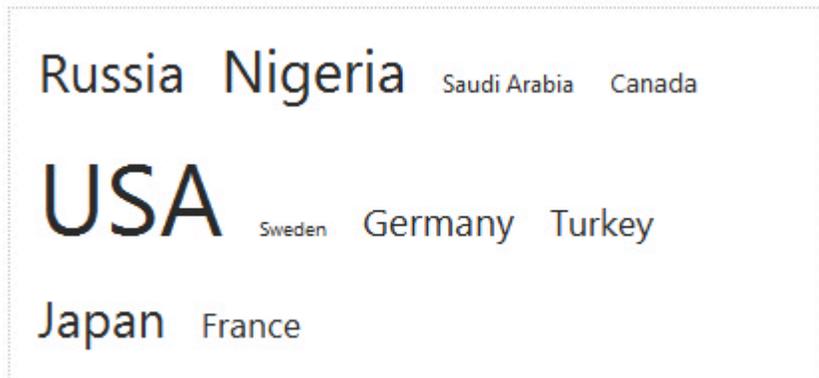
Key Features:

- **Distribution** - specifies how the font size will be distributed among the items. When set to *Linear* the font size is distributed linearly and in the case of *Logarithmic* the items are weighted logarithmically. **Sorting** - specifies in what order the TagCloud items will be listed. By default they are not sorted. The user can choose to sort them alphabetically or based on their weight, in ascending or descending order. Possible values for this property are: *NotSorted* (default), *AlphabeticAsc*, *AlphabeticDsc*, *WeightedAsc* and *WeightedDsc*.
- **Filtering Of The Items** - Three properties control the filtering of the items: **MinimalWeightAllowed**, **MaxNumberOfItems** and **TakeTopWeightedItems**.
- **MinimalWeightAllowed** - specifies the lower bound for the item Weight. If the Weight of the item is smaller than this bound, the tag will not appear in the cloud. The default value is 0.0, which means the items will not be filtered.
- **MaxNumberOfItems** - specifies the maximal number of items that can (will) be shown in the cloud. If the **TakeTopWeightedItems** property is set to true, the items with the highest weight will be taken. The default value is 0, which means the items will not be filtered.
- **MinFontSize and MaxFontSize** - specify the range of the font size, the TagCloud items could have. The default values are 10px and 20px, respectively. These properties accept values of type `System.Web.UI.WebControls.Unit` and the font-size of the TagCloud items will have the same `System.Web.UI.WebControls.UnitType` as the one of the properties. The value of **MaxFontSize** must be greater or equal than the one of **MinFontSize**.

31.3 Getting Started

The following tutorial demonstrates how to set up a page with **RadTagCloud** and manually populate the control with keywords. The walk-through will also show how to sort the items alphabetically in ascending order.

1. In the default page of a new ASP.NET AJAX-enabled Web Application add a RadTagCloud control.



2. In the Source view of the .aspx page, find the definition of the TagCloud, and add the <Items></Items> inner property.
3. Between the opening and the closing tag of the <Items> property add the following list of items. Every item represents a country, with the **Weight** of the item equal representing the millions of people living there, and the **NavigateUrl** pointing to the country's Wikipedia article.

Inner Items tags of RadTagCloud

```
<Items>
  <telerik:RadTagCloudItem Text="Russia" Weight="141.9"
  NavigateUrl="http://en.wikipedia.org/wiki/Russia" />
  <telerik:RadTagCloudItem Text="Nigeria" Weight="154.7"
  NavigateUrl="http://en.wikipedia.org/wiki/Nigeria" />
  <telerik:RadTagCloudItem Text="Saudi Arabia" Weight="28.6"
  NavigateUrl="http://en.wikipedia.org/wiki/Saudi Arabia" />
  <telerik:RadTagCloudItem Text="Canada" Weight="34.1"
  NavigateUrl="http://en.wikipedia.org/wiki/Canada" />
  <telerik:RadTagCloudItem Text="USA" Weight="309.4"
  NavigateUrl="http://en.wikipedia.org/wiki/USA" />
  <telerik:RadTagCloudItem Text="Sweden" Weight="9.3"
  NavigateUrl="http://en.wikipedia.org/wiki/Sweden" />
  <telerik:RadTagCloudItem Text="Germany" Weight="81.7"
  NavigateUrl="http://en.wikipedia.org/wiki/Germany" />
  <telerik:RadTagCloudItem Text="Turkey" Weight="72.5"
  NavigateUrl="http://en.wikipedia.org/wiki/Turkey" />
  <telerik:RadTagCloudItem Text="Japan" Weight="127.3"
  NavigateUrl="http://en.wikipedia.org/wiki/Japan" />
  <telerik:RadTagCloudItem Text="France" Weight="65.4"
  NavigateUrl="http://en.wikipedia.org/wiki/France" />
</Items>
```

4. In the Properties Window of RadTagCloud set the following properties
 1. **Width**="400px"
 2. **MaxFontSize**="50px"

3. `Sorting="AlphabeticAsc"`
5. The definition of the TagCloud should look like the following:

RadTagCloud Declaration

```
<telerik:RadTagCloud ID="RadTagCloud1" runat="server" Width="400px" MaxFontSize="50px"
Sorting="AlphabeticAsc">
  <Items>
    <%-- TagCloud items --%>
  </Items>
</telerik:RadTagCloud>
```

6. Press F5 to run the Application. When a tag is clicked the browser navigates to the respective Wikipedia article.



31.4 Important Properties

RadTagCloud properties:

- **Distribution** - type: enumerator - Gets or sets a value indicating how the font-size will be distributed among the different words (items). Values:
 - **Linear** - The font-size is **linearly** distributed among the different words based on their weight.
 - **Logarithmic** - The font-size is **logarithmically** distributed among the different words based on their weight.
- **MinFontSize** - type: Unit - Unit values - Gets or sets the font-size to the least important (frequent) item.
- **MaxFontSize** - type: Unit - Unit values - Gets or sets the font-size to the most important (frequent) item.
- **MinimalWeightAllowed** - type: Double - Gets or sets the minimal weight a TagCloud item could have. If the weight of the item is less than this value, the keyword will not appear in the cloud. The default value is 0.0, which means the items will appear in the cloud regardless of their weight.
- **MaxNumberOfItems** - type: Integer - Gets or sets the maximal number items that can appear in the cloud. The default value is 0, which means the items will appear in the cloud no matter their count.
- **TakeTopWeightedItems** - type: Boolean - Should be used with MaxNumberOfItems property. Gets or sets a bool value indicating whether the [MaxNumberOfItems] visible items will be the ones with the biggest weight, or the ones that occur first in the DataSource. The default value is false (i.e. the items are the first that appear in the DataSource).
- **RenderItemWeight** - type: Boolean - Gets or sets a bool value indicating whether the item weight will be rendered. It is rendered right next to the item's text.
- **Sorting** - type: enumerator - Gets or sets a value indicating how the TagCloud items will be sorted. Values:

- **NotSorted** - The TagCloud items are left as they appear in the Items collection (DataSource).
- **AlphabeticAsc** - The TagCloud items are sorted alphabetically in ascending order.
- **AlphabeticDsc** - The TagCloud items are sorted alphabetically in descending order.
- **WeightedAsc** - The TagCloud items are sorted based on their Weight in ascending order.
- **WeightedDsc** - The TagCloud items are sorted based on their Weight in descending order.

Server-Side Events:

- **ItemDataBound** - Adds or removes an event handler method from the ItemDataBound event. The event is fired right after RadTagCloudItem is databound.
- **ItemClick** - Adds or removes an event handler method from the ItemClick event. The event is fired after RadTagCloudItem is clicked.

RadTagCloudItem properties:

- **DataItem** - type: object - Gets or sets the data object (from the data source) associated with the TagCloud item.
- **NavigateUrl** - type: string - Gets or sets the URL of the TagCloud item.
- **Text** - type: string - Gets or sets the text that is displayed in the TagCloud item.
- **Weight** - type: double - Gets or sets the weight, that determines how the TagCloud item (tag, keyword) will be styled. Greater value means, the value of the font size will be closer to the one of the RadTagCloud's MaxFontSize property.

31.5 Databinding

RadTagCloud supports binding to all ASP.NET DataSource components, including

- AccessDataSource
- SqlDataSource
- XmlDataSource
- ObjectDataSource
- SiteMapDataSource
- LinqDataSource

To bind to a DataSource component, you need to set the following properties:

1. **DataSource** - Set to an instance of your data source. This is mandatory when binding the RadTagCloud at runtime.
2. **DataSourceID** - Set to the ID of your data source. This is mandatory when binding the RadTagCloud declaratively.
3. **DataMember** - If the data source is a DataSet and DataMember is set, then the RadTagCloud is bound to the DataTable with the respective name in the DataSet. If DataMember is not set, the TagCloud is bound to the first DataTable in the DataSet.
4. **DataTextField** - This is the field name from the data source that populates each item's Text property during binding.

5. **DataWeightField** - This is the field name from the data source that populates each item's Weight property during binding.
6. **DataNavigateUrlField** - This is the field name from the data source that populates each item's NavigateUrl property during binding.
7. **DataBind** - Call this method after you have set the aforementioned properties when binding at runtime. This method is mandatory for binding at runtime.

Here is an example that shows how to bind the tagCloud to an ObjectDataSource. In a similar way the control can be bound to any of the above mentioned DataSource components.

Default.aspx

```
<div>
<telerik:RadTagCloud ID="RadTagCloud2" runat="server" Width="400px" MaxFontSize="50px"
  Sorting="AlphabeticAsc" DataSourceID="ObjectDataSource1" DataTextField="Text"
  DataWeightField="Weight" DataNavigateUrlField="NavigateUrl">
</telerik:RadTagCloud>
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server" SelectMethod="GetSiteData"
  TypeName="TagCloudDataItem"></asp:ObjectDataSource>
</div>
```

TagCloudDataItem.cs

```
using System.Collections.Generic;
/// <summary>
/// This class is only for demonstration purposes. The class used in this example resides in
the App_Code folder
/// </summary>
public class TagCloudDataItem
{
  private string _text;
  private string _navigateUrl;
  private double _weight;
  public string Text
  {
    get { return _text; }
    set { _text = value; }
  }

  public string NavigateUrl
  {
    get { return _navigateUrl; }
    set { _navigateUrl = value; }
  }
  public double Weight
  {
    get { return _weight; }
    set { _weight = value; }
  }
  public TagCloudDataItem(string text, string navigateUrl, double weight)
  {
    _text = text;
    _navigateUrl = navigateUrl;
    _weight = weight;
  }
}
```

```
}
public static List<TagCloudDataItem> GetSiteData()
{
    List<TagCloudDataItem> siteData = new List<TagCloudDataItem>();
    siteData.Add(new TagCloudDataItem("Russia", "http://en.wikipedia.org/wiki/Russia", 141.9));
    siteData.Add(new TagCloudDataItem("Nigeria", "http://en.wikipedia.org/wiki/Nigeria",
154.7));
    siteData.Add(new TagCloudDataItem("Saudi Arabia",
"http://en.wikipedia.org/wiki/Saudi Arabia", 28.6));
    siteData.Add(new TagCloudDataItem("Canada", "http://en.wikipedia.org/wiki/Canada", 34.1));
    siteData.Add(new TagCloudDataItem("USA", "http://en.wikipedia.org/wiki/USA", 309.4));
    siteData.Add(new TagCloudDataItem("Sweden", "http://en.wikipedia.org/wiki/Sweden", 9.3));
    siteData.Add(new TagCloudDataItem("Germany", "http://en.wikipedia.org/wiki/Germany",
81.7));
    siteData.Add(new TagCloudDataItem("Turkey", "http://en.wikipedia.org/wiki/Turkey", 72.5));
    siteData.Add(new TagCloudDataItem("Japan", "http://en.wikipedia.org/wiki/Japan", 127.3));
    siteData.Add(new TagCloudDataItem("France", "http://en.wikipedia.org/wiki/France", 65.4));
    return siteData;
}
}
```

Example Title

```
Imports System.Collections.Generic
''' <summary>
''' This class is only for demonstration purposes. The class used in this example resides in
the App_Code folder
''' </summary>
Public Class TagCloudDataItem
    Private _text As String
    Private _navigateUrl As String
    Private _weight As Double
    Public Property Text() As String
        Get
            Return _text
        End Get
        Set
            _text = value
        End Set
    End Property

    Public Property NavigateUrl() As String
        Get
            Return _navigateUrl
        End Get
        Set
            _navigateUrl = value
        End Set
    End Property

    Public Property Weight() As Double
        Get
            Return _weight
        End Get
        Set
            _weight = value
        End Set
    End Property
End Class
```

```

End Set
End Property
Public Sub New(text As String, navigateUrl As String, weight As Double)
    _text = text
    _navigateUrl = navigateUrl
    _weight = weight
End Sub
Public Shared Function GetSiteData() As List(Of TagCloudDataItem)
    Dim siteData As New List(Of TagCloudDataItem)()
    siteData.Add(New TagCloudDataItem("Russia", "http://en.wikipedia.org/wiki/Russia", 141.9))
    siteData.Add(New TagCloudDataItem("Nigeria", "http://en.wikipedia.org/wiki/Nigeria",
154.7))
    siteData.Add(New TagCloudDataItem("Saudi Arabia",
"http://en.wikipedia.org/wiki/Saudi_Arabia", 28.6))
    siteData.Add(New TagCloudDataItem("Canada", "http://en.wikipedia.org/wiki/Canada", 34.1))
    siteData.Add(New TagCloudDataItem("USA", "http://en.wikipedia.org/wiki/USA", 309.4))
    siteData.Add(New TagCloudDataItem("Sweden", "http://en.wikipedia.org/wiki/Sweden", 9.3))
    siteData.Add(New TagCloudDataItem("Germany", "http://en.wikipedia.org/wiki/Germany",
81.7))
    siteData.Add(New TagCloudDataItem("Turkey", "http://en.wikipedia.org/wiki/Turkey", 72.5))
    siteData.Add(New TagCloudDataItem("Japan", "http://en.wikipedia.org/wiki/Japan", 127.3))
    siteData.Add(New TagCloudDataItem("France", "http://en.wikipedia.org/wiki/France", 65.4))
    Return siteData
End Function
End Class

```

31.6 Filtering and Sorting of the TagCloud Items

RadTagCloud provides an easy way of organizing the tags, by setting a couple of properties. This way the user can choose which tags and in what order they will appear in the cloud.

The sorting of the items is controlled by the **Sorting** property. By setting it to one of the possible values:

- *NotSorted* (default)
- *AlphabeticAsc*
- *AlphabeticDsc*
- *WeightedAsc*
- *and WeightedDsc*

the user can choose how the items will be listed in the cloud. The items can be sorted alphabetically or based on their weight, in ascending or descending order.

Items can be filtered by setting either of the following properties:

- **MinimalWeightAllowed**- specifies the lower bound for the item Weight. If the Weight of the item is smaller than this bound, the tag will not appear in the cloud. The default value is 0.0, which means the items will not be filtered.
- **MaxNumberOfItems** - specifies the maximal number of items that can (will) be shown in the cloud. If the **TakeTopWeightedItems** property is set to true, the items with the highest weight will be taken. The default value is 0, which means the items will not be filtered.

Note: Please note that, neither the filtering, nor the sorting, modifies the **Items** collection of the TagCloud,

but it only displays the items that satisfy the conditions and values, set by the respective (Sorting, MinimalWeightAllowed, etc.) properties. This is because the user should be able to return to the previous state of the Items collection.

To get the items that are filtered use the *RadTagCloud.Items.Filter()* method, and for the sorted use *RadTagCloud.Items.Sort()* method. Both methods return a collection of TagCloud items.

31.7 Generating TagCloud from External Sources

RadTagCloud provides an easy way to generate tags from external sources. By setting the corresponding property, you can generate tags from text file, direct input (text) and from a web site. To configure **RadTagCloud** to use external sources you need to set one or more of the following properties:

- **Text** - sets text value for direct input generation source
- **TextFile** - specifies the location of the file to be used as a generation source
- **TextUrl** - specifies the URL of the web site to be used as a generation source

If more than one of these properties are set **RadTagCloud** will combine the sources when generating the tags.

In the example below, you can generate tags from a web site by setting an absolute URL in the input field:

ASPX

Enter a valid URL and press the Update button
 to populate the Tag Cloud control below:


```
<asp:TextBox ID="urlField" runat="server" TextMode="SingleLine" Width="285px" ToolTip="The URL must start with http://"></asp:TextBox>
<asp:RegularExpressionValidator ID="urlValidator" runat="server" SetFocusOnError="true" ErrorMessage="Valid URL should start with http://" ControlToValidate="urlField" ValidationExpression="http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?"></asp:RegularExpressionValidator>
<br />
<br />
<asp:Button ID="urlButton" runat="server" Text="Generate" OnClick="urlButton_Click" />
<br />
<br />
<telerik:RadTagCloud Text="Tag Cloud" ID="RadTagCloud1" runat="server" MaxNumberOfItems="30" TakeTopWeightedItems="true" PunctuationCharactersValid=".'#$$€€ Width="200px">
</telerik:RadTagCloud>
```

C# Codebehind

```
protected void urlButton_Click(object sender, EventArgs e)
{
    RadTagCloud1.TextUrl = urlField.Text;
}
```

VB.NET Codebehind

```
Protected Sub urlButton_Click(sender As Object, e As EventArgs)
    RadTagCloud1.TextUrl = urlField.Text
End Sub
```

31.8 Client-Side Data Binding

You can load child items in a tag cloud dynamically through a Web service. The following steps describe how to

configure **RadTagCloud** so that it can use a Web service to supply child items:

1. In the properties pane for the **RadTagCloud** component, set the **WebServiceSettings** property to identify the Web service and service method:
 1. Set the **Path** sub-property to the URL for the Web service.
 2. Set the **Method** sub-property to the name of the method of the Web service that supplies child items.
 3. Set the **UseHttpGet** sub-property to True to change the default HTTP method (POST).

ASPX

```
<telerik:RadTagCloud ID="RadTagCloud1" runat="server"
  OnClientItemsRequesting="itemsRequesting"
  OnClientItemsRequested="itemsRequested"
  OnClientItemsRequestFailed="itemsRequestFailed">
  <WebServiceSettings Path="VehiclesWeightByRating.aspx" Method="GetRadTagCloudItems"
</telerik:RadTagCloud
```

2. When the **WebServiceSettings** property is set, an empty context request will be initiated automatically. You can trigger requests to the service by calling the *requestItems()* client-side method. This method has a single parameter, which is sent as an argument to the Web service method. Keep in mind that all current items will be removed before the new ones are populated.

JavaScript

JavaScript

```
function clientFunction()
{
  //...
  var context = "some value";
  tagCloud.requestItems(context);
  //...
}
```

3. Optionally, set the **OnClientItemsRequesting** property to a client-side event handler that passes context information to the Web service. The Web service can use this context information to determine what child items to return or what properties to set on those child items.

JavaScript

```
//Fired before the request is sent to the Web Service
function itemsRequesting(sender, args)
{
  //If you want to cancel the request use
  //args.set_cancel(true);
  //The args.get_context()/args.set_context(value) methods get/set the parameter which is
  be sent to the Web Service.
  var context = args.get_context();
}
```

4. Optionally, set the **OnClientItemsRequested** and **OnClientItemsRequestFailed** properties to client-side event handlers that respond when the Web service has successfully loaded child items or when the Web service has generated an error while trying to service the item request, respectively.

JavaScript

```
//Fired when the items are successfully loaded.
function itemsRequested(sender, args)
```

```
{
    //...
}

//Fired when the request for the items fails.
function itemsRequestFailed(sender, args)
{
    // Disable the notifying error alert.
    args.set_cancelErrorAlert(true);
    //...
}
```

5. To use the integrated WebService support of RadTagCloud, the WebService should have the following signature:

C#

```
[ScriptService]
public class WebServiceName : WebService
{
    [WebMethod]
    public TagCloudDataItem[] GetRadTagCloudItems(Object context)
    {
        List<TagCloudDataItem> result = new List<TagCloudDataItem>();
        //.....
        TagCloudDataItem item = new TagCloudDataItem();
        item.Text = "Item";
        item.Weight = 6.6;
        item.NavigateUrl = "NavigateUrl";
        item.AccessKey = "AccessKey";
        item.TabIndex = 5;
        item.ToolTip = "ToolTip";
        item.Value = "ToolTip";
        result.Add(item);
        //.....
        return result.ToArray();
    }
}

/// <summary>
/// This class is only for demonstration purposes.
/// The class used in this example resides in the App_Code/TagCloud folder
/// </summary>
public class TagCloudDataItem
{
    private string _text;
    private double _weight;
    private string _navigateUrl;
    private string _accessKey;
    private short _tabIndex;
    private string _toolTip;
    private string _value;
    public string Text
    {
        get { return _text; }
        set { _text = value; }
    }
}
```

```

    }
    public double Weight
    {
        get { return _weight; }
        set { _weight = value; }
    }
    public string NavigateUrl
    {
        get { return _navigateUrl; }
        set { _navigateUrl = value; }
    }
    public string AccessKey
    {
        get { return _accessKey; }
        set { _accessKey = value; }
    }
    public short TabIndex
    {
        get { return _tabIndex; }
        set { _tabIndex = value; }
    }
    public string ToolTip
    {
        get { return _toolTip; }
        set { _toolTip = value; }
    }
    public string Value
    {
        get { return _value;}
        set { _value = value; }
    }
    public TagDataItem()
    {
    }
    public TagDataItem(string text, double weight)
    {
        _text = text;
        _weight = weight;
    }
    public TagDataItem(string text, string navigateUrl, double weight)
    {
        _text = text;
        _navigateUrl = navigateUrl;
        _weight = weight;
    }
}

```

VB.NET

```

<ScriptService()> _
Public Class WebServiceName
    Inherits WebService
    <WebMethod()> _
    Public Function GetRadTagCloudItems(ByVal context As [Object]) As TagCloudDataItem()

```

```

        Dim result As New List(Of TagCloudDataItem)()
        '.....
        Dim item As New TagCloudDataItem()
        item.Text = "Item"
        item.Weight = 6.6
        item.NavigateUrl = "NavigateUrl"
        item.AccessKey = "AccessKey"
        item.TabIndex = 5
        item.ToolTip = "ToolTip"
        item.Value = "ToolTip"
        result.Add(item)
        '.....
        Return result.ToArray()
    End Function
End Class

''' <summary>
''' This class is only for demonstration purposes.
''' The class used in this example resides in the App_Code/TagCloud folder
''' </summary>
Public Class TagCloudDataItem
    Private _text As String
    Private _weight As Double
    Private _navigateUrl As String
    Private _accessKey As String
    Private _tabIndex As Short
    Private _toolTip As String
    Private _value As String

    Public Property Text() As String
        Get
            Return _text
        End Get
        Set(ByVal value As String)
            _text = value
        End Set
    End Property
    Public Property Weight() As Double
        Get
            Return _weight
        End Get
        Set(ByVal value As Double)
            _weight = value
        End Set
    End Property
    Public Property NavigateUrl() As String
        Get
            Return _navigateUrl
        End Get
        Set(ByVal value As String)
            _navigateUrl = value
        End Set
    End Property
    Public Property AccessKey() As String
        Get

```

```

        Return _accessKey
    End Get
    Set(ByVal value As String)
        _accessKey = value
    End Set
End Property
Public Property TabIndex() As Short
    Get
        Return _tabIndex
    End Get
    Set(ByVal value As Short)
        _tabIndex = value
    End Set
End Property
Public Property ToolTip() As String
    Get
        Return _toolTip
    End Get
    Set(ByVal value As String)
        _toolTip = value
    End Set
End Property
Public Property Value() As String
    Get
        Return _value
    End Get
    Set(ByVal value As String)
        _value = value
    End Set
End Property
Public Sub New()
End Sub
Public Sub New(ByVal text As String, ByVal weight As Double)
    _text = text
    _weight = weight
End Sub
Public Sub New(ByVal text As String, ByVal navigateUrl As String, ByVal weight As
Double)
    _text = text
    _navigateUrl = navigateUrl
    _weight = weight
End Sub
End Class

```

You can also use a WCF service to load the items in the **RadTagCloud**. The following steps describe how to achieve this:

1. In the properties pane for the **RadTagCloud** component, set the **WebServiceSettings** property to identify the WCF Web service and service method:
 1. Set the Path sub-property to the URL for the Web service.
 2. Set the Method sub-property to the name of the method of the WCF Web service that supplies child items

ASPX

```
<telerik:RadTagCloud ID="RadTagCloud1" runat="server"
  OnClientItemsRequesting="itemsRequesting"
  OnClientItemsRequested="itemsRequested"
  OnClientItemsRequestFailed="itemsRequestFailed">
  <WebServiceSettings Path="TagCloudWcfService.svc" Method="GetRadTagCloudItems" />
</telerik:RadTagCloud>
```

where the WCF WebService must be in the website, e.g.:

ASPX

```
<%@ ServiceHost Language="C#" Debug="true" Service="TagCloudWcfService"
  CodeBehind="~/App_Code/TagCloudWcfService.cs" %>
```

2. When the **WebServiceSettings** property is set, an empty context request will be initiated automatically. You can trigger requests to the service by calling the **requestItems()** client-side method. This method has a single parameter, which is sent as an argument to the Web service method.

JavaScript

```
function clientFunction()
{
  //...
  var context = { minUnitPrice: "500" };
  tagCloud.requestItems(context);
  //...
}
```

3. Optionally, set the **OnClientItemsRequesting** property to a client-side event handler that passes context information to the Web service. The Web service can use this context information to determine what child items to return or what properties to set on those child items. Note that setting the context requires an object as shown above.

JavaScript

```
//Fired before the request is sent to the Web Service
function itemsRequesting(sender, args)
{
  //If you want to cancel the request use
  //args.set_cancel(true);
  //The args.get_context()/args.set_context(value) methods get/set the parameter which is
  be sent to the Web Service.
  var context = args.get_context();
}
```

4. Optionally, set the **OnClientItemsRequested** and **OnClientItemsRequestFailed** properties to client-side event handlers that respond when the Web service has successfully loaded child items or when the Web service has generated an error while trying to service the item request, respectively.

JavaScript

```
//Fired when the items are successfully loaded.
function itemsRequested(sender, args)
{
  //...
}
```

```
//Fired when the request for the items fails.
function itemsRequestFailed(sender, args)
{
    // Disable the notifying error alert.
    args.set_cancelErrorAlert(true);
    //...
}
```

5. To use the integrated WCF WebService support of **RadTagCloud**, the WCF WebService should have the following signature:

C#

```
[ServiceContract(Namespace = "")]
[AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
public class TagCloudWcfService
{
    [OperationContract]
    public TagDataItem[] GetRadTagCloudItems(IDictionary context)
    {
        string argument = (string)context["minUnitPrice"];
        List<TagCloudDataItem> result = new List<TagCloudDataItem>();
        //.....
        TagCloudDataItem item = new TagCloudDataItem();
        item.Text = "Item";
        item.Weight = 6.6;
        item.NavigateUrl = "NavigateUrl";
        item.AccessKey = "AccessKey";
        item.TabIndex = 5;
        item.ToolTip = "ToolTip";
        item.Value = "ToolTip";
        result.Add(item);
        //.....
        return result.ToArray();
    }
}
```

VB.NET

```
<ServiceContract([Namespace] := "")> _
<AspNetCompatibilityRequirements
(RequirementsMode:=AspNetCompatibilityRequirementsMode.Allowed)> _
Public Class TagCloudWcfService
    <OperationContract()> _
    Public Function GetRadTagCloudItems(context As IDictionary) As TagDataItem()
        Dim argument As String = DirectCast(context("minUnitPrice"), String)
        Dim result As New List(Of TagCloudDataItem)()
        '.....
        Dim item As New TagCloudDataItem()
        item.Text = "Item"
        item.Weight = 6.6
        item.NavigateUrl = "NavigateUrl"
        item.AccessKey = "AccessKey"
        item.TabIndex = 5
    End Function
End Class
```

```
        item.ToolTip = "ToolTip"  
        item.Value = "ToolTip"  
        result.Add(item)  
        '.....'  
        Return result.ToArray()  
    End Function  
End Class
```

32 RadRating

32.1 Objectives

- Explore features of the RadRating control.
- Learn how to configure RadRating.
- Explore the RadRating design time interface including the Smart Tag and major property groups.
- Learn some advance customizations.
- Learn how to control RadRating using its client-side API.

32.2 Introduction

Telerik RadRating is a flexible UI component that allows users to intuitively rate by selecting the number of items [stars] from a predefined maximum number of items. The user can fully customize the control by configuring its orientation, rating precision, direction etc.

Key features:

- **Horizontal/Vertical Orientation** - depending on your needs, RadRating can be displayed horizontally or vertically on the page by setting the **Orientation** property.
- **Direction** - you can configure the RadRating control to reverse its standard direction using its **IsDirectionReversed** property. The standard direction is from left to right (or from top to bottom if it has vertical orientation).
- **Maximum Number of Items** - by setting a value to the **ItemCount** property you can easily choose the maximum number of items the user can rate from.
- **Selection Mode** - it can be Single or Continuous. In Single mode a single item [star] is marked as selected and in Continuous mode all items, starting from the first one, are marked as selected.
- **Rating Precision**- the RadRating control enables the users to select their rating value precisely. By setting the **Precision** property to one of the following: Exact, Half, Item - you can rate by selecting: a precise part of the star [Exact], half a star [Half] or the whole star [Item].

32.3 Getting Started

In this section you will become familiar with the RadRating control.

Setting up RadRating

Below are the basic steps needed to install and configure the **RadRating** control in Visual Studio:

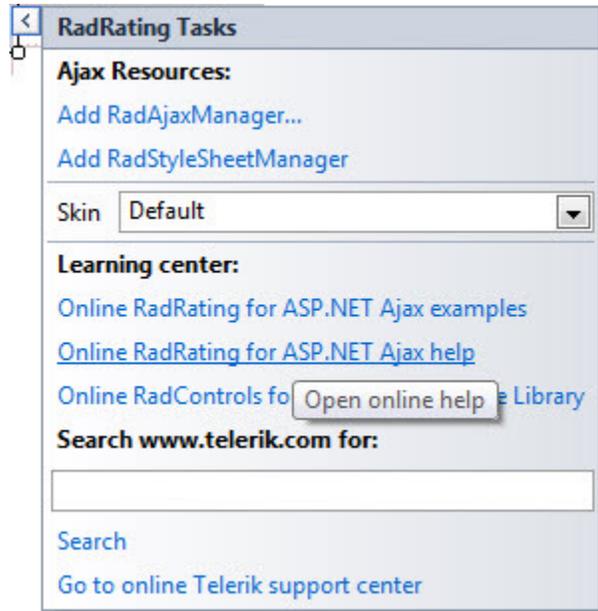
1. Drag ASP ScriptManager or -RadScriptManager control from the VS Toolbox to the page.
2. Drag RadRating from VS Toolbox to the page
3. Set **RadRating's** common properties
4. Build and view the result in the browser

Designer Interface

Smart tags

The RadRating Smart Tag contains only the common elements of RadControls Smart Tags: the Ajax Resources,

Skin selection, and Learning center:



Property window

Configuration

- You can specify the number of the rating items by setting value to **ItemCount** property
- Changing the voting direction from right to left is easily done by turning **IsDirectionReversed** property on
- If you want you can change the rating's precision to Item, Half-item or Exact using the **Precision** property
- In order to disable the rating and use it to display its current value you need to turn on the **ReadOnly** property
- Turning **AutoPostBack** property on will cause the RadRating control to trigger a postback when the user rate

Appearance

This group of properties controls appearance on several levels:

- Individual property settings such as **BorderColor**, **BorderWidth**, **ForeColor**, etc. These properties will work in limited scenarios where the styles or skins are not already at work and where you have a property that already addresses the visual change you need to make.
- Skins: You can set the **Skin** to an predefined value to get a coordinated look-and-feel. You can also customize an existing skin or build your own from scratch. Skins provide a generalized framework

Client-Side Events

We will explore these events in the upcoming section on Client-Side Programming. For now, just know the events fire on the client when Rating is first loaded, and when the user rates.

32.4 Server-Side Programming

Items Collection

From the server you can work with RadRating's Items collection. It enables the user to individually customize every single item. By setting RadRatingItem's properties you can easily give diverse look and functionality to different the items:

- **Value** - the decimal value associated with the RadRatingItem.

- **ToolTip** - the tooltip shown when the mouse pointer is hovered over the item.
- **CssClass** - the CSS class applied to the item.
- **ImageUrl** - the URL of the image displayed when item is not rated.
- **SelectedImageUrl** - the URL of the image displayed when the item is selected (rated).
- **HoveredImageUrl** - the URL of the image displayed when the item is not rated but the mouse pointer is over the item.
- **HoveredSelectedImageUrl** - the URL of the image displayed when the item is selected (rated) and the mouse pointer is over the item.
- **ItemHeight(RadRating)** - the height of every item. (The property is set to the RadRating control and not to the Item.)
- **ItemWidth (RadRating)** - the width of every item. (The property is set to the RadRating control and not to the Item.)



Server-side Events

RadRating offers one server-side event **Rate**. This event can be used in combination with **AutoPostBack=true**, this way you can handle on the server when user has rated.

C# Declaring RadRating with custom Items server-side and assigning handler to the Rate event

```
protected void Page_Load(object sender, EventArgs e)
{
    RadRating RadRating1 = new RadRating();
    RadRating1.SelectionMode = RatingSelectionMode.Single;
    RadRating1.AutoPostBack = true;
    RadRating1.Rate += new EventHandler(RadRating1_Rate);
    RadRatingItem negativeVote = new RadRatingItem();
    negativeVote.Value = -1;
    negativeVote.ImageUrl = "Images/down.png";
    negativeVote.HoveredImageUrl = "Images/downh.png";
    negativeVote.HoveredSelectedImageUrl = "Images/downh.png";
    negativeVote.SelectedImageUrl = "Images/downh.png";
    negativeVote.ToolTip = "No";
    RadRating1.Items.Add(negativeVote);
    RadRatingItem emptyVote = new RadRatingItem();
    emptyVote.Value = 0;
    emptyVote.ImageUrl = "Images/0.png";
    emptyVote.HoveredImageUrl = "Images/0h.png";
    emptyVote.HoveredSelectedImageUrl = "Images/0h.png";
    emptyVote.SelectedImageUrl = "Images/0.png";
    emptyVote.ToolTip = "Reset Current Rating";
    RadRating1.Items.Add(emptyVote);
    RadRatingItem positiveVote = new RadRatingItem();
    positiveVote.Value = 1;
}
```

```
positiveVote.ImageUrl = "Images/up.png";
positiveVote.HoveredImageUrl = "Images/uph.png";
positiveVote.HoveredSelectedImageUrl = "Images/uph.png";
positiveVote.SelectedImageUrl = "Images/uph.png";
positiveVote.ToolTip = "Yes";
RadRating1.Items.Add(positiveVote);
Page.Form.Controls.Add(RadRating1);
}
```

VB Declaring RadRating with custom Items server-side and assigning handler to the Rate event

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
    Dim RadRating1 As New RadRating()
    RadRating1.SelectionMode = RatingSelectionMode.[Single]
    RadRating1.AutoPostBack = True
    AddHandler RadRating1.Rate, AddressOf Me.RadRating1_Rate
    Dim negativeVote As New RadRatingItem()
    negativeVote.Value = -1
    negativeVote.ImageUrl = "Images/down.png"
    negativeVote.HoveredImageUrl = "Images/downh.png"
    negativeVote.HoveredSelectedImageUrl = "Images/downh.png"
    negativeVote.SelectedImageUrl = "Images/downh.png"
    negativeVote.ToolTip = "No"
    RadRating1.Items.Add(negativeVote)
    Dim emptyVote As New RadRatingItem()
    emptyVote.Value = 0
    emptyVote.ImageUrl = "Images/0.png"
    emptyVote.HoveredImageUrl = "Images/0h.png"
    emptyVote.HoveredSelectedImageUrl = "Images/0h.png"
    emptyVote.SelectedImageUrl = "Images/0.png"
    emptyVote.ToolTip = "Reset Current Rating"
    RadRating1.Items.Add(emptyVote)
    Dim positiveVote As New RadRatingItem()
    positiveVote.Value = 1
    positiveVote.ImageUrl = "Images/up.png"
    positiveVote.HoveredImageUrl = "Images/uph.png"
    positiveVote.HoveredSelectedImageUrl = "Images/uph.png"
    positiveVote.SelectedImageUrl = "Images/uph.png"
    positiveVote.ToolTip = "Yes"
    RadRating1.Items.Add(positiveVote)
    Page.Form.Controls.Add(RadRating1)
End Sub
```

32.5 Client-Side Programming

RadRating creates a client side object with ClientID of the menu. You can obtain the reference to this client-side object like with all the RadControls using the `$find()` method:

Getting Reference to the RadRating Client-Side Object

```
var rating = $find("<%= RadRating1.ClientID %>");
```

Then you can use the client-side API of the control to achieve various scenarios. The following example

demonstrates how to toggle the RadRating's readOnly mode from a button on the page:

Toggle ReadOnly mode example

```
<telerik:RadRating ID="RadRating1" runat="server">
</telerik:RadRating>
<asp:Button ID="ToggleReadOnly" runat="server" Text="Toggle ReadOnly mode"
OnClick="toggleReadOnly(); return false;" />
<script type="text/javascript">
function toggleReadOnly()
{
var rating = $find("<%= RadRating1.ClientID %>");
rating.set_readOnly(!rating.get_readOnly());
}
</script>
```

You can use the Client-side API to attach multiple event handlers to the RadRating's client-side events or by providing an event handler to the specific event property.

Client-Side Events

```
<telerik:RadRating ID="RadRating1" runat="server" OnClientLoad="OnClientLoad">
</telerik:RadRating>
<script type="text/javascript">
function OnClientLoad(sender, args)
{
alert("RadRating client-side object is loaded");
sender.add_rating(function ()
{
alert("This code will be executed before the rate");
});
var ratedHandler = function ()
{
alert("This code will be executed right after the rate");
}
sender.add_rated(ratedHandler);
}
</script>
```

32.6 Summary

In this chapter you looked at the RadRating control and saw some of the powerful features it provides. We explored the client-side and server-side properties of the control. You also learned how to configure the control server-side or using the Property window and saw how to use the server-side event Rate, as well as how to configure the control to use custom items programmatically.

33 RadRibbonBar

33.1 Objectives

- Explore the basic structure of the RadRibbonBar control and learn more about the different items that the control holds.
- Learn how to create a simple RadRibbonBar control.
- Explore some of the client-side methods for working with the items collection and some of the client-side events.
- Explore the server-side methods for working with the items collection and some server-side events.
- Learn how to use the ItemTemplate of the RadRibbonBar control.
- Explore RibbonBar's Image Rendering Mode and learn how to add correctly images to Buttons.

33.2 Introduction

The Telerik ASP.NET AJAX RibbonBar control allows you to easily organize the navigation of your application in a simple, structured way. The control includes enhanced navigation capabilities by elegantly grouping menu items in a RibbonBarMenu. The RibbonBarMenu along with the different types of buttons like SplitButtons and ToggleButtons are put into nicely styled groups. Furthermore, the content of these groups can easily be accessed simply by clicking on their tabs. The smartly designed layout allows each item to have various image sizes. Thus you can set the different images accordingly depending on its size.

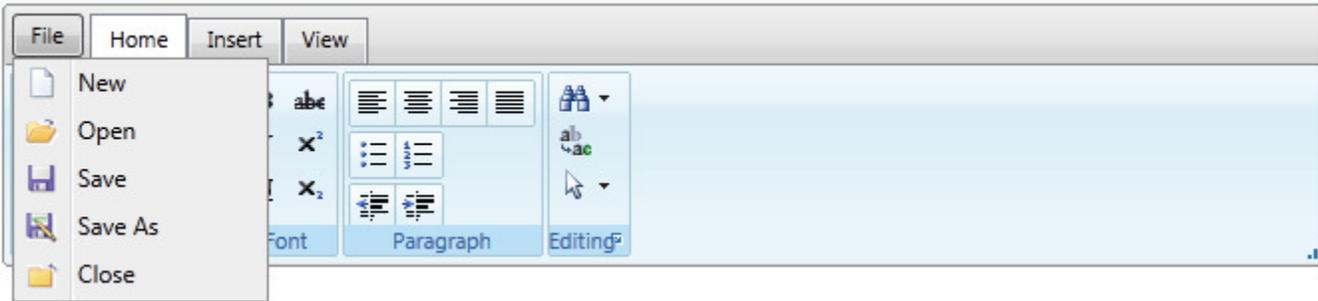
Here is a more detailed description of the RibbonBar items. First we will start with the **RibbonBarApplicationMenu**.

In fact the **Application Menu** is specially designed to be used only within the RadRibbonBar control. With this new addition, you can enrich the RibbonBar experiences of your pages by adding a list of 'application commands' to the tab row of the control. These commands are contained in a drop down, which opens when clicking on the "application menu button", which resides right in-front of the first 'tab' of the RibbonBar. The commands have **Text**, **Value** and **ImageUrl** properties.

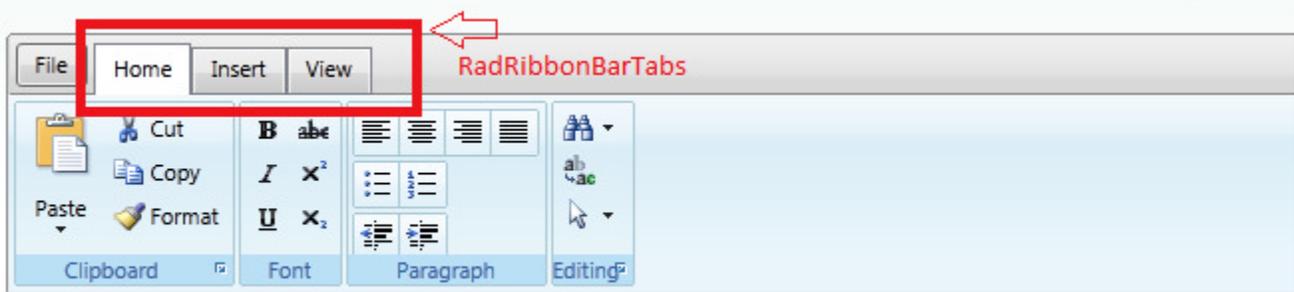
ASPX

```
<telerik:RadRibbonBar ID="RadRibbonBar1" runat="server" SelectedTabIndex="0">
  <ApplicationMenu ID="RibbonBarApplicationMenu1" runat="server" Text="File" >
    <Items>
      <telerik:RibbonBarApplicationMenuItem Text="New" ImageUrl="icons/file/New.gif" />
      <telerik:RibbonBarApplicationMenuItem Text="Open" ImageUrl="icons/file/Open.gif" />
      <telerik:RibbonBarApplicationMenuItem Text="Save" ImageUrl="icons/file/Save.gif" />
      <telerik:RibbonBarApplicationMenuItem Text="Save As"
ImageUrl="icons/file/SaveAs.gif" />
      <telerik:RibbonBarApplicationMenuItem Text="Close" ImageUrl="icons/file/Close.gif" />
    </Items>
  </ApplicationMenu>
  <Tabs>
    <telerik:RibbonBarTab Text="Tab1">
      <telerik:RibbonBarGroup Text="Group1" Value="1">
        <Items>
          <telerik:RibbonBarButton Text="New" />
          <telerik:RibbonBarButton Text="Edit" />
        </Items>
      </telerik:RibbonBarGroup>
    </telerik:RibbonBarTab>
  </Tabs>
</RadRibbonBar>
```

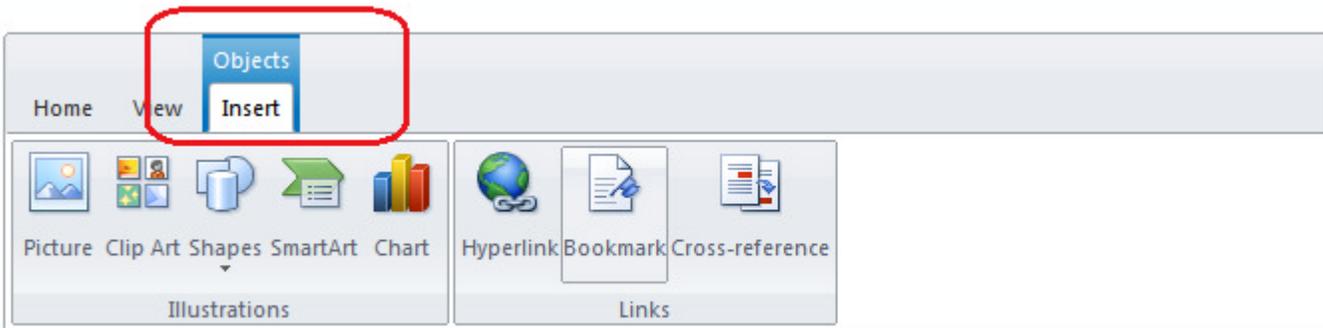
```
</telerik:RadRibbonBar>
```



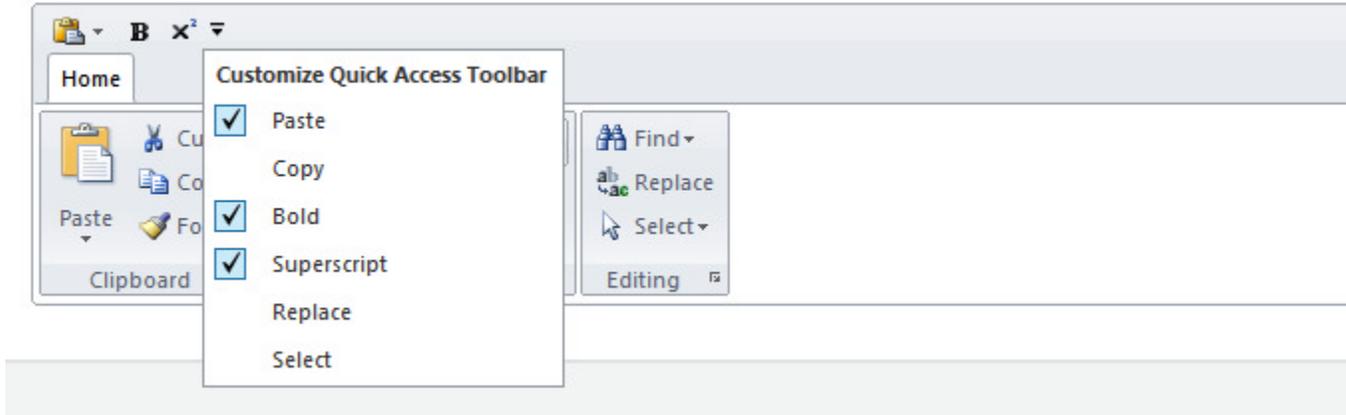
The **RibbonBarTab** on its hand stays on the same level as the Application Menu with the slight difference that it should be placed (in the markup) in the RadRibbonBar control.



RadRibbonBar's **contextual tabs** allows you to group a number of tabs based on some context. Contextual tabs are contained in a contextual tab group, and, following Microsoft's Ribbon specification, are always positioned last (after the normal set of tabs). The contextual tab groups are inactive by default and in order to enable them you will need to set their *Active* property to *true*.



The **Quick Access Toolbar** is also listed in the title bar of the control. It allows you to choose the most used RadRibbonBar commands that are currently available and put their shortcuts in the title bar of the control. The shortcuts function in the same way as the original commands - they fire both their client-side and server-side events. All types of commands (Button, Split Button, Menu, Toggle Button) are supported.



The items are separated in **RibbonBarGroups**, which nicely form collections of similar tools.

Here is a list of the items that can be placed in a RibbonBarGroup

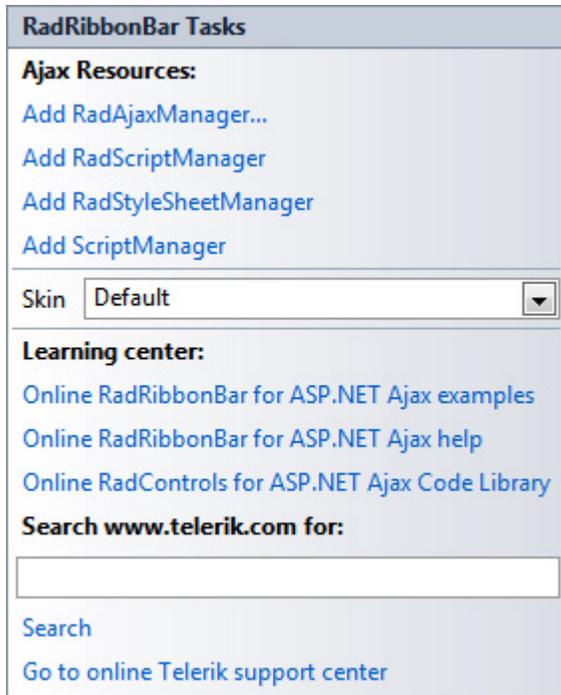
- **RibbonBarButton** - The most used control in every toolbox;
- **RibbonBarSplitButton** - A combination between menu and button;
- **RibbonBarToggleButton** - A button with toggle state;
- **RibbonBarToggleList** - List of ToggleButtons with at most one selected item at any time (similar to `OptionButtonList`);
- **RibbonBarButtonStrip** - A container for buttons with specific appearance;
- **RibbonBarMenu** - Light-weight menu designed specifically for the needs of `RadRibbonBar`;
- **RibbonBarTemplateItem** - For anything else that you would be missing in the rich, but still finite toolbox of `RadRibbonBar`.

33.3 Getting Started

Using RadRibbonBar

The minimum steps to get the `RadRibbonBar` up and running in a browser are:

1. In a new ASP.NET Web Application, drag a `RadRibbonBar` to the default page.
2. Using the Smart tag you can add the `RadScriptManager` to the page.



3. Add a RibbonBarTab to the RadRibbonBar control.
4. Add a RibbonBarGroup inside the above mentioned RadRibbonTab.
5. Add any of the different buttons that you may need in the RibbonBarGroup.

ASPX

```
<telerik:RadRibbonBar ID="RadRibbonBar1" runat="server" SelectedTabIndex="0">
  <telerik:RibbonBarTab>
    <telerik:RibbonBarGroup Text="Group1" >
      <Items>
        <telerik:RibbonBarButton Text="New" />
        <telerik:RibbonBarButton Text="Edit"/>
      </Items>
    </telerik:RibbonBarGroup>
  </telerik:RibbonBarTab>
</telerik:RadRibbonBar>
```

33.4 Server-Side Programming

Working with the Items collection

RadRibbonBar control supports a number of methods for locating items placed in the control. For example here are some of them:

- **FindButtonByValue** - returns a reference to a button given the value of its Value property.
- **FindControl** - returns a reference to an item given the value of its ID.
- **FindGroupByValue** - returns a reference to a group given the value of its Value property.
- **FindMenuItemByValue** - returns a reference to a menu item given the value of its Value property.
- **FindTabByValue** - returns a reference to a tab given the value of its Value property.
- **FindToggleButtonByValue** - returns a reference to a toggle button given the value of its Value property.

Once you have located an item, you can use its properties to change its value, select it, disable it, delete it and so on.



You can find the complete source for this project at:

`\VS Projects\RibbonBar\FindItem`

- **ApplicationMenuItemClick** - Occurs when an item of the Application Menu is clicked
- **ButtonClick** - Occurs when a Button is clicked
- **ButtonToggle** - Occurs when a Toggle Button is toggled
- **LauncherClick** - Occurs when a Group Launcher is clicked
- **MenuItemClick** - Occurs when a Menu Item is clicked
- **SelectedTabChange** - Occurs when a non-selected Tab is clicked
- **SplitButtonClick** - Occurs when a Split Button or a button inside it is clicked
- **ToggleListToggle** - Occurs when a Toggle Button in a Toggle List is toggled

33.5 Client-Side Programming

Working with the Items collection

RadRibbonBar control supports a number of methods for locating items placed in the control. Here are some that could be used on the client-side:

- **findItemByText** - returns the first found item containing the specified text.
- **findTabByValue** - returns a reference to a tab given the value of its Value property.
- **findGroupByValue** - returns a reference to a group given the value of its Value property.
- **findButtonByValue** - returns a reference to a button given the value of its Value property.
- **findToggleButtonByValue** - returns a reference to a toggle button given the value of its Value property.
- **findMenuItemByValue** - returns a reference to a menu item given the value of its Value property.

Once you have located an item, you can use its properties to change its value, text, select it, disable it, delete it etc.



You can find the complete source for this project at:

`\VS Projects\RibbonBar\DisableItemClientSide`

Besides the methods described above there are also various events that could be used for manipulating the different items and buttons in the RadRibbonBar control. Currently the RadRibbonBar supports the following events:

- **OnClientApplicationMenuItemClicked**
- **OnClientApplicationMenuItemClicking**
- **OnClientButtonClicked**
- **OnClientButtonClicking**
- **OnClientButtonToggled**
- **OnClientButtonToggling**
- **OnClientLauncherClicked**
- **OnClientLauncherClicking**

- OnClientLoad
- OnClientMenuItemClicked
- OnClientMenuItemClicking
- OnClientSelectedTabChanged
- OnClientSelectedTabChanging
- OnClientSplitButtonClicked
- OnClientSplitButtonClicking
- OnClientToggleListToggled
- OnClientToggleListToggling

33.6 How -to

Using a RadComboBox control in a Template



You can find the complete source for this project at:
 \VS Projects\RibbonBar\FindControlInsideTemplate

How to set width of the RibbonBarButtons?

RadRibbonBar doesn't support for setting the width in pixels for the RibbonBarButtons or regular RibbonBarItem, in general (exception is made only for RibbonBarTemplateItem). The approach that we adopt here is to give the choice of preset Sizes (Small, Medium and Large).

Image Rendering

Since RadRibbonBar is a control that follows very tightly the Microsoft's "Ribbons" specification and guide-lines it enforces usage of images for almost all RibbonBarItem.

As there is a notion for Size (RibbonBarItemSize) as property of all items, the need for different images for all sizes (3 sizes using all together 2 sizes for images - 16x16 for Small and Medium and 32x32 for Large) emerges.

In order to provide more, we have implemented the ability to set images for the disabled state of RibbonBarItem (in all sizes) as well. Which means the images can be as many as 4 per item. This variety of choices has its own problems. Setting all the images in properties one-by-one (and storing them in separate files), proves to be hard. And in some cases even requires additional work to be done - some graphic libraries are provided by vendors in as "clip" images (small and large image in a small sprite - in 1 file). Still having the ability to change a single image, without the need of setting (creating) a new clip, proves to be more flexible, therefore potentially very useful. In order to satisfy both the camps, we decided to implement Image Rendering Mode, in order to switch between single-image model (called Dual) and clip-image (called Clip).

Image Rendering Mode is implemented on both control and item level. And in both places the property is called the same ImageRenderingMode and has the same values (binded from the enumeration RibbonBarImageRenderingModes). The values of the enumeration are as follows:

- **Auto** (ImageRenderingMode.Auto) - When no mode is explicitly selected (or Auto is), RadRibbonBar tries to determine the appropriate rendering mode on per-item basis. We strongly recommend to always explicitly set on of the other two modes - Clip and Dual. Simply because there are items with very hard to determine mode scenarios. Perhaps the best example of such item is RibbonBarSplitButton - it has a selected button, but at the same time it has the possibility of setting all images directly on it. This makes the determining of the Rendering Mode far too complex and many scenarios cannot be covered automatically. Otherwise the algorithm is basically working as following: if there is a large image set (ImageUrlLarge or DisabledImageUrlLarge), the mode is Dual. If there are no

large images set, but there is a small image set - the mode is Clip. If no images are set - the mode is Dual again and default RibbonBar images are displayed.

- **Clip** (`ImageRenderingMode.Clip`) - In this mode the images are assigned through the `ImageUrl` and `DisabledImageUrl`. `ImageUrl` image contains both small and large images for the enabled state of the item, and the `DisabledImageUrl` contains the images for the disabled state. Images set in the `ImageUrlLarge` and `DisabledImageUrlLarge` are disregarded.
- **Dual** (`ImageRenderingMode.Dual`) - When Dual is explicitly set (or resolved to, from Auto), small images are set through `ImageUrl` and `DisabledImageUrl` and large images are set through `ImageUrlLarge` and `DisabledImageUrlLarge`.

By default the selected value is Auto.

As previously mentioned, the `ImageRenderingMode` can be set on `RadRibbonBar` and on any `RibbonBarItem`. This means that you can set a general rule on `RibbonBar` level and make exceptions on the level of item. In order to fully show the power of this approach, you can find an example here:



You can find the complete source for this project at:
`\VS Projects\RibbonBar\ImageRendering`

33.7 Summary

In this chapter you explored the structure of the `RadRibbonBar` and saw some of the powerful features it provides. You learned how to create a `RadRibbonBar`. Moreover, you've learned some of the server-side and client-side properties and events of the control. You also learned how to manipulate different items of the `RadRibbonBar` using both server-side and client-side code. Finally, you've learned how to use Templates in `RadRibbonBar`. You have built a simple application that used templates and learned how to find controls in a `RadRibbonBar` `ItemTemplate` with a nested control.

34 RadOrgChart

34.1 Objectives

- Explore the most common features of our new control - RadOrgChart. Including what the control is primarily designed for and how to use it.
- Create your first **OrgChart** using **Simple Data Binding**
- Learn what rendered fields are and how to use them.
- Get acquainted to templates and get to know how to apply them.

34.2 Introduction

Telerik **RadOrgChart** for ASP.NET AJAX is a flexible organizational chart control for ASP.NET applications. It is specially designed to represent a structure of an organization in terms of relationships among personnel and/or departments. It is a powerful and at the same time easy-to-use control that represents data in the most intuitive way so that the final user can understand the structure of an organization at a single glance.

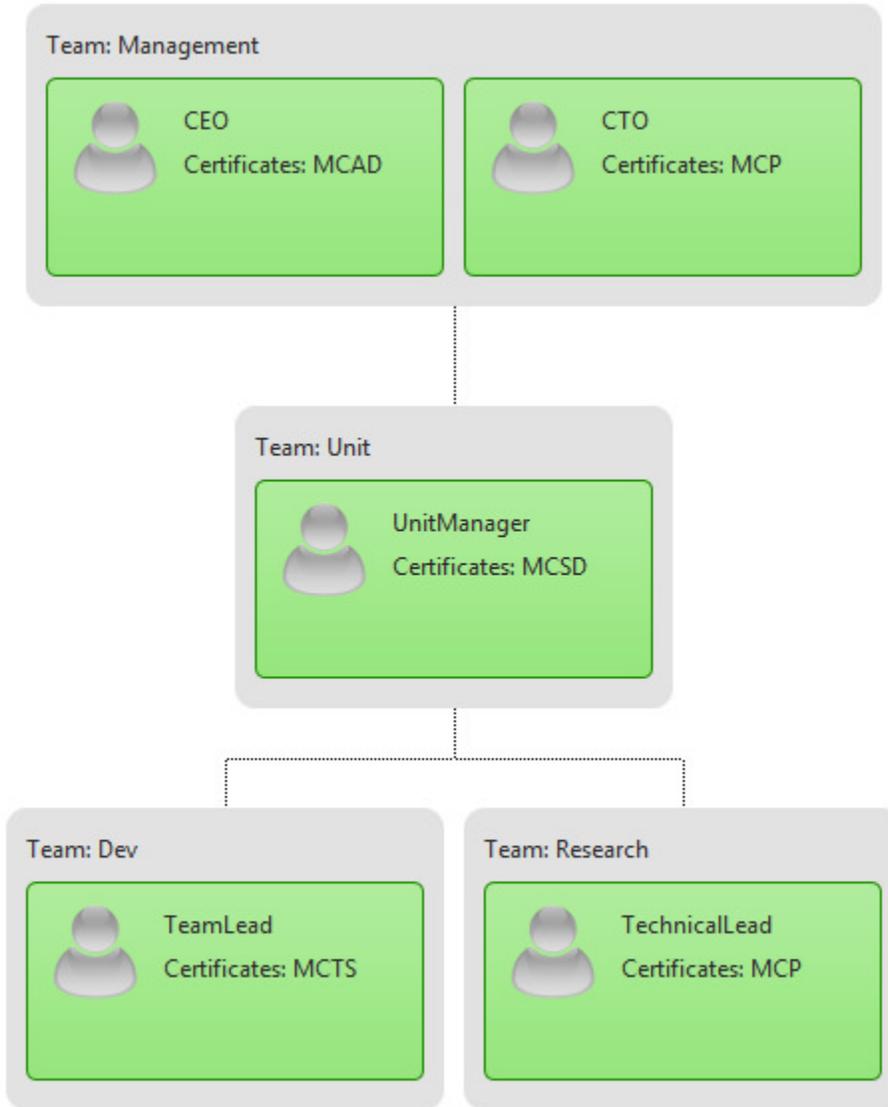
RadOrgChart control supports **drag-and-drop** of Nodes and GroupItems. By setting **EnableDragAndDrop** property to true the drag-and-drop functionality is enabled. To further customize this functionality of the RadOrgchart control you can use the **NodeDrop** and **GroupItemDrop** server side events. When a node is dropped the NodeDrop event is fired; when a GroupItem is dropped the GroupItemDrop event is fired. Note also that in order to change the OrgChart hierarchy you need to handle the events and update the underlying data sources. No automatic updates are available.

RadOrgChart supports **Expand/Collapse** of the RadOrgChart hierarchical tree. The functionality can be enabled by setting the **EnableCollapsing** property to true. The controls also supports **Expand/Collapse of a Group**. To enable the functionality you need to set the **EnableGroupCollapsing** property to true. The functionality, however, is only available when a Node has more than one GroupItem.

RadOrgChart supports various bindings to different data sources which are relatively simple to setup. Here are some of the supported ones for example:

- Declarative DataSources
 - ObjectDataSource
 - SQLDataSource
- Entity DataSource
- Linq DataSource
- IEnumerable (Programmatic DataBinding)

Not only does the **RadOrgChart** control support the above mentioned data bindings, but it also supports exporting and importing to and from Xml. Its innovative rendering is represented by some clever features like **RenderedFields**, **Templates** (on 3 different levels), **Groups**, **Column Count** (a property of OrgChartNode used to specify the number of columns locally for the group) etc. More detailed description of some these features is provided in the Control Specifics section below.



34.3 Getting Started

In general the **RadOrgChart** control is designed to be bound to a Data Source; however, you could also use the markup to create a simple hierarchy with the RadOrgchart like the following one:

ASPX

```

<telerik:RadOrgChart runat="server" ID="RadOrgChart1" GroupColumnCount="3">
  <Nodes>
    <telerik:OrgChartNode>
      <GroupItems>
        <telerik:OrgChartGroupItem Text="John Bravo" />
        <telerik:OrgChartGroupItem Text="Nancy Davolio" />
        <telerik:OrgChartGroupItem Text="Andrew Fuller" />
        <telerik:OrgChartGroupItem Text="Margaret Peacock" />
        <telerik:OrgChartGroupItem Text="Michael Suyama" />
        <telerik:OrgChartGroupItem Text="Janet Leverling" />
      </GroupItems>
    </telerik:OrgChartNode>
  </Nodes>
</telerik:RadOrgChart>
  
```

```

</GroupItems>
<Nodes>
  <telerik:OrgChartNode ColumnCount="2">
    <GroupItems>
      <telerik:OrgChartGroupItem Text="Don Marko" />
      <telerik:OrgChartGroupItem Text="Sony Gustavo" />
      <telerik:OrgChartGroupItem Text="Boni Taylor" />
      <telerik:OrgChartGroupItem Text="Sara Darkman" />
    </GroupItems>
    <Nodes>
      <telerik:OrgChartNode>
        <GroupItems>
          <telerik:OrgChartGroupItem Text="Hun-ni Ho" />
          <telerik:OrgChartGroupItem Text="Lukas Brezina" />
          <telerik:OrgChartGroupItem Text="Viktor Varga" />
          <telerik:OrgChartGroupItem Text="Marianna Weissova" />
          <telerik:OrgChartGroupItem Text="David Maly" />
          <telerik:OrgChartGroupItem Text="Lin-Sheng Fen" />
        </GroupItems>
      </telerik:OrgChartNode>
    </Nodes>
  </telerik:OrgChartNode>
</Nodes>
</telerik:RadOrgChart>

```

34.4 Control Specifics

Basic structure of the RadOrgChart

RadOrgChart consists of two main objects: items and nodes. Why did we decide to include both? Well, the reason is very simple, for example: in a real world company it is very likely that a department of people is represented by a single unit and therefore we needed to structure these units. Hence we decided to include both nodes and items in order to avoid confusion and nesting of nodes. Below you can see how the actual object model looks like:

```

RadOrgChart
  NodesCollection
  Node
    NodesCollection
    GroupItemsCollection
    GroupItem

```

Rendered Fields

The control offers an API that allows the user to include custom data fields in the default rendering of both Nodes and Items and hence making the control extremely easy to use and customize. For instance, all of the desired changes (that could be displaying additional data for each Group Item) could be achieved only by using a single tag and its properties. Using RenderedFields you can add extra information about every Node or Item in RadOrgChart. However, if you need to set custom fields on a Node while binding RadOrgChart's data, you'll need to use Group-Enabled Binding.

RenderedFields for Nodes or Items can be set either from the mark up in section RenderedFields or in code bind. Below are the properties that could be applied:

- DataField - this is the field name from the data source that populates each custom field's text property

during binding.

- **Label** - short description about custom field's text. It is optional.



You can find the complete source for this project at:
`\VS Projects\OrgChart\RenderedFields`

Data Binding

The **RadOrgChart** control provides many different kinds of data bindings and at the same time ease and simplicity in setting up the binding. In order to support a simple, straight-forward binding to a self-referenced data, without complex relations such as groups, **RadOrgChart** supports 2 different kinds of binding: first of which we called Simple Data Binding and Group-Enabled Binding.



You can find the complete source for this project at:
`\VS Projects\OrgChart\SimpleDataBinding`

34.5 Server-Side Programming

RadOrgChart introduces several server-side events for customizing the behavior of the nodes. Here are several that you may find useful:

- **OnGroupItemDataBound** - occurs when **OrgChartGroupItem** is created from a data source and added to its parent-node's **GroupItems** Collection;
- **OnNodeDataBound** - occurs after **OrgChartNode** is created from a data source;
- **GroupItemDrop** - occurs when a **GroupItemDrop** is dragged and dropped on **Node** different than the **Node** which contains the item;
- **NodeDrop** - occurs when a **Node** is dragged and dropped on non-child **Node**.

34.6 How-to

Add a Simple Template

RadOrgChart is a highly customizable control. Besides the built-in skins that you can apply, you can further alter the appearance of the nodes by using templates. For example, if the default size of images does not apply to an already collected number of employee profile pictures, a template can be created without the restraint of 48x48 pixels (which is the default image size for **RadOrgChart**).

In order to facilitate that, we implemented a 3-level templating engine giving the power of applying templates on:

1. **OrgChart** level (**ItemTemplate**) - for all items that doesn't have template nor does their node;
2. **Node** level (**ItemTemplate**) - for all items within the node which doesn't have template;
3. **GroupItem** level (simply **Template**) - for the item, with disregard of node or global (**OrgChart.ItemTemplate**) template.



You can find the complete source for this project at:
`\VS Projects\OrgChart\Templates`

34.7 Summary

In this chapter you explored the purpose and application of the **RadOrgChart** control. Then you learned how to build a simple RadOrgChart control using only its markup. A few of the control's details that you explored included reviewing of the **RenderedFields** property, the different data bindings that the control supports, and its basic structure: nodes and items.

35 RadPivotGrid

35.1 Objectives

- Explore the features of the RadPivotGrid control.
- Explore the RadPivotGrid design time interface, including Smart Tag and Editor.
- Create simple application for binding data using the RadPivotGrid and presents the most common features.

35.2 Introduction

In Q2 2012 SP1 a CTP version of new **RadPivotGrid** control was released. RadPivotGrid is a data summarization control where users can break down raw data in any manner they want. A pivot table can help quickly summarize the reports and highlight the desired information. It displays data in formats such as spreadsheets or business intelligence applications.

The key features of the RadPivotGrid control are:

- Various field types
- Codeless data-binding using the DataSourceControls in ASP.NET 3.5/4.0
- Data-Binding to various data sources which implement the IEnumerable, IList, IQueryable or ICustomTypeDescriptor interfaces
- Integrated paging
- Integrated sorting
- Integrated scrolling
- Interoperability with RadAjax and loading indicators - dramatically improves the responsiveness of the component, simulates Windows-application like behavior, and minimizes the traffic to the server
- Easily customizable skinning mechanism (setting single Skin property of the pivotgrid)
- The expanded state of the items is persisted while navigating through pages.

35.3 Getting Started

Here we will describe the main features of the RadPivotGrid and the properties/methods you should know to enable them.

Paging

RadPivotGrid has built-in pager functionality which is available out of the box and is controlled with the **AllowPaging** property. Paging functionality allows the user to fetch and display data by chunks. This behavior provides better performance and ease of use for the user.

You can control how many items will be fetched and displayed by the **PageSize** property. This property is used by **RadPivotGrid** to split the returned result set of the datasource. RadPivotGrid also supports different pager styles that you can choose from. For more information you can see [this article \(http://www.telerik.com/help/aspnet-ajax/pivotgrid-pager-item.html\)](http://www.telerik.com/help/aspnet-ajax/pivotgrid-pager-item.html).

Sorting

To enable pivotgrid sorting functionality in **RadPivotGrid** you should set the **AllowSorting** property to True. After that arrow buttons appear in row headers that are used to select a sort mode. The available options are:

- **Ascending**
- **Descending**

RadPivotGrid can sort columns and rows. By default when sorting is enabled, the results are returned in ascending order. The sorting operation is executed when you click on the field that you want to sort:

Quantity	Year ▼		Quarter ▲				
Category ▼	▲ 1998		1998 Total	▲ 1997			
	Quarter 1	Quarter 2		Quarter 1	Quarter 2	Quarter 3	Quarter 4
Seafood	25.89	20.78	24.25	16.41	23.51	25.39	19.57
Produce	17.84	22.89	19.95	20.53	24.26	21.81	31.50
Meat/Poultry	18.82	29.61	23.04	29.21	21.52	23.83	21.58
Grains/Cereals	22.48	21.65	22.21	25.92	28.88	23.37	18.36
Dairy Products	26.60	25.37	26.11	26.50	23.50	23.50	22.56
Confections	23.01	23.73	23.19	26.80	22.77	22.90	24.79
Condiments	21.77	18.04	20.30	37.00	25.78	22.89	28.56
Beverages	24.30	25.25	24.63	26.87	20.50	21.88	23.71
Grand Total	23.55	23.44	23.51	26.15	23.53	23.37	23.53

Scrolling

You can easily make RadPivotGrid scrollable by setting the ClientSettings -> Scrolling -> AllowVerticalScroll property to True (By default its value is False.) The Horizontal scroll is enabled by default and will appear when the total width of the columns exceeds the width of the pivotgrid.

The ClientSettings->Scrolling->ScrollHeight property specifies the height value beyond which scrolling is turned on. The default value is 300px.

The ClientSettings->Scrolling->SaveScrollPosition property keep the scroll position during postbacks.

 To optimize the RadPivotGrid loading time when scrolling is enabled, you may consider defining ColumnHeaderCellStyle.Width and RowHeaderCellStyle.Height properties. Thus the pivotgrid will not execute additional scripts for aligning.

Caching

RadPivotGrid's aggregate calculations and grouping are driven by a powerful data engine which produces a special pivot view model. The latter feeds the aggregate values into the final output by the control. At times, however, when the data to be aggregated is quite large, the creation of the pivot view model can get pretty demanding in terms of CPU resources. In order to avoid the recalculation of large data that does not change very often, RadPivotGrid allows the caching of the pivot view model into the session state.

Caching is enabled through the EnableCaching property of the control. For large sets of data, using it will result in considerable speeding up of any operations that require the rebinding of the pivot grid.

However, there are trade-offs that should be carefully considered before opting for the employment of this feature:

- **Memory consumption** is very likely to increase significantly if the web page that RadPivotGrid is placed on experiences intense traffic.
- There is evidently **no guarantee the data to be displayed will be up-to-date.**

35.4 RadPivotGrid Fields

UI for ASP.NET AJAX

Pivot Grid Fields represent data source fields and provide specific data to RadPivotGrid. The Fields headers are used for presenting the different fields that can be moved between control areas using drag-and-drop.

Fields:

To presents specific data in the RadPivotGrid, fields should be created and placed in the appropriated areas.

The screenshot shows a RadPivotGrid with the following structure:

- Data Header Area:** Contains the 'TotalPrice' field.
- Row Header Area:** Contains 'Category' and 'ProductName' fields.
- Column Header A:** Contains 'Year' and 'Quarter' fields.

		Year ▲	Quarter ▼		
		► 1996	▲ 1997		
			Quarter 1	Quarter 2	Quarter 3
► Beverages		\$53,879.20	\$38,342.00	\$34,977.00	\$25,429.75
▲ Condiments	Aniseed Syrup	\$240.00	\$560.00	\$740.00	\$260.00
	Chef Anton's Cajun Seasoning	\$1,883.20	\$281.60	\$3,740.00	\$1,716.00
	Chef Anton's Gumbo Mix	\$2,193.00			\$405.65
	Genen Shouyu	\$310.00		\$728.50	\$775.00
	Grandma's Boysenberry Spread	\$720.00			\$2,500.00
	Gula Malacca	\$2,139.00	\$2,433.50	\$1,556.00	\$2,898.05
	Louisiana Fiery Hot Pepper Sauce	\$2,604.00	\$1,394.40	\$3,135.85	\$3,641.65
	Louisiana Hot Spiced	\$408.00	\$1,632.00	\$544.00	\$68.00

Here are the available fields in RadPivotGrid:

- **DataFields** - The PivotGrid calculates summaries against these fields. Visually they can be placed into the Data Header Area.

TotalPrice		Year	Quarter				
Category	ProductName	▲ 1996			1996 Total	▲ 1997	
		Quarter 2	Quarter 3	Quarter 4		Quarter 1	Quar
▲ Beverages	Chai		\$1,584.00	\$216.00	\$1,800.00	\$705.60	\$1,56
	Chang	\$1,596.00	\$1,459.20	\$380.00	\$3,435.20	\$2,660.00	\$418.
	Chartreuse verte	\$691.20	\$2,851.20	\$288.00	\$3,830.40	\$662.40	\$450.
	Côte de Blaye		\$21,080.00	\$8,432.00	\$29,512.00	\$27,193.20	\$16,8
	Guaraná Fantástica	\$154.80	\$306.00	\$108.00	\$568.80	\$550.80	\$675.
	Ipoh Coffee		\$5,004.80		\$5,004.80	\$1,398.40	\$4,60
	Lakkalikööri	\$216.00	\$1,382.40	\$504.00	\$2,102.40	\$1,353.60	\$3,74
	Laughing Lumberjack Lager		\$56.00		\$56.00		\$518.
	Outback Lager	\$492.00	\$660.00	\$720.00	\$1,872.00	\$1,260.00	\$630.
	Rhönbräu Klosterbier		\$744.00		\$744.00	\$458.80	\$2,41

- **ColumnFields** - The PivotGrid represents row headers from these fields. Visually they can be placed into the **Column Header Area**. The ColumnFields control the PivotGrid Columns which can be nested.

Year ▲		Quarter ▼					
▶ 1996	▲ 1997				1997 Total	▲ 1998	▲
	Quarter 1	Quarter 2	Quarter 3	Quarter 4		Quarter 1	
\$53,879.20	\$38,342.00	\$34,977.00	\$25,429.75	\$11,675.25	\$110,424.00	\$93,156.75	
\$240.00	\$560.00	\$740.00	\$260.00	\$200.00	\$1,760.00	\$790.00	
\$1,883.20	\$281.60	\$3,740.00	\$1,716.00		\$5,737.60	\$1,232.00	
\$2,193.00			\$405.65		\$405.65	\$1,067.50	
\$310.00		\$728.50	\$775.00		\$1,503.50		
\$720.00			\$2,500.00		\$2,500.00	\$3,600.00	
\$2,139.00	\$2,433.50	\$1,556.00	\$2,898.05	\$194.50	\$7,082.05	\$778.00	
\$2,604.00	\$1,394.40	\$3,135.85	\$3,641.65	\$1,726.10	\$9,898.00	\$1,662.95	

- **RowFields** - The PivotGrid represents column header from these fields. Visually they can be placed into the Row Header Area.

		▲ 1996						Sum Total
		Quarter 2		Quarter 3		Quarter 4		
		Sum of TotalPrice	Sum of Quantity	Sum of TotalPrice	Sum of Quantity	Sum of TotalPrice	Sum of Quantity	
	Jack's New England Clam Chowder	\$392.70	51	\$423.50	55	\$61.60	8	\$877
	Konbu			\$201.60	42	\$230.40	48	\$432
	Nord-Ost Matjeshering	\$1,242.00	60	\$1,531.80	74	\$372.60	18	\$3,14
	Röd Kaviar			\$300.00	25			\$300
	Rogede sild					\$114.00	15	\$114
	Spegesild			\$432.00	45	\$700.80	73	\$1,13
Seafood Total		\$2,770.50	187	\$15,394.70	811	\$3,424.40	288	\$21,5
Grand Total		\$30,192.10	1462	\$145,153.00	5919	\$50,953.40	2200	\$226

- **Drag and Drop Fileds**- different fields can be moved between control areas using drag-and-drop.This

functionality is used through the context menu of the desired field, as soon as you right click on the field.

Quantity	Year ▲	Quarter			1997 Total	▲ 1998	
Category ▲	▲ 1997					Quarter 1	Quarter 2
	Quarter 1	Quarter 2	Quarter 3	Quarter 4			
Beverages	26.87	20.50	21.88	23.71	22.70	24.30	25.25
Condiments	37.00	25.78	22.89	28.56	27.31	21.77	18.04
Confections	26.80	22.77	22.90	24.79	24.19	23.01	23.73
Dairy Products	26.50	23.50	23.50	22.56	24.03	26.60	25.37
Grains/Cereals	25.92	28.88	23.37	18.36	25.10	22.48	21.65
Meat/Poultry	29.21	21.52	23.83	21.58	24.32	18.82	29.61
Produce	20.53	24.26	21.81	31.50	23.63	17.84	22.89
Seafood	16.41	23.51	25.39	19.57	22.71	25.89	20.78
Grand Total	26.15	23.53	23.37	23.53	24.07	23.55	23.44

To show a hidden field you need to use the Fields Window and drag the desired field to its new location:

Quantity	Year ▲	Quarter				
Category ▲	▲ 1997					
	Quarter 1	Quarter 2	Quarter 3	Quarter 4		
Beverages	26.87	20.50	21.88	23.71		
Condiments	37.00	25.78	22.89	28.56		
Confections	26.80	22.77	22.90	24.79		
Dairy Products	26.50	23.50	23.50	22.56		
Grains/Cereals	25.92	28.88	23.37	18.36		
Meat/Poultry	29.21	21.52	23.83	21.58		
Produce	20.53	24.26	21.81	31.50		
Seafood	16.41	23.51	25.39	19.57		
Grand Total	26.15	23.53	23.37	23.53		

RadPivotGrid Fields Wind...

TotalPrice

35.5 Summary

In this chapter we looked at the RadPivotGrid control and explored its most commonly used features like paging, sorting, scrolling, caching and drag drop fields.

You learned how to use RadPivotGrid in Design Time and build its layout with ease.

You saw how to implement a sample project on how to manipulate the RadPivotGrid appearance.

36 RadSocialShare

36.1 Objectives

- Explore the features of the **RadSocialShare** control
- Understand the use of its properties
- Get familiar with the different types of buttons and the button collections
- Choose the buttons
- Configure the buttons
- Get a simple example running
- Learn how to control the URL and Title of the user's post
- See the functionality of the third-party buttons

36.2 Introduction

The **RadSocialShare** is a control that allows you to easily connect your site with popular social networks or let the user send an e-mail with a link. It creates a centralized bar in which you can choose in which networks your users will be able to share the content. You are also allowed to make only a handful of the buttons visible initially and place the rest in a popup with a search box.

Key features:

- Allows sharing of current page or custom url on different social networks
- Allows the developer to preset the title of the post the user will share - you can override the page title/URL
- Visual designer that allows you to quickly and easily configure all the buttons
- Compact popup with filter capabilities to save space
- Tell a friend dialog for sending e-mails
- Offers styled buttons as well as standard third party buttons
- Easy to configure both styled and standard third party buttons through simple properties
- Easy to customize the visual appearance of styled buttons
- XHTML compliant implementation of third party buttons
- Full multi-trackers Google Analytics support

36.3 Button Types And Button Collections

Buttons in the **RadSocialShare** are grouped in two inner tags: the **MainButtons** and the **CompactButtons**. In the first group are the ones that are always visible on the page and the latter are the ones that can be shown in a **RadWindow** popup which also provides a search box to filter the available buttons by their name (label). To trigger the popup you need a **CompactButton** in the **MainButtons** collection. If the **CompactButtons** collection is left empty, all the **Styled Buttons** that are not in the **MainButtons** collection will be automatically populated in the popup.

The **Styled Buttons** are the ones that are built-in and provide a consistent look and feel. They utilize the public API that the different social networks provide and their appearance can easily be customized by the developer.

There are two specific **Styled Buttons** available, which send an e-mail either via the user's machine, or with a built-in mail form. The third special **Styled Button** is the **Compact Button**.

The **Standard Buttons** are created by external scripts from the respective social network and offer a larger set of options. There are three sites that offer this functionality - Facebook, Twitter and Google. These buttons can only be present in the **MainButtons** collection.

36.4 Important Properties

RadSocialShare is ready to be used once you add the buttons you desire. It is as simple as that. When none of its properties are set explicitly they take their default values and the most important one - the **UrlToShare** is taken from the current page's URL.

Here follows a list with the most important properties and their effect:

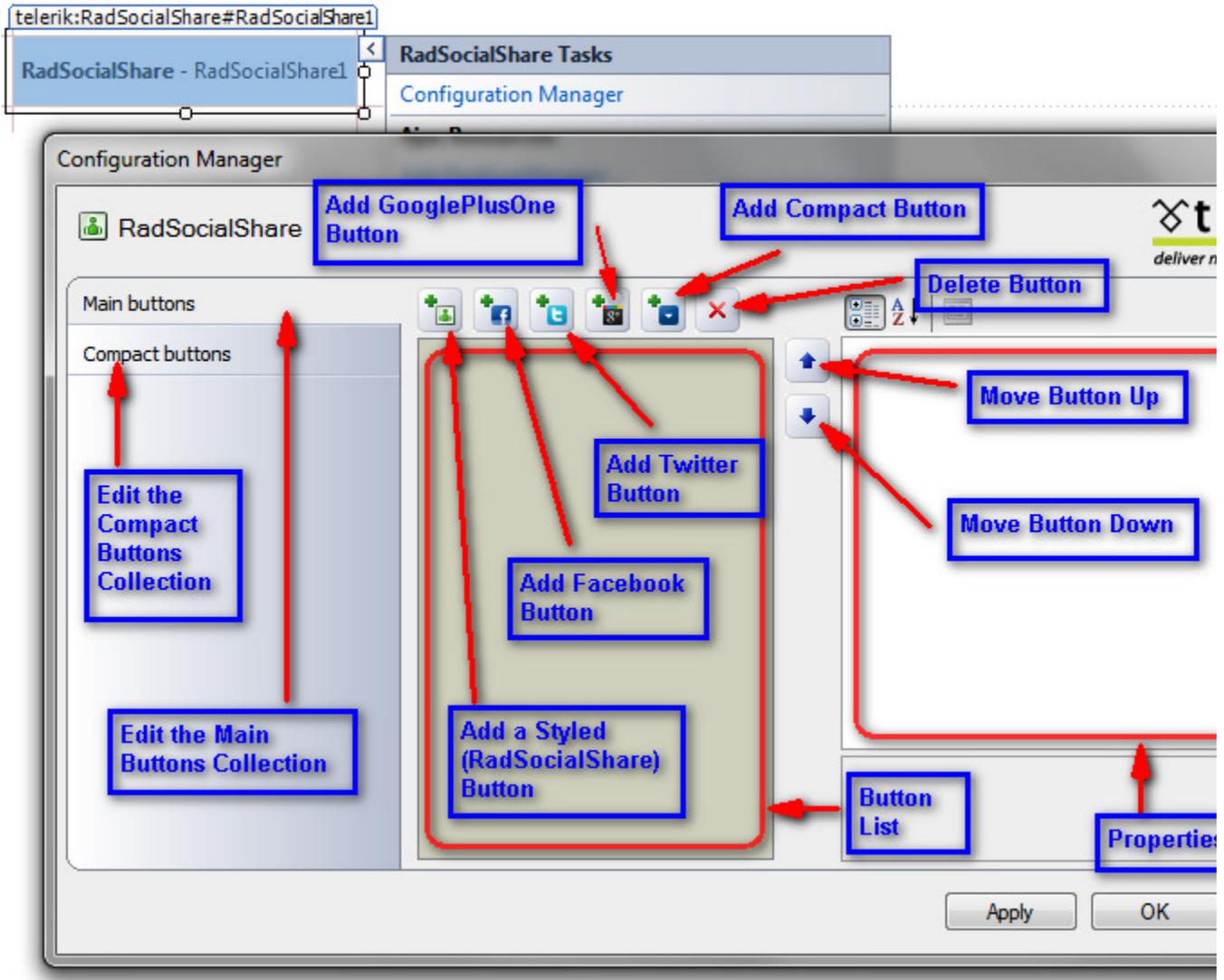
- **UrlToShare** - presets the URL the end user shares in the social network. Uses the current page URL if left empty
- **TitleToShare** - presets the title of the user's post / subject of the mail
- **Orientation** - controls whether the RadSocialShare main bar will be vertical or horizontal
- **Height** and **Width** - control the dimensions of the RadSocialShare
- **DialogHeight** and **DialogWidth** - set the size of the popups in which the social network site opens. By default this is 500px
- **DialogLeft** and **DialogTop** - set the position of the browser popup for the social network. By default it opens centered on the screen
- The buttons also expose some of the main properties which override the global ones - **UrlToShare**, **TitleToShare**, **DialogHeight**, **DialogWidth**, **DialogLeft**, **DialogTop**

Each **Styled Button** exposes several properties that allow the developer to customize their appearance:

- **CustomIconUrl** - an URL to the custom image that should be used
- **CustomIconHeight** and **CustomIconWidth** - the custom dimensions you wish the button to have
- **LabelText** - the text next to the button that can also be clicked
- **ToolTip** - the tooltip when the button or the label is hovered

36.5 Using The Configurator

The **RadSocialShare** control offers a visual configurator for the buttons. It is easy to use and with just a few clicks you can setup the collections as you please. Its main features are outlined below:



The left column lets you choose which of the **Button Collections** (Section 36.3) you will modify. By default the **MainButtons** collection is selected.

In the middle pane you see a list with the already added buttons and a name corresponding to the type of the button - the **SocialNetType** property for the **Styled Buttons** (Section 36.3) and the name for the Standard Buttons.

You can add a Styled Button by pressing the first button; the next three are, respectively, the **Standard Buttons** (Section 36.3) for Facebook, Twitter and GooglePlusOne. The fifth button adds the RadCompactButton and the sixth removes the selected RadSocialButton.

You can choose which network the button connects to by directly typing the Standard Buttons's name (or **SocialNetType** property for the Styled Button) in the list, or you can select this from the dropdown in the right pane where you can choose all other options.

If you type in a name that does not exist as a possible value for these properties the input will not be taken and the button will be reset to its previous state. Note that the names are case-sensitive. By default the **GoogleBookmarks** Styled Button is added as it is the first one in the alphabetical order.

If the button type is changed via the properties pane this change is automatically reflected in its name in the list and vice versa.

You can reorder the buttons in the collection by using the two arrows on the right of the list - each click moves the selected button one position up or down the list.

All other properties can be controlled via the right pane, which is the standard Properties pane of the Visual Studio. By default only the **SocialNetType** and the **ToolTip** are set for each Styled Button and are rendered in the markup. For the Facebook Standard Button only the **ButtonType** property is selected by default and the Twitter and GooglePlusOne buttons do not need any additional properties initially. You can leave this as-is, or modify the properties as needed.

When working with the **CompactButtons** collection you can only choose from the Styled Buttons, as they are the only ones that are acceptable for it. Therefore, if a name for a Standard Button is entered it will not be taken by the Configurator.

36.6 First Steps

The following tutorial demonstrates how a simple **RadSocialShare** control can be used to share an URL.

1. In a new AJAX-Enabled Web Application add a **RadSocialShare** control to the default web page (either by dragging it from the Toolbox, or by simply typing in the markup):

```
<telerik:RadSocialShare runat="server" ID="RadSocialShare1"></telerik:RadSocialShare>
```

2. Add a ScriptManager to the beginning of the page. You can do this via the control's SmartTag as well
3. Set the **UrlToShare** property to <http://www.telerik.com/products/aspnet-ajax.aspx> .
4. Set the **TitleToShare** property to ASP.NET AJAX Controls, .NET Web UI Components | Telerik
5. Add some buttons to the **MainButtons** collection. You can choose the social network which they target via the **SocialNetType** property. For example add a Facebook share button, a Twitter tweet button and a Blogger button. The last button is the **RadCompactButton** so that you can pop up a **RadWindow** with the rest of the networks:

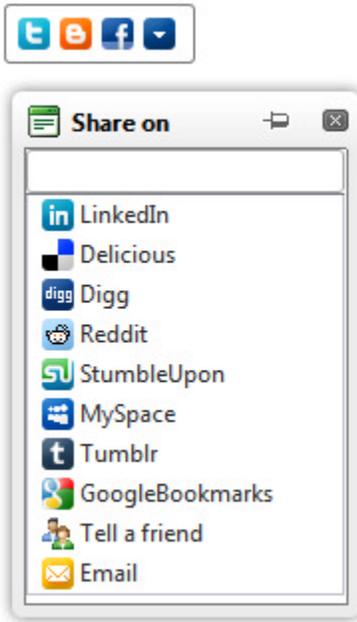
RadSocialShare declaration

```
<telerik:RadSocialShare runat="server" ID="RadSocialShare1"
UrlToShare="http://www.telerik.com/products/aspnet-ajax.aspx"
TitleToShare="ASP.NET AJAX Controls, .NET Web UI Components | Telerik">
  <MainButtons>
    <telerik:RadSocialButton SocialNetType="ShareOnTwitter" />
    <telerik:RadSocialButton SocialNetType="Blogger" />
    <telerik:RadSocialButton SocialNetType="ShareOnFacebook" />
    <telerik:RadCompactButton />
  </MainButtons>
</telerik:RadSocialShare>
```

You can do this via the Configurator from the Smart Tag as well.

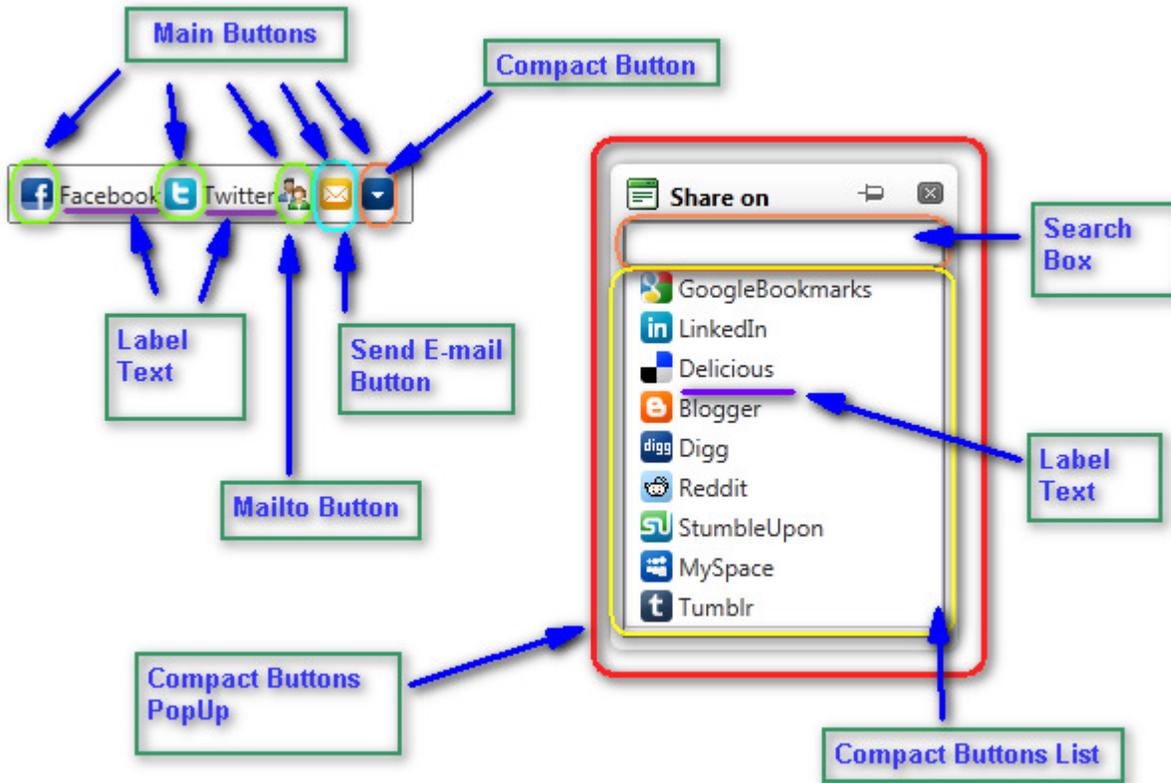
6. Press F5 to run the application. You will see a simple bar in the top left corner of the browser. If you click the last button the popup with the rest shows up.

The end result will be similar to the following image once the CompactButtons are shown:

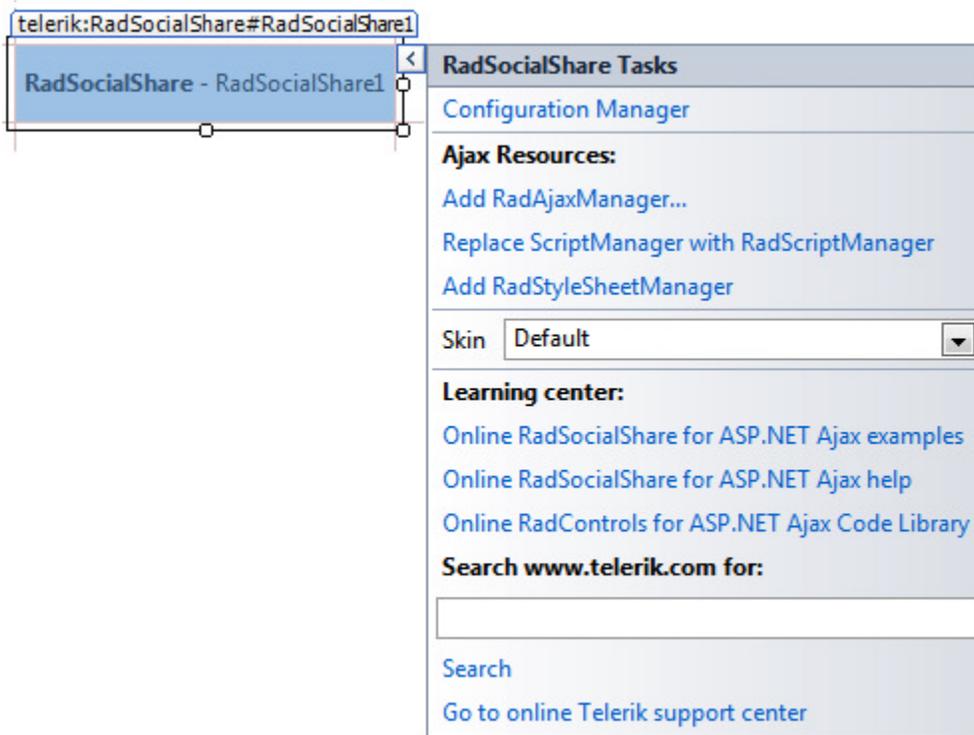


The main visual elements of the RadSocialShare control are:

- **MainButtons** - these are the buttons that are always visible on the page
- **CompactButton** - this is an extra button that is not used for sharing, but to show the other available buttons
- **CompactButtons** list - these are the buttons that are not initially visible on the page, yet are easily reachable in a movable popup
- **SearchBox** - you can start typing a social network's name and the CompactButtons will be filtered accordingly
- **CompactButtons** popup - the RadWindow that holds the additional buttons
- **Send E-mail button** - a button that pops up a form in a RadWindow that allows you to send an e-mail via a dedicated server
- **Mailto button** - a button that triggers the system's default mail client to send an e-mail via the user's machine
- **Label Text** - the text associated with the button. It can be set explicitly via a property. The CompactButtons have a predefined value which is used for the Search Box even if you do not set the label explicitly.



The Smart Tag of **RadSocialShare** lets you easily select the available buttons, change the skin for your control or quickly get help. You can display the Smart Tag by right clicking on a **RadSocialShare** control and choosing "Show Smart Tag", or by clicking the small rightward-pointing arrow located in the upper right corner of the control.



36.7 Controlling the URL and the Title

RadSocialShare allows you to preselect the URL and the title/mail subject of the user's post. By default the **UrlToShare** property takes the URL from the current page where the control resides, yet you can easily change this if you wish the post to link to another article, site or section. Simply set the **UrlToShare** property to the desired URL - full (i.e. starting with `http://` and followed by the full address), or a relative server path (e.g. `~/Shared/PageToShare.aspx`). You can also preselect the title of the user's post for sites that support such a feature and for the e-mail by setting the **TitleToShare** property to the desired string. Here follows the logic by which these properties are determined:

1. **UrlToShare** is not set - the current page URL and title are used. If the page has no title the URL becomes the title
2. **UrlToShare** is set:
 1. **TitleToShare** is not set - the **UrlToShare** property is used as both URL and Title
 2. **TitleToShare** is set: - the **UrlToShare** property is used as the URL and the **TitleToShare** is used as the title

With the following simple markup you can have the RadSocial share point to an explicit site and predefine the title (which can, of course, be changed by the end user):

Setting **UrlToShare** and **TitleToShare** properties

```
<telerik:RadSocialShare ID="RadSocialShare1" runat="server"
    UrlToShare="http://www.telerik.com/products/aspnet-ajax.aspx"
    TitleToShare="Check out the awesome UI components for ASP.NET AJAX from Telerik">
    <MainButtons>
        <telerik:RadSocialButton SocialNetType="Blogger" LabelText="Blogger">

```

```

ToolTip="Share on Blogger" />
    <telerik:RadSocialButton SocialNetType="GoogleBookmarks" LabelText="Google
Bookmarks"
        ToolTip="Add a Google Bookmark" />
    <telerik:RadSocialButton SocialNetType="MailTo" ToolTip="Tell a friend" />
</MainButtons>
</telerik:RadSocialShare>

```

These properties can also be set for each individual button and then they will override the global setting. This allows you to tailor the posts for a specific network as desired.

36.8 Using Third Party Buttons

The **RadSocialShare** control also offers integration with the scripts offered by different social networks. Instead of using their public API, which is sometimes limited, you can utilize the full functionality their scripts provide. Currently there are three networks that offer such functionality - Facebook, Twitter and GooglePlusOne. This may change in the future, however, and it is out of our control.

The buttons generated via these external scripts are called **Standard Buttons** and can only be present in the **Main Buttons** collection, since their rendering is not done by the **RadSocialShare**.

Some of the functionalities these buttons provide are counters (for all three networks), information popups (for the Facebook and GooglePlusOne buttons), and extended options to control the way content is posted (e.g. Facebook offers share, recommend or like posts). Also, instead of a browser popup, these features are often wrapped in a small tooltip-like popup.

The bonus **RadSocialShare** offers is that this functionality is wrapped in separate classes, so you can activate different features by simply choosing the properties we expose from their API. This can be done with a single line of code or even easier - via the visual Designer.

Here follows a list with some of the most important features and properties that are available:

1. **Facebook's buttons** - they are activated by adding a `<telerik:RadFacebookButton />` tag. The options they provide are:
 1. **ButtonType** - determines the exact functionality the button will offer:
 1. *FacebookShare* - creates a Share button
 2. *FacebookRecommend* - creates a Recommend button
 3. *FacebookLike* - creates a Like button
 4. *FacebookSend* - creates a Send button. Note that if both a *FacebookLike* and *FacebookSend* buttons are present Facebook automatically combines them in a new, bigger button even if they are not adjacent.
 2. **ButtonLayout** - determines the layout of the button (whether and where counters are shown, whether text and and counters are available, etc.)
 3. **ShowFaces** - specifies whether to display profile photos below the button (standard layout only). True by default.
2. **Google Plus One Button** - To add this button you need the the `<telerik:RadGoogleButton />` tag. Its options are:
 1. **ButtonSize** - the size of the button:
 1. *Small*
 2. *Medium*

3. *Standard*
 4. *Tall*
2. **AnnotationType** - the annotation to display next to the button:
1. *None* - no additional information
 2. *Bubble* - displays only the number of people who have shared this
 3. *Inline* - displays profile pictures and the count of the people who have +1-ed this
3. **Twitter Button** - to show this button add the `<telerik:RadTwitterButton />` tag. Its only option is the
1. **CounterMode**
 1. *Horizontal*
 2. *Vertical*
 3. *None*

Here follows some simple markup that will allow you to see this functionality in action:

ASPX

```
<telerik:RadSocialShare ID="RadSocialShare1" runat="server"
UrlToShare="http://www.telerik.com (http://www.telerik.com/)"
TitleToShare=".NET UI Controls, Reporting, Visual Studio Tools, Agile Project
Management, Automated Testing, ASP.NET Web CMS by Telerik"
Width="300">
  <MainButtons>
    <telerik:RadFacebookButton ButtonType="FacebookLike" ButtonLayout="Standard"
Width="300"
      ShowFaces="true" />
    <telerik:RadFacebookButton ButtonType="FacebookSend" />
    <telerik:RadGoogleButton AnnotationType="Bubble" ButtonSize="Medium" />
    <telerik:RadTwitterButton CounterMode="Horizontal" />
    <telerik:RadFacebookButton ButtonType="FacebookShare"
ButtonLayout="ButtonCount" />
  </MainButtons>
</telerik:RadSocialShare>
```

37 RadTreeList

37.1 Objectives

- Explore the features of the RadTreeList control.
- Explore the RadTreeList design time interface, including Smart Tag and Properties View.
- Create simple application for binding data using the RadTreeList
- Become familiar with the most common features of RadTreeList

37.2 Introduction

Telerik RadTreeList is a hybrid control combining treeview and grid in one. It gives you the opportunity for hierarchical representation of the underlying data, like in a treeview. In addition, it can have multiple columns and provides you with the ability to perform advanced operations like paging, selecting items, etc.

The key features of the RadTreeList control are:

- Various column types
- Codeless data-binding using the DataSourceControls in ASP.NET 2.0/3.5
- Data-Binding to various data sources which implement the IEnumerable, IList or ICustomTypeDescriptor interfaces
- Integrated paging
- Integrated sorting
- Easily customizable skinning mechanism (setting single Skin property of the treelist)
- The ShowOuterBorders, ShowTreeLines and GridLines properties allow you to quickly change the appearance
- Interoperability with RadAjax and loading indicators - dramatically improves the responsiveness of the component, simulates Windows-application like behavior, and minimizes the traffic to the server
- Single and Multi-Row Server-Side and Client-Side Selection
- The selected and the expanded state of the items is persisted while navigating through pages.

37.3 Getting-Started

Here we will describe the main features of the RadTreeList and the properties/methods you should know to enable them.

Paging

RadTreeList supports paging functionality which allows the users to view the data, separated in chunks. To enable this functionality in RadTreeList, you should set the AllowPaging property to true.

The following methods and properties are exposed in the **RadTreeList's** server-side Pager API:

PageSize	Determines the maximum items displayed on a single page
PagerStyle-FirstPageToolTip	The text that is displayed when hovering the FirstPage button
PagerStyle-NextPageToolTip	The text that is displayed when hovering the NextPage button
PagerStyle-PrevPageToolTip	The text that is displayed when hovering the PrevPage button
PagerStyle-LastPageToolTip	The text that is displayed when hovering the LastPage button
PageButtonCount	The number of numeric buttons in the pager
	Determines the position of the Pager in RadTreeList

UI for ASP.NET AJAX

- Position
- Top
 - Bottom
 - TopAndBottom
- Mode
- NextPrev
 - NumericPages
 - NextPrevAndNumeric
 - NextPrevNumericAndAdvanced
 - Advanced
 - Slider

This property sets the appearance of the Pager. The available modes are:

Sorting

RadTreeList offers sorting capabilities that allows the users to conveniently order the items in the desired direction. To enable this functionality you just have to set AllowSorting property to true and the control will handle the sorting operations automatically.

There are three sort modes:

- **Ascending** - orders the items in ascending order
- **Descending** - orders the items in descending order
- **None** - the items are ordered in the way they came from the datasource ("Natural" sort)

RadTreeList also supports sorting by multiple datafields - this is the so-called **Multi-column sorting**. To enable this mode, set the **AllowMultiColumnSorting** to true.

Due to the self-referencing nature of the control, the sorting takes effect "per-level". Basically, this means that each level of the hierarchical structure is sorted independently.

LocationID	Name	Population
2	Africa	922,011,000
5	Antarctica	1,000
1	Asia	3,879,000,000
8	China	1,340,480,000
10	India	1,189,870,000
9	Japan	127,380,000
13	Thailand	67,070,000
7	Australia	22,000,000
6	Europe	731,000,000
3	North America	528,720,588

Navigation controls: [Home] [Previous] [1] [2] [Next] [End] Page size: 10

Sorting API:

RadTreeList exposes the following properties and methods:

AllowMultiColumnSorting Determines whether the multi-column sorting functionality is enabled.

AllowNaturalSort Enables or disables the "natural" sort mode where the items are ordered in the way they come from the datasource.

AllowSort	Enables the sorting functionality in RadTreeList.
SortExpressions	SortExpressions collection. Contains the expressions that are applied to the control.

Selecting

Telerik **RadTreeList** has built-in mechanism for items selection. You can select items either on the client or on the server as per your requirements.

1. Client-Side Selection:

To enable the RadTreeList client-side selection you need to set the **ClientSettings.Selecting.AllowItemSelection** to true. This will allow you to select an item on mouse click. As a result the **OnItemClick**, **OnItemSelecting** and **OnItemSelected** client-side events of the RadTreeList will be fired so you can perform further actions and handle the item selection in a custom manner.

You can also use the below settings to enable additional modes of the client-side selection:

- **ClientSettings.Selecting.AllowToggleSelection** - When set to **true** (the default value is false) enables you to deselect an item by clicking on one that is already selected.
- **ClientSettings.Selecting.UseSelectColumnOnly** - When set to **true** (the default value is false) prevents users from selecting items on mouse click and forces them to use the **TreeListSelectColumn** for that purpose.

With RadTreeList you might want to provide the ability for multi-item selection. This is done by setting its **AllowMultiItemSelection** property to true (its default value is false). And to select a few items at a time, one can use the **[Ctrl]** and **[Shift]** keys as in Windows Explorer. Or, in case the AllowToggleSelection property is true, just click on the desired items to select them.

 Note that when client-side selection is enabled through the AllowItemSelection property, the TreeListSelectColumn selects the items on the client. If the AllowItemSelection property is false, server-side selection is performed.

2. Server-Side Selection:

There might be scenarios where you need to perform server-side selection for the RadTreeList items. For that purpose, you can use one of the below approaches:

- Add a **TreeListSelectColumn** and provide the ability to the user to select the desired items through it
- Use server-side code to programmatically select the treelist items

In both cases, to enable multi-item selection, you need to set the RadTreeList **AllowMultiItemSelection** property to **true**.

2.1. Using the TreeListSelectColumn:

RadTreeList server-side selection is enabled for the users once you add the **TreeListSelectColumn** to the RadTreeList Columns collection. You do not need to set any additional properties. Then checking the checkbox rendered in the column marks the corresponding item as selected. As a result, postback is performed and the **ItemCommand** event is fired with command name **RadTreeList.SelectCommandName**. To deselect an item, one should uncheck the checkbox in the select column. Then again postback is performed and the **ItemCommand** event is fired with command name **RadTreeList.DeselectCommandName**.

ASPX

```
<telerik:RadTreeList ID="RadTreeList1" runat="server" DataKeyNames="EmployeeID"
  DataSourceID="SqlDataSource1" ParentDataKeyNames="ReportsTo"
  AllowMultiItemSelection="True"
  OnItemCommand="RadTreeList1_ItemCommand">
  <Columns>
    <telerik:TreeListSelectColumn UniqueName="SelectColumn">
```

```
        </telerik:TreeListSelectColumn>
    </Columns>
</telerik:RadTreeList>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<$ ConnectionStrings>"
    SelectCommand="SELECT [EmployeeID], [LastName], [FirstName], [Title], [TitleOfCourtesy],
[City], [ReportsTo] FROM [Employees]">
</asp:SqlDataSource>
```

C#

```
protected void RadTreeList1_ItemCommand(object sender,
Telerik.Web.UI.TreeListCommandEventArgs e)
{
    if (e.CommandName == RadTreeList.SelectCommandName)
    {
        //item is being selected
    }
    if (e.CommandName == RadTreeList.DeselectCommandName)
    {
        //item is being deselected
    }
}
```

VB

```
Protected Sub RadTreeList1_ItemCommand(sender As Object, e As
Telerik.Web.UI.TreeListCommandEventArgs)
    If e.CommandName = RadTreeList.SelectCommandName Then
        'item is being selected
    End If
    If e.CommandName = RadTreeList.DeselectCommandName Then
        'item is being deselected
    End If
End Sub
```

2.2. Programmatic items selection:

To select an item/items with server-side code, you can set the particular item/items **Selected** property to **true**:

C#

```
protected void Page_PreRender(object sender, EventArgs e)
{
    TreeListDataItem item = RadTreeList1.Items[0];
    item.Selected = true;
}
```

VB

```
Protected Sub Page_PreRender(sender As Object, e As EventArgs) Handles Me.PreRender
    Dim item As TreeListDataItem = RadTreeList1.Items(0)
    item.Selected = True
End Sub
```

2.3. Deselecting items programmatically

To deselect an item/items programmatically, set its **Selected** property to **false**.

You can also call the **ClearSelectedItems()** method of the RadTreeList control to deselect all selected items which are visible on the current page.

3. Recursive Selection

As part of its server-side selection RadTreeList gives you the ability to select items recursively. To enable the RadTreeList recursive selection, set the **AllowRecursiveSelection** property to **true**.

✎ When **AllowRecursiveSelection** is set to **true**, this implicitly enables multi-item selection for the RadTreeList.

To select an item/items, you can either use the **TreeListSelectColumn** or set the **Selected** property of the item/items to **true**.

When recursive selection is enabled for the RadTreeList and you select an item, all its nested items are selected, no matter on which nested level they are on (visible or not). Also, if selecting an item makes all items on the same level selected, their parent item will be marked as selected as well. The opposite is true as well; deselecting item from a nested level will invoke deselecting of its parent item in case it is previously selected.

However, for the recursive selection to work and all items' state to be updated properly, a postback is required. Thus if client-side selection is enabled, it is automatically turned off. Also when items are selected/deselected, an implicit rebind is invoked for the RadTreeList control.

To deselect all items, you can call the **ClearSelectedItems()** method of the RadTreeList. As a result all selected items will be deselected, be they on the current page or not, visible or not.

✎ Note that, if you want to traverse the RadTreeList items in a foreach loop and change their selected state in the loop while recursive selection is enabled, you need to rebind the RadTreeList first.

- **Exporting**

Since Q3 2011 the Telerik RadTreeList can export your data to PDF after a call to its **ExportToPdf()** method. Export to Excel functionality is added since Q1 2012. The corresponding method is **ExportToExcel()**.

- **Common properties:**

In addition to the export format's specific properties, the **ExportSettings** group exposes several common properties:

ExportOnlyData - this is an enumeration with four possible values described below:

- **DefaultContent** - the whole data and content of the RadTreeList are sent for export, without removing or replacing anything;
- **RemoveControls** - removes all controls that implement the *IButton*, *ITextBox*, *ICheckBox* and *IScriptControl* interfaces;
- **ReplaceControls** - tries to replace all controls that implement the *IButton*, *ITextBox*, *ICheckBox* and *IScriptControl* interfaces with their text;
- **RemoveAll** - removes all non-text controls.

IgnorePaging - when you enable it, the RadTreeList will rebind before export in order to fetch all the data from your datasource.

OpenInNewWindow - by default, the exported file will be handled by the program associated with the appropriate file type. If you prefer to give the user the option to choose whether to save, open (inline) or cancel, you can enable this property.

FileName - This is helpful when you want to give a predefined name for your file. Please note that the file name cannot be longer than 256 symbols. **Unicode** names are not supported out-of-the-box for *Internet Explorer 6* and *7*. Of course you can manually encode the file name and it will be shown

properly in the "Save" dialog (*OpenInNewWindow="true"*):
HttpUtility.UrlEncode("unicode string", System.Text.Encoding.UTF8);

- Pdf Export

- Basics

Name	Description
AllowAdd / AllowCopy / AllowModify / AllowPrinting	Boolean properties that determines whether the corresponding action is allowed for the generated <i>PDF</i> file
Author / Creator / Keywords / Producer / Subject / Title	<i>PDF</i> document specific information
DefaultFontFamily	Specifies the default font
PageTitle	Sets the page title (appears on the top of the page)
PaperSize / PageWidth / PageHeight	These properties are related to the size of the generated page. You can either use the <i>PaperSize</i> to supply a predefined value (<i>A4</i> , <i>Letter</i> , <i>JIS B5</i> , etc) or define the size manually using <i>PageWidth/PageHeight</i> . Please note that the values set to the <i>PageWidth/PageHeight</i> properties have a higher priority than the <i>PaperSize</i>
PageBottomMargin / PageTopMargin / PageLeftMargin / PageRightMargin / PageFooterMargin / PageHeaderMargin	All the page margins could be controlled via these settings
RotatePaper	You can switch the orientation of the page through this property
UserPassword	Used to set a password and enable password protection for the exported file

- ASPX

```
<ExportSettings>  
  <Pdf PageTitle="My Page" PaperSize="A4" RotatePaper="true" />  
</ExportSettings>
```

- Exporting HTML tables

There are a few rules that should be followed when exporting HTML tables to PDF:

- The table should define <colgroup> and <col> elements
- The number of col elements should be equal to the number of the columns in the table body
- Width in pixels (or another absolute units) should be set to the table

XML

```
<table width="300px">
<colgroup>
    <col />
    <col />
</colgroup>
<tr>
    <td>
    Cell1
    </td>
    <td>
    Cell2
    </td>
</tr>
</table>
```

■ Appearance and styling

RadTreeList does not export any external styles. This means that your skins will not appear in the generated file. However, the control offers the following options for customizing the appearance in the exported file:

- Styles set in the code-behind;
- Styles set in the Pdf category of the ExportSettings;
- Styles set to the TreeList.

The priority follows the above order.

■ Setting styles in the code-behind

Different approaches for setting styles in the code-behind could be used depending whether the RadTreeList will be rebound before export (when IgnorePaging is set to true or when you rebound manually). The following code can be used in both cases:

C#

```
bool isExport = false;
protected void Button1_Click(object sender, EventArgs e)
{
    isExport = true;
    RadTreeList1.ExportToPdf();
}

protected void RadTreeList1_ItemCreated(object sender, TreeListItemCreatedEventArgs e)
{
    if (e.Item is TreeListDataItem && isExport)
        e.Item.Style["background-color"] = "#888888";
}
```

Example Title

```

Private isExport As Boolean = False
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
    isExport = True
    RadTreeList1.ExportToPdf()
End Sub

Protected Sub RadTreeList1_ItemCreated(ByVal sender As Object, ByVal e As
TreeListItemCreatedEventArgs)
    If TypeOf e.Item Is TreeListItem AndAlso isExport Then
        e.Item.Style("background-color") = "#888888"
    End If
End Sub

```

■ Setting styles in the Pdf category

The TreeList offers the following built-in style descriptors for export to PDF:

- ItemStyle
- AlternatingItemStyle
- HeaderStyle
- ExpandCollapseCellStyle

The first three inherit the `TableItemStyle` (<http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.tableitemstyle.aspx>) and style the corresponding treelist rows. The `ExpandCollapseCellStyle` styles only the Expand/Collapse cells in the control. You should set either the expand/collapse text, or the expand/collapse images (not both). The path for the expand/collapse images should be relative, otherwise they will not be found and exported on the production server.

ASPX

```

<ExportSettings>
  <Pdf>
    <ItemStyle BackColor="Green" ForeColor="DarkGreen" />
    <AlternatingItemStyle BackColor="WhiteSmoke" ForeColor="Black" />
    <HeaderStyle Font-Size="Large" />
    <ExpandCollapseCellStyle ExpandText="(+)" CollapseText="(-)" />
  </Pdf>
</ExportSettings>

```

■ Setting styles declaratively to the TreeList

ASPX

```

<telerik:RadTreeList runat="server" ID="RadTreeList1" DataSourceID="SqlDataSou
AllowPaging="true" PageSize="5" DataKeyNames="id" ParentDataKeyNames="{
AutoGenerateColumns="false">
  <HeaderStyle ForeColor="BlueViolet" />
  <AlternatingItemStyle Font-Size="Small" />
  <Columns>
    <telerik:TreeListBoundColumn DataField="id" UniqueName="id" Hea
ReadOnly="true" />
    <telerik:TreeListBoundColumn DataField="Text" UniqueName="Text"
HeaderText="Name">
      <ItemStyle Font-Italic="true" />
      <HeaderStyle BackColor="BlanchedAlmond" />
    </telerik:TreeListBoundColumn>
    <telerik:TreeListBoundColumn DataField="parentid" UniqueName="{
HeaderText="Parent ID"

```

```

        ReadOnly="true" />
    </Columns>
</telerik:RadTreeList>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%"$
ConnectionString:TelerikConnectionString %">
    SelectCommand="SELECT * FROM [Links]"></asp:SqlDataSource>

```

- Excel Export

- Appearance and styling

RadTreeList does not export any external styles. This means that your skins will not appear in the generated file. However, the control offers the following options for customizing the appearance in the exported file:

- Styles set in the Excel category of the ExportSettings;
- Styles set to the TreeList.

The priority follows the above order.

- Setting styles in the Excel category

The TreeList offers the following built-in style descriptors for export to Excel:

- ItemStyle
- AlternatingItemStyle
- HeaderStyle
- ExpandCollapseCellStyle

The first three inherit the `TableItemStyle` (<http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.tableitemstyle.aspx>) and style the corresponding treelist rows. The `ExpandCollapseCellStyle` styles only the Expand/Collapse cells in the control. You should set either the expand/collapse text, or the expand/collapse images (not both). The path for the expand/collapse images should be relative, otherwise they will not be found and exported on the production server.

ASPX

```

<ExportSettings>
  <Excel>
    <ItemStyle BackColor="Green" ForeColor="DarkGreen" />
    <AlternatingItemStyle BackColor="WhiteSmoke" ForeColor="Black" />
    <HeaderStyle Font-Size="Large" />
    <ExpandCollapseCellStyle ExpandText="(+)" CollapseText="(-)" />
  </Excel>
</ExportSettings>

```

- Setting styles

Another option to export styled treelist is to specify the styles declaratively to the control:

ASPX

```

<telerik:RadTreeList runat="server" ID="RadTreeList1" DataSourceID="SqlDataSource1"
  AllowPaging="true" PageSize="5" DataKeyNames="id" ParentDataKeyNames="parentId"
  AutoGenerateColumns="false">

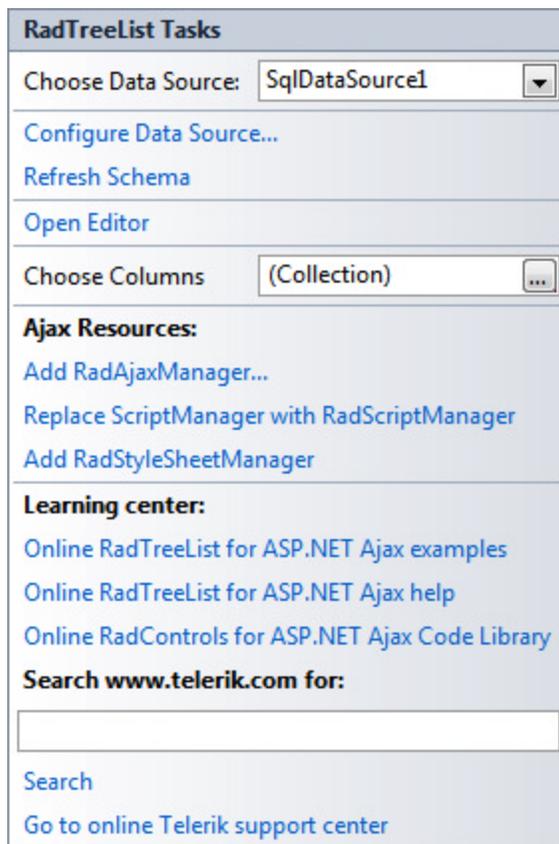
```

```
<HeaderStyle ForeColor="BlueViolet" />
<AlternatingItemStyle Font-Size="Small" />
<Columns>
  <telerik:TreeListBoundColumn DataField="id" UniqueName="id" HeaderText="ID"
  ReadOnly="true" />
  <telerik:TreeListBoundColumn DataField="Text" UniqueName="Text" HeaderText="Name";
  <ItemStyle Font-Italic="true" />
  <HeaderStyle BackColor="BlanchedAlmond" />
</telerik:TreeListBoundColumn>
  <telerik:TreeListBoundColumn DataField="parentid" UniqueName="parentid"
  HeaderText="Parent ID"
  ReadOnly="true" />
</Columns>
</telerik:RadTreeList>
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%=
ConnectionString:TelerikConnectionString %>"
  SelectCommand="SELECT * FROM [Links]"></asp:SqlDataSource>
```

37.4 Using the design-time interface

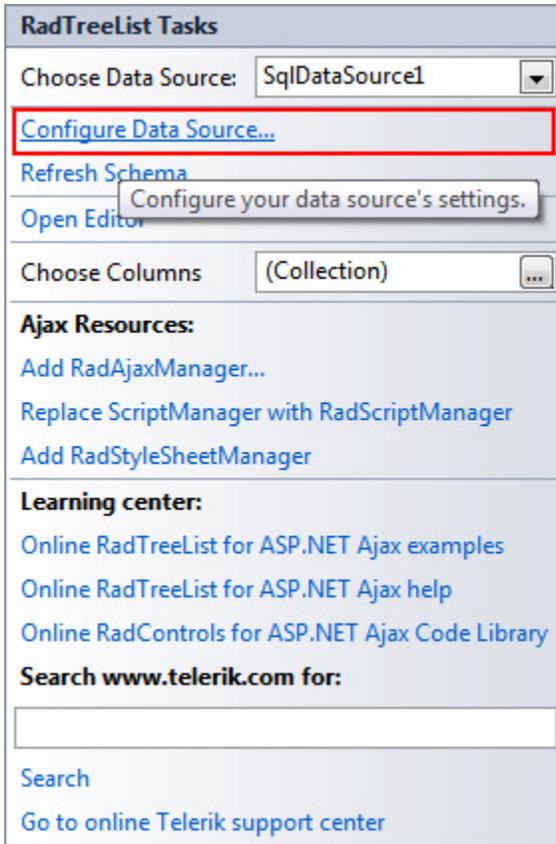
Smart Tag

The RadTreeList Smart Tag provides convenient access to the most common settings for the control. You can display the Smart Tag by right clicking on the RadTreeList in the design window, and choosing the "Show Smart Tag" option from its context menu.



Configure Data Source

You can easily configure declarative data source by choosing Configure Data Source link from the Smart Tag of the RadTreeList control:



Open Editor

Open Editor link displays RadTreeList wizard with Functionality and Appearance sections which lets you customize/configure the RadTreeList control.

RadTreeList Tasks

Choose Data Source:

[Configure Data Source...](#)

[Refresh Schema](#)

[Open Editor](#)

Choose Columns

Ajax Resources:

[Add RadAjaxManager...](#)

[Replace ScriptManager with RadScriptManager](#)

[Add RadStyleSheetManager](#)

Learning center:

[Online RadTreeList for ASP.NET Ajax examples](#)

[Online RadTreeList for ASP.NET Ajax help](#)

[Online RadControls for ASP.NET Ajax Code Library](#)

Search [www.telerik.com](#) for:

[Search](#)

[Go to online Telerik support center](#)

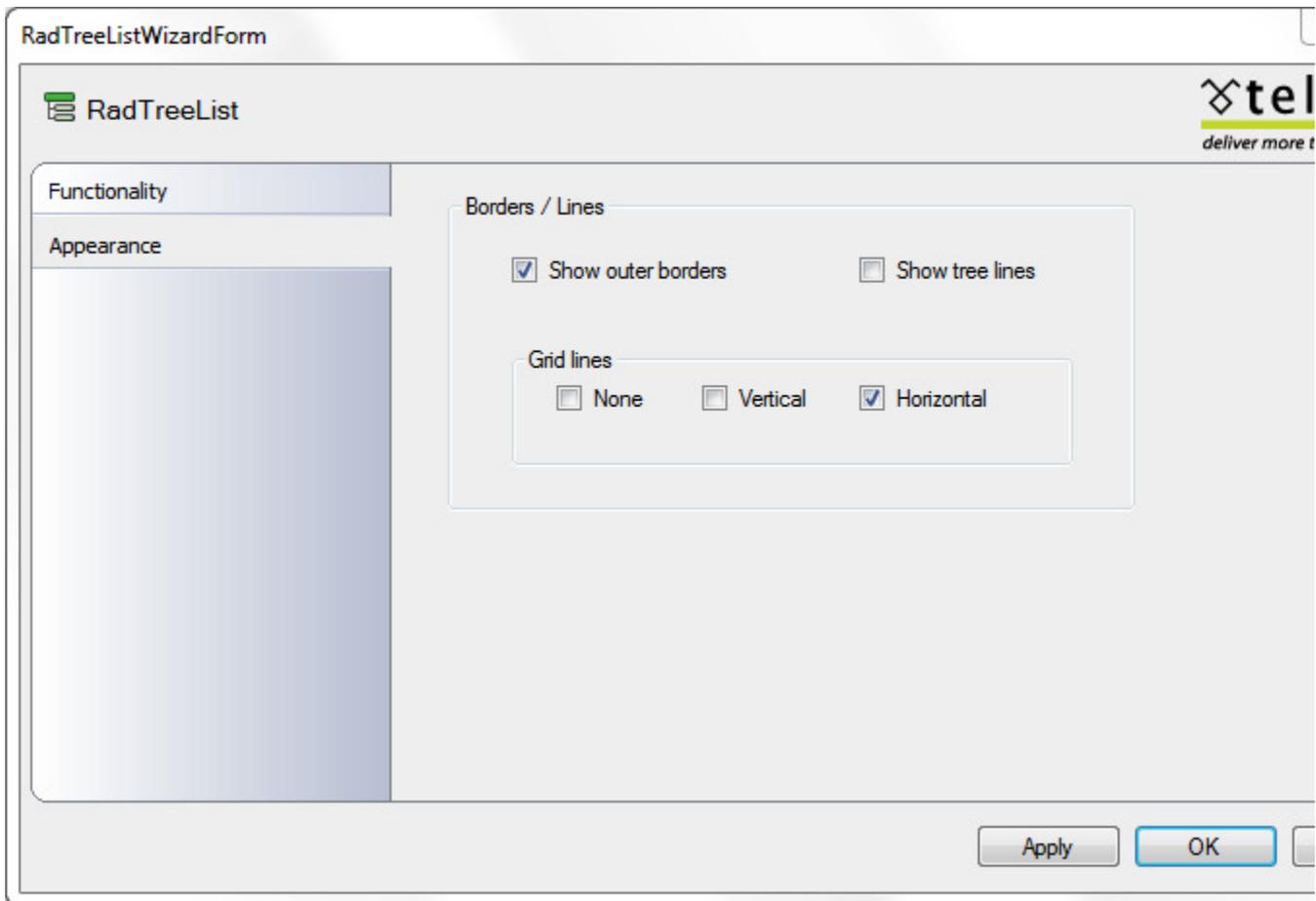
Functionality

The Functionality section allows you to Enable paging and specify the PagerMode and PageSize properties of the treelist. Also you have the ability to choose the server-side selection type.

The screenshot shows the 'RadTreeListWizardForm' window. On the left, there is a sidebar with two tabs: 'Functionality' and 'Appearance'. The 'Appearance' tab is selected. The main area of the form is divided into two sections: 'Paging' and 'Selection'. In the 'Paging' section, there is a checked checkbox for 'Enable paging', a dropdown menu for 'Pager mode' set to 'NextPrevAndNumeric', and a spinner control for 'Page size' set to '10'. In the 'Selection' section, there are three radio buttons: 'None', 'Single item' (which is selected), and 'Multiple items'. At the bottom right of the form, there are 'Apply' and 'OK' buttons.

Appearance

In this section you can set the appearance options for the RadTreeList.



Choose Columns

The Choose Columns option allows you to set the way columns are generated and visualized.

RadTreeList Tasks

Choose Data Source:

[Configure Data Source...](#)

[Refresh Schema](#)

[Open Editor](#)

Choose Columns

Ajax Resources:

[Add RadAjaxManager...](#)

[Replace ScriptManager with RadScriptManager](#)

[Add RadStyleSheetManager](#)

Learning center:

[Online RadTreeList for ASP.NET Ajax examples](#)

[Online RadTreeList for ASP.NET Ajax help](#)

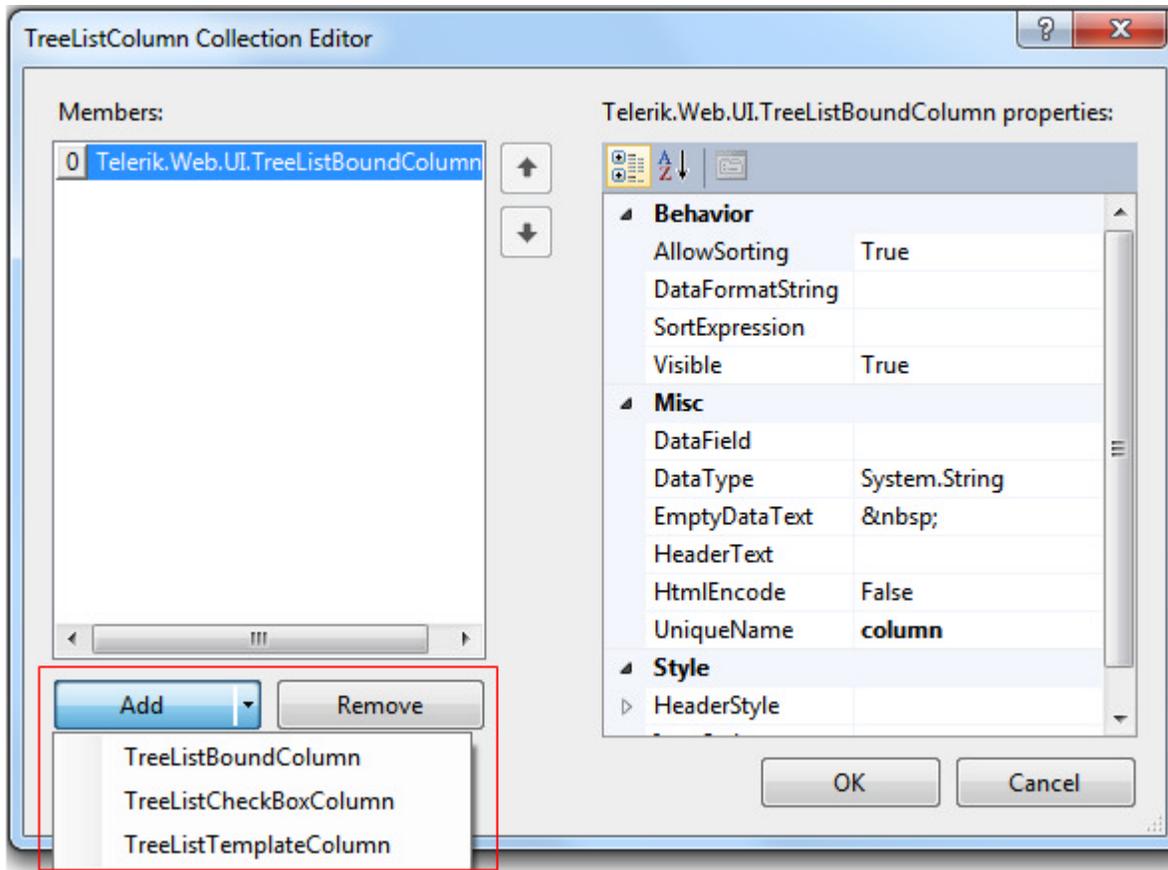
[Online RadControls for ASP.NET Ajax Code Library](#)

Search [www.telerik.com](#) for:

[Search](#)

[Go to online Telerik support center](#)

RadTreeListColumn Collection Editor will let you display only specific data fields from a given database. Moreover, you can define custom properties for the columns that present these fields. You can use the Add button to create different columns in the treelist. The columns will appear in the "Selected Columns" list. You can use the Up and Down buttons to re-order the columns and the **Remove** button to remove a column from this list. From the "Members" list choose the columns, which you want to bind (display).



Ajax Resources

- Add RadAjaxManager... - adds a RadAjaxManager component to your Web page, and displays the RadAjaxManager Property Builder where you can configure it.
- Replace ScriptManager with RadScriptManager - replaces the default ScriptManager component that is added for AJAX-enabled Web sites with RadScriptManager.
- Add RadStyleSheetManager - adds a RadStyleSheetManager to your Web page.

Learning Center

Links navigate you directly to RadTreeList examples, help, or code library. You can also search the Telerik web site for a given string.

37.5 Data Editing

One of the main features of RadTreeList are data editing operations, auto-generated and custom edit forms support, as well as different edit modes. The RadTreeList control supports the below edit modes:

- InPlace - you need to set the EditMode property of your RadTreeList control to InPlace.
- EditForms - you need to set the EditMode property to EditForms.
- PopUp -you need to set the EditMode property to PopUp.

The default **EditMode** of the treelist is **EditForms**. To specify which edit mode will your control, you can set its EditMode property to one of the above values.

The edit form types of the RadTreeList are:

- AutoGenerated
- Template
- WebUserControl

The default **EditFormType** of the treelist is **AutoGenerated**. To specify which edit form type will your control, you can set its **EditFormSettings-EditFormType** property to one of the above values. When the **EditFormType** is set to **AutoGenerated** (the default value), **RadTreeList** will generate the edit form for you.

Automatic data operations

RadTreeList provides an API for inserting new data, updating existing data and deleting data from the specified data source. You can use these features while writing very little code. The only requirement is binding the treelist to a declarative data source using the **DataSourceID** property of the control.

You also need to set the **DataKeyNames** and **ParentDataKeyNames** properties of the **RadTreeList** control so that the insert, update, and delete operations perform as expected. A live example is available in **this online demo** (<http://demos.telerik.com/aspnet-ajax/treelist/examples/dataediting/net2automaticdataediting/defaultcs.aspx>).

Manual data editing

This demo (<http://demos.telerik.com/aspnet-ajax/treelist/examples/dataediting/manualdataediting/defaultcs.aspx>) shows how to manually update/insert items to the database. The new values are extracted from the current item using the **ExtractValues** method. Note that, when inserting a child item to a parent data item, the foreign key values (the **ReportsTo** field in our case, specified by the **ParentDataKeyNames** array in **RadTreeList**) are extracted into the **Hashtable** with **ExtractValues**. We need to check if the foreign key is present in the **Hashtable** and add it to the insert parameters. When inserting a root item, however, foreign keys are not populated and thus, **DBNull** should be explicitly added as a foreign key.

Custom editors

RadTreeList provides a straightforward way to specify a non-default editor for an editable column. The **RadTreeList.CreateColumnEditor** event fires whenever a column editor needs to be initialized. The event argument object of type **TreeListCreateColumnEventArgs** provides the following properties:

- **Column** - the **TreeListEditableColumn** instance for which a column editor will be initialized.
- **DefaultEditor** - the default **ITreeListColumnEditor** instance that the column provides.
- **CustomEditorInitializer** - a delegate that does not accept parameters and returns an instance of type **ITreeListColumnEditor**.

You should provide a delegate function to **e.CustomEditorInitializer** that instantiates and returns an **ITreeListColumnEditor** object. A column editor instance usually accepts the target editable column in its constructor. With the attached sample code, we provide a custom column editor for the "Notes" column in the **RadTreeList**:



You can find the complete source for this project at:
 \VS Projects\TreeList\RadTreeListCustomEditors

37.6 Appearance and Styling

The predefined layouts of **RadTreeList**, which in turn have a predefined HTML rendering, enable you to use the

control's embedded skins to achieve a consistent look of RadTreeList with the other RadControls on the page. The control also provides properties for quickly changing the RadTreeList appearance:

- ShowOuterBorders
- ShowTreeLines
- GridLines

The difference from the other RadControls is that the radTreeList provides a **DetailTemplate** which gives you the freedom to create and design one extra row for each treelist item. This additional detail row allows data-binding the controls within it to the data fields of its parent. Thus, based on your custom preferences you can model the look and feel of the detail item in a non-table-dependant format while at the same time filling it with content related to the parent row.

The following sample illustrates one possible usage of the detail item feature and the other appearance options of RadTreeList:

1. Create a new project in Visual Studio and add a RadScriptManager on top of the form. You can also add RadSkinManager and set its Skin property, and a RadFormDecorator control for page styling.
2. Then add the RadTreeList control itself using the below code for it:

ASPX

```
<telerik:RadTreeList runat="server" ID="RadTreeList1" DataSourceID="SqlDataSource1"
    AutoGenerateColumns="false" AllowPaging="true" PageSize="5" DataKeyNames="EmployeeID"
    ParentDataKeyNames="ReportsTo">
  <Columns>
    <telerik:TreeListBoundColumn DataField="EmployeeID" HeaderText="EmployeeID"
      UniqueName="EmployeeID">
    </telerik:TreeListBoundColumn>
    <telerik:TreeListBoundColumn DataField="LastName" HeaderText="Last Name"
      UniqueName="LastName">
    </telerik:TreeListBoundColumn>
    <telerik:TreeListBoundColumn DataField="FirstName" HeaderText="First Name"
      UniqueName="FirstName">
    </telerik:TreeListBoundColumn>
    <telerik:TreeListBoundColumn DataField="Title" HeaderText="Title" UniqueName="Title">
    </telerik:TreeListBoundColumn>
    <telerik:TreeListBoundColumn DataField="ReportsTo" HeaderText="ReportsTo"
      UniqueName="ReportsTo">
    </telerik:TreeListBoundColumn>
  </Columns>
</telerik:RadTreeList>
<asp:SqlDataSource ID="SqlDataSource1" ConnectionString="<%=
  ConnectionStrings:NorthwindConnectionString %>"
  ProviderName="System.Data.SqlClient" SelectCommand="SELECT EmployeeID, LastName, FirstNa
  Title, ReportsTo, Notes FROM Employees"
  runat="server"></asp:SqlDataSource>
```

3. Wrap the RadTreeList into RadAjaxPanel to omit the page flickering when RadTreeList items are changing their modes.
4. Set the ShowOuterBorders, ShowTreeLines and GridLines properties:

ASPX

```
<telerik:RadTreeList runat="server" ID="RadTreeList1" DataSourceID="SqlDataSource1"
    AutoGenerateColumns="false" AllowPaging="true" ShowOuterBorders="true"
    ShowTreeLines="false" GridLines="None" PageSize="5" DataKeyNames="EmployeeID"
    ParentDataKeyNames="ReportsTo">
  <Columns>
```

```

<telerik:TreeListBoundColumn DataField="EmployeeID" HeaderText="EmployeeID"
UniqueName="EmployeeID">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="LastName" HeaderText="Last Name"
UniqueName="LastName">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="FirstName" HeaderText="First Name"
UniqueName="FirstName">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="Title" HeaderText="Title" UniqueName="Title">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="ReportsTo" HeaderText="ReportsTo"
UniqueName="ReportsTo">
</telerik:TreeListBoundColumn>
</Columns>
</telerik:RadTreeList>
<asp:SqlDataSource ID="SqlDataSource1" ConnectionString="<%=
ConnectionString:NorthwindConnectionString %>"
ProviderName="System.Data.SqlClient" SelectCommand="SELECT EmployeeID, LastName, FirstNa
r
Title, ReportsTo, Notes FROM Employees"
runat="server"></asp:SqlDataSource>

```

5. Enable the DetailTemplate feature using the following code:

ASPX

```

<telerik:RadTreeList runat="server" ID="RadTreeList1" DataSourceID="SqlDataSource1"
AutoGenerateColumns="false" AllowPaging="true" ShowOuterBorders="true"
ShowTreeLines="false" GridLines="None" PageSize="5" DataKeyNames="EmployeeID"
ParentDataKeyNames="ReportsTo">
<DetailTemplate>
<table>
<tr>
<td>
<img src='<%= Page.ResolveUrl("~/Img/") + Eval("EmployeeID") %>.jpg' alt='
Eval("LastName") + " " + Eval("FirstName") %>' />
</td>
<td>
<asp:Label ID="lblNotes" runat="server" Text='<%= Eval("Notes") %
>'></asp:Label>
</td>
</tr>
</table>
</DetailTemplate>
</Columns>
<telerik:TreeListBoundColumn DataField="EmployeeID" HeaderText="EmployeeID"
UniqueName="EmployeeID">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="LastName" HeaderText="Last Name"
UniqueName="LastName">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="FirstName" HeaderText="First Name"
UniqueName="FirstName">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="Title" HeaderText="Title" UniqueName="Title">
</telerik:TreeListBoundColumn>
<telerik:TreeListBoundColumn DataField="ReportsTo" HeaderText="ReportsTo"

```

```
UniqueName="ReportsTo">
  </telerik:TreeListBoundColumn>
</Columns>
</telerik:RadTreeList>
<asp:SqlDataSource ID="SqlDataSource1" ConnectionString="<%=
ConnectionString:NorthwindConnectionString %>"
ProviderName="System.Data.SqlClient" SelectCommand="SELECT EmployeeID, LastName, FirstNa
Title, ReportsTo, Notes FROM Employees"
runat="server"></asp:SqlDataSource>
```

37.7 Summary

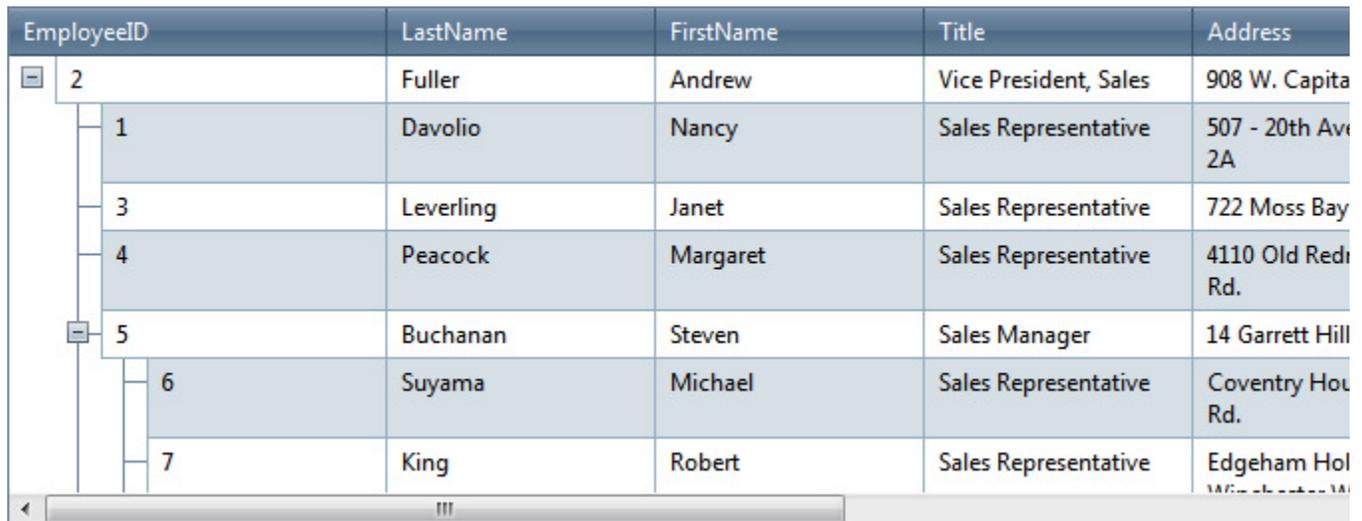
In this chapter we looked at the RadTreeList control and explored its most commonly used features like paging, sorting and items selection.

You learned how to use RadTreeList in Design Time, and to build its layout with ease.

You saw how to implement a sample project on how to manipulate the RadTreeList appearance and create a DetailTemplate.

37.8 Scrolling

Often, when constructing a Web page that contains a treelist, there are design limitations regarding the size of the treelist. In such cases, you may need to enable client-side treelist scrolling so that the treelist can fit in the allowed space. You can enable scrolling by setting the **ClientSettings.Scrolling.AllowScroll** property to **True** (By default its value is **False**).



EmployeeID	LastName	FirstName	Title	Address
2	Fuller	Andrew	Vice President, Sales	908 W. Capita
1	Davolio	Nancy	Sales Representative	507 - 20th Ave 2A
3	Leverling	Janet	Sales Representative	722 Moss Bay
4	Peacock	Margaret	Sales Representative	4110 Old Red Rd.
5	Buchanan	Steven	Sales Manager	14 Garrett Hill
6	Suyama	Michael	Sales Representative	Coventry Hou Rd.
7	King	Robert	Sales Representative	Edgeham Hol Wickhampton W

When scrolling is enabled, the treelist columns should declare **HeaderStyle.Width**.

Using static headers

The most common problem while scrolling is losing the context of the current column. This context is supplied by the column header. RadTreeList lets you keep the header at the top even when scrolling the treelist. To enable this feature, set the **ClientSettings.Scrolling.UseStaticHeaders** property to **True** (Its default value is **False**).

When **UseStaticHeaders** is **True**, the header row is still visible, even when the treelist is scrolled:

EmployeeID	LastName	FirstName	Title	Address	Report
3	Leverling	Janet	Sales Representative	722 Moss Bay Blvd.	2
4	Peacock	Margaret	Sales Representative	4110 Old Redmond Rd.	2
5	Buchanan	Steven	Sales Manager	14 Garrett Hill	2
6	Suyama	Michael	Sales Representative	Coventry House Miner Rd	5

Navigation: [K] [←] [1] [→] [>] Page size: 10

When `UseStaticHeaders` is `False`, the header scrolls along with the data rows:

EmployeeID	LastName	FirstName	Title	Address	Report
3	Leverling	Janet	Sales Representative	722 Moss Bay Blvd.	2
4	Peacock	Margaret	Sales Representative	4110 Old Redmond Rd.	2
5	Buchanan	Steven	Sales Manager	14 Garrett Hill	2
6	Suyama	Michael	Sales Representative	Coventry House Miner Rd	5

Setting height to the scrollable RadTreeList

`ClientSettings.Scrolling.ScrollHeight` property determines the height of the control's scrollable area when scrolling is enabled. Depending on whether static headers are enabled or not, the scrollable area includes different portions of the `RadTreeList` control:

- If static headers are enabled, the scrollable container includes only the data and footer items. The header and pager are "static" (not scrolled).
- If static headers are not enabled, the scrollable container includes everything.

Saving scroll position

`ClientSettings.Scrolling.SaveScrollPosition` property gets or sets a value indicating whether `RadTreeList` will keep the scroll position during postbacks.

You can set the scrolling properties as below:

ASPX

```
<telerik:RadTreeList ID="RadTreeList1" runat="server">
  <ClientSettings>
    <Scrolling AllowScroll="true" UseStaticHeaders="true" SaveScrollPosition="true"
    ScrollHeight="350px" />
  </ClientSettings>
</telerik:RadTreeList>
```

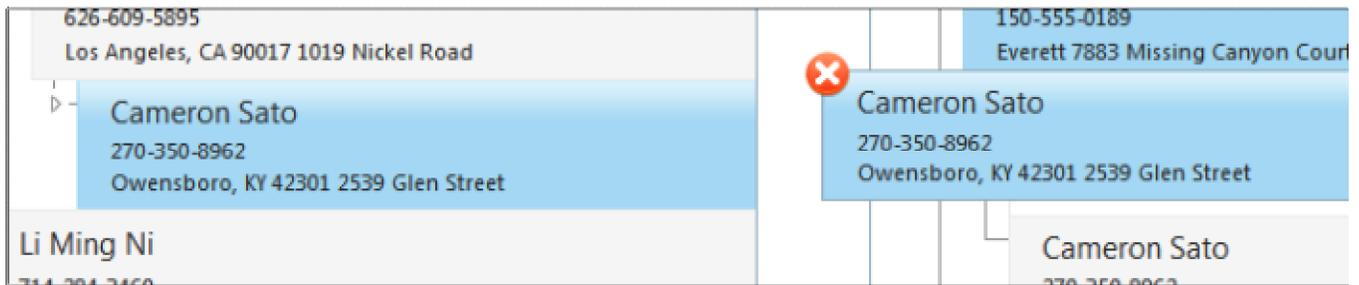
37.9 Items Drag and Drop

The rich drag and drop API facilitate the developers when implementing copy/move operations between different `RadTreeList` and other controls. Among the standard server and client methods and properties you will also find some useful options like:

- automatic items reorder when using `SqlDataSource` control.

UI for ASP.NET AJAX

- option to display a special icon (drop clue) that can be changed (manually) depending on the hovered container



Configuring RadTreeList for Drag and Drop

To enable this functionality in RadTreeList, you need to:

- set ClientSettings-AllowItemsDragDrop property to true
- enable client selection by setting the ClientSettings-Selecting-AllowItemSelection property to true

Drag and Drop

```
<ClientSettings AllowItemsDragDrop="true">  
  <Selecting AllowItemSelection="True" />  
</ClientSettings>
```

Item Reordering

RadTreeList supports automatic item reordering when SqlDataSource is used. In this case, the developer needs to configure the datasource control to have valid update command (as for automatic operations). When binding to another type of datasource, the reordering should be handled manually via the client/server API.

Server-Side API

- **OnItemDrop** - this event occurs when a RadListView item is dragged and dropped over HTML element (if not cancelled via the client-side API)
- **AllowItemsDragDrop** - property which is used to enable dragged and dropped over HTML element (if not cancelled via the client-side API)

Client-Side API

Below you can find a list of all client-side events that can be used when implementing drag and drop operations. It is important to mention that OnItemDragging and OnItemDropping events can be cancelled by setting the set_cancel property available in their event arguments.

A convenient feature of the client-side API is that the control automatically detects some of the invalid reordering operations, such as trying to drag a parent item onto its child or trying to drop an item over itself. In these cases, the get_canDrop property will return false when invoked in the OnItemDragging event.

- **OnItemDragStarted** - event is fired when a drag action is started.
- **OnItemDragging** - event is fired when a TreeListDataItem is being dragged.
- **OnItemDropping** - event is fired when a TreeListDataItem is being dropped.
- **OnItemDropped** - event is fired when a TreeListDataItem has been dropped.

37.10 Load On Demand

Starting with the Q2 2011 release **RadTreeList** supports a new **Load-on-Demand** functionality. It allows child nodes to be added on the fly as parent nodes are expanded. This mode is useful when you need to fill sub nodes only, when the parent node is expanded or the data source contains thousands of records:

Load On Demand mechanism

To use the Load-On-Demand mechanism:

1. Set **AllowLoadOnDemand** property to **true**
2. Get the root items from the datasource and assign them to the **RadTreeList.DataSource** into **RadTreeList.NeedDataSource** event handler:

C#

```
protected void RadTreeList1_NeedDataSource(object sender, TreeListNeedDataSourceEventArgs e)
{
    RadTreeList1.DataSource = GetDataTable("SELECT * FROM TestItems WHERE ParentID IS NULL",
    null);
}
```

VB.NET

```
Protected Sub RadTreeList1_NeedDataSource(ByVal sender As Object, ByVal e As
TreeListNeedDataSourceEventArgs)
    RadTreeList1.DataSource = GetDataTable("SELECT * FROM TestItems WHERE ParentID IS NULL",
Nothing)
End Sub
```

3. Handle the **RadTreeList.ChildItemsDataBind** event and select the subset of items related to the expanded item. Assign them to the child items datasource property that is available through the second argument passed to the event handler:

C#

```
protected void RadTreeList1_ChildItemsDataBind(object sender,
TreeListChildItemsDataBindEventArgs e)
{
    int id = Convert.ToInt32(e.ParentDataKeyValues["ID"].ToString());
    e.ChildItemsDataSource = GetDataTable("SELECT * FROM TestItems WHERE ParentID = " + id);
}
```

VB.NET

```
Protected Sub RadTreeList1_ChildItemsDataBind(ByVal sender As Object, ByVal e As
TreeListChildItemsDataBindEventArgs)
    Dim id As Integer = Convert.ToInt32(e.ParentDataKeyValues("ID").ToString())
    e.ChildItemsDataSource = GetDataTable("SELECT * FROM TestItems WHERE ParentID = " & id)
End Sub
```

In addition, the **RadTreeList** control always shows **ExpandCollapse** button in front of each item. When the item is expanded and there are no nested items, by default nothing will be displayed below the expanded item, but the **ExpandCollapse** button will stay. If you want to hide the expand button in this case you can set **HideExpandCollapseButtonIfNoChildren** property to **true**.

For a live example illustrating this approach you can see **TreeList / Load on Demand** (<http://demos.telerik.com/aspnet-ajax/treelist/examples/databinding/loadondemand/defaultcs.aspx>) demo.

37.11 Columns

This article will introduce you to the main specifics of **RadTreeList** columns.

These are the different column types supported by the **RadTreeList** control which you can use in order to

display your data:

- **TreeListBoundColumn** - this is a column bound to a field in the data source.
- **TreeListButtonColumn** - displays a button for each entry in the column.
- **TreeListCalculatedColumn** - displays a value that is calculated based on one or more fields and an expression that indicates how to calculate the display value.
- **TreeListCheckBoxColumn** - it displays a checkbox used to represent a boolean value from the data source.
- **TreeListDateTimeColumn** - a column type used for displaying and editing DateTime values.
- **TreeListEditCommandColumn** - enables you to fire an Edit or InitInsert command.
- **TreeListHyperLinkColumn** - used to display a hyperlink in each cell.
- **TreeListImageColumn** - displays an image in each column cell.
- **TreeListNumericColumn** - this column is used for displaying and editing numeric values.
- **TreeListSelectColumn** - allows client-side or server-side row selection depending on the selecting settings of the RadTreeList control.
- **TreeListTemplateColumn** - lets you specify an item template which determines how will each cell of the column be displayed.

Also, there are different ways to create the RadTreeList columns - to auto-generate them, to declare them in mark-up or add them dynamically to the **Columns** collection. More information about treelist column types is available in this [help article \(http://www.telerik.com/help/aspnet-ajax/treelist-column-types.html\)](http://www.telerik.com/help/aspnet-ajax/treelist-column-types.html) and [online example \(http://demos.telerik.com/aspnet-ajax/treelist/examples/columns/columnntypes/defaultcs.aspx\)](http://demos.telerik.com/aspnet-ajax/treelist/examples/columns/columnntypes/defaultcs.aspx).

Column Resizing

If you want the columns in your treelist to be resizable, set the **ClientSettings.Resizing.AllowColumnResize** property to **True**. When AllowColumnResize is True, users can resize columns by dragging the handle between column headers. The default value for this property is false.

The resizing feature can be adjusted using the following properties:

- When resizing is enabled (AllowColumnResize is True), you can disable column resizing for individual columns by setting the column's **Resizable** property to False. Setting a column's Resizable property has no effect if AllowColumnResize is False.
- To specify whether columns are resized using real-time resizing, set the **ClientSettings.Resizing.EnableRealTimeResize** property. The default value for this property is False.
- There are three modes of column resizing:
 - **NoScroll** (this is the default value) - No changes in the width of the TreeList. The resized column changes width, while the other columns are squeezed at the two ends.
 - **AllowScroll** - Works only when scrolling is turned on. Does not change the width of the treelist, only the width of its inner table and adds scroll. The resized column changes width while the other columns stay the same.
 - **ResizeTreeList** - The whole control changes width together with the resized column. Other columns stay the same width.
- Treelist columns also support minimum / maximum width only for the currently resized column by setting **MinWidth /MaxWidth** properties.

Column Reordering

You can allow users to set the order of the treelist columns by dragging and dropping them. Just set the **ClientSettings.Reordering.AllowColumnsReorder** property to **True**. There are two possible modes for column reordering: client and server-side. If you want to reorder columns on client, set the

ClientSettings.Reordering.ReorderColumnsOnClient property to True.

- When columns are reordered on the client, the **ClientSettings.Reordering.ColumnsReorderMethod** property determines what happens when the user drops a column in a new position. When **ColumnsReorderMethod** is "Swap" (the default) it switches places of two columns. Columns between them do not change order. When **ColumnsReorderMethod** is "Reorder" it places the first (dragged) column at the place of the second (dropped on) column. Columns between them also change order.
- When columns are reordered on the server, the treelist uses the "swap" method multiple times to re-order columns.

You can see column resizing and reordering in action in the online demo available [here](http://demos.telerik.com/aspnet-ajax/treelist/examples/client/resizing/defaultcs.aspx) (<http://demos.telerik.com/aspnet-ajax/treelist/examples/client/resizing/defaultcs.aspx>).

Aggregates

RadTreeList provides the option to display column aggregates. The calculated total values are displayed in the footer item at the end of each level. The **Aggregate** property can be set to any of the following values: "Avg", "Count", "CountDistinct", "First", "Last", "Max", "Min", "Sum".

You can set the **Aggregate** property of a bound column to the function that you want to be used in calculating the aggregated value. Then just set **ShowFooter="true"** in the **RadTreeList** declaration to start showing aggregates. A footer will appear at the bottom of each level in the treelist showing the totals from the items in this level.

The Column Aggregates feature is demonstrated in the online demo available [here](http://demos.telerik.com/aspnet-ajax/treelist/examples/columns/aggregates/defaultcs.aspx) (<http://demos.telerik.com/aspnet-ajax/treelist/examples/columns/aggregates/defaultcs.aspx>).

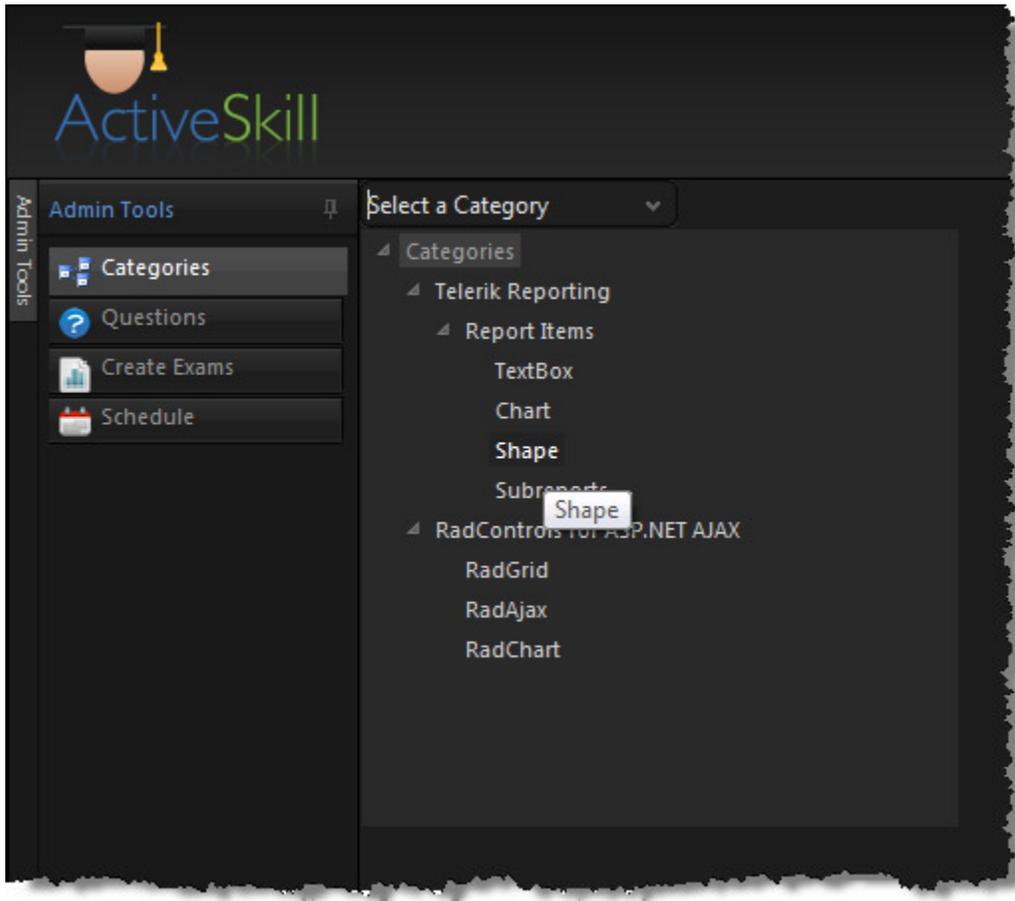
38 ActiveSkill: Database Maintenance

38.1 Objectives

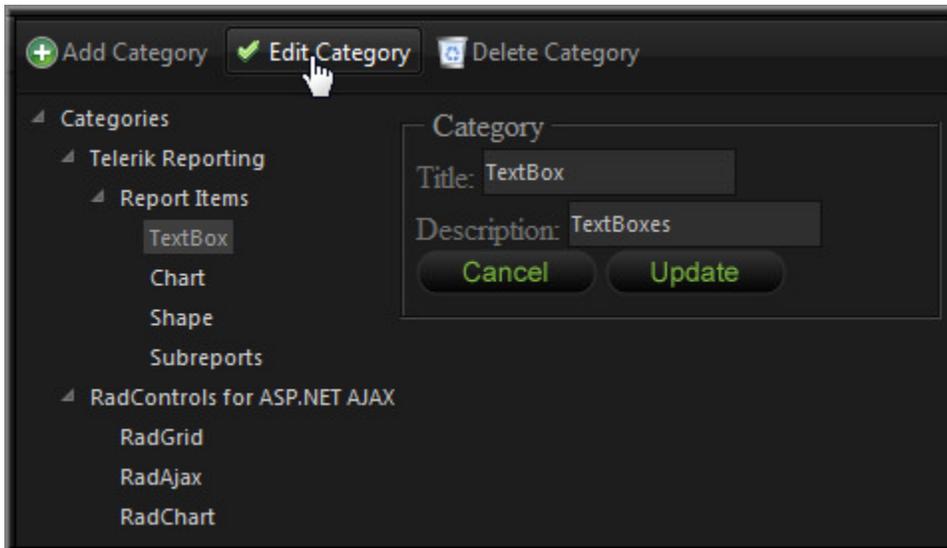
- Build a web user control that contains a RadComboBox with a RadTreeView inside. Reuse this control in several locations.
- Use RadControls within a standard FormView control. Use Eval() and Bind() expressions.
- Build a single grid containing master and detail data with full CRUD functionality.
- Build two related grids, one containing master data and the other containing detail data, both with CRUD functionality. Make use of template columns containing check boxes.

38.2 Introduction

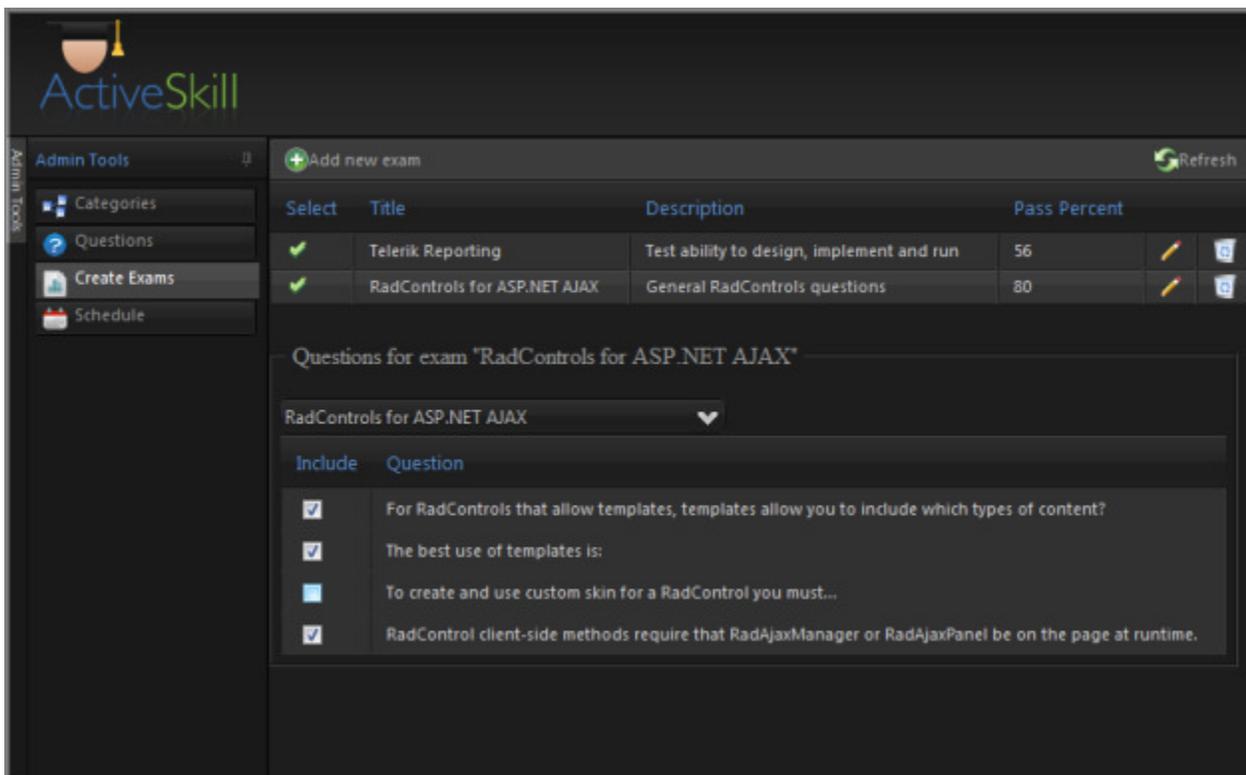
The material in this chapter may be somewhat heavy going, as you will be building user controls that represent "pages" that display in response to the clicking on the tab strip. Each "page" handles full CRUD functionality for Categories, Questions and Exam tables. You will be creating a control that displays all categories in a RadTreeView that displays within a RadComboBox or on its own. This control will be reused in all three "pages".



We will use the new "CategoriesTree" control together with a standard ASP.NET FormView to add, edit and delete categories. The screenshot below shows RadTextBox controls within the FormView. Each of the FormView templates contains RadTextBox controls bound within the markup using Bind() and Eval() binding expressions.

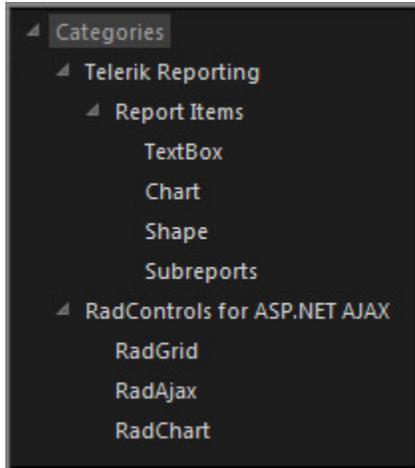


We will be using the grid heavily to access its powerful editing and viewing capabilities. For example, the screenshot below shows exams in one grid and questions in another grid. The questions are filtered by the CategoriesTree user control we will build beforehand. The check boxes in the "Include" column will be checked if a "ExamQuestions" join table record exists for the exam and question combination. You will be using much of the built-in "automatic" ability, but also using template columns to get very specific behaviors from your RadGrid. In fact the "Include" checkboxes are standard checkboxes accessed within code to deliver custom functionality.



38.3 Building the Categories Tree Control

We will need a tree view control bound to all the possible question categories for several of our maintenance pages. The control will need to display as both a combo so the tree view can drop down and as a stand alone tree view.



You can find the complete source for this project at:
\\VS Projects\ActiveSkill Database Maintenance\Categories Tree

Prepare the Control Layout

1. In the \Controls folder of the ActiveSkillUI project, add a new user control and name the class file "CategoriesTree.ascx".
2. In the design view for the user control, add a **RadComboBox**.
 - o Set the **Text** property to "Select a Category", **DropDownWidth** to "300px" and **AllowCustomText** to true.
 - o Open up the **CollapseAnimation** and **ExpandAnimation** properties. Set the **Type** sub-property to be "None".
 - o Using the **RadComboBox Item Builder** dialog (from the Smart Tag **Build RadComboBox...** option), add a single item. Set the **Text** and **Value** item properties blank.
3. Add a **RadTreeView** just below the combo box.
 - o Set the **ID** property to "StandAloneTreeView".
 - o Set the **DataFieldID** property to "ID", **DataFieldParentID** to "ParentID", **DataTextField** to "Title" and **ValueField** to "ID".
 - o Using the events (⚡) button of the Properties window, locate the **NodeClick** event. Type in "CategoriesNodeClick" and hit **Enter**. This will create a server-side event handler that we will code later.
 - o Using the events (⚡) button of the Properties window, locate the **NodeDataBound** event. Type in "CategoriesNodeDataBound" and hit **Enter**. This will create a server-side event handler that we will code later.
4. Select the RadTreeView, right-click and select **Copy** from the context menu.

5. From the RadComboBox Smart Tag, select **Edit Templates**.
6. Right-click in the open template area and add a Div component from the HTML tab of the Toolbox. Right-click inside the div and select **Paste** from the context menu.
7. Set the **ID** property for the new treeview to "DropDownTreeView".
8. Set the **OnClientNodeClicking** event to "ClientNodeClicking". We will create a client-side event handler for this in later steps.
9. Set the **OnClientNodeClicked** event to "ClientNodeClicked". We will create a client-side event handler for this in later steps.
10. Select **End Template Editing** from the RadComboBox Smart Tag.

The markup so far should look like the example below:

[ASP.NET] Markup for Combo and Tree View Controls

```
<!--Drop down treeview-->
<telerik:RadComboBox ID="cboxCategories" runat="server"
  Text="Select a Category"
  AllowCustomText="True"
  DropDownWidth="300px" Width="300px"
  >
  <Items>
    <telerik:RadComboBoxItem runat="server" />
  </Items>
  <ItemTemplate>

    <div id="divTreeView" >

      <telerik:RadTreeView ID="DropDownTreeView" runat="server" Height="300px"
        DataFieldID="ID"
        DataFieldParentID="ParentID"
        DataTextField="Title"
        DataValueField="ID"
        OnNodeClick="CategoriesNodeClick"
        OnNodeDataBound="CategoriesNodeDataBound"
        OnClientNodeClicked="ClientNodeClicked"
        OnClientNodeClicking="ClientNodeClicking"
      >
    </telerik:RadTreeView>

  </div>

</ItemTemplate>
<CollapseAnimation Type="None" />
<ExpandAnimation Type="None" />
</telerik:RadComboBox>
<!--Stand alone treeview-->
<telerik:RadTreeView ID="StandAloneTreeView" runat="server"
  DataFieldID="ID"
  DataFieldParentID="ParentID"
  DataTextField="Title"
  DataValueField="ID"
  OnNodeClick="CategoriesNodeClick"
  OnNodeDataBound="CategoriesNodeDataBound"
  >
```

```
>  
</telerik:RadTreeView>
```

Add Server Properties

1. Add the **Telerik.Web.UI** and **System.Data** namespaces to the "Imports" (VB) or "uses" (C#) section of code.
2. Add properties to the CategoriesTreeView code behind.

These properties keep track of the "DisplayMode" which can be "DropDown" or "TreeView". "IsRootNode" is a shortcut for consumers of the control to know if the top level node has been selected. "ID" and "CategoryID" properties are shortcuts to values stored in the selected node and the parent node of the selected node. These last two properties are used when binding to declarative data sources and in code-behind.

[VB] CategoriesTree Properties

```
#region properties  
' "DisplayMode" is used to signal which treeview we're  
' showing, the one within the combo or the stand alone  
' version.  
Public Enum DisplayModes  
    DropDown  
    TreeViewOnly  
End Enum  
Const DisplayModeKey As String = "DisplayModeKey"  
Public Property DisplayMode() As DisplayModes  
    Get  
        Return IIf(ViewState(DisplayModeKey) = Nothing, DisplayModes.TreeViewOnly, DirectCast  
(ViewState(DisplayModeKey), DisplayModes))  
    End Get  
    Set  
        ViewState(DisplayModeKey) = value  
    End Set  
End Property  
' returns the visible treeview as controlled by  
' the DisplayMode  
Public ReadOnly Property TreeView() As RadTreeView  
    Get  
        Return IIf(Me.DisplayMode = DisplayModes.DropDown, TryCast(Me.cboxCategories.Items  
(0).FindControl("DropDownTreeView"), RadTreeView), Me.StandAloneTreeView)  
    End Get  
End Property  
' true if the root node of the tree is selected  
Public ReadOnly Property IsRootSelected() As Boolean  
    Get  
        Return TreeView.SelectedValue = "-1"  
    End Get  
End Property  
' Returns the category id stored in the selected  
' node's value. Used by data source parameters.  
Public Property CategoryID() As String  
    Get  
        Return TreeView.SelectedValue  
    End Get  
    Set  
        TreeView.SelectedNode.Value = value  
    End Set  
End Property
```

```

End Set
End Property
' Returns the parent category id stored in the selected
' node's, parent node value. Used by data source parameters.
Public ReadOnly Property ParentCategoryID() As String
Get
Return TreeView.SelectedNode.ParentNode.Value
End Get
End Property
#End Region properties

```

[C#] CategoriesTree Properties

```

#region properties
// "DisplayMode" is used to signal which treeview we're
// showing, the one within the combo or the stand alone
// version.
public enum DisplayModes { DropDown, TreeViewOnly };
const string DisplayModeKey = "DisplayModeKey";
public DisplayModes DisplayMode
{
get
{
return ViewState[DisplayModeKey] == null ?
DisplayModes.TreeViewOnly : (DisplayModes)ViewState[DisplayModeKey];
}
set
{
ViewState[DisplayModeKey] = value;
}
}
// returns the visible treeview as controlled by
// the DisplayMode
public RadTreeView TreeView
{
get
{
return this.DisplayMode == DisplayModes.DropDown ?
this.cboxCategories.Items[0].FindControl("DropDownTreeView") as RadTreeView :
this.StandAloneTreeView;
}
}
// true if the root node of the tree is selected
public bool IsRootSelected
{
get { return TreeView.SelectedValue == "-1"; }
}
// Returns the category id stored in the selected
// node's value. Used by data source parameters.
public string CategoryID
{
get { return TreeView.SelectedValue; }
set { TreeView.SelectedNode.Value = value; }
}
// Returns the parent category id stored in the selected

```

```
// node's, parent node value. Used by data source parameters.
public string ParentCategoryID
{
    get { return TreeView.SelectedNode.ParentNode.Value; }
}
#endregion properties
```

Add Public Methods

Add public methods to the CategoriesTree code-behind.

Consumers of the control will need to add, edit and delete nodes from the active tree view. Also, consumers will need to load the treeview with data from a data source. Note that a dependency exists between the columns in the data source fed to this control and the Data properties of the tree view, i.e. where DataTextField = "Title", DataFieldID = "ID" and DataFieldParentID = "ParentID".

[VB] CategoriesTree Public Methods

```
#region public methods
' Allow consumers of this control to insert a node.
Public Sub InsertCategoryNode(ByVal title As String, ByVal description As String)
    ' create new node and move selection to new node before expanding parent node
    Dim node As New RadTreeNode(title)
    node.ToolTip = description
    TreeView.SelectedNode.Nodes.Add(node)
    node.Selected = True
    node.ParentNode.ExpandChildNodes()
End Sub
' Allow consumers of this control to delete a node
Public Sub DeleteCategoryNode()
    Dim parentNode As RadTreeNode = TreeView.SelectedNode.ParentNode
    TreeView.SelectedNode.Remove()
    If parentNode <> Nothing Then
        parentNode.Selected = True
    End If
End Sub
' Allow consumers of this control to update a node
Public Sub UpdateCategoryNode(ByVal title As String, ByVal description As String)
    TreeView.SelectedNode.Text = title
    TreeView.SelectedNode.ToolTip = description
End Sub
' Allow consumers of this control to initialize the
' control with data. Note: there is a dependency here
' that the data have column names corresponding to the
' treeview Data properties, i.e. DataTextField, etc.
Public Sub InitialLoad(ByVal dataSource As Object)
    TreeView.DataSource = dataSource
    TreeView.DataBind()
    TreeView.Nodes(0).Selected = True
End Sub
#End Region
```

[C#] CategoriesTree Public Methods

```
#region public methods
```

```

// Allow consumers of this control to insert a node.
public void InsertCategoryNode(string title, string description)
{
    // create new node and move selection to new node before expanding parent node
    RadTreeNode node = new RadTreeNode(title);
    node.ToolTip = description;
    TreeView.SelectedNode.Nodes.Add(node);
    node.Selected = true;
    node.ParentNode.ExpandChildNodes();
}
// Allow consumers of this control to delete a node
public void DeleteCategoryNode()
{
    RadTreeNode parentNode = TreeView.SelectedNode.ParentNode;
    TreeView.SelectedNode.Remove();
    if (parentNode != null)
    {
        parentNode.Selected = true;
    }
}
// Allow consumers of this control to update a node
public void UpdateCategoryNode(string title, string description)
{
    TreeView.SelectedNode.Text = title;
    TreeView.SelectedNode.ToolTip = description;
}
// Allow consumers of this control to initialize the
// control with data. Note: there is a dependency here
// that the data have column names corresponding to the
// treeview Data properties, i.e. DataTextField, etc.
public void InitialLoad(object dataSource)
{
    TreeView.DataSource = dataSource;
    TreeView.DataBind();
    TreeView.Nodes[0].Selected = true;
}
#endregion

```

Add Server Events

1. Add a single event type to CategoriesTree so that consumers of the control can respond to node clicks. This event will be triggered in response to the tree view NodeClick event.

[VB] Declaring the NodeClick Event

```

#region events
' Reuse the same event type as the tree view itself
Public Event NodeClick As RadTreeViewEventHandler
#End Region events

```

[C#] Declaring the NodeClick Event

```

#region events
// Reuse the same event type as the tree view itself
public event RadTreeViewEventHandler NodeClick;
#endregion events

```

Add Page Events

Add the following event handlers for page and control events.

In the Page_Load event, the appropriate treeview (based on DisplayMode setting) is made visible. The CategoriesNodeClick event handler (which we earlier hooked up to the NodeClick event of both tree views) simply fires the CategoriesTreeView Nodeclick even—if the control's consumer has defined an event handler for it. In the NodeDataBound event, the tree view ToolTip and Value properties are populated from the database.

[VB] Handling Page and Control Events

```
#region page events
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' make the appropriate controls visible based on DisplayMode
    Me.cboxCategories.Visible = Me.DisplayMode = DisplayModes.DropDown
    Me.StandAloneTreeView.Visible = Me.DisplayMode = DisplayModes.TreeViewOnly
End Sub
' allow the consumer of this control to respond to tree view node clicks
Protected Sub CategoriesNodeClick(ByVal sender As Object, ByVal e As RadTreeNodeEventArgs)
    If NodeClick <> Nothing Then
        NodeClick(sender, e)
    End If
End Sub
' Save the category ID in the value of the node. Also store
' the category description in the tooltip.
Protected Sub tvCategories_NodeDataBound(ByVal sender As Object, ByVal e As
RadTreeNodeEventArgs)
    Dim drv As DataRowView = TryCast(e.Node.DataItem, DataRowView)
    e.Node.ToolTip = drv("Description").ToString()
    e.Node.Value = drv("ID").ToString()
    e.Node.Expanded = True
End Sub
#End Region
```

[C#] Handling Page and Control Events

```
#region page events
protected void Page_Load(object sender, EventArgs e)
{
    // make the appropriate controls visible based on DisplayMode
    this.cboxCategories.Visible = this.DisplayMode == DisplayModes.DropDown;
    this.StandAloneTreeView.Visible = this.DisplayMode == DisplayModes.TreeViewOnly;
}
// allow the consumer of this control to respond to tree view node clicks
protected void CategoriesNodeClick(object sender, RadTreeNodeEventArgs e)
{
    if (NodeClick != null)
    {
        NodeClick(sender, e);
    }
}
// Save the category ID in the value of the node. Also store
// the category description in the tooltip.
protected void tvCategories_NodeDataBound(object sender, RadTreeNodeEventArgs e)
```

```

{
  DataRowView drv = e.Node.DataItem as DataRowView;
  e.Node.ToolTip = drv["Description"].ToString();
  e.Node.Value = drv["ID"].ToString();
  e.Node.Expanded = true;
}
#endregion

```

Handle Client Events

As you may remember from the RadComboBox chapter, controls within the RadComboBox ItemTemplate that are also AJAX-enabled can use a bit of special handling so that the combo box and the control it wraps work smoothly as a unit.

1. Add an event handler "onclick="StopPropagation(event)" to "divTreeView". This div wraps the "drop down" treeview inside the combo box template. The event handler will consume mouse clicks made just outside the treeview but still inside the combo box drop down area. We will see how this plays out a little later when we test the control.

[ASP.NET] Adding the onclick Event Handler

```
<div id="divTreeView" onclick="StopPropagation(event)">
```

2. Add three event handlers. Two are for the treeview client events ClientNodeClicking and ClientNodeClicked. The last event handler StopPropagation() handles clicks in the combo box that are outside the treeview area.

[JavaScript] Handling TreeView Events

```

<script type="text/javascript" language="javascript">
function ClientNodeClicked(sender, args) {
  var combo = $find('<%=cboxCategories.ClientID%>');
  // if a node has been selected in the treeview,
  // get the text for the selected node and populate
  // the combo text with it. Close the drop down
  // and do not pass any more events to the combo
  if (sender.get_selectedNode() != null) {
    var text = sender.get_selectedNode().get_text();
    combo.set_text(text);
    combo.hideDropDown();
    // stop event propagation to the combo box.
    // if the combo had a OnClientSelectedIndexChanged defined
    // it will not run if stopPropatation() is called.
    args.get_domEvent().stopPropagation();
  }
}
function ClientNodeClicking(sender, args) {
  var combo = $find('<%=cboxCategories.ClientID%>');
  combo.attachDropDown();
}
function StopPropagation(e) {
  e.cancelBubble = true;
  if (e.stopPropagation) {
    e.stopPropagation();
  }
}
</script>

```

Databind and Use the Control

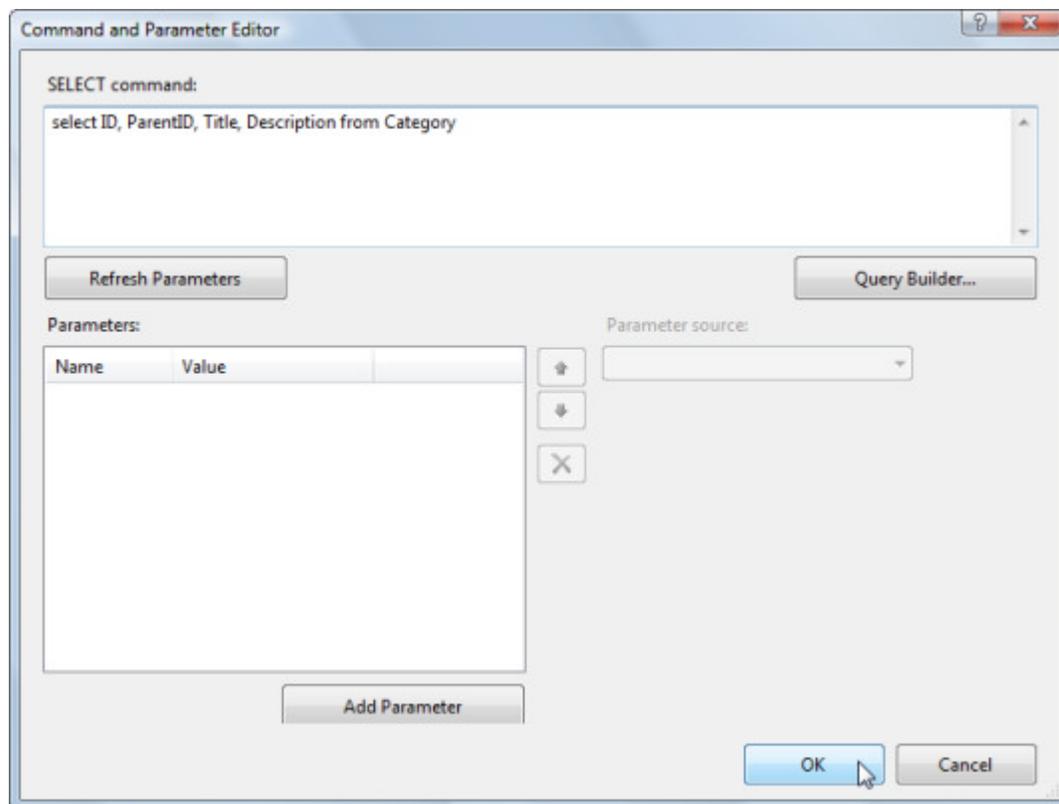
To consume the CategoriesTree control we need to add it to the Categories.ascx "page" and bind it to data.

1. In the design view of the Categories.ascx "page", drop an **SqlDataSource** control from the Data tab of the Toolbox to the design surface. Use the Properties window to set the following:
 - o Set the **ID** property to dsAllCategories.
 - o Set the **ConnectionString** property to "ActiveSkillConnectionString" from the drop down list.
 - o Click the ellipses on the **SelectQuery** property. This brings up the Command and Parameter Editor dialog. Enter the select command:

[T-SQL] Select All Categories

```
select ID, ParentID, Title, Description from Category
```

- o Click the **OK** button to close the dialog.



2. Below the data source, drop a **CategoriesTree** control from the Solution Explorer. Set the **DisplayMode** to "TreeViewOnly".
3. In the "Categories.aspx" code behind add the code below to the FirstLoad() method implementation.

[VB] Coding the FirstLoad() Method

```
Public Sub FirstLoad(ByVal args As Dictionary(Of String, String))  
    CategoriesTree1.InitialLoad(dsAllCategories)  
End Sub
```

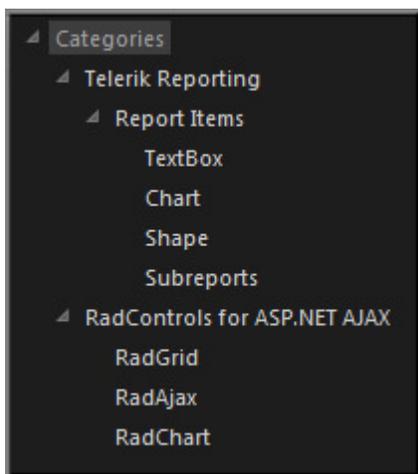
[C#] Coding the FirstLoad() Method

```
public void FirstLoad(Dictionary<string, string> args)
{
    CategoriesTree1.InitialLoad(dsAllCategories);
}
```

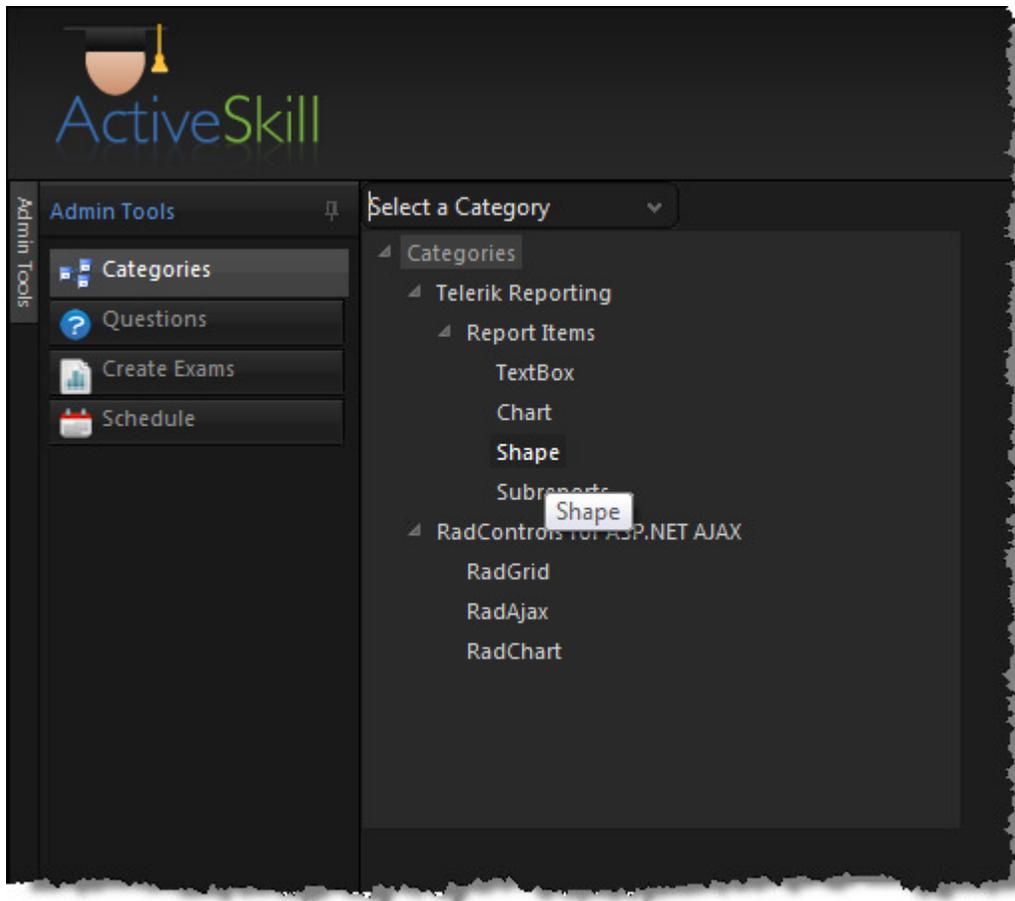
- Press **Ctrl-F5** to run the application. Hover the mouse over some of the treeview entries to see the tool tips that were added during the data bind event.



- Stop the application. The print for the treeview entries is a little dim, so let's make it slightly brighter. In the Solution Explorer, navigate to the `\skins\ActiveSkill\TreeView.ActiveSkill.css` file and open it. At the top of the file is a CSS selector for `".RadTreeView_ActiveSkill .rtEdit .rtIn input"`. Change the color for that style from `"color:#9F9F9F;"` to `"color:#CFCFCF;"`. This will brighten the text very slightly:



- Set the `CategoriesTree DisplayMode` property to `"DropDown"` and re-run the application. Try selecting different items and also click off to the side of the treeview but still within the combo drop down area.



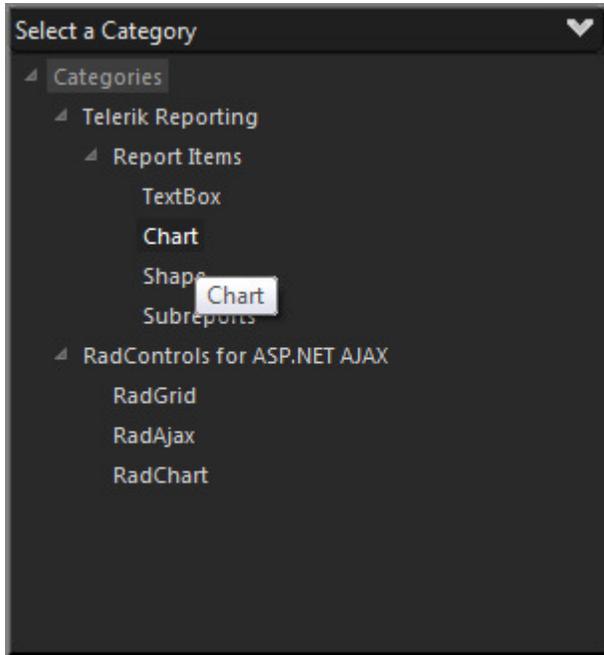
7. The downward pointing arrow for the combo box drop down is a little hard to see so we will replace the graphic to make it a little brighter.



The image file that controls the appearance for this button is called `rcbArrowCell.gif`, and looks like the image below when you open it in a graphics editing application (e.g. PhotoShop). This kind of image is generically called a "CSS Sprite" where the image represents all of the states that a screen element can be. Using CSS sprites enhances performance by allowing all the images for a screen element to be downloaded at one time. The CSS styles determine which chunk of the image will be displayed. You can edit these files using PhotoShop or any other image editing utility.



Copy the image "`rcbArrowCell.gif`" from the folder "`VS Projects\Images\ActiveSkill\Admin Database`" to "`Skins\ActiveSkill\ComboBox`" in your ActiveSkillUI project. This will replace the file that already exists there. Now the combo arrow will look something like this:



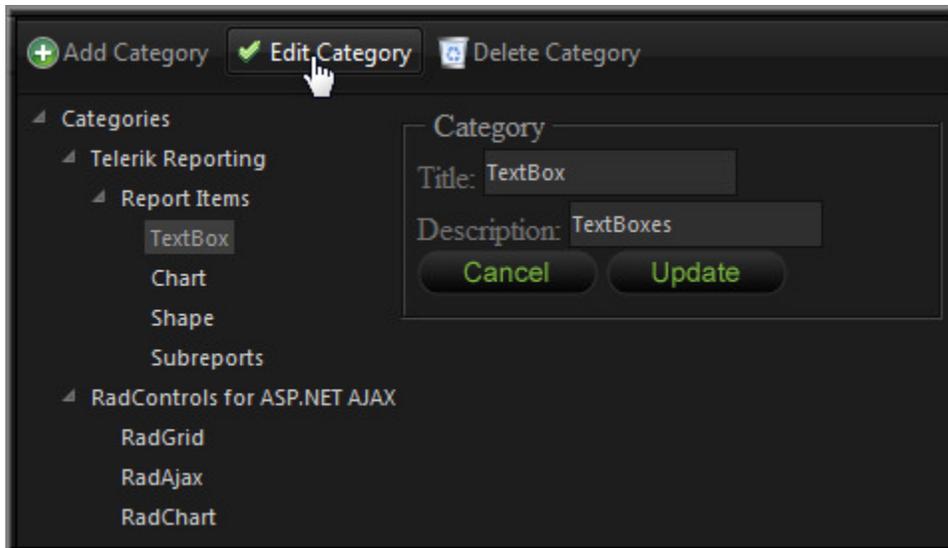
8. While you're testing out the drop-down functionality of this control, now is a good time to look at some of the decisions made in configuring and coding this control, particularly regarding how the combo box and tree view working together. Starting with the client-side code:
- In the ClientNodeClicked event we have a call to stopPropagation() for the combo box's DOM element. This code stops the event from bubbling out from the treeview to the combo box and causing unintended behaviors.
 - In the ClientNodeClicking we call the combo's attachDropDown() method. Without this call, clicking on the treeview will trigger a postback instead of a callback.
 - The click event for the div surrounding the treeview makes a call to StopPropagation(). This ensures that clicking just outside the treeview (but still inside the combo box) is ignored. You get some odd behavior if you don't make this call where the concatenated contents of the tree view appear to be displayed in the combo text:



38.4 Implement Categories Control

Now that we have a working CategoriesTree user control we need to provide facilities for maintaining the category table. We will include Add/Edit/Delete buttons using a RadToolBar and a standard ASP FormView control to host the templates for adding and editing. Within the FormView templates will be RadTextBoxes for category titles and descriptions.

When we are finished we should have a working page that furnishes add/edit/delete against the Category table. The page will look something like the example below. The Add button will be available at all times while the Edit and Delete buttons will only display when a category below the root is selected. Added categories are always appended as a child below the selected node.



You can find the complete source for this project at:
\\VS Projects\ActiveSkill Database Maintenance\Categories

Prepare Layout

RoadMap

Before we get going, here's a general road map that shows how the markup for Categories.aspx will be structured. The screenshot below shows the major parts of the markup commented and with the elements collapsed so that you can see all portions of the page at one time.

- **Data Sources:** You have already defined the "dsAllCategories" data source when testing the CategoriesTree user control. You will add a second data source "dsCategory" to handle the add/update/delete jobs for a single category record.
- **Toolbar with add, edit and delete buttons:** The tool bar buttons have values corresponding to constants on the server for CRUD operations. The images for the button are actually retrieved from the Grid skin images.
- **Treeview with Categories:** This is the CategoriesTree control you added and tested on this page.
- **Fieldset containing FormView:** The standard ASP.NET FormView control has Item, Edit and Insert templates that can hold RadTextBoxes and allow binding to database values on the server using binding expressions. The FormView is the area where the admin actually enters a category name, category description and clicks the "Update" or "Cancel" buttons. The FormView also has an OnItemCreated event that gives us a chance to retrieve a newly generated category ID to plug back into our treeview.

```

<!--Data sources-->
<asp:SqlDataSource ID="dsCategory" ...>...</asp:SqlDataSource>
<asp:SqlDataSource ID="dsAllCategories" ...>...</asp:SqlDataSource>

<!--toolbar with add, edit, delete buttons-->
<telerik:RadToolBar ID="tbarCategories" ...>...</telerik:RadToolBar>

<!--treeview with categories-->
<div id="CategoriesTreeDiv" style="float: left;">
  <ucl:CategoriesTree ID="CategoriesTree1" runat="server"
    DisplayMode="TreeViewOnly" />
</div>

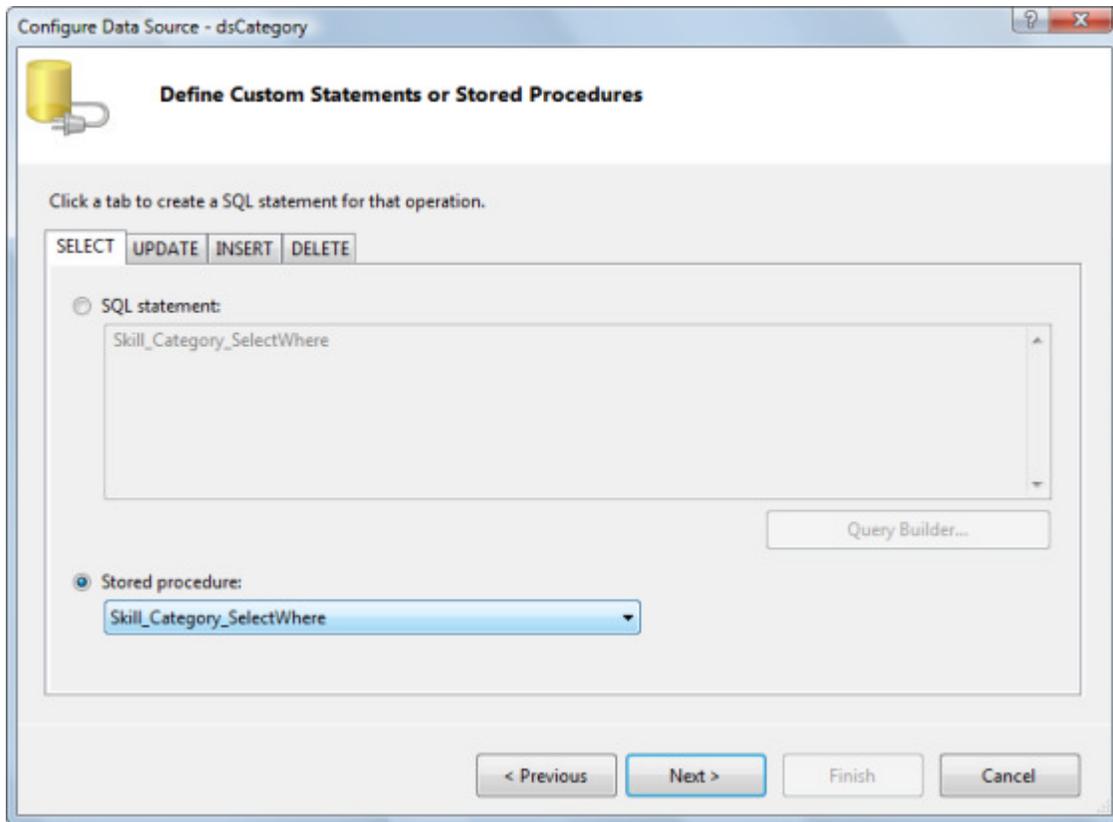
<!--fieldset containing formview with editable area and buttons-->
<fieldset>
  <legend>Category</legend>
  <asp:FormView ID="fvCategories" runat="server"
    DataSourceID="dsCategory"
    OnItemCreated="fvCategories_ItemCreated">
    <ItemTemplate>...</ItemTemplate>
    <EditItemTemplate>...</EditItemTemplate>
    <InsertItemTemplate>...</InsertItemTemplate>
  </asp:FormView>
</fieldset>

```

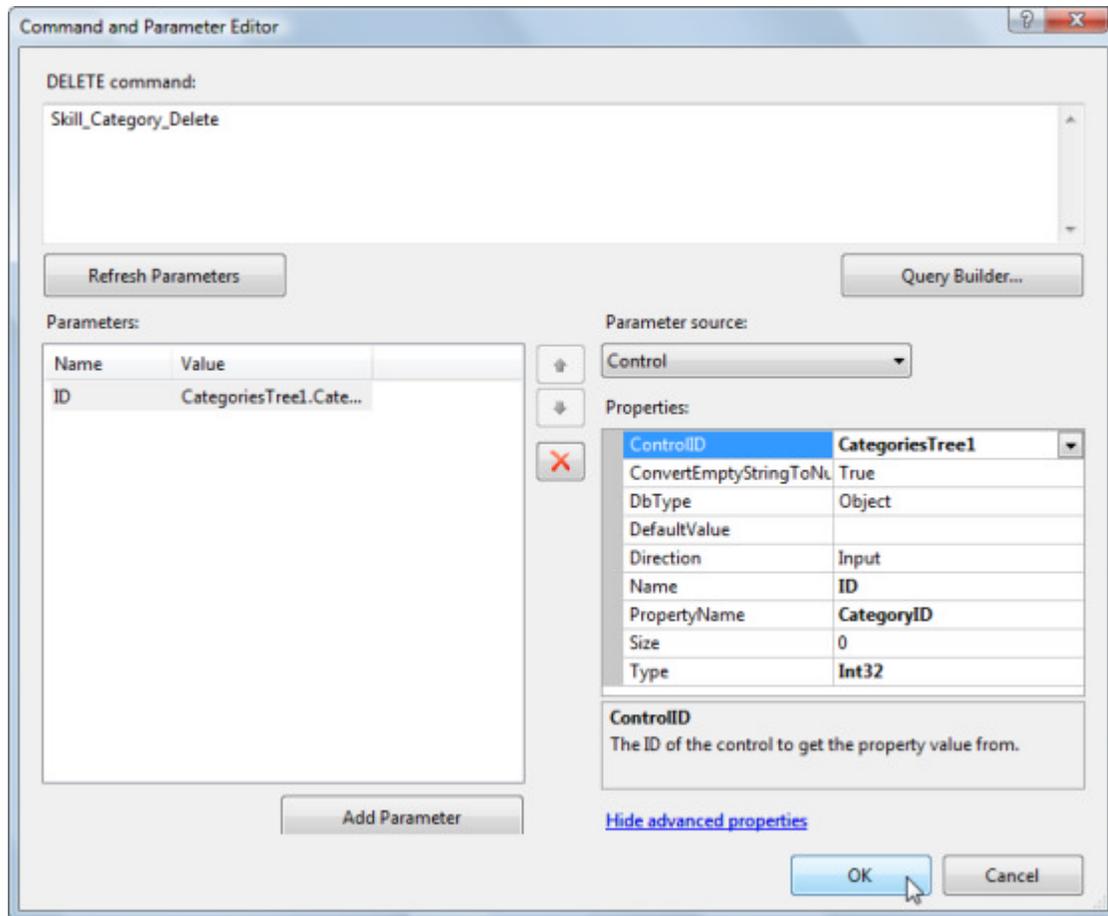
Adding the DataSource

This set of steps configures the "dsCategory" data. The data source will be used for all CRUD operations on the Category table. There are a set of stored procedures already defined when you ran the database scripts that will now be hooked up to each of the CRUD operations for the data source.

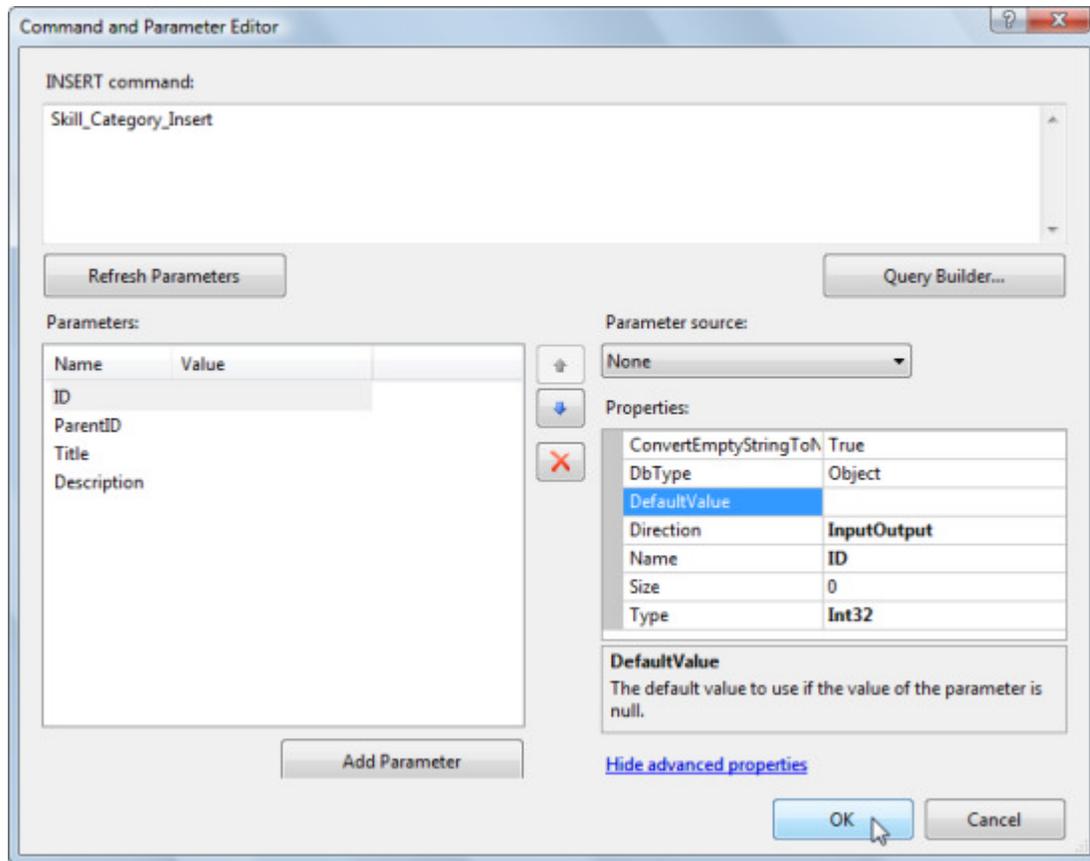
1. Drop a new **SqlDataSource** control on the web page from the ToolBox Data tab and set the ID property to "dsCategory".
2. Using the Smart Tag, select **Configure Data Source...**
3. In the "Choose your Data Connection" page of the Data Source Wizard select "ActiveSkillConnectionString" from the drop down list. Click the **Next** button.
4. In the "Configure the Select" statement page, select the "select a custom SQL statement or stored procedure" radio button option. Click the **Next** button.
5. In the "Define Custom Statements or Stored Procedures" page:
 - o On the Select tab, select the "Stored Procedure" radio button. From the drop down list select "Skill_Category_SelectWhere".



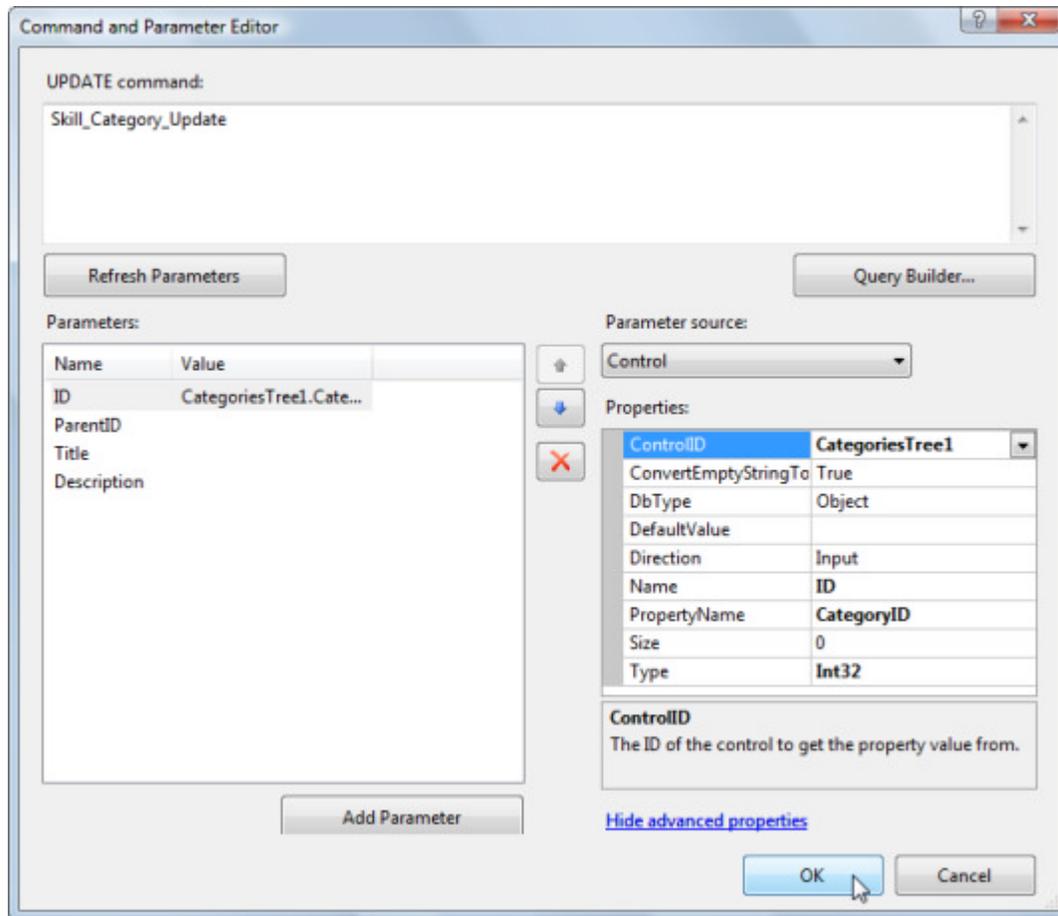
- On the Update tab, select the "Stored Procedure" radio button. From the drop down list select "Skill_Category_Update".
 - On the Insert tab, select the "Stored Procedure" radio button. From the drop down list select "Skill_Category_Insert".
 - On the Delete tab, select the "Stored Procedure" radio button. From the drop down list select "Skill_Category_Delete".
 - Click the **Next** button.
6. Click the **Next** button.
 7. Click the **Finish** button to close the dialog.
 8. In the Properties window for the SqlDataSource, click the **DeleteQuery** property ellipses. *This will again display the Command and Parameter editor for the delete query.*
 9. In the "Parameter source:" drop down list, select "Control".
 - Set the **ControlID** property to "CategoriesTree1".
 - Set the **Name** property to "ID".
 - Set the **PropertyName** to "CategoryId"
 - Set the **Type** to "Int32".



10. Click the **OK** button to close the dialog.
11. In the Properties window for the `SqlDataSource`, click the **InsertQuery** property ellipses. In this case you will have to set properties for more than one parameter and these parameters will be populated from code. Configure each parameter as follows:
 - **ID**: Leave the "Parameter Source" as "None", **Direction** to "InputOutput", **Name** to "ID", **Type** to "Int32".
 - **ParentID**: Leave the "Parameter Source" as "None", **Direction** to "Input", **Name** to "ParentID", **Type** to "Int32".
 - **Title**: Leave the "Parameter Source" as "None", **Direction** to "Input", **Name** to "Title", **Type** to "String".
 - **Description**: Leave the "Parameter Source" as "None", **Direction** to "Input", **Name** to "Description", **Type** to "String".



12. Click the **OK** button to close the dialog.
13. In the Properties window for the SqlDataSource, click the **UpdateQuery** property ellipses. Configure the parameters as follows:
 - **ID:** Set the "Parameter Source" as "Control", **ControlID** as "CategoriesTree1", **Property Name** as "CategoryID", **Direction** to "Input", **Name** to "ID", **Type** to "Int32".
 - **ParentID:** Set the "Parameter Source" as "Control", **ControlID** as "CategoriesTree1", **Property Name** as "ParentCategoryID", **Direction** to "Input", **Name** to "ParentID", **Type** to "Int32".
 - **Title:** Leave the "Parameter Source" as "None", **Direction** to "Input", **Name** to "Title", **Type** to "String".
 - **Description:** Leave the "Parameter Source" as "None", **Direction** to "Input", **Name** to "Description", **Type** to "String".



14. Click the OK button to close the dialog. At this point the markup for "dsCategory" is:

[ASP.NET] The Markup for dsCategory

```
<asp:SqlDataSource ID="dsCategory" runat="server" ConnectionString="<%=
ConnectionStrings:ActiveSkillConnectionString %>"
DeleteCommand="Skill_Category_Delete" InsertCommand="Skill_Category_Insert"
UpdateCommand="Skill_Category_Update"
DeleteCommandType="StoredProcedure" InsertCommandType="StoredProcedure"
UpdateCommandType="StoredProcedure"
CancelSelectOnNullParameter="False" OnInserted="dsCategory_Inserted"
SelectCommand="Skill_Category_SelectWhere" SelectCommandType="StoredProcedure">
<SelectParameters>
  <asp:Parameter Name="ID" Type="Int32" />
</SelectParameters>
<DeleteParameters>
  <asp:ControlParameter ControlID="CategoriesTree1" Name="ID" PropertyName="CategoryID"
    Type="Int32" />
</DeleteParameters>
<UpdateParameters>
  <asp:ControlParameter ControlID="CategoriesTree1" Name="ID"
    PropertyName="CategoryID" Type="Int32" />
  <asp:ControlParameter ControlID="CategoriesTree1" Name="ParentID"
    PropertyName="ParentCategoryID" Type="Int32" />
  <asp:Parameter Name="Title" Type="String" />
</UpdateParameters>
```

```
<asp:Parameter Name="Description" Type="String" />
</UpdateParameters>
<InsertParameters>
  <asp:Parameter Direction="InputOutput" Name="ID" Type="Int32" />
  <asp:Parameter Name="ParentID" Type="Int32" />
  <asp:Parameter Name="Title" Type="String" />
  <asp:Parameter Name="Description" Type="String" />
</InsertParameters>
</asp:SqlDataSource>
```



You can mix and match which parameters are set up declaratively and which are handled at runtime. For example, the `SelectQuery` of this data source runs automatically before a `CategoryID` is available from the tree so it was easier here to handle programmatically in the node click.

Issues with data source controls declared within updating areas via AJAX can be difficult to debug. If necessary, you can get better control and more complete information by setting those parameters to "None" and populating them in code. Make sure the `DataSourceID` property is empty and instead set the `DataSource` property in code. Just before you call `DataBind()` you can review the parameters in the debugger and get a good idea of what is going on.

When in doubt, handle it yourself in code.

Adding the RadToolBar

1. Add a `RadToolBar` to the page and set the `ID` to "tbarCategories". Using the Smart Tag **Build RadToolBar...** option, add three tool bar buttons.
 - Set the `Text` property to "Add Category", `ImageUrl` to "../Skins/ActiveSkill/Grid/AddRecord.gif" and `Enabled` to false.
 - Set the `Text` property to "EditCategory", `ImageUrl` to "../Skins/ActiveSkill/Grid/Update.gif" and `Enabled` to false.
 - Set the `Text` property to "Delete Category", `ImageUrl` to "../Skins/ActiveSkill/Grid/Delete.gif" and `Enabled` to false.
2. You will need to finish the definition of the tool bar within the markup because it requires applying a style and a few binding expressions. First add `Style="float: none"` to the `RadToolBar` element. Then add binding expressions to the `Value` properties of all three buttons:
 - `Value='<%=#INSERT_CATEGORY%>'`
 - `Value='<%=#EDIT_CATEGORY%>'`
 - `Value='<%=#DELETE_CATEGORY%>'`

The finished tag should look like the markup below. Note that we will need to call `Page.DataBind()` for the binding expressions to be evaluated:

[ASP.NET] Defining the RadToolBar

```
<!--toolbar with add, edit, delete buttons-->
<telerik:RadToolBar ID="tbarCategories" runat="server"
  OnButtonClick="tbarCategories_ButtonClick"
  Style="float: none" >
  <items>
    <telerik:RadToolBarButton runat="server" Text="Add Category"
      Value='<%=#INSERT_CATEGORY%>'
      ImageUrl="../Skins/ActiveSkill/Grid/AddRecord.gif" Enabled="False">
    </telerik:RadToolBarButton>
    <telerik:RadToolBarButton runat="server" Text="Edit Category"
```

```

        Value='<#EDIT_CATEGORY%>'
        ImageUrl=" ../Skins/ActiveSkill/Grid/Update.gif" Enabled="False">
    </telerik:RadToolBarButton>
    <telerik:RadToolBarButton runat="server" Text="Delete Category"
        Value='<#DELETE_CATEGORY%>'
        ImageUrl=" ../Skins/ActiveSkill/Grid/Delete.gif" Enabled="False">
    </telerik:RadToolBarButton>
</items>
</telerik:RadToolBar>

```

The CategoriesTree

After the RadToolBar should come the CategoriesTree control you added when testing the control. Verify that the DisplayMode is set to TreeViewOnly. Surround the CategoriesTree control with a div. The div ID should be "CategoriesTreeDiv" and have a style of "float: left". The style keeps the FormView (which comes next) from being bumped off onto the next line.

[ASP.NET] The CategoriesTree

```

<%--treeview with categories--%>
<div id="CategoriesTreeDiv" style="float: left;">
    <uc1:CategoriesTree ID="CategoriesTree1" runat="server"
        DisplayMode="TreeViewOnly" />
</div>

```

Defining the FormView



You can skip past the steps explaining how to work in the FormView here if you just want to paste the completed ASP.NET markup title "Defining the Formview" below.

1. After the CategoriesTree, add a `<fieldset>` from the HTML tab of the toolbox. Double-click on the legend area in the upper left of the fieldset and type "Category".
2. Inside the fieldset add a **FormView** from the Data tab of the ToolBox and set the ID to "fvCategories".
3. From the FormView Smart Tag set the **Choose Data Source** drop down to "dsCategory".
4. Also from the Smart Tag click the **Edit Templates...** link.
5. In the **ItemTemplate** add two **RadTextBox** controls and name them "tbTitle" and "tbDescription". Also set their **Label** properties to "Title:" and "Description" respectively. Select both of these text boxes (you can select one and then with the control key held down, select the second), and from Visual Studio Edit menu select **Copy**.
6. In the **EditItemTemplate**, use the Visual Studio Edit menu to paste the text boxes.
7. Add a **Div** control from the HTML tab of the ToolBox just below the two text boxes.
8. Within the Div add two **ImageButton** controls from the Standard tab of the ToolBox. Set their ID properties to "btnEditCancel" and "btnEditSave" respectively.
 - o Set their **ImageUrl** properties to "../Images/cancel_btn_2.png" and "../Images/update_btn_2.png" respectively.
 - o Double-click both the cancel and update buttons to create **OnClick** event handlers for each.
 - o Copy the two text boxes and the div holding the two buttons onto the clipboard.
9. Using the FormView Smart Tag, navigate to the **InsertItemTemplate** and paste the contents of the clipboard, i.e. the two text boxes, the div and the two buttons.
10. Navigate back to the **ItemTemplate** and set the two RadTextBox **ReadOnly** properties to true and the **Enabled** properties to false.
11. Go to the source for the page to work with the FormView markup and add binding expressions.

- In the **ItemTemplate**, set the **Text** properties for the two RadTextBox controls to '<## Eval("Title") %>' and '<## Eval("Description") %>' respectively. *This will display the Category table Title and Description columns when the FormView is in ReadOnly mode.*
- In the **EditItemTemplate** set the **Text** properties for the two RadTextBox controls to '<## Bind("Title") %>' and '<## Bind("Description") %>' respectively. *This will allow editing of the Category table Title and Description columns when the FormView is in Edit mode.*
- Repeat the step above and add binding expressions for the **InsertItemTemplate**.

The completed markup should look like the example below:

[ASP.NET] Defining the FormView

```

<!--fieldset containing formview with editable area and buttons-->
<fieldset>
  <legend>Category</legend>
  <asp:FormView ID="fvCategories" runat="server"
    DataSourceID="dsCategory"
    OnItemCreated="fvCategories_ItemCreated">
    <ItemTemplate>
      <div>
        <telerik:RadTextBox ID="tbTitle" runat="server"
          Label="Title:"
          ReadOnly="true"
          Text='<%=# Eval("Title") %>'
          Enabled="False">
        </telerik:RadTextBox>
      </div>
      <div>
        <telerik:RadTextBox ID="tbDescription" runat="server"
          Label="Description:"
          ReadOnly="true"
          Text='<%=# Eval("Description") %>'
          Enabled="False">
        </telerik:RadTextBox>
      </div>
    </ItemTemplate>
    <EditItemTemplate>
      <div>
        <telerik:RadTextBox ID="tbTitle" runat="server"
          Label="Title:"
          Text='<%=# Bind("Title") %>'
        </telerik:RadTextBox>
      </div>
      <div>
        <telerik:RadTextBox ID="tbDescription" runat="server"
          Label="Description:"
          Text='<%=# Bind("Description") %>'
        </telerik:RadTextBox>
      </div>
      <div id="divButtons">
        <asp:ImageButton ID="btnEditCancel" runat="server"
          ImageUrl="~/Images/cancel_btn_2.png"
          OnClick="btnCancel_Click" />
        <asp:ImageButton ID="btnEditSave" runat="server"
          ImageUrl="~/Images/update_btn_2.png"
          OnClick="btnSaveEdit_Click" Height="23px" Width="91px" />
      </div>
    </EditItemTemplate>
    <InsertItemTemplate>
      <div>
        <telerik:RadTextBox ID="tbTitle" runat="server"
          Label="Title:"
          Text='<%=# Bind("Title") %>'
        </telerik:RadTextBox>
      </div>
      <div>
        <telerik:RadTextBox ID="tbDescription" runat="server"
          Label="Description:"

```

```
        Label="Description:"
        Text='<%# Bind("Description") %>'>
    </telerik:RadTextBox>
</div>
<div id="divButtons">
    <asp:ImageButton ID="btnInsertCancel" runat="server"
        ImageUrl="~/Images/cancel_btn_2.png"
        OnClick="btnCancel_Click" />
    <asp:ImageButton ID="btnInsertSave" runat="server"
        ImageUrl="~/Images/update_btn_2.png"
        OnClick="btnSaveInsert_Click" />
</div>
</InsertItemTemplate>
</asp:FormView>
</fieldset>
```

Declare Constants and Properties

In the code-behind for Categories.ascx add the constants and properties shown below:

The constants are used in binding expressions in the markup for the RadToolBar and the matching constants are used to evaluate which tool bar button was clicked. The CategoryTitle and CategoryDescription properties are shortcuts to finding the corresponding RadTextBox controls within the FormView.

[VB] Defining Constants and Properties

```
#region constants
Public Const INSERT_CATEGORY As String = "INSERT_CATEGORY"
Public Const EDIT_CATEGORY As String = "EDIT_CATEGORY"
Public Const DELETE_CATEGORY As String = "DELETE_CATEGORY"
#End Region
#region properties
Private ReadOnly Property CategoryTitle() As RadTextBox
    Get
        Return TryCast(fvCategories.FindControl("tbTitle"), RadTextBox)
    End Get
End Property
Private ReadOnly Property CategoryDescription() As RadTextBox
    Get
        Return TryCast(fvCategories.FindControl("tbDescription"), RadTextBox)
    End Get
End Property
#End Region properties
```

[C#] Defining Constants and Properties

```
#region constants
public const string INSERT_CATEGORY = "INSERT_CATEGORY";
public const string EDIT_CATEGORY = "EDIT_CATEGORY";
public const string DELETE_CATEGORY = "DELETE_CATEGORY";
#endregion
#region properties
private RadTextBox CategoryTitle
{
    get { return fvCategories.FindControl("tbTitle") as RadTextBox; }
}
private RadTextBox CategoryDescription
{
```

```

    get { return fvCategories.FindControl("tbDescription") as RadTextBox; }
}
#endregion properties

```

Add Private Methods

Add a private method to set the toolbar button state based on the selected node.

Usability for the toolbar works like this: Insert and Delete are always allowed if some node is selected. Edit is allowed if a node is selected and it's not the root node. The root node acts simply as a heading "Categories", much like the "MailBox" node in Outlook. You can add and delete nodes under "MailBox" but not the "MailBox" node itself.

[VB] Private Methods

```

#region private methods
' Determine if there is a selected node and if
' the node is the root node. Enable the tool bar
' buttons accordingly.
Private Sub UpdateButtonState()
    Dim nodeIsSelected As Boolean = CategoriesTree1.TreeView.SelectedNode <> Nothing
    tbarCategories.Items.FindItemByValue(INSERT_CATEGORY).Enabled = nodeIsSelected
    tbarCategories.Items.FindItemByValue(DELETE_CATEGORY).Enabled = nodeIsSelected And Not
CategoriesTree1.IsRootSelected
    tbarCategories.Items.FindItemByValue(EDIT_CATEGORY).Enabled = nodeIsSelected And Not
CategoriesTree1.IsRootSelected
End Sub
#End Region

```

[VB] Private Methods

```

#region private methods
// Determine if there is a selected node and if
// the node is the root node. Enable the tool bar
// buttons accordingly.
private void UpdateButtonState()
{
    bool nodeIsSelected = CategoriesTree1.TreeView.SelectedNode != null;
    tbarCategories.Items.FindItemByValue(INSERT_CATEGORY).Enabled =
        nodeIsSelected;
    tbarCategories.Items.FindItemByValue(DELETE_CATEGORY).Enabled =
        nodeIsSelected & !CategoriesTree1.IsRootSelected;
    tbarCategories.Items.FindItemByValue(EDIT_CATEGORY).Enabled =
        nodeIsSelected & !CategoriesTree1.IsRootSelected;
}
#endregion

```

Handle DataSource CRUD Operations

The CategoriesTree is bound to data when it first loads but does not do a full refresh after that. Instead, we maintain the database records and the node properties in parallel. The database operations are handled through the data source methods Delete(), Insert() and Update(). Add the two methods below that wrap the Update and Insert operations:

Most of the parameters are handled declaratively. The controls within the FormView are a little harder to get at so we load them here just before making the Update() or Insert() method calls explicitly.

[VB] DataSource CRUD Methods

```

#region Datasource CRUD operations

```

```
Private Sub UpdateDB()  
    dsCategory.UpdateParameters("Title").DefaultValue = Me.CategoryTitle.Text  
    dsCategory.UpdateParameters("Description").DefaultValue = Me.CategoryDescription.Text  
    dsCategory.Update()  
End Sub  
Private Sub InsertDB()  
    dsCategory.InsertParameters("ParentID").DefaultValue = CategoriesTree1.ParentCategoryID  
    dsCategory.InsertParameters("Title").DefaultValue = Me.CategoryTitle.Text  
    dsCategory.InsertParameters("Description").DefaultValue = Me.CategoryDescription.Text  
    dsCategory.Insert()  
End Sub  
#End Region CRUD operations  
[VB] DataSource CRUD Methods  
#region Datasource CRUD operations  
private void UpdateDB()  
{  
    dsCategory.UpdateParameters["Title"].DefaultValue = this.CategoryTitle.Text;  
    dsCategory.UpdateParameters["Description"].DefaultValue = this.CategoryDescription.Text;  
    dsCategory.Update();  
}  
private void InsertDB()  
{  
    dsCategory.InsertParameters["ParentID"].DefaultValue = CategoriesTree1.ParentCategoryID;  
    dsCategory.InsertParameters["Title"].DefaultValue = this.CategoryTitle.Text;  
    dsCategory.InsertParameters["Description"].DefaultValue = this.CategoryDescription.Text;  
    dsCategory.Insert();  
}  
#endregion CRUD operations
```

Handle Page Events

Add the page event handlers below to the code-behind:

- **Page_Load:** The NodeClick handler for the CategoriesTree needs to be re-added on every postback.
- **Cancel and Update Click Events:** These events occur in response to the admin clicking on cancel or update buttons within the FormView. Canceling works the same for both Insert and Edits. The FormView mode is changed back to ReadOnly and the toolbar button state is refreshed to reflect this. The save of an insert causes a new node to be created using the Title and Description entry from the FormView. The record is inserted and the FormView is returned to a read-only state. Saving an edit is quite similar to insert except the corresponding update methods of the CategoriesTree and DataSource are called instead.
- **Inserted Event Handler:** When the category is inserted to the database we need to find out what the new generated ID is and get that back to the tree node Value property. You can retrieve the generated value of an InputOutput parameter using the SqlDataSource **OnInserted** event. OnInserted has a **SqlDataSourceStatusEventArgs** argument that lets you access the parameters of the stored procedure responsible for inserting the category.
- **ToolBar Button Click Event Handler:** When a button on the toolbar is clicked, this event handler retrieves the **Value** property that contains the operation to perform. The **Value** property is populated by a binding expression that uses the same three constants used in this event handler. If Insert or Edit buttons were clicked, the FormView mode is changed so that the admin can enter values. If the Delete button is clicked, the delete happens immediately.
- **FormView OnItemCreated Event Handler:** As items in the FormView are created, this event handler attaches attributes so that the image buttons can respond to onmouseover and onmouseout client events. When the user hovers over a button, the button responds visually.

[VB] Handling Page Events

```

#region page events
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Add this event handler every time the page loads
    AddHandler CategoriesTree1.NodeClick, AddressOf CategoriesTree1_NodeClick
End Sub
' An insert or edit has been canceled so
' change the form view mode to read-only
' and update the tool bar buttons to reflect this
Public Sub btnCancel_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs)
    fvCategories.ChangeMode(FormViewMode.[ReadOnly])
    UpdateButtonState()
End Sub
' An edit is being saved so copy the title and description
' to the node text and tooltip, then update the database.
' change the form view mode back to read-only
' and update the tool bar buttons to reflect this
Public Sub btnSaveEdit_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs)
    CategoriesTree1.UpdateCategoryNode(Me.CategoryTitle.Text, Me.CategoryDescription.Text)
    Me.UpdateDB()
    fvCategories.ChangeMode(FormViewMode.[ReadOnly])
    UpdateButtonState()
End Sub
' An insert is being saved so copy the title and description
' to the node text and tooltip, then insert to the database.
' change the form view mode back to read-only
' and update the tool bar buttons to reflect this
Public Sub btnSaveInsert_Click(ByVal sender As Object, ByVal e As
System.Web.UI.ImageClickEventArgs)
    CategoriesTree1.InsertCategoryNode(Me.CategoryTitle.Text, Me.CategoryDescription.Text)
    Me.InsertDB()
    fvCategories.ChangeMode(FormViewMode.[ReadOnly])
    UpdateButtonState()
End Sub
' A category has just been inserted to the database
' and the newly generated ID is being returned in this event
' as an argument. We do this to keep the new database record
' for the category in sync with the CategoryID in the CategoriesTree
' control
Protected Sub dsCategory_Inserted(ByVal sender As Object, ByVal e As
SqlDataSourceStatusEventArgs)
    CategoriesTree1.CategoryID = e.Command.Parameters("@ID").Value.ToString()
End Sub
' In response to clicking the tool bar, get the
' tool bar button Value (which has these same
' three constants injected using binding expressions)
' and perform the appropriate operation.
' Insert and Edit simply change the FormView mode
' so the user can enter values while Delete
' happens immediately.
Protected Sub tbarCategories_ButtonClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadToolBarEventArgs)
    Select Case e.Item.Value

```

```
Case INSERT_CATEGORY
    fvCategories.ChangeMode(FormViewMode.Insert)
Exit Select
Case EDIT_CATEGORY
    fvCategories.ChangeMode(FormViewMode.Edit)
Exit Select
Case DELETE_CATEGORY
    ' delete from the db first (the datasource
    ' depends on treeview values), then delete node
    dsCategory.Delete()
    CategoriesTree1.DeleteCategoryNode()
    fvCategories.ChangeMode(FormViewMode.[ReadOnly])
    UpdateButtonState()
Exit Select
End Select
End Sub
' As FormView items are created, attach attributes for
' onmouseover and onmouseout client events so that the
' buttons respond visually.
Protected Sub fvCategories_ItemCreated(ByVal sender As Object, ByVal e As EventArgs)
    If fvCategories.CurrentMode = FormViewMode.Edit Then
        Dim btnEditCancel As ImageButton = TryCast(fvCategories.FindControl("btnEditCancel"),
ImageButton)
        Dim btnEditSave As ImageButton = TryCast(fvCategories.FindControl("btnEditSave"),
ImageButton)
        btnEditCancel.Attributes.Add("onmouseover", "this.src='../Images/cancel_btn_1.png'")
        btnEditCancel.Attributes.Add("onmouseout", "this.src='../Images/cancel_btn_2.png'")
        btnEditSave.Attributes.Add("onmouseover", "this.src='../Images/update_btn_1.png'")
        btnEditSave.Attributes.Add("onmouseout", "this.src='../Images/update_btn_2.png'")
    End If
    If fvCategories.CurrentMode = FormViewMode.Insert Then
        Dim btnInsertCancel As ImageButton = TryCast(fvCategories.FindControl("btnInsertCancel"),
ImageButton)
        Dim btnInsertSave As ImageButton = TryCast(fvCategories.FindControl("btnInsertSave"),
ImageButton)
        btnInsertCancel.Attributes.Add("onmouseover", "this.src='../Images/cancel_btn_1.png'")
        btnInsertCancel.Attributes.Add("onmouseout", "this.src='../Images/cancel_btn_2.png'")
        btnInsertSave.Attributes.Add("onmouseover", "this.src='../Images/update_btn_1.png'")
        btnInsertSave.Attributes.Add("onmouseout", "this.src='../Images/update_btn_2.png'")
    End If
End Sub
#End Region
```

[C#] Handling Page Events

```
#region page events
protected void Page_Load(object sender, EventArgs e)
{
    // Add this event handler every time the page loads
    CategoriesTree1.NodeClick +=
        new RadTreeViewEventHandler(CategoriesTree1_NodeClick);
}
// An insert or edit has been canceled so
// change the form view mode to read-only
// and update the tool bar buttons to reflect this
```

```

public void btnCancel_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
    fvCategories.ChangeMode(FormViewMode.ReadOnly);
    UpdateButtonState();
}
// An edit is being saved so copy the title and description
// to the node text and tooltip, then update the database.
// change the form view mode back to read-only
// and update the tool bar buttons to reflect this
public void btnSaveEdit_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
    CategoriesTree1.UpdateCategoryNode(this.CategoryTitle.Text,
        this.CategoryDescription.Text);
    this.UpdateDB();
    fvCategories.ChangeMode(FormViewMode.ReadOnly);
    UpdateButtonState();
}
// An insert is being saved so copy the title and description
// to the node text and tooltip, then insert to the database.
// change the form view mode back to read-only
// and update the tool bar buttons to reflect this
public void btnSaveInsert_Click(object sender, System.Web.UI.ImageClickEventArgs e)
{
    CategoriesTree1.InsertCategoryNode(this.CategoryTitle.Text,
        this.CategoryDescription.Text);
    this.InsertDB();
    fvCategories.ChangeMode(FormViewMode.ReadOnly);
    UpdateButtonState();
}
// A category has just been inserted to the database
// and the newly generated ID is being returned in this event
// as an argument. We do this to keep the new database record
// for the category in sync with the CategoryID in the CategoriesTree
// control
protected void dsCategory_Inserted(object sender,
    SqlDataSourceStatusEventArgs e)
{
    CategoriesTree1.CategoryID =
        e.Command.Parameters["@ID"].Value.ToString();
}
// In response to clicking the tool bar, get the
// tool bar button Value (which has these same
// three constants injected using binding expressions)
// and perform the appropriate operation.
// Insert and Edit simply change the FormView mode
// so the user can enter values while Delete
// happens immediately.
protected void tbarCategories_ButtonClick(object sender,
    Telerik.Web.UI.RadToolBarEventArgs e)
{
    switch (e.Item.Value)
    {
        case INSERT_CATEGORY:
            fvCategories.ChangeMode(FormViewMode.Insert);
            break;
    }
}

```

```
case EDIT_CATEGORY:
    fvCategories.ChangeMode(FormViewMode.Edit);
    break;
case DELETE_CATEGORY:
    // delete from the db first (the datasource
    // depends on treeview values), then delete node
    dsCategory.Delete();
    CategoriesTree1.DeleteCategoryNode();
    fvCategories.ChangeMode(FormViewMode.ReadOnly);
    UpdateButtonState();
    break;
}
}
// As FormView items are created, attach attributes for
// onmouseover and onmouseout client events so that the
// buttons respond visually.
protected void fvCategories_ItemCreated(object sender, EventArgs e)
{
    if (fvCategories.CurrentMode == FormViewMode.Edit)
    {
        ImageButton btnEditCancel = fvCategories.FindControl("btnEditCancel") as ImageButton;
        ImageButton btnEditSave = fvCategories.FindControl("btnEditSave") as ImageButton;
        btnEditCancel.Attributes.Add("onmouseover", "this.src='../Images/cancel_btn_1.png'");
        btnEditCancel.Attributes.Add("onmouseout", "this.src='../Images/cancel_btn_2.png'");
        btnEditSave.Attributes.Add("onmouseover", "this.src='../Images/update_btn_1.png'");
        btnEditSave.Attributes.Add("onmouseout", "this.src='../Images/update_btn_2.png'");
    }
    if (fvCategories.CurrentMode == FormViewMode.Insert)
    {
        ImageButton btnInsertCancel = fvCategories.FindControl("btnInsertCancel") as ImageButton;
        ImageButton btnInsertSave = fvCategories.FindControl("btnInsertSave") as ImageButton;
        btnInsertCancel.Attributes.Add("onmouseover", "this.src='../Images/cancel_btn_1.png'");
        btnInsertCancel.Attributes.Add("onmouseout", "this.src='../Images/cancel_btn_2.png'");
        btnInsertSave.Attributes.Add("onmouseover", "this.src='../Images/update_btn_1.png'");
        btnInsertSave.Attributes.Add("onmouseout", "this.src='../Images/update_btn_2.png'");
    }
}
}
#endregion
```

Handle CategoriesTreeView NodeClick Event

Add the CategoriesTree NodeClick event handler.

First load up the data source select parameter so that as the admin clicks on nodes, the FormView re-bind has the correct CategoryID to work with. If the user is already editing a category and clicks on another node, the edit is abandoned and the FormView is returned to ReadOnly mode. The FormView is rebound to the new category and the tool bar button state is refreshed.

[VB] Handling the CategoriesTree NodeClick

```
Sub CategoriesTree1_NodeClick(ByVal sender As Object, ByVal e As RadTreeNodeEventArgs)
    ' setup the select using the currently selected node value
    dsCategory.SelectParameters("ID").DefaultValue = CategoriesTree1.CategoryID
    ' If the user is in the middle of the edit,
    ' cancel the edit by changing to readonly
    fvCategories.ChangeMode(FormViewMode.[ReadOnly])
    fvCategories.DataBind()
```

```
UpdateButtonState()
End Sub
```

[C#] Handling the CategoriesTree NodeClick

```
void CategoriesTree1_NodeClick(object sender, RadTreeNodeEventArgs e)
{
    // setup the select using the currently selected node value
    dsCategory.SelectParameters["ID"].DefaultValue = CategoriesTree1.CategoryID;
    // If the user is in the middle of the edit,
    // cancel the edit by changing to readonly
    fvCategories.ChangeMode(FormViewMode.ReadOnly);
    fvCategories.DataBind();
    UpdateButtonState();
}
```

Finish Implementing FirstLoad()

When you first tested the CategoriesTree, you simply called the CategoriesTree InitialLoad() method and passed the "dsAllCategories" data source. You'll keep that code but add:

- this.DataBind() to bind the page. This will allow the binding expression in the RadToolBar to occur.
- Perform the Initial bind of the FormView.
- Update the button state to agree with the editing mode.

Update your FirstLoad() implementation with the code below:

[VB] Implementing FirstLoad()

```
#region IASControl Members
Public Sub FirstLoad(ByVal args As System.Collections.Generic.Dictionary(Of String, String))
    ' bind the page to emit binding expressions into the markup
    ' bind here instead of page_load because categoriestreeview is counting on
    ' the values already being populated by the call to InitialLoad()
    Me.DataBind()
    CategoriesTree1.InitialLoad(dsAllCategories)
    fvCategories.DataBind()
    UpdateButtonState()
End Sub
#End Region
```

[C#] Implementing FirstLoad()

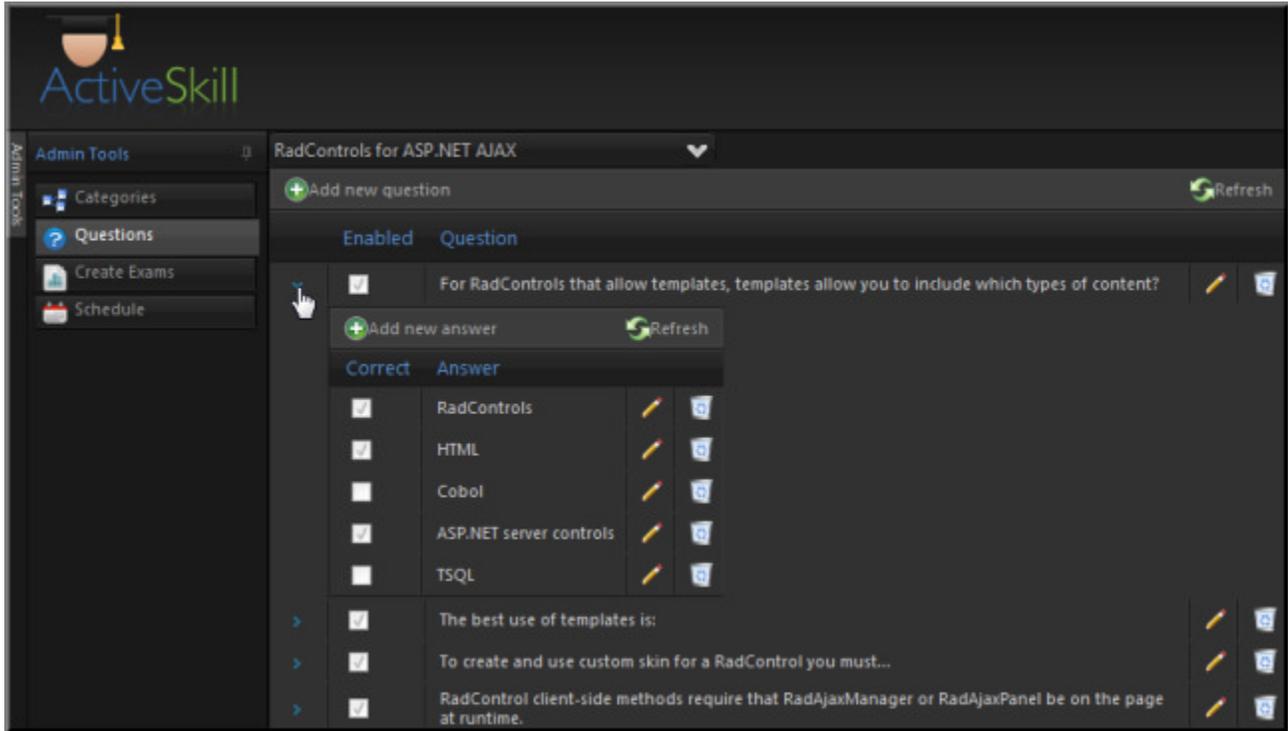
```
#region IASControl Members
public void FirstLoad(System.Collections.Generic.Dictionary<string, string> args)
{
    // bind the page to emit binding expressions into the markup
    // bind here instead of page_load because categoriestreeview is counting on
    // the values already being populated by the call to InitialLoad()
    this.DataBind();
    CategoriesTree1.InitialLoad(dsAllCategories);
    fvCategories.DataBind();
    UpdateButtonState();
}
#endregion
```

38.5 Implement Questions Control

We can re-use our CategoryTree control when filtering questions for a particular category. The Questions.ascx

UI for ASP.NET AJAX

control will display the list of categories as a drop down list at the top of the screen and a master-detail grid below it containing questions and responses.



You can find the complete source for this project at:
\\VS Projects\ActiveSkill Database Maintenance\Questions

Prepare Layout

Roadmap

The overall layout of the markup for this user control looks something like the condensed view shown in the screenshot below. The user control has three data sources: the "all categories" data source that supplies the CategoriesTree, dsCategoryQuestions that feeds the MasterTableView of the grid and dsResponse that feeds the detail table of the grid.

```
<%--Data Sources--%>
<asp:SqlDataSource ID="dsAllCategories" ...>...</asp:SqlDataSource>
<asp:SqlDataSource ID="dsCategoryQuestions" ...>...</asp:SqlDataSource>
<asp:SqlDataSource ID="dsResponse" ...>...</asp:SqlDataSource>

<%--Categories Tree View--%>
<uc1:CategoriesTree ID="CategoriesTree1" runat="server"
    DisplayMode="DropDown" />

<%--Questions Grid --%>
<telerik:RadGrid ID="gridQuestions" ...>...</telerik:RadGrid>
```

Add Data Sources

Add data source controls for the Categories tree, the questions grid and the responses detail portion of the grid. You have already walked through how to configure a SqlDataSource by hand so you can add this directly into the markup.

- **dsAllCategories:** supplies all categories to the CategoriesTree control.
- **dsCategoryQuestions:** supplies questions for the category selected in the CategoriesTree to a RadGrid. The questions are displayed in the MasterTableView. The data source provides add, update and delete functionality for the grid. The category ID is supplied by the CategoryTree and the other parameters are assigned programmatically.
- **dsResponse:** This data source returns a list of responses for a given question and can add, update and delete responses.

[ASP.NET] Adding the DataSource Controls

```
<%--Data Sources--%>
<asp:SqlDataSource ID="dsAllCategories" runat="server"
  ConnectionString="<%$ ConnectionStrings:ActiveSkillConnectionString %>"
  SelectCommand="SELECT [ID], [ParentID], [Title], [Description] FROM [Category]">
</asp:SqlDataSource>
<asp:SqlDataSource ID="dsCategoryQuestions" runat="server"
  ConnectionString="<%$ ConnectionStrings:ActiveSkillConnectionString %>"
  SelectCommand="Skill_QuestionsByCategory_Select"
  SelectCommandType="StoredProcedure" DeleteCommand="Skill_Question_Delete"
  DeleteCommandType="StoredProcedure" InsertCommand="Skill_Question_Insert"
  InsertCommandType="StoredProcedure" UpdateCommand="Skill_Question_Update"
  UpdateCommandType="StoredProcedure">
  <SelectParameters>
    <asp:Parameter Name="CategoryID" Type="Int32" />
  </SelectParameters>
  <DeleteParameters>
    <asp:ControlParameter ControlID="CategoriesTree1" Name="ID"
      PropertyName="CategoryID" Type="Int32" />
  </DeleteParameters>
  <UpdateParameters>
    <asp:Parameter Name="HTML" Type="String" />
    <asp:Parameter Name="Enabled" Type="Boolean" />
    <asp:ControlParameter ControlID="CategoriesTree1" Name="CategoryID"
      PropertyName="CategoryID" Type="Int32" />
    <asp:Parameter Name="ID" Type="Int32" />
  </UpdateParameters>
  <InsertParameters>
    <asp:Parameter Direction="InputOutput" Name="ID" Type="Int32" />
    <asp:ControlParameter ControlID="CategoriesTree1" Name="CategoryID"
      PropertyName="CategoryID" Type="Int32" />
    <asp:Parameter Name="HTML" Type="String" />
    <asp:Parameter Name="Enabled" Type="Boolean" />
  </InsertParameters>
</asp:SqlDataSource>

<asp:SqlDataSource ID="dsResponse" runat="server"
  ConnectionString="<%$ ConnectionStrings:ActiveSkillConnectionString %>"
  DeleteCommand="Skill_Response_Delete" DeleteCommandType="StoredProcedure"
  InsertCommand="Skill_Response_Insert" InsertCommandType="StoredProcedure">
```

```
SelectCommand="Skill_ResponsesByQuestion_SelectWhere"
SelectCommandType="StoredProcedure" UpdateCommand="Skill_Response_Update"
UpdateCommandType="StoredProcedure">
<SelectParameters>
  <asp:Parameter Name="QuestionID" Type="Int32" />
</SelectParameters>
<DeleteParameters>
  <asp:Parameter Name="ID" Type="Int32" />
</DeleteParameters>
<UpdateParameters>
  <asp:Parameter Name="ID" Type="Int32" />
  <asp:Parameter Name="QuestionID" Type="Int32" />
  <asp:Parameter Name="Correct" Type="Boolean" />
  <asp:Parameter Name="HTML" Type="String" />
</UpdateParameters>
<InsertParameters>
  <asp:Parameter Direction="Output" Name="ID" Type="Int32" />
  <asp:Parameter Name="QuestionID" Type="Int32" />
  <asp:Parameter Name="Correct" Type="Boolean" />
  <asp:Parameter Name="HTML" Type="String" />
</InsertParameters>
</asp:SqlDataSource>
```

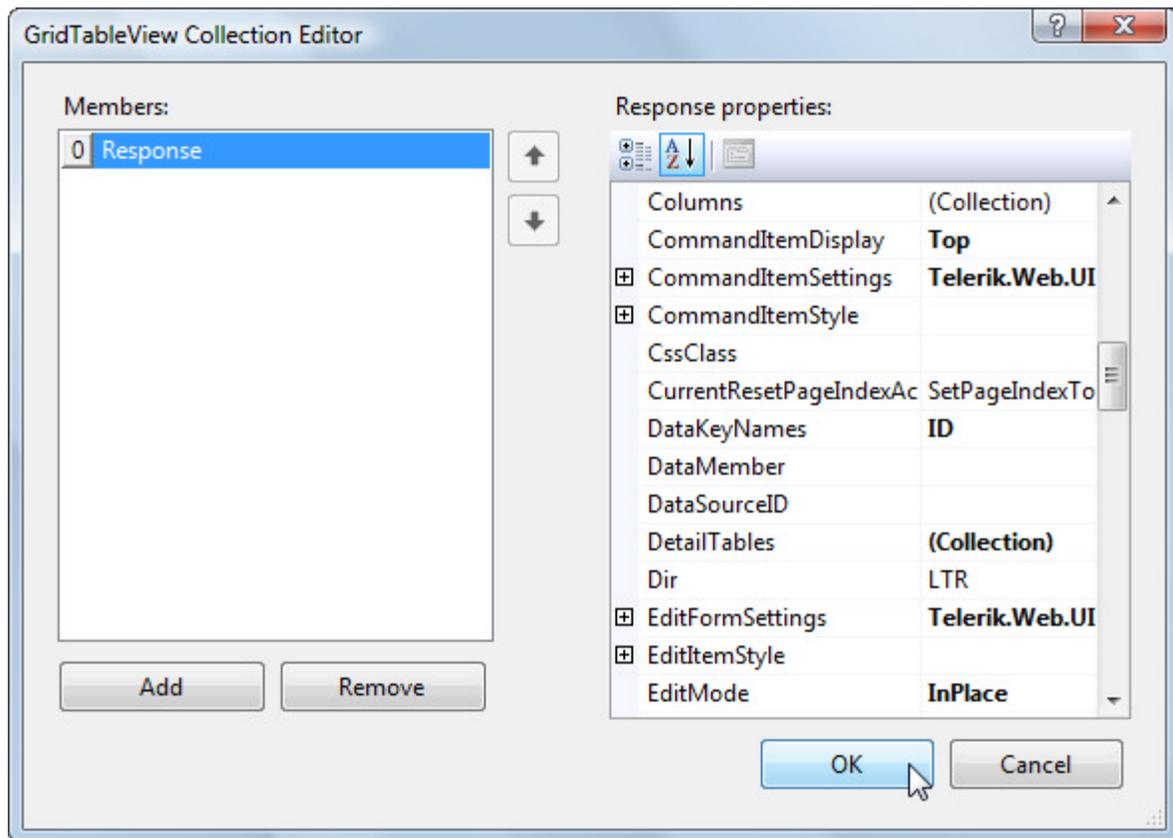
Add Categories Tree

Add a CategoriesTree user control from the Solution Explorer to the web page. Set the **DisplayMode** to "DropDown".

Add and Configure RadGrid

1. Below the CategoriesTree add a RadGrid control. In the Properties window set these properties for the grid:
 - **ID:** "gridQuestions".
 - **AllowAutomaticDeletes:** true.
 - **AutoGenerateColumns:** false.
 - **GridLines:** None.
 - **Skin:** leave this blank.
2. Open up the MasterTableView property and set the following sub properties:
 - **CommandItemDisplay:** Top
 - **DataKeyNames:** "ID"
 - **EditMode:** InPlace
 - **Name:** "Question" (*note: this will be used to identify the table view in some of the command events*)
 - **NoMasterRecordsText:** "No questions to display for this category"
3. Also in **MasterTableView**, open up the **DetailTables** sub-property, add a detail table and set the following properties to the new detail table:
 - **CommandItemDisplay:** Top
 - **DataKeyNames:** "ID"
 - **EditMode:** InPlace
 - **GridLines:** None

- Name: "Response"
- NoDetailRecordsText: "No answers to display."



4. In the markup, add the columns below to the MasterTableView <Columns>. The markup can be added just inside the closing </MasterTableView> tag.

You can also build the column list from the Property Builder or Columns collection editor. At the time of this writing, the GridHTMLColumn type was not available from these two options, so it will be more straightforward to define all columns at once here.

 The GridHTMLColumn is designed with a limited number of tools due to performance reasons so that instances rendered for each edited row are loaded faster. In most cases you don't need the entire set of RadEditor tools when using it as column editor in RadGrid. If you do require a larger toolset, embed a RadEditor using a template column. See this article for more on **using a RadEditor in a template column** (<http://www.telerik.com/support/kb/article/b454K-tae-b454T-cbb-b454c-cbb.aspx>).

There are a few things to notice about the columns defined below. The Visible property for key column "ID" is false. The "HTML" column uses a GridHTMLColumn type. The columns that have images have image URLs that point to \Skins\ActiveSkill\Grid directory.

[ASP.NET] Adding Columns

```
<Columns>
<telerik:GridBoundColumn DataField="ID" DataType="System.Int32"
  HeaderText="ID" SortExpression="ID" UniqueName="ID"
  Visible="False">
</telerik:GridBoundColumn>
```

```
<telerik:GridCheckBoxColumn DataField="Enabled" DataType="System.Boolean"
  HeaderText="Enabled" SortExpression="Enabled" UniqueName="Enabled">
</telerik:GridCheckBoxColumn>
<telerik:GridHTMLEditorColumn DataField="HTML" HeaderText="Question"
  SortExpression="HTML" UniqueName="HTML">
</telerik:GridHTMLEditorColumn>
<telerik:GridEditCommandColumn ButtonType="ImageButton"
  EditImageUrl=" ../Skins/ActiveSkill/Grid/Edit.gif"
  InsertImageUrl=" ../Skins/ActiveSkill/Grid/Update.gif"
  UpdateImageUrl=" ../Skins/ActiveSkill/Grid/Update.gif"
  CancelImageUrl=" ../Skins/ActiveSkill/Grid/Cancel.gif"
  >
</telerik:GridEditCommandColumn>
<telerik:GridButtonColumn ButtonType="ImageButton"
  CommandName="Delete" ImageUrl=" ../Skins/ActiveSkill/Grid/Delete.gif"
  Text="Delete" UniqueName="column">
</telerik:GridButtonColumn>
</Columns>
```

5. Add the detail columns for the question responses inside the tag `<telerik:GridTableView Name="Response"...`:

[ASP.NET] Add the Detail Columns

```
<Columns>
<telerik:GridBoundColumn DataField="ID" UniqueName="ResponseID"
  Visible="False">
</telerik:GridBoundColumn>
<telerik:GridBoundColumn DataField="QuestionID" UniqueName="QuestionID"
  Visible="False">
</telerik:GridBoundColumn>
<telerik:GridCheckBoxColumn DataField="Correct" UniqueName="Correct"
  DataType="System.Boolean" HeaderText="Correct">
</telerik:GridCheckBoxColumn>
<telerik:GridHTMLEditorColumn DataField="HTML" HeaderText="Answer"
  SortExpression="HTML" UniqueName="Answer">
</telerik:GridHTMLEditorColumn>
<telerik:GridEditCommandColumn ButtonType="ImageButton"
  EditImageUrl=" ../Skins/ActiveSkill/Grid/Edit.gif"
  InsertImageUrl=" ../Skins/ActiveSkill/Grid/Update.gif"
  UpdateImageUrl=" ../Skins/ActiveSkill/Grid/Update.gif"
  CancelImageUrl=" ../Skins/ActiveSkill/Grid/Cancel.gif"
  >
</telerik:GridEditCommandColumn>
<telerik:GridButtonColumn ButtonType="ImageButton"
  CommandName="Delete" ImageUrl=" ../Skins/ActiveSkill/Grid/Delete.gif"
  Text="Delete" UniqueName="column">
</telerik:GridButtonColumn>
</Columns>
```

6. Using the Properties window Events button () , add handlers for the following events:
 - o DeleteCommand
 - o InsertCommand
 - o UpdateCommand

- DataTableDataBind
- ItemCreated
- NeedDataSource

In the later sections we will implement these event handlers.

Handle Page Events

Add the code below to the Page_Load event handler. *Attach the CategoriesTree NodeClick every time the page loads.*

[VB] Handling the Page_Load Event

```
#region page events
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' reattach the click event handler every page load
    AddHandler CategoriesTree1.NodeClick, AddressOf CategoriesTree1_NodeClick
End Sub
#End Region
```

[C#] Handling the Page_Load Event

```
#region page events
protected void Page_Load(object sender, EventArgs e)
{
    // reattach the click event handler every page load
    CategoriesTree1.NodeClick +=
        new RadTreeViewEventHandler(CategoriesTree1_NodeClick);
}
#endregion
```

Handle Grid Events

Add the code below to the grid event handlers.

gridQuestions_NeedDataSource: *Using the IsFromDetailTable property of the args passed in, assign just the master table view data source.*

gridQuestions_DetailTableDataBind: *This event fires when the user expands one of the questions to reveal the responses. Take this opportunity to retrieve the primary key ("QuestionID") from the parent item passed in ("e.DetailTableView.ParentItem"), assign it to the data source for "responses" and finally assign the data source to detail table view.*

gridQuestions_ItemCreated: *The "Enabled" checkbox for each question should be checked by default. Do this by verifying that the ItemCreated event is firing for a "Questions" row, locate the checkbox in the passed in "e.item" and set the Checked value to true.*

[VB] Handling Grid Events

```
#region grid events
Protected Sub gridQuestions_NeedDataSource(ByVal source As Object, ByVal e As
GridNeedDataSourceEventArgs)
    ' assign the data source for the master table only
    If Not e.IsFromDetailTable Then
        (TryCast(source, RadGrid)).MasterTableView.DataSource = dsCategoryQuestions
    End If
End Sub
Protected Sub gridQuestions_DetailTableDataBind(ByVal source As Object, ByVal e As
GridDetailTableDataBindEventArgs)
    ' Get the parent item primary key value,
```

```
' set to the detail data source select parameter
' and assign the detail data source.
e.DetailTableView.DataSource = dsResponse
Dim parentItem As GridDataItem = TryCast(e.DetailTableView.ParentItem, GridDataItem)
dsResponse.SelectParameters("QuestionID").DefaultValue = parentItem.GetDataKeyValue
("ID").ToString()
End Sub
Protected Sub gridQuestions_ItemCreated(ByVal sender As Object, ByVal e As
GridItemEventArgs)
' Gotcha! Once you add detail tables, you need to distinguish
' which level in the hierarchy initiated the event.
' Check OwnerTableView.DataSourceID.
' You can also use the table Name property
If e.Item.OwnerTableView.Name = "Question" Then
' initialize "enabled" checkbox
If TypeOf e.Item Is GridDataItem Then
Dim cbEnabled As CheckBox = (TryCast((TryCast(e.Item, GridDataItem))("Enabled").Controls
(0), CheckBox))
cbEnabled.Checked = True
End If
End If
End Sub
#End Region
```

[C#] Handling Grid Events

```
#region grid events
protected void gridQuestions_NeedDataSource(object source,
GridNeedDataSourceEventArgs e)
{
// assign the data source for the master table only
if (!e.IsFromDetailTable)
{
(source as RadGrid).MasterTableView.DataSource =
dsCategoryQuestions;
}
}
protected void gridQuestions_DetailTableDataBind(object source,
GridDetailTableDataBindEventArgs e)
{
// This fires when the user expands a question to reveal the
// responses.
// Get the parent item primary key value,
// set to the detail data source select parameter
// and assign the detail data source.
e.DetailTableView.DataSource = dsResponse;
GridDataItem parentItem = e.DetailTableView.ParentItem as GridDataItem;
dsResponse.SelectParameters["QuestionID"].DefaultValue =
parentItem.GetDataKeyValue("ID").ToString();
}
protected void gridQuestions_ItemCreated(object sender, GridItemEventArgs e)
{
// Gotcha! Once you add detail tables, you need to distinguish
// which level in the hierarchy initiated the event.
// Check OwnerTableView.DataSourceID.
// You can also use the table Name property
```

```

if (e.Item.OwnerTableView.Name == "Question")
{
    // initialize "enabled" checkbox
    if (e.Item is GridDataItem)
    {
        CheckBox cbEnabled =
            ((e.Item as GridDataItem)["Enabled"].Controls[0] as CheckBox);
        cbEnabled.Checked = true;
    }
}
}
#endregion

```

Handle Grid CRUD Commands

Add code to the event handlers that express the CRUD commands Insert, Update and Delete. *Be on the lookout for these details:*

- Each event handler has its own unique Item object. Looking at the insert and update commands, they use **GridDataInsertItem** and **GridEditableItem** respectively. The item's **ExtractValues()** method is used to fill a HashTable with key/value pairs. You can see how the HashTable is used to extract column data using the column name to index the HashTable.
- To know which table is to be updated or inserted to you can get the **OwnerTableView.Name** from the e.Item parameter passed in. When you prepared the layout for this grid, the **Name** for the MasterTableView and the detail table view was assigned at that time.
- When inserting to the detail table you need to know the foreign key to the master table "QuestionID". You can get that key value by traversing up from the **OwnerTableView** to the **ParentItem** and then extracting from the **DataKeyValue** property. See the **InsertCommand** handler in the 'Case "Response" for example of this technique. The **UpdateCommand** handler does not need to do this because "QuestionID" is already populated in the row for the response.
- The **DeleteCommand** only needs the key value for the record being deleted. Here you can first extract the **OwnerTableView DataKeyValues** array. Index into the array using the current row (e.Item.ItemIndex) and the key column using the column name ("ID").

[VB] Handling the Grid CRUD Commands

```

#region grid crud commands
Protected Sub gridQuestions_InsertCommand(ByVal source As Object, ByVal e As
GridCommandEventArgs)
    ' Get the item that appears when grid is in Insert Mode.
    ' Use the item object ExtractValues()method
    ' to fill a HashTable with values for the current row.
    Dim insertItem As GridDataInsertItem = DirectCast((e.Item.OwnerTableView.GetInsertItem()),
GridDataInsertItem)
    Dim ht As New Hashtable()
    insertItem.ExtractValues(ht)
    ' Navigate to the OwnerTableView for the Name property.
    ' In this case the Name will be "Question", the master table view,
    ' or "Response", the detail table view.
    ' Load up the appropriate data source parameters from the hash table
    ' and call Insert() method.
    Select Case e.Item.OwnerTableView.Name
    Case "Question"
        dsCategoryQuestions.InsertParameters("HTML").DefaultValue = ht("HTML").ToString()
        dsCategoryQuestions.InsertParameters("Enabled").DefaultValue = ht("Enabled").ToString()
        dsCategoryQuestions.Insert()
    
```

```

Exit Select
Case "Response"
' In the detail table you get at the QuestionID but
' traversing up from the OwnerTableView to the ParentItem and
' then extracting the DataKeyValue.
Dim parentItem As GridDataItem = TryCast(e.Item.OwnerTableView.ParentItem, GridDataItem)
dsResponse.InsertParameters("QuestionID").DefaultValue = parentItem.GetDataKeyValue
("ID").ToString()
dsResponse.InsertParameters("Correct").DefaultValue = ht("Correct").ToString()
dsResponse.InsertParameters("HTML").DefaultValue = ht("HTML").ToString()
dsResponse.Insert()
Exit Select
End Select
' The underlying data has now changed so rebind
e.Item.OwnerTableView.Rebind()
End Sub
Protected Sub gridQuestions_UpdateCommand(ByVal source As Object, ByVal e As
GridCommandEventArgs)
' Get the item that appears when grid is in Update Mode.
' Use the item object ExtractValues() method
' to fill a Hashtable with values for the current row.
Dim item As GridEditableItem = TryCast(e.Item, GridEditableItem)
Dim ht As New Hashtable()
item.ExtractValues(ht)

' Navigate to the OwnerTableView for the Name property.
' In this case the Name will be "Question", the master table view,
' or "Response", the detail table view.
' Load up the appropriate data source parameters from the hash table
' and call Update() method.
Select Case e.Item.OwnerTableView.Name
Case "Question"
dsCategoryQuestions.UpdateParameters("ID").DefaultValue = ht("ID").ToString()
dsCategoryQuestions.UpdateParameters("HTML").DefaultValue = ht("HTML").ToString()
dsCategoryQuestions.UpdateParameters("Enabled").DefaultValue = ht("Enabled").ToString()
dsCategoryQuestions.Update()
Exit Select
Case "Response"
' When updating, we already have the question ID and don't need to
' traverse back up to the master table.
dsResponse.UpdateParameters("ID").DefaultValue = ht("ID").ToString()
dsResponse.UpdateParameters("QuestionID").DefaultValue = ht("QuestionID").ToString()
dsResponse.UpdateParameters("Correct").DefaultValue = ht("Correct").ToString()
dsResponse.UpdateParameters("HTML").DefaultValue = ht("HTML").ToString()
dsResponse.Update()
Exit Select
End Select
e.Item.OwnerTableView.Rebind()
End Sub
Protected Sub gridQuestions_DeleteCommand(ByVal source As Object, ByVal e As
GridCommandEventArgs)
Dim dataKeys As GridDataKeyArray = e.Item.OwnerTableView.DataKeyValues
Select Case e.Item.OwnerTableView.Name
Case "Question"
Dim customerID As String = dataKeys(e.Item.ItemIndex)("ID").ToString()

```

```

    dsCategoryQuestions.DeleteParameters("ID").DefaultValue = customerID
Exit Select
Case "Response"
    Dim responseID As String = dataKeys(e.Item.ItemIndex)("ID").ToString()
    dsResponse.DeleteParameters("ID").DefaultValue = responseID
Exit Select
End Select
e.Item.OwnerTableView.Rebind()
End Sub
#End Region

```

[C#] Handling the Grid CRUD Commands

```

#region grid crud commands
protected void gridQuestions_InsertCommand(object source, GridCommandEventArgs e)
{
    // Get the item that appears when grid is in Insert Mode.
    // Use the item object ExtractValues()method
    // to fill a HashTable with values for the current row.
    GridDataInsertItem insertItem =
        (GridDataInsertItem)(e.Item.OwnerTableView.GetInsertItem());
    Hashtable ht = new Hashtable();
    insertItem.ExtractValues(ht);
    // Navigate to the OwnerTableView for the Name property.
    // In this case the Name will be "Question", the master table view,
    // or "Response", the detail table view.
    // Load up the appropriate data source parameters from the hash table
    // and call Insert() method.
    switch (e.Item.OwnerTableView.Name)
    {
        case "Question":
            dsCategoryQuestions.InsertParameters["HTML"].DefaultValue =
                ht["HTML"].ToString();
            dsCategoryQuestions.InsertParameters["Enabled"].DefaultValue =
                ht["Enabled"].ToString();
            dsCategoryQuestions.Insert();
            break;
        // In the detail table you get at the QuestionID but
        // traversing up from the OwnerTableView to the ParentItem and
        // then extracting the DataKeyValue.
        case "Response":
            GridDataItem parentItem =
                e.Item.OwnerTableView.ParentItem as GridDataItem;
            dsResponse.InsertParameters["QuestionID"].DefaultValue =
                parentItem.GetDataKeyValue("ID").ToString();
            dsResponse.InsertParameters["Correct"].DefaultValue =
                ht["Correct"].ToString();
            dsResponse.InsertParameters["HTML"].DefaultValue =
                ht["HTML"].ToString();
            dsResponse.Insert();
            break;
    }
    // The underlying data has now changed so rebind
    e.Item.OwnerTableView.Rebind();
}
protected void gridQuestions_UpdateCommand(object source, GridCommandEventArgs e)

```

```
{
    // Get the item that appears when grid is in Update Mode.
    // Use the item object ExtractValues() method
    // to fill a Hashtable with values for the current row.
    GridEditableItem item = e.Item as GridEditableItem;
    Hashtable ht = new Hashtable();
    item.ExtractValues(ht);

    // Navigate to the OwnerTableView for the Name property.
    // In this case the Name will be "Question", the master table view,
    // or "Response", the detail table view.
    // Load up the appropriate data source parameters from the hash table
    // and call Update() method.
    switch (e.Item.OwnerTableView.Name)
    {
        case "Question":
            dsCategoryQuestions.UpdateParameters["ID"].DefaultValue =
                ht["ID"].ToString();
            dsCategoryQuestions.UpdateParameters["HTML"].DefaultValue =
                ht["HTML"].ToString();
            dsCategoryQuestions.UpdateParameters["Enabled"].DefaultValue =
                ht["Enabled"].ToString();
            dsCategoryQuestions.Update();
            break;
        // When updating, we already have the question ID and don't need to
        // traverse back up to the master table.
        case "Response":
            dsResponse.UpdateParameters["ID"].DefaultValue =
                ht["ID"].ToString();
            dsResponse.UpdateParameters["QuestionID"].DefaultValue =
                ht["QuestionID"].ToString();
            dsResponse.UpdateParameters["Correct"].DefaultValue =
                ht["Correct"].ToString();
            dsResponse.UpdateParameters["HTML"].DefaultValue =
                ht["HTML"].ToString();
            dsResponse.Update();
            break;
    }
    e.Item.OwnerTableView.Rebind();
}
protected void gridQuestions_DeleteCommand(object source, GridCommandEventArgs e)
{
    GridDataKeyArray dataKeys = e.Item.OwnerTableView.DataKeyValues;
    switch (e.Item.OwnerTableView.Name)
    {
        case "Question":
            string customerID = dataKeys[e.Item.ItemIndex]["ID"].ToString();
            dsCategoryQuestions.DeleteParameters["ID"].DefaultValue =
                customerID;
            break;
        case "Response":
            string responseID = dataKeys[e.Item.ItemIndex]["ID"].ToString();
            dsResponse.DeleteParameters["ID"].DefaultValue =
                responseID;
            break;
    }
}
```

```

}
e.Item.OwnerTableView.Rebind();
}
#endregion

```

Implement the CategoryTree InitialLoad Method

Add the code below to the CategoriesTree NodeClick event handler.

The data source select query CategoryID parameter will be loaded using the value from the currently selected CategoriesTree node value, the grid will be re-bound and the grid will be disabled if the selected node is the root node. Note that the call to Rebind() will also trigger a call to the grid's NeedDataSource event.

[VB] Handling the NodeClick Event

```

#region CategoriesTree events
Sub CategoriesTree1_NodeClick(ByVal sender As Object, ByVal e As RadTreeNodeEventArgs)
' load the select query parameters with the currently selected
' category node's ID, rebind the grid and set the grid to be
' enabled if the treeview node is not the root node.
dsCategoryQuestions.SelectParameters("CategoryID").DefaultValue =
CategoriesTree1.CategoryID
gridQuestions.Rebind()
gridQuestions.Enabled = Not CategoriesTree1.IsRootSelected
End Sub
#End Region

```

[C#] Handling the NodeClick Event

```

#region CategoriesTree events
void CategoriesTree1_NodeClick(object sender, RadTreeNodeEventArgs e)
{
// load the select query parameters with the currently selected
// category node's ID, rebind the grid and set the grid to be
// enabled if the treeview node is not the root node.
dsCategoryQuestions.SelectParameters["CategoryID"].DefaultValue =
CategoriesTree1.CategoryID;
gridQuestions.Rebind();
gridQuestions.Enabled = !CategoriesTree1.IsRootSelected;
}
#endregion

```

Styling the Grid

Again, the skin can be tweaked slightly to suit our purposes. Locate the Grid.ActiveSkill.css in the \skins\ActiveSkill directory of the ActiveSkillUI project. Find the .RadGrid_ActiveSkill element that defines the background and font color. Set the color "color:#CFCFCF;"

[CSS] Changing Grid Font Color

```

.RadGrid_ActiveSkill
{
background:#313131;
color:#CFCFCF;
}

```

Locate the GridHeader style selector that handles font color. Change color to "color:#598FD3;"

[CSS] Changing Grid Header Font Color

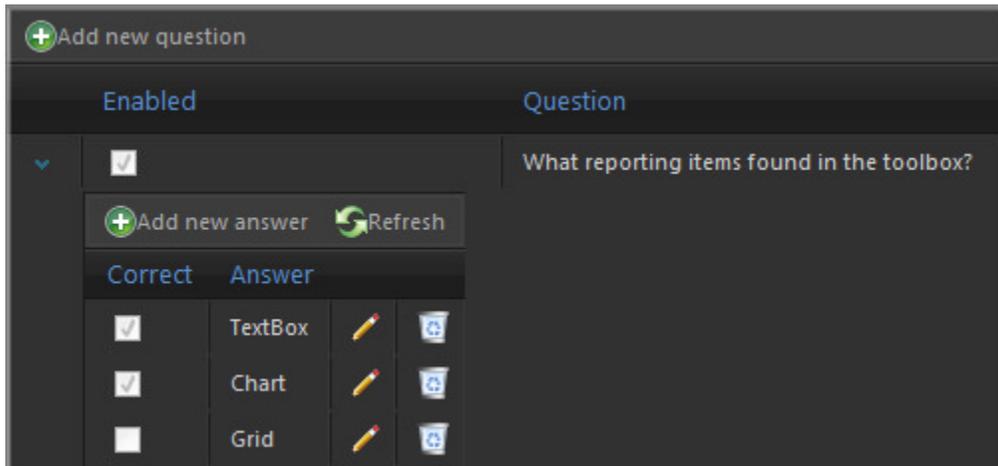
```

.GridHeader_ActiveSkill,

```

```
.GridHeader_ActiveSkill a
{
color:#598FD3;
text-decoration:none;
}
```

When you run the application, the question and response text will be more visible but still keeping with the overall look-and-feel for the skin. Also, the heading font will be slightly brighter to match the titling on the "Admin Tools" panel.



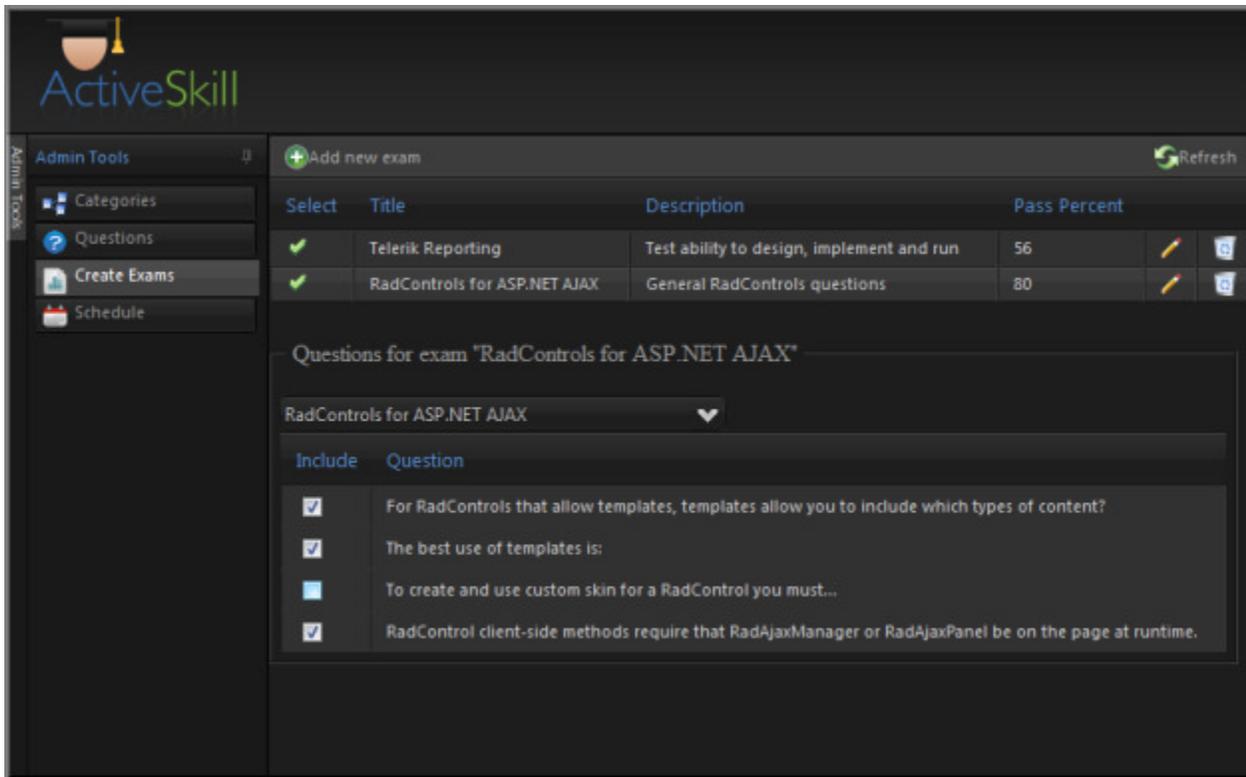
Test the Application

Run the application and work with both the CategoriesTree selection and the grid functions. Add, edit and delete questions. Notice when adding or editing that the record display is "in-line" due to the **EditMode** setting.

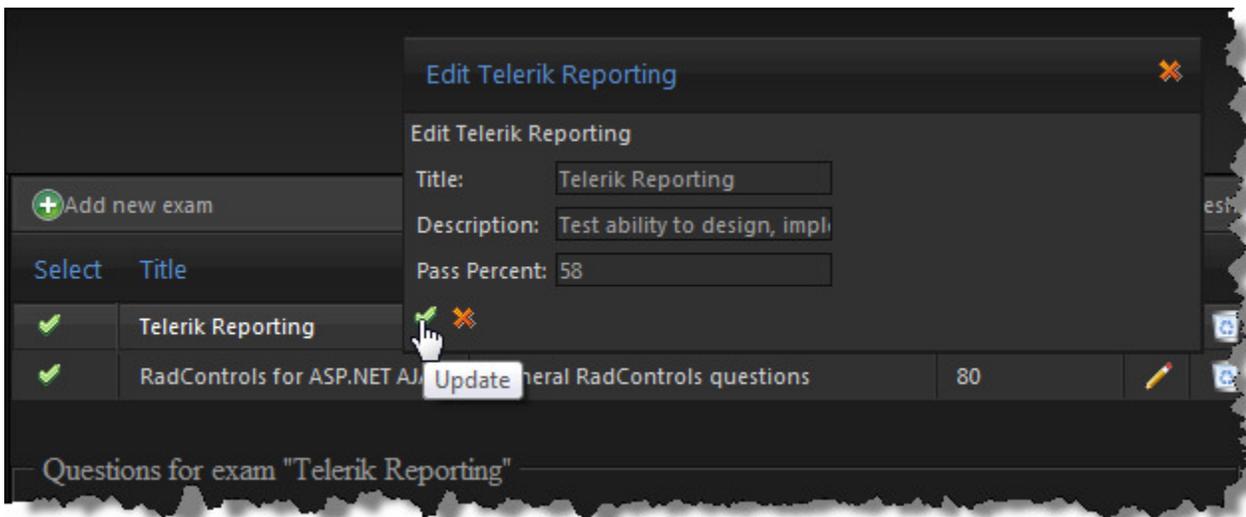
38.6 Implement CreateExams Control

This next control that implements the "page" for creating exams uses two RadGrids. The two grids are related, but not nested, as in the "Questions" control. The top grid lists all the available exams and allows the user to add, edit and delete exam records.

Below that, a CategoriesTree is used to filter the second grid that shows questions that apply to a given exam. All questions for a given category are shown and checkboxes in a grid template column indicate if they are included in the selected exam.



The edit mode for Insert and Edit will show in a pop-up dialog as shown in the screen shot below:



You can find the complete source for this project at:
 \VS Projects\ActiveSkill Database Maintenance\Create Exams

Prepare Page Layout

RoadMap

The overall layout of this user control includes a grid to list all of the exams and a <fieldset> element to contain a CategoriesTree that filters a second grid of questions only for that category. The data sources feed the CategoriesTree, the exams grid and the exam questions grid respectively.

```
<%--Data Sources--%>
<asp:SqlDataSource ID="dsAllCategories" ...>...</asp:SqlDataSource>
<asp:SqlDataSource ID="dsExams" ...>...</asp:SqlDataSource>
<asp:SqlDataSource ID="dsExamQuestions" ...>...</asp:SqlDataSource>

<%--Exam grid--%>
<telerik:RadGrid ID="gridExams" ...>...</telerik:RadGrid>
<br />

<%--Exam Title and Questions Grid--%>
<fieldset>
  <legend id="legendTitle" runat="server">Questions for
    exam "<%=this.ExamTitle %%"</legend>
  <br />
  <uc1:CategoriesTree ID="CategoriesTree1" runat="server"
    DisplayMode="DropDown" />
  <telerik:RadGrid ID="gridExamQuestions" ...>...</telerik:RadGrid>
</fieldset>
```

Add Data Sources

Add data source controls for the Categories tree, Exam and Questions Grid using the markup below.

- dsAllCategories: Supplies all categories to the CategoriesTree control.
- dsExams: Supplies all available exams. Handles add/update/delete operations.
- dsExamQuestions: Returns a list of questions for the selected exam and category. Handles add and delete operations.

[ASP.NET] Adding the DataSource Controls

```
<%@ Control Language="C#" AutoEventWireup="true" CodeBehind="CreateExams.ascx.cs"
  Inherits="Telerik.ActiveSkill.UI.Admin.CreateExams" %>
<%@ Register Src="../Controls/CategoriesTree.ascx"
  TagName="CategoriesTree" TagPrefix="uc1" %>
<%@ Register Assembly="Telerik.Web.UI, Version=2008.2.723.35, Culture=neutral,
  PublicKeyToken=121fae78165ba3d4"
  Namespace="Telerik.Web.UI" TagPrefix="telerik" %>

<%--Data Sources--%>
<asp:SqlDataSource ID="dsAllCategories" runat="server"
  ConnectionString="<%= $ConnectionStrings.ActiveSkillConnectionString %%"
  SelectCommand="SELECT [ID], [ParentID], [Title], [Description] FROM [Category]">
</asp:SqlDataSource>
<asp:SqlDataSource ID="dsExams" runat="server"
  ConnectionString="<%= $ConnectionStrings.ActiveSkillConnectionString %%"
  SelectCommand="SELECT [ID], [Title], [Description], [PassPercent], [ModifyDate] FROM
```

```
[Exam]"
DeleteCommand="Skill_Exam_Delete" DeleteCommandType="StoredProcedure"
InsertCommand="Skill_Exam_Insert" InsertCommandType="StoredProcedure"
UpdateCommand="Skill_Exam_Update" UpdateCommandType="StoredProcedure">
<DeleteParameters>
  <asp:Parameter Name="ID" Type="Int32" />
</DeleteParameters>
<UpdateParameters>
  <asp:Parameter Name="ID" Type="Int32" />
  <asp:Parameter Name="Title" Type="String" />
  <asp:Parameter Name="Description" Type="String" />
  <asp:Parameter Name="PassPercent" Type="Int32" />
</UpdateParameters>
<InsertParameters>
  <asp:Parameter Direction="InputOutput" Name="ID" Type="Int32" />
  <asp:Parameter Name="Title" Type="String" />
  <asp:Parameter Name="Description" Type="String" />
  <asp:Parameter Name="PassPercent" Type="Int32" />
</InsertParameters>
</asp:SqlDataSource>
<asp:SqlDataSource ID="dsExamQuestions" runat="server"
  ConnectionString="<%= $ConnectionStrings.ActiveSkillConnectionString %>"
  SelectCommand="Skill_Exam_Question_SelectWhere" SelectCommandType="StoredProcedure"
  DeleteCommand="Skill_Exam_Question_Delete" DeleteCommandType="StoredProcedure"
  InsertCommand="Skill_Exam_Question_Insert" InsertCommandType="StoredProcedure">
  <SelectParameters>
    <asp:Parameter Name="CategoryID" Type="Int32" />
    <asp:Parameter Name="ExamID" Type="Int32" />
  </SelectParameters>
  <DeleteParameters>
    <asp:ControlParameter ControlID="gridExams" Name="ExamID"
      PropertyName="SelectedValue" Type="Int32" />
    <asp:Parameter Name="QuestionID" Type="Int32" />
  </DeleteParameters>
  <InsertParameters>
    <asp:ControlParameter ControlID="gridExams" Name="ExamID"
      PropertyName="SelectedValue" Type="Int32" />
    <asp:Parameter Name="QuestionID" Type="Int32" />
  </InsertParameters>
</asp:SqlDataSource>
```

Add the "Exams" Grid

1. Add a RadGrid control. In the Properties window set these properties for the grid:
 - **ID:** "gridExams".
 - **AllowAutomaticDeletes:** true.
 - **AutoGenerateColumns:** false.
 - **GridLines:** None.
2. Open up the MasterTableView property and set the following sub properties:
 - **CommandItemDisplay:** Top
 - **DataKeyNames:** "ID"
 - **EditMode:** Popup

- **Name:** "Exam"
- **NoMasterRecordsText:** "No exams to display"

3. Using the Properties window Events button () , add handlers for the following events:

- OnInsertCommand
- OnNeedDataSource
- OnDeleteCommand
- OnUpdateCommand
- OnSelectedIndexChanged
- OnDataBound

In the later sections we will implement these event handlers as well as add the columns.

Setup up the Exam Questions Area

In the markup, add a **FieldSet** HTML element below the exams grid. Set the legend to bind to a property of the page "this.ExamTitle". Below that, add a **CategoriesTree** and set the **DisplayMode** property to "DropDown".

[ASP.NET] Adding the FieldSet and CategoriesTree

```
<fieldset>
  <legend id="legendTitle" runat="server">Questions for
    exam "%=this.ExamTitle %"</legend>
  <br />
  <uc1:CategoriesTree ID="CategoriesTree1" runat="server"
    DisplayMode="DropDown" />
  <!-- exam questions grid goes here -->
</fieldset>
```

Add the "Exam Questions" Grid

1. Back in the designer, add a **RadGrid** control below the **CategoriesTree** within the **FieldSet**. In the Properties window set these properties for the grid:
 - **ID:** "gridExamQuestions".
 - **AutoGenerateColumns:** false.
 - **AllowPaging:** true.
 - **GridLines:** None.
2. Open up the **MasterTableView** property and set the following sub properties:
 - **CommandItemDisplay:** Top
 - **DataKeyNames:** "ID"
 - **EditMode:** InPlace
 - **Name:** "Question"
 - **NoMasterRecordsText:** "No questions to display"
3. Using the Properties window Events button () , add handlers for the following events:
 - OnNeedDataSource
 - OnItemDataBound

In the later sections we will implement these event handlers.

Adding Columns

Add columns to the exam grid:

[ASP.NET] Adding Columns to gridExams

```
<Columns>
  <telerik:GridButtonColumn CommandName="Select" Text="Select"
    UniqueName="column1" HeaderText="Select"
    ImageUrl="../Skins/ActiveSkill/Grid/Update.gif"
    ButtonType="ImageButton">
  </telerik:GridButtonColumn>
  <telerik:GridBoundColumn DataField="Title" HeaderText="Title"
    UniqueName="Title">
  </telerik:GridBoundColumn>
  <telerik:GridBoundColumn DataField="Description" HeaderText="Description"
    UniqueName="Description">
  </telerik:GridBoundColumn>
  <telerik:GridBoundColumn DataField="PassPercent" DataType="System.Int32"
    HeaderText="Pass Percent" UniqueName="PassPercent">
  </telerik:GridBoundColumn>
  <telerik:GridEditCommandColumn ButtonType="ImageButton"
    CancelImageUrl="../Skins/ActiveSkill/Grid/Cancel.gif"
    EditImageUrl="../Skins/ActiveSkill/Grid/Edit.gif"
    InsertImageUrl="../Skins/ActiveSkill/Grid/Update.gif"
    UpdateImageUrl="../Skins/ActiveSkill/Grid/Update.gif">
  </telerik:GridEditCommandColumn>
  <telerik:GridButtonColumn UniqueName="ExamsDeleteColumn"
    ButtonType="ImageButton" CommandName="Delete"
    ImageUrl="../Skins/ActiveSkill/Grid/Delete.gif"
    Text="Delete">
  </telerik:GridButtonColumn>
</Columns>
```

Add columns to the exam questions grid:

[ASP.NET] Adding Columns to gridExamQuestions

```
<Columns>
  <telerik:GridBoundColumn UniqueName="ID" DataField="ID"
    DataType="System.Int32" Display="False" Visible="False"
    FilterImageUrl="../Skins/ActiveSkill/Grid/Filter.gif"
    SortAscImageUrl="../Skins/ActiveSkill/Grid/SortAsc.gif"
    SortDescImageUrl="../Skins/ActiveSkill/Grid/SortDesc.gif">
  </telerik:GridBoundColumn>
  <telerik:GridBoundColumn UniqueName="IsInExam" DataField="IsInExam"
    DataType="System.boolean" Display="False">
  </telerik:GridBoundColumn>
  <telerik:GridTemplateColumn UniqueName="IsInExamColumn"
    HeaderText="Include">
    <ItemTemplate>
      <asp:CheckBox ID="cbIsInExam" runat="server" AutoPostBack="true"
        OnCheckedChanged="IsInExamCheckChanged"></asp:CheckBox>
    </ItemTemplate>
  </telerik:GridTemplateColumn>
  <telerik:GridHTMLEditorColumn DataField="HTML" HeaderText="Question">
```

```
SortExpression="HTML" UniqueName="HTML" ReadOnly="true"
FilterImageUrl=" ../Skins/ActiveSkill/Grid/Filter.gif"
SortAscImageUrl=" ../Skins/ActiveSkill/Grid/SortAsc.gif"
SortDescImageUrl=" ../Skins/ActiveSkill/Grid/SortDesc.gif">
</telerik:GridHTMLEditorColumn>
</Columns>
```

Review the layout

Take a look at a few critical pieces of the layout before moving on to coding:

- The **EditMode** property of the MasterTableView is set to "PopUp".
- The **EditFormSettings** tag of the Exam grid has a **CaptionDataField** property set to "Title". The exam "Title" column will display in the grid pop-up caption area. Note that you can also use **CaptionFormatString** along with **CaptionDataField**, e.g. "The title of the exam is {0}".
- The **NoMasterRecordsText** property of the Exam grid MasterTableView lets you set custom text to display when there are no records, rather than the generic default message.
- The "cbIsInExam" check box is found within a **GridTemplateColumn** of the questions grid. You will be accessing this check box in code later to implement custom behavior, so take a quick look at it now.
- The **AllowAutomaticDeletes** property is enabled for the exam grid. This allows us to leave out the data source Delete() method call. Note that you still have to set up the data source delete parameters and rebind the grid afterward.
- In the **GridEditCommandColumn** that displays the edit, insert or cancel buttons has a **ButtonType** of "ImageButton" and that the image URLs reuse grid images found in the \skins directory.
- The field set surrounding the exam questions has a legend element that is bound to the ExamTitle property of the "page".

Add Properties

This page uses the property ExamTitle in a binding expression within the legend. The property is refreshed whenever a new row is selected in the exam grid.

[C#] Defining Properties

```
#region properties
Private Const ExamTitleKey As String = "ExamTitleKey"
Public Property ExamTitle() As String
    Get
        Return IIf(ViewState(ExamTitleKey) = Nothing, "", ViewState(ExamTitleKey).ToString())
    End Get
    Set
        ViewState(ExamTitleKey) = value
    End Set
End Property
#End Region properties
```

[C#] Defining Properties

```
#region properties
private const string ExamTitleKey = "ExamTitleKey";
public string ExamTitle
{
    get
    {
        return ViewState[ExamTitleKey] == null ? "" :
            ViewState[ExamTitleKey].ToString();
    }
}
```

```

}
set
{
    ViewState[ExamTitleKey] = value;
}
}
#endregion properties

```

Add Private Methods

Add the ExamQuestionsRefresh() method shown below.

This method ties together the exams grid, the legend and the exams-questions grid. The method first checks that something is selected in the grid. Then the selected item is cast to GridDataItem to access the "Title" column. "Title" is assigned to the ExamTitle property and bound so that it shows up in the legend (by way of a binding expression). You can find "ExamTitle" being used in a binding expression within "CreateExams.ascx".

The data source for the "detail" grid displaying questions relies on both the currently selected category ID and the exam ID selected in the grid. Here, you assign those two values to the data source SelectParameters and rebind the grid to display the questions for that exam/category combination.

[VB] Handling Private Events

```

#region private methods
' Handles common logic to refresh the grid for ExamQuestions
Private Sub ExamQuestionsRefresh()
' if an item is selected in the exams grid,
' save the ExamTitle for use the legend binding expression
' the legend), set the parameters for gridExamQuestions and rebind
If gridExams.SelectedItems.Count > 0 Then
    Dim item As GridDataItem = TryCast(gridExams.SelectedItems(0), GridDataItem)
    Me.ExamTitle = item("Title").Text
    legendTitle.DataBind()
    dsExamQuestions.SelectParameters("CategoryID").DefaultValue = CategoriesTree1.CategoryID
    dsExamQuestions.SelectParameters("ExamID").DefaultValue = gridExams.SelectedValue.ToString
()
    gridExamQuestions.Rebind()
End If
End Sub
#End Region

```

[C#] Handling Private Events

```

#region private methods
// Handles common logic to refresh the grid for ExamQuestions
private void ExamQuestionsRefresh()
{
    // if an item is selected in the exams grid,
    // save the ExamTitle for use the legend binding expression
    // the legend), set the parameters for gridExamQuestions and rebind
    if (gridExams.SelectedItems.Count > 0)
    {
        GridDataItem item = gridExams.SelectedItems[0] as GridDataItem;
        this.ExamTitle = item["Title"].Text;
        legendTitle.DataBind();
        dsExamQuestions.SelectParameters["CategoryID"].DefaultValue = CategoriesTree1.CategoryID;
        dsExamQuestions.SelectParameters["ExamID"].DefaultValue =
gridExams.SelectedValue.ToString();
        gridExamQuestions.Rebind();
    }
}

```

```
}  
}  
#endregion
```

Handle Page Events

- Add the code below to the **Page_Load** event handler. *Attach the CategoriesTree NodeClick every time the page loads.*
- Add the code for "IsInExamCheckChanged" to handle the **OnCheckChanged** event for the "cbIsInExam" checkbox.

 How do you get the data associated with the checkbox? There is no `CheckBox.Tag` or `Value`. Instead, use the naming container that contains the checkbox, which happens to be a `GridDataItem`.

This check box is contained in a questions grid template column. To get at data for the row associated with the checkbox, snag the sender parameter and cast it to be `CheckBox`. Then step up to the checkbox `NamingContainer` and cast that to be a `GridDataItem`. You can use the `GridDataItem.GetDataKeyValue()` to extract the "QuestionID" for the row. If the check box is being checked, a record is added to the `ExamQuestions` join table, otherwise the record is deleted.

[VB] Handling Page Events

```
#region page events  
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)  
    ' add the node click event handler every page load  
    AddHandler CategoriesTree1.NodeClick, AddressOf CategoriesTree1_NodeClick  
End Sub  
' OnCheckChanged event handler  
Public Sub IsInExamCheckChanged(ByVal sender As Object, ByVal e As EventArgs)  
    ' Toggle the "IsInExam" checkbox and either insert or  
    ' delete the corresponding record  
    Dim cb As CheckBox = (TryCast(sender, CheckBox))  
    ' How do you get the data associated with the checkbox?  
    ' There is no CheckBox.Tag or Value.  
    ' Use the naming container that holds the checkbox,  
    ' which happens to be a GridDataItem.  
    Dim item As GridDataItem = DirectCast(cb.NamingContainer, GridDataItem)  
    Dim id As String = item.GetDataKeyValue("ID").ToString()  
    If cb.Checked Then  
        dsExamQuestions.InsertParameters("QuestionID").DefaultValue = id  
        dsExamQuestions.Insert()  
    Else  
        dsExamQuestions.DeleteParameters("QuestionID").DefaultValue = id  
        dsExamQuestions.Delete()  
    End If  
End Sub  
#End Region
```

[C#] Handling Page Events

```
#region page events  
protected void Page_Load(object sender, EventArgs e)  
{  
    // add the node click event handler every page load  
    CategoriesTree1.NodeClick +=  
        new RadTreeViewEventHandler(CategoriesTree1_NodeClick);  
}  
}
```

```

// OnCheckChanged event handler
public void IsInExamCheckChanged(object sender, EventArgs e)
{
    // Toggle the "IsInExam" checkbox and either insert or
    // delete the corresponding record
    CheckBox cb = (sender as CheckBox);
    // How do you get the data associated with the checkbox?
    // There is no CheckBox.Tag or Value.
    // Use the naming container that holds the checkbox,
    // which happens to be a GridDataItem.
    GridDataItem item = (GridDataItem)cb.NamingContainer;
    string id = item.GetDataKeyValue("ID").ToString();
    if (cb.Checked)
    {
        dsExamQuestions.InsertParameters["QuestionID"].DefaultValue = id;
        dsExamQuestions.Insert();
    }
    else
    {
        dsExamQuestions.DeleteParameters["QuestionID"].DefaultValue = id;
        dsExamQuestions.Delete();
    }
}
#endregion

```

Handle Categories Tree Node Click

Add the code below to handle the categories tree NodeClick event. *The event handler will only call the ExamQuestionsRefresh() method to sync the legend and grid to the first row of the exam grid.*

[C#] Handling the Categories Tree Node Click

```

#region categories tree events
Sub CategoriesTree1_NodeClick(ByVal sender As Object, ByVal e As RadTreeNodeEventArgs)
    ' category has changed so run common logic
    ExamQuestionsRefresh()
End Sub
#End Region

```

[C#] Handling the Categories Tree Node Click

```

#region categories tree events
void CategoriesTree1_NodeClick(object sender, RadTreeNodeEventArgs e)
{
    // category has changed so run common logic
    ExamQuestionsRefresh();
}
#endregion

```

Handle Exam Grid Events

Add the code below to handle the Exam grid events.

- **The CRUD command events** are essentially the same structure as used in the Questions maintenance. For **InsertCommand** and **UpdateCommand**, the operation-appropriate item object is retrieved, its **ExtractValues()** method called to fill a hash table and the hash table is used to fill data source parameters before calling **Insert()** or **Update()**. **DeleteCommand** is also the same pattern from Questions maintenance. Here the **OwnerTableView DataKeyValues** array is used to extract the primary key value, and that value is used to populate data source parameters. Because the **AllowAutomaticDeletes** property

was set to true, the data source Delete() method is not called explicitly.

- `gridExams_NeedDataSource` simply assigns the exam grid data source.
- `gridExams_SelectedIndexChanged` calls the private method `ExamQuestionsRefresh()` to resync the legend and the questions grid with the current selected row of the exam grid.
- Once the exam grid is bound, `gridExams_DataBound` selects the first exam row if nothing else has been selected.

[VB] Handling Exam Grid Events

```
#region exams grid events
Protected Sub gridExams_InsertCommand(ByVal source As Object, ByVal e As
Telerik.Web.UI.GridCommandEventArgs)
    ' Get the item that appears when grid is in Insert Mode.
    ' Use the item object ExtractValues()method
    ' to fill a Hashtable with values for the current row.
    Dim insertItem As GridEditFormInsertItem = DirectCast((e.Item.OwnerTableView.GetInsertItem
()), GridEditFormInsertItem)
    Dim ht As New Hashtable()
    insertItem.ExtractValues(ht)
    ' Load data source parameters from the hash table and insert the record.
    dsExams.InsertParameters("Title").DefaultValue = ht("Title").ToString()
    dsExams.InsertParameters("Description").DefaultValue = ht("Description").ToString()
    dsExams.InsertParameters("PassPercent").DefaultValue = ht("PassPercent").ToString()
    dsExams.Insert()
    ' The underlying data has now changed so rebind
    e.Item.OwnerTableView.Rebind()
End Sub
Protected Sub gridExams_UpdateCommand(ByVal source As Object, ByVal e As
GridCommandEventArgs)
    ' Get the item that appears when grid is in Update Mode.
    ' Use the item object ExtractValues()method
    ' to fill a Hashtable with values for the current row.
    Dim item As GridEditableItem = TryCast(e.Item, GridEditableItem)
    Dim ht As New Hashtable()
    item.ExtractValues(ht)
    ' Load data source parameters from the hash table and update the record.
    Dim dataKeys As GridDataKeyArray = e.Item.OwnerTableView.DataKeyValues
    Dim id As String = dataKeys(e.Item.ItemIndex)("ID").ToString()
    dsExams.UpdateParameters("ID").DefaultValue = id
    dsExams.UpdateParameters("Title").DefaultValue = ht("Title").ToString()
    dsExams.UpdateParameters("Description").DefaultValue = ht("Description").ToString()
    dsExams.UpdateParameters("PassPercent").DefaultValue = ht("PassPercent").ToString()
    dsExams.Update()
    ' The underlying data has now changed so rebind
    e.Item.OwnerTableView.Rebind()
End Sub
Protected Sub gridExams_DeleteCommand(ByVal source As Object, ByVal e As
GridCommandEventArgs)
    ' Extract the OwnerTableView DataKeyValues array.
    ' Index into the array using the current row (e.Item.ItemIndex)
    ' and the key column of the primary key("ID").
    Dim dataKeys As GridDataKeyArray = e.Item.OwnerTableView.DataKeyValues
    Dim id As String = dataKeys(e.Item.ItemIndex)("ID").ToString()
    ' You don't need to call the data source Delete() here
```

```

' because you already set AllowAutomaticDeletes to true.
e.Item.OwnerTableView.Rebind()
End Sub

Protected Sub gridExams_NeedDataSource(ByVal source As Object, ByVal e As
Telerik.Web.UI.GridNeedDataSourceEventArgs)
    gridExams.DataSource = dsExams
End Sub
Protected Sub gridExams_SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)
' the row in the exams grid has changed,
' so resync the other parts of the page.
ExamQuestionsRefresh()
End Sub
Protected Sub gridExams_DataBound(ByVal sender As Object, ByVal e As EventArgs)
' There are items, but none selected yet. Select the first row as a default
If (gridExams.Items.Count > 0) And (gridExams.SelectedIndexes.Count = 0) Then
    gridExams.Items(0).Selected = True
    ExamQuestionsRefresh()
End If
End Sub
#End Region

```

[C#] Handling Exam Grid Events

```

#region exams grid events
protected void gridExams_InsertCommand(object source, Telerik.Web.UI.GridCommandEventArgs e)
{
    // Get the item that appears when grid is in Insert Mode.
    // Use the item object ExtractValues()method
    // to fill a HashTable with values for the current row.
    GridEditFormInsertItem insertItem =
        (GridEditFormInsertItem)(e.Item.OwnerTableView.GetInsertItem());
    Hashtable ht = new Hashtable();
    insertItem.ExtractValues(ht);
    // Load data source parameters from the hash table and insert the record.
    dsExams.InsertParameters["Title"].DefaultValue = ht["Title"].ToString();
    dsExams.InsertParameters["Description"].DefaultValue = ht["Description"].ToString();
    dsExams.InsertParameters["PassPercent"].DefaultValue = ht["PassPercent"].ToString();
    dsExams.Insert();
    // The underlying data has now changed so rebind
    e.Item.OwnerTableView.Rebind();
}
protected void gridExams_UpdateCommand(object source, GridCommandEventArgs e)
{
    // Get the item that appears when grid is in Update Mode.
    // Use the item object ExtractValues()method
    // to fill a HashTable with values for the current row.
    GridEditableItem item = e.Item as GridEditableItem;
    Hashtable ht = new Hashtable();
    item.ExtractValues(ht);
    // Load data source parameters from the hash table and update the record.
    GridDataKeyArray dataKeys = e.Item.OwnerTableView.DataKeyValues;
    string id = dataKeys[e.Item.ItemIndex]["ID"].ToString();
    dsExams.UpdateParameters["ID"].DefaultValue = id;
    dsExams.UpdateParameters["Title"].DefaultValue = ht["Title"].ToString();
    dsExams.UpdateParameters["Description"].DefaultValue = ht["Description"].ToString();
}

```

```
dsExams.UpdateParameters["PassPercent"].DefaultValue = ht["PassPercent"].ToString();
dsExams.Update();
// The underlying data has now changed so rebind
e.Item.OwnerTableView.Rebind();
}

protected void gridExams_DeleteCommand(object source, GridCommandEventArgs e)
{
    // Extract the OwnerTableView DataKeyValues array.
    // Index into the array using the current row (e.Item.ItemIndex)
    // and the key column of the primary key("ID").
    GridDataKeyArray dataKeys = e.Item.OwnerTableView.DataKeyValues;
    string id = dataKeys[e.Item.ItemIndex]["ID"].ToString();
    // You don't need to call the data source Delete() here
    // because you already set AllowAutomaticDeletes to true.
    e.Item.OwnerTableView.Rebind();
}

protected void gridExams_NeedDataSource(object source,
Telerik.Web.UI.GridNeedDataSourceEventArgs e)
{
    gridExams.DataSource = dsExams;
}
protected void gridExams_SelectedIndexChanged(object sender, EventArgs e)
{
    // the row in the exams grid has changed,
    // so resync the other parts of the page.
    ExamQuestionsRefresh();
}
protected void gridExams_DataBound(object sender, EventArgs e)
{
    // There are items, but none selected yet. Select the first row as a default
    if ((gridExams.Items.Count > 0) & (gridExams.SelectedIndexes.Count == 0))
    {
        gridExams.Items[0].Selected = true;
        ExamQuestionsRefresh();
    }
}
}
#endregion
```

Handle Questions Grid Events

Add the event handling code below to populate the questions grid.

*The **NeedDataSource** event simply assigns the appropriate data source. **ItemDataBound** fires for each row and allows the current "IsInExam" value to be reflected in the checkbox found in a template column with the **UniqueName** property "IsInExamColumn".*

 Use **UniqueName** to identify columns in a grid. Each column in a RadGrid has an **UniqueName** property. This property is assigned automatically by the designer (or the first time you want to access the columns if built dynamically).

[VB] Handling Questions Grid Events

```
#region questions grid events
Protected Sub gridQuestions_NeedDataSource(ByVal source As Object, ByVal e As
GridNeedDataSourceEventArgs)
    gridExamQuestions.DataSource = dsExamQuestions
```

```

End Sub
Protected Sub gridQuestions_ItemDataBound(ByVal sender As Object, ByVal e As
GridItemEventArgs)
    If TypeOf e.Item Is GridDataItem Then
        Dim item As GridDataItem = TryCast(e.Item, GridDataItem)
        ' Extract "IsInExam" boolean value
        Dim ht As New Hashtable()
        item.ExtractValues(ht)
        Dim isInExam As Boolean = IIf(ht("IsInExam") = Nothing, False, Convert.ToBoolean(ht
("IsInExam")))
        ' get checkbox control and set checked according to extracted "IsInExam" value.
        Dim cbIsInExam As CheckBox = TryCast(item("IsInExamColumn").FindControl("cbIsInExam"),
CheckBox)
        cbIsInExam.Checked = isInExam
    End If
End Sub
#End Region

```

[C#] Handling Questions Grid Events

```

#region questions grid events
protected void gridQuestions_NeedDataSource(object source, GridNeedDataSourceEventArgs e)
{
    gridExamQuestions.DataSource = dsExamQuestions;
}
protected void gridQuestions_ItemDataBound(object sender, GridItemEventArgs e)
{
    if (e.Item is GridDataItem)
    {
        GridDataItem item = e.Item as GridDataItem;
        // Extract "IsInExam" boolean value
        Hashtable ht = new Hashtable();
        item.ExtractValues(ht);
        bool isInExam = ht["IsInExam"] == null ? false : Convert.ToBoolean(ht["IsInExam"]);
        // get checkbox control and set checked according to extracted "IsInExam" value.
        CheckBox cbIsInExam = item["IsInExamColumn"].FindControl("cbIsInExam") as CheckBox;
        cbIsInExam.Checked = isInExam;
    }
}
#endregion

```

Implement the FirstLoad Method

As in the previous maintenance user controls, add the code below to call the categories tree InitialLoad() method within the FirstLoad() implementation.

[VB] Implementing the FirstLoad() Method

```

#region IASControl Members
Public Sub FirstLoad(ByVal args As System.Collections.Generic.Dictionary(Of String, String))
    CategoriesTree1.InitialLoad(dsAllCategories)
End Sub
#End Region

```

[C#] Implementing the FirstLoad() Method

```

#region IASControl Members
public void FirstLoad(System.Collections.Generic.Dictionary<string, string> args)
{

```

UI for ASP.NET AJAX

```
CategoriesTree1.InitialLoad(dsAllCategories);
}
#endregion
```

Make the Selected Row More Visible

The skin as it exists now already highlights the selected row slightly. To make the currently selected row more visible you can change the skin by...

- Altering the graphic elements of the row background.
- Changing the font of the selected text within the CSS.

In this case we're going to take the easier and shorter way out by just changing the font to a brighter color. The general pattern for changing the graphic was touched on when we first built the CategoryTree control in "DataBind and Use the Control" where we changed the default drop down arrow.

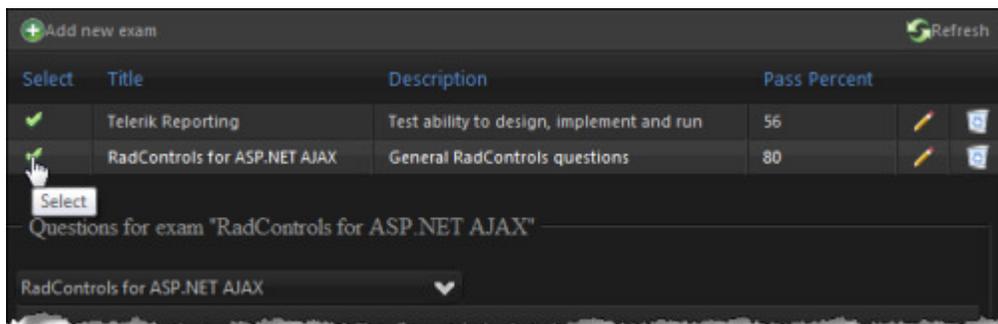
✍ If you want to alter the selected row graphic in PhotoShop or some other graphics utility, look in the \ActiveSkillUI\Skins\Grid folder for the file called "Sprite.gif".

1. Locate Grid.ActiveSkill.CSS in the \skins folder and open it.
2. Find the first instance of ".SelectedRow_ActiveSkill" in the file.
3. Change the color to "color:#FFF".

[CSS] Changing the Selected Row Style

```
.SelectedRow_ActiveSkill
{
background:url('Grid/sprite.gif') 0 -300px repeat-x #343434;
/* RadControls Step by Step Tutorial */
color:#FFF;
}
```

The new style should make the font for the selected row pop out a little more like the example below.



Test the Control

Run the application and test the "page":

- Add, edit and delete exams. Notice that the record displays in a pop-up due to the EditMode property setting.
- Filter questions using the CategoriesTree control.
- Check mark some of the questions to include them in an exam.
- Switch selection between exams to see the effect on the checkboxes. If you are on "Exam A" and check "Question 1", move to "Exam B" and back to "Exam A", the check mark should persist.

38.7 Summary

In this chapter you built complete maintenance functionality for categories, questions and exam related tables. You used the RadGrid heavily to leverage its powerful CRUD handling abilities, creating both master-detail in a single grid and in two related grids. You used RadControls within a standard ASP.NET FormView along with Eval () and Bind() binding expressions. You also built a user control that combined the RadComboBox with the RadTreeView and reused your control throughout the application.

39 ActiveSkill: User Functionality

39.1 Objectives

- Build the exam taking functionality.
- Use JavaScript objects to wrap client code. This will include using MS AJAX Library functionality including registering of namespaces, classes, inheritance and events.
- Bind RadGrid data on the client.
- Consume web services on the client.
- Use LINQ to SQL to retrieve the exam data.

39.2 Build the User Home Page

The user portion of ActiveSkill is made up of two main functions: Exam taking and scheduling. The exam functionality actually consists of three pages. A listing of exam summaries showing title, description and pass percent, a page for displaying and taking input for each question of the exam and finally a "finish" page that summaries the exam results and displays the score by category in a RadChart. The general steps for this stage of the application will be:

- Add the user controls for each "page". Implement the IASControl interface as we did in the Admin page.
- Define the user home page markup and code behind. This step will be similar to the Admin page, but we will trigger the page change from a client event that triggers an AjaxRequest event. Using the client event/AjaxRequest combination will let us change user controls from just about any user interface trigger, including from the RadGrid row click, the RadTabStrip tab click and from standard HTML buttons firing JavaScript. We will wrap the JavaScript required for this mechanism neatly in a JavaScript object.
- Add the code-behind for the user home page. This again will be quite similar to the Admin home page, but the server TabClick event will be replaced with a AjaxRequest server event. We will also add the ability to pass arguments that travel from the client, through the AjaxRequest event and passed along to the IASControl FirstLoad() method. For example, when the user clicks the row of the grid containing an exam, the exam ID is picked up and passed to the AjaxRequest to the "question" page FirstLoad() where the exam ID is retrieved and used to populate a data structure with the data for the entire exam.



You can find the complete source for this project at:
\\VS Projects\ActiveSkill Adding User Functionality\CS\001

Add User Controls

Add four User Controls to the \User directory of the ActiveSkillUI project. These will represent the four "pages" of user functionality.

1. Add the first control, naming it "TakeExamChoose.ascx". In the code-behind, implement the IASControl interface as we did in the Admin Page chapter. Add "Telerik.ActiveSkill.Common" to the "Imports" (VB) or "uses" (C#) section of the code. The code should look like the example below:

[VB] Implementing IASControl

```
Imports System
Imports System.Collections.Generic
Imports System.Linq
Imports System.Web
Imports System.Web.UI
Imports System.Web.UI.WebControls
```

```
Imports Telerik.ActiveSkill.Common
Namespace Telerik.ActiveSkill.UI.User
    Public Partial Class TakeExamChoose
        Inherits System.Web.UI.UserControl
        Implements IASControl
        Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
        End Sub
        #region IASControl Members
        Public Sub FirstLoad(ByVal args As Dictionary(Of String, String))
        End Sub
    #End Region
End Class
End Namespace
```

[C#] Implementing IASControl

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using Telerik.ActiveSkill.Common;
namespace Telerik.ActiveSkill.UI.User
{
    public partial class TakeExamChoose : System.Web.UI.UserControl, IASControl
    {
        protected void Page_Load(object sender, EventArgs e)
        {
        }
        #region IASControl Members
        public void FirstLoad(Dictionary<string, string> args)
        {
        }
        #endregion
    }
}
```

2. Repeat this step for the following controls: TakeExamQuestion.ascx, TakeExamFinish.ascx, ScheduleExams.ascx.

Trigger Page Changes Client Side

The dynamic changing of user control "pages" will be handled by a JavaScript object we will call "DynamicControl".

1. Add a new JScript item to the ActiveSkillUI project \Scripts directory. Name it "DynamicControl.js".
2. Add the code below to the DynamicControl.js file. *Notice that we're using the Microsoft AJAX Library Type object to register a "ActiveSkill" namespace. By pre-pending all of our objects with "ActiveSkill." we make the object distinct from others that might appear in a large project. The constructor simply saves a reference to the RadAjaxManager for use later. The built-in "prototype" object is used to tack on properties and methods of our DynamicControl object. We add only a single method "load()" that formats the arguments and triggers the ajax request. When all is said and done, we call notifyScriptLoaded() to notify the ScriptManager that the JavaScript is loaded.*

[JavaScript]

```
Type.registerNamespace("ActiveSkill");
// DynamicControl wraps the ChangeControl() JavaScript
// method used in the Admin page.
/* -- DynamicControl constructor -- */
ActiveSkill.DynamicControl = function(ajaxManager)
{
    // save a reference to the AjaxManager
    this._ajaxManager = ajaxManager;
}
ActiveSkill.DynamicControl.prototype = {
// the load() method passes a path to a user control
// and any arguments specified.
load: function(path, args)
{
    var ajaxArgs = "&ControlName=" + path + args;
    this._ajaxManager.ajaxRequest(ajaxArgs);
}
}

// notify script manager that this js is loaded
if (typeof (Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();
```

3. For JavaScript handled through the ScriptManager, we must add a reference to the script. You can do this using the ScriptManager Scripts collection or add it to the markup:

[ASP.NET] Adding the Script Reference

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Scripts>
    <asp:ScriptReference Path="~/scripts/DynamicControl.js" />
  </Scripts>
</asp:ScriptManager>
```



Gotcha! Do *not* include a standard JavaScript tag to include your js file. It will interfere with the ScriptManager loading your JavaScript and prevent Microsoft AJAX Library functions from being loaded. For example, the "Type" object would not be recognized. Load your js file once only, using a ScriptManager reference.

Later we will replace our client methods with a call to this object with code similar to the example below:

[JavaScript] DynamicControl Usage Example

```
function pageLoad() {
    if (window.DynamicControl == null) {
        window.DynamicControl = new ActiveSkill.DynamicControl(
            $find("<%= RadAjaxManager1.ClientID %>"));
    }
}
function MyFunction(myArgs) {
    window.DynamicControl.load('MyUserControlPath.ascx', myArgs);
}
```

Define User Home Page Markup

1. Copy the markup from "AdminHome.aspx" to "UserHome.aspx".
2. Change the RadSlidingZone "MasterSlidingPane" DockText property to "My Skills".
3. Modify the RadTabStrip "tsMain".

- Change the RadTabStrip "tsMain" tabs collection to point at our new user controls "TakeExamChoose" and "ScheduleExams".
- The tsMain ClickSelectedTab property should be set to true. *Setting ClickSelectedTab to true will allow us to click the tab and have the click events execute even when the clicked tab is already selected. This is the behavior we want if we're in the middle of an exam and want to choose a new exam.*
- Add a OnClientTabSelected event handler and name it "ClientTabSelected". Eliminate the server TabClick event. The markup for tsMain should look like the example below.

[ASP.NET] Changing the TabStrip

```
<telerik:RadTabStrip ID="tsMain" runat="server" Orientation="VerticalRight"
ClickSelectedTab="true"
SelectedIndex="0" OnClientTabSelected="ClientTabSelected">
  <Tabs>
    <telerik:RadTab runat="server" Text="Take Exam" Value="TakeExamChoose.ascx"
      ImageUrl="~/images/Exams.png">
    </telerik:RadTab>
    <telerik:RadTab runat="server" Text="Schedule" Value="ScheduleExams.ascx"
      ImageUrl="~/images/Schedule.png">
    </telerik:RadTab>
  </Tabs>
</telerik:RadTabStrip>
```

4. Add this block of JavaScript code just inside the <body> tag. *pageLoad() is an event that fires courtesy of having the ScriptManager on the page. This provides an opportunity to create an instance of our DynamicControl object that can be used everywhere in the exam pages. The OnClientTabSelected event handler fires when the user clicks a tab, which in turns causes DynamicControl to load a new user control.*

[JavaScript] Handling the PageLoad and the OnClientTabSelected Events

```
<telerik:RadScriptBlock ID="RadScriptBlock1" runat="server">
  <script type="text/javascript">
    /* -- Client event handlers -- */
    function pageLoad() {
      // if there isn't a global instance of DynamicControl,
      // create it and pass a reference to the RadAjaxManager client
      // object.
      if (window.DynamicControl == null) {
        window.DynamicControl = new ActiveSkill.DynamicControl(
          $find("<%= RadAjaxManager1.ClientID %>"));
      }
    }
    function ClientTabSelected(sender, args) {
      // use the DynamicControl load() method to swap user controls.
      window.DynamicControl.load(args.get_tab().get_value(), '');
    }
  </script>
</telerik:RadScriptBlock>
```

5. Add references to the ScriptManager. This can be done either in the Properties window using the Services and Scripts collections or within the markup. *Later, the TakeExamQuestion.ascx control will be consuming a web service and several JavaScript files. The web service path is the same path that can be used directly in a browser to display the methods available for the server and to test that the web service is working.*

[ASP.NET] Adding ScriptManager References

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="http://localhost/ActiveSkillWS/service1.asmx" />
  </Services>
  <Scripts>
    <asp:ScriptReference Path="~/scripts/DynamicControl.js" />
    <asp:ScriptReference Path="~/scripts/ExamManager.js" />
    <asp:ScriptReference Path="~/scripts/UIManager.js" />
  </Scripts>
</asp:ScriptManager>
```

Add User Home Code-Behind

Copy over the logic from AdminHome.aspx. This logic will remain the same except for the following:

1. Replace the LoadUserControl() method with a new version that passes a dictionary of arguments:

[VB] Loading the User Control

```
' The same LoadUserControl method as defined previously, but passes
' a dictionary object. You must also include a reference to System.Collections.Generic.
' When FirstLoad() is called, the dictionary is passed to the user control.
```

```
Public Function LoadUserControl(ByVal parentControl As Control, ByVal newControlPath As
String, ByVal isFirstLoad As Boolean, ByVal args As Dictionary(Of String, String)) As
Control
  Dim control As Control = Page.LoadControl(newControlPath)
  control.ID = newControlPath
  If isFirstLoad Then
    control.EnableViewState = False
  End If
  parentControl.Controls.Clear()
  parentControl.Controls.Add(control)
  If isFirstLoad Then
    control.EnableViewState = True
    (TryCast(control, IASControl)).FirstLoad(args)
  End If
  Return control
End Function
```

[C#] Loading the User Control

```
// The same LoadUserControl method as defined previously, but passes
// a dictionary object. You must also include a reference to System.Collections.Generic.
// When FirstLoad() is called, the dictionary is passed to the user control.
public Control LoadUserControl(Control parentControl, string newControlPath, bool
isFirstLoad,
Dictionary<string, string> args)
{
  Control control = Page.LoadControl(newControlPath);
  control.ID = newControlPath;
  if (isFirstLoad)
  {
    control.EnableViewState = false;
  }
}
```

```

parentControl.Controls.Clear();
parentControl.Controls.Add(control);
if (isFirstLoad)
{
    control.EnableViewState = true;
    (control as IASControl).FirstLoad(args);
}
return control;
}

```

2. Be sure that System.Collections.Generic is added to the "Imports" (VB) or "uses" (C#) section of the code. This will support the Dictionary object.
3. Remove the old TabClick event handler and replace it with a AjaxRequest event handler. Also add a new utility method "UnpackArgs()" to convert a string of "&" delimited name/value pairs to a Dictionary object. Again, the logic is similar to the AdminPage version except that its triggered through the AjaxRequest event which passes an Argument property. The client packages up the args to look something like "&ControlName=TakeExamQuestion.ascx&id=123". This string is broken apart and repackaged as a dictionary and passed to FirstLoad() where it can be addressed using the name of each element, e.g. "args ["id"];"

[VB] Handling the AjaxRequest

```

' Convert a string passed from the client to a Dictionary
' and call LoadUserControl with the dictionary of arguments
Protected Sub RadAjaxManager1_AjaxRequest(ByVal sender As Object, ByVal e As
Telerik.Web.UI.AjaxRequestEventArgs)
    Dim args As Dictionary(Of String, String) = UnpackArgs(e.Argument)
    CurrentControl = args("ControlName")
    LoadUserControl(Placeholder1, CurrentControl, True, args)
End Sub
' Take a string formatted somewhat like a query string,
' i.e. delimited with "&", break it up into an array
' parse and load a Dictionary object with key and value
' pairs.
Public Function UnpackArgs(ByVal arguments As String) As Dictionary(Of String, String)
    Dim result As New Dictionary(Of String, String)()
    Dim split As String() = arguments.Split("&"C)
    For Each str As String In split
        Dim pair As String() = str.Split("="C)
        If pair.Length = 2 Then
            If result.ContainsKey(pair(0)) Then
                result(pair(0)) = pair(1)
            Else
                result.Add(pair(0), pair(1))
            End If
        End If
    Next
    Return result
End Function

```

[C#] Handling the AjaxRequest

```

// Convert a string passed from the client to a Dictionary
// and call LoadUserControl with the dictionary of arguments
protected void RadAjaxManager1_AjaxRequest(object sender,
Telerik.Web.UI.AjaxRequestEventArgs e)
{

```

```
Dictionary<string, string> args = UnpackArgs(e.Argument);
CurrentControl = args["ControlName"];
LoadUserControl(Placeholder1, CurrentControl, true, args);
}
// Take a string formatted somewhat like a query string,
// i.e. delimited with "&", break it up into an array
// parse and load a Dictionary object with key and value
// pairs.
public Dictionary<string, string> UnpackArgs(string arguments)
{
    Dictionary<string, string> result = new Dictionary<string, string>();
    string[] split = arguments.Split('&');
    foreach (string str in split)
    {
        string[] pair = str.Split('=');
        if (pair.Length == 2)
        {
            if (result.ContainsKey(pair[0]))
                result[pair[0]] = pair[1];
            else
                result.Add(pair[0], pair[1]);
        }
    }
    return result;
}
```

Test the Navigation and Parameters

Test the application so far. You will want to be sure that navigation is working and that parameters are being passed:

1. Add the following button element to the "TakeExamChoose.ascx" markup that will navigate and pass parameters to the TakeExamQuestion page.

[ASP.NET] Adding Stub Client

```
<input id="Button1" type="button" value="button"
onclick="window.DynamicControl.load(
'TakeExamQuestion.ascx',
'&ControlName=TakeExamQuestion.ascx&id=123');" />
```

2. Add a literal and a TextBox to the TakeExamQuestion control:

[ASP.NET] Adding a TextBox

```
question
<asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>
```

3. Add code to the FirstLoad() method that populates the textbox with parameters sent from TakeExamQuestions.ascx.

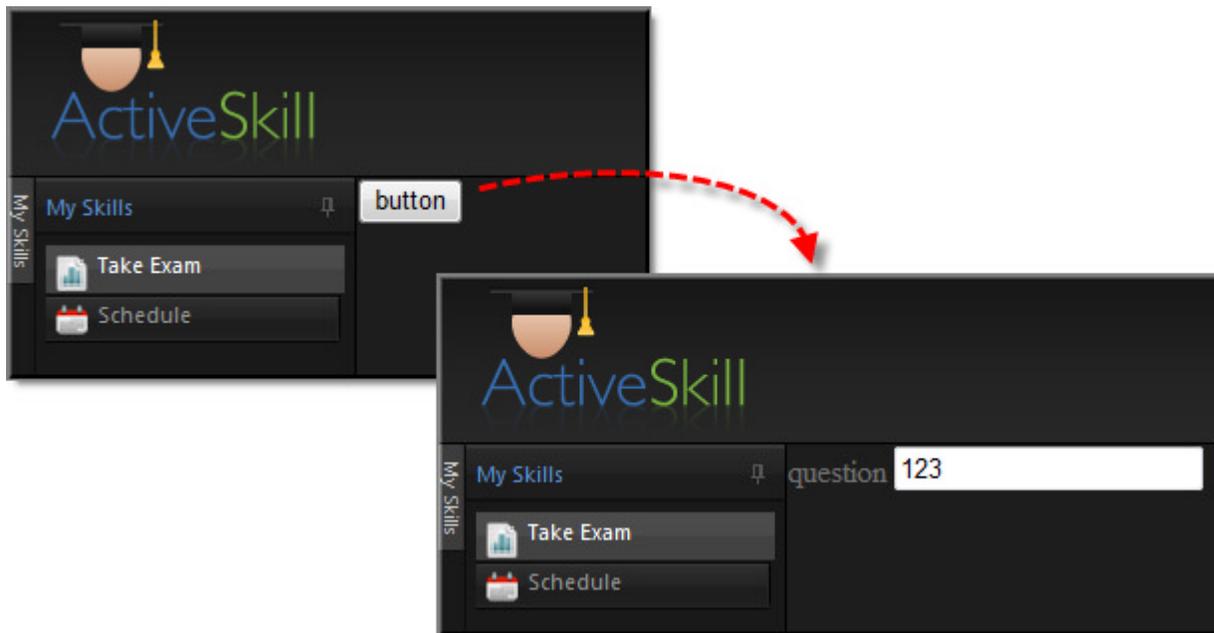
[VB] Passing Args

```
Public Sub FirstLoad(ByVal args As Dictionary(Of String, String))
    TextBox1.Text = args("id")
End Sub
```

[C#] Passing Args

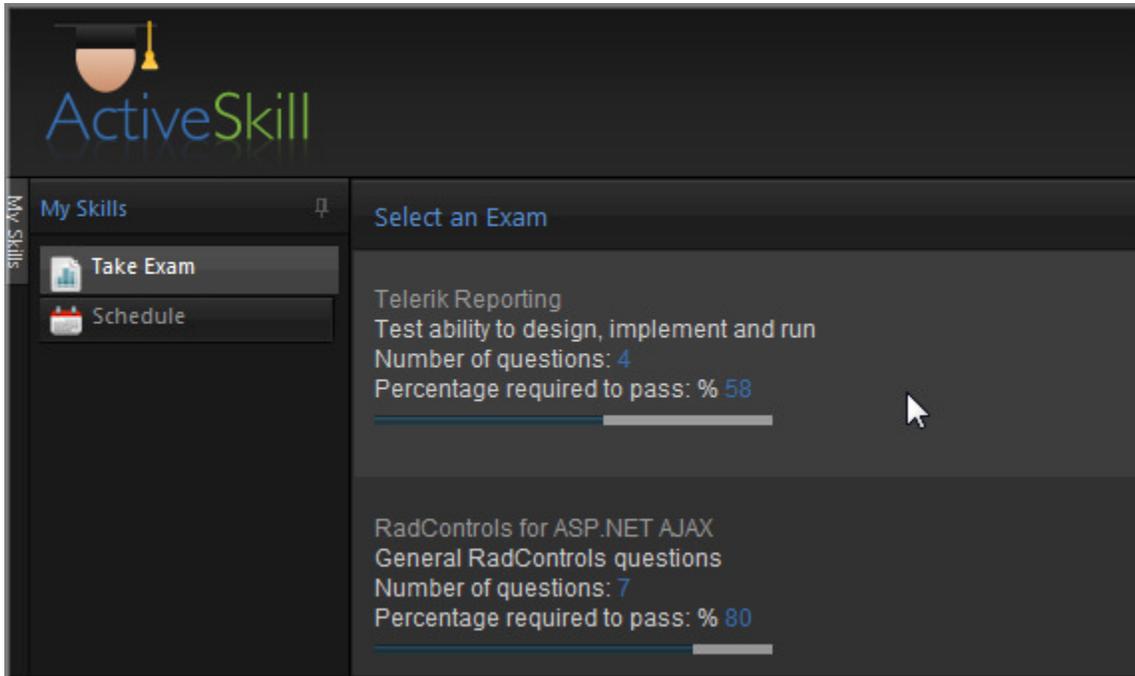
```
public void FirstLoad(Dictionary<string, string> args)
{
    TextBox1.Text = args["id"];
}
```

4. In the Solution Explorer, right-click the ActiveSkillUI project and select **Set as Startup Project** from the context menu. Right-click UserHome.aspx and select **Set as Start Page** from the context menu. Press **Ctrl-F5** to run the application.



39.3 Build the Choose Exam Control

Now we can fill out the Choose Exam control using a RadGrid and a template. Each row only has a single column, but the template allows us to arrange bound controls any way we like. The finished control will display exam information to appear like the example screenshot below. You'll notice that the row with the mouse over it is slightly highlighted.



You can find the complete source for this project at:
VS Projects\ActiveSkill Adding User Functionality\CS\002

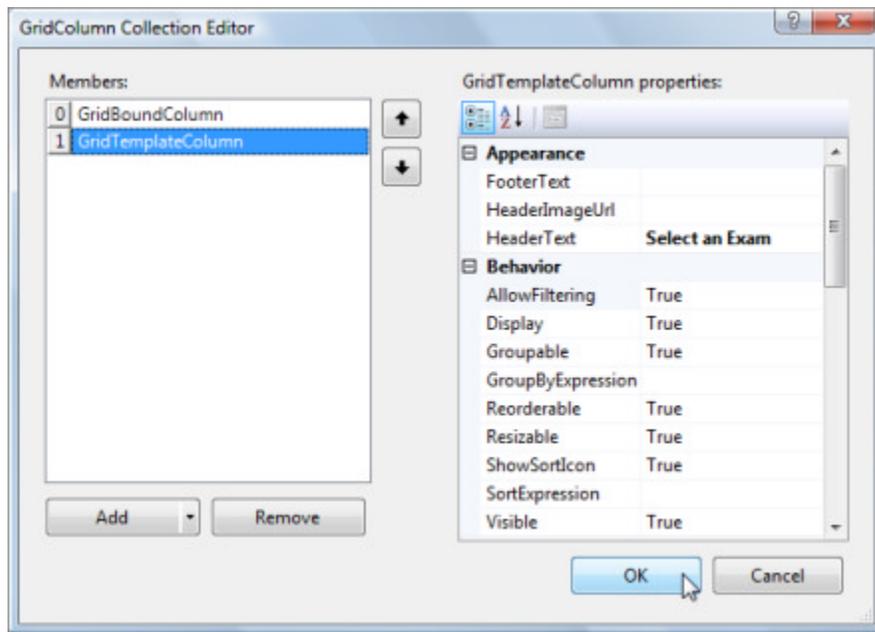
1. Add a `SqlDataSource` to the control. Set the `ID` property to be "dsExam". Point the `ConnectionString` property at the `ActiveSkillConnectionString` we've been using thus far. Set the `SelectCommand` property to "Skill_Exam_Select" and `SelectCommandType` to "StoredProcedure". You can configure the data source in the designer or in the markup. The markup should look something like the example below when you're finished.

[ASP.NET] Defining the `SqlDataSource`

```
<!--Data sources --%>  
<asp:SqlDataSource ID="dsExam" runat="server"  
  ConnectionString="<%$ ConnectionStrings:ActiveSkillConnectionString %>"  
  SelectCommand="Skill_Exam_Select" SelectCommandType="StoredProcedure">  
</asp:SqlDataSource>
```

2. Add a `RadGrid` to the control. Set the `ID` property of the grid to "gridExam". Set the other `RadGrid` properties:
 - o `AllowPaging`: true.
 - o `AutoGenerateColumns`: false.
 - o `GridLines`: None.
 - o `PageSize`: 3.
3. In the `MasterTableView` property for the grid:
 - o `ClientDataKeyNames`: "id"
 - o `DataKeyNames`: "ID"

4. In the Properties window, click the MasterTableView **Columns** property ellipses. Add two columns:
 - A GridBoundColumn with **DataField** property "ID" and **Visible** property "false".
 - A GridTemplateColumn with **UniqueName** "TemplateColumn" and **HeaderText** "Select an Exam".

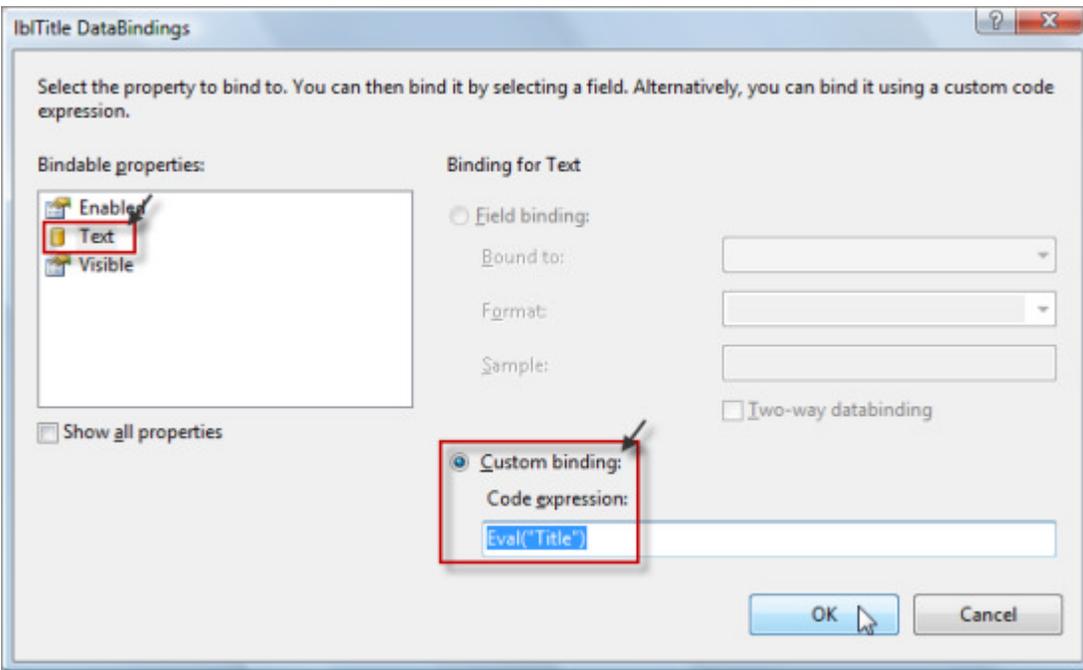


5. In the **ClientSettings** property of the grid:
 - **EnableRowHoverStyle**: true.
 - **ClientEvents.OnRowSelected**: "RowSelected".
 - **Selecting.AllowRowSelect**: true.

 **EnableRowHoverStyle** when true highlights the row that the mouse is currently over. If **Selecting.AllowRowSelect** is true, you can still select rows on the client side, but there's no visual feedback as the mouse passes over. Finally, you need to respond to the **ClientEvents.OnRowSelected** event. You can use the args passed to the event to get at the key value for the row (specified by **ClientDataKeyNames**):

```
mykeyvalue = args.GetDataKeyValue("id");
```

6. From the Smart Tag select **Edit Templates**. Using the Smart Tag menu, navigate to the **ItemTemplate**.
7. Drop a standard ASP.NET Label control on the ItemTemplate. Set the **ID** property to "lblTitle" and **CssClass** property to "skillTitle". From the label's Smart Tag select **Edit Bindings...** Provide a custom data binding 'Eval("Title")' to the **Text** property of the label:



8. Use the enter key to add a hard break (
) after this label.
9. Drop five more Labels in the template and set the properties as follows:
 - ID: "lblDescription", **CssClass:** "skillNormal", **Text** binding expression: '<%# Eval("Description") %>'. Use the enter key to add a hard break (
) after this label.
 - **CssClass:** "skillNormal", **Text:** 'Number of questions:'.
 - ID: "lblNumberOfQuestions", **CssClass:** "skillBlue", **Text** binding expression: '<%# Eval("NumberOfQuestions") %>'. Use the enter key to add a hard break (
) after this label.
 - **CssClass:** "skillNormal", **Text:** "Percentage required to pass: %".
 - ID: "lblPercent", **CssClass:** "skillNormal", **Text** binding expression: '<%# Eval("PassPercent") %>'. Use the enter key to add a hard break (
) after this label.
10. Add a RadSlider after the last label. Set the ID property to "sldPassPercent", Enabled to "false", MaximumValue to "100" and MinimumValue to "0". Set properties ShowDecreaseHandle, ShowDraghandle and ShowIncreaseHandle all to "false". Bind the Value property to the PassPercent. You can use the **Edit Bindings...** dialog in the same way you did with the previous Label controls, but in this case, check the "Show all Properties" checkbox, locate Value in the Bindable Properties list, then set the custom binding to '<%# Eval("PassPercent") %>'. You can also populate the bindings directly in the markup.

The markup for the grid should look like the example below.

[ASP.NET] The RadGrid Definition

```
<!--Grid-->
<telerik:RadGrid ID="gridExam" runat="server" AutoGenerateColumns="False"
GridLines="None" OnNeedDataSource="gridExam_NeedDataSource"
PageSize="3" AllowPaging="True">
<PagerStyle FirstPageImageUrl=" ../Skins/ActiveSkill/Grid/PagingFirst.gif"
LastPageImageUrl=" ../Skins/ActiveSkill/Grid/PagingLast.gif"
NextPageImageUrl=" ../Skins/ActiveSkill/Grid/PagingNext.gif"
PrevPageImageUrl=" ../Skins/ActiveSkill/Grid/PagingPrev.gif" />
<MasterTableView ClientDataKeyNames="id" DataKeyNames="ID">
<Columns>
```

```

<telerik:GridBoundColumn DataField="ID" UniqueName="column1" Visible="False">
</telerik:GridBoundColumn>
<telerik:GridTemplateColumn UniqueName="TemplateColumn" HeaderText="Select an Exam":
  <ItemTemplate>
    <br />
    <asp:Label ID="lblTitle" runat="server" CssClass="skillTitle"
      Text='<%=# Eval("Title") %>'></asp:Label>
    <br />
    <asp:Label ID="lblDescription" runat="server" CssClass="skillNormal"
      Text='<%=# Eval("Description") %>'></asp:Label>
    <br />
    <asp:Label ID="Label1" runat="server" CssClass="skillNormal"
      Text="Number of questions:"></asp:Label>
    <asp:Label ID="lblNumberOfQuestions" runat="server"
      CssClass="skillBlue" Text='<%=# Eval("NumberOfQuestions") %>'></asp:Label>
    <br />
    <asp:Label ID="lblPassPercent" runat="server" CssClass="skillNormal"
      Text="Percentage required to pass: %"></asp:Label>
    <asp:Label ID="lblPercent" runat="server" CssClass="skillBlue"
      Text='<%=# Eval("PassPercent") %>'></asp:Label>
    <telerik:RadSlider ID="sldPassPercent" runat="server"
      Enabled="False" MaximumValue="100" MinimumValue="0"
      ShowDecreaseHandle="False" ShowDragHandle="False"
      ShowIncreaseHandle="False" Value='<%=# Eval("PassPercent") %>'
      />
    <br />
  </ItemTemplate>
</telerik:GridTemplateColumn>
</Columns>
</MasterTableView>
<ClientSettings EnableRowHoverStyle="True">
  <ClientEvents OnRowSelected="RowSelected" />
  <Selecting AllowRowSelect="True" />
</ClientSettings>
</telerik:RadGrid>

```

11. In the code-behind for TakeExamChoose.ascx, add a NeedDataSource event handler. *This will simply assign the "dsExam" data source to pull in all of the Exam table records.*

[VB] Handling the Grid NeedDataSource Event

```

Protected Sub gridExam_NeedDataSource(ByVal source As Object, ByVal e As
Telerik.Web.UI.GridNeedDataSourceEventArgs)
  (TryCast(source, RadGrid)).DataSource = dsExam
End Sub

```

[C#] Handling the Grid NeedDataSource Event

```

protected void gridExam_NeedDataSource(object source,
Telerik.Web.UI.GridNeedDataSourceEventArgs e)
{
  (source as RadGrid).DataSource = dsExam;
}

```

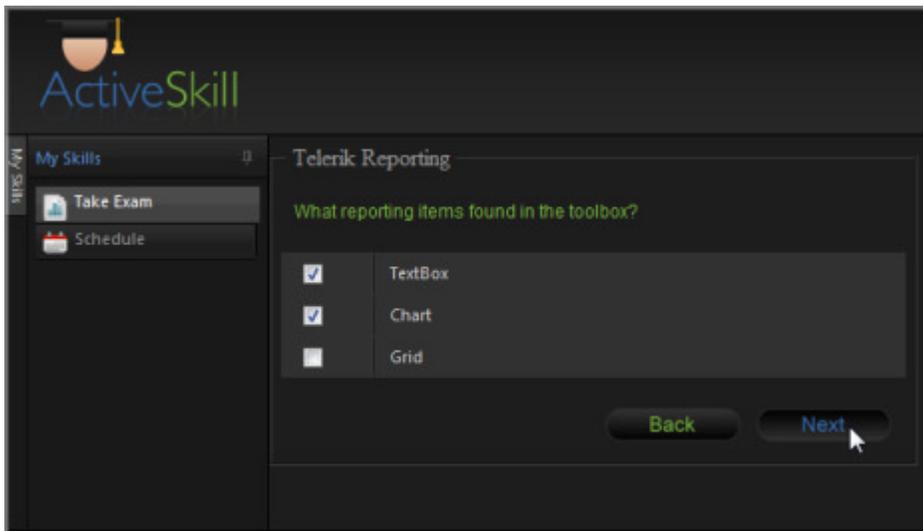
12. Press Ctl-F5 to run the example. You should be able to select an exam by clicking one of the rows. The TakeExamQuestion.ascx control should display and be populated with the ID for the selected exam.

39.4 Build the Exam Question Control

Part of the purpose for the exam taking controls is to demonstrate how we can work entirely on the client side without coming up for air. When the Exam Question control first loads, the only thing that happens on the server is to take the exam id from the Dictionary arguments passed in and stuff it in a hidden field on the page. From there on, it's all client side. Even binding the data to the grid. When the exam question page loads on the client, a web service is called that returns the exam record, all of the questions and all of the responses for each question. In addition, we tack on a response column to store the user's choices as they take the exam.

We're also going to use the MS AJAX Library heavily in this example to wrap the functionality surrounding the exam. Not just the exam itself, but where we are in the exam, navigating forward and back through the questions and returning results when we complete. We also use a JavaScript object to wrap the UI elements of the page in one convenient spot.

The user interface for the question "page" will display a FieldSet with the title of the exam in the legend, the question text, a series of responses and back/next buttons.



The general steps to build this page are:

- Build and test the web service.
- Define the markup for the exam question control.
- Create the ExamManager client object.
- Create the UIManager client object.
- Add client code to the exam question control that will consume the new client objects.
- Code the FirstLoad() method in the code behind.
- Test taking an exam all the way through to navigation on to the "finish" control.

Build and Test the Web Service.

The web service will be minimal and have only a single method that returns all the data for a single exam. The MS AJAX Library provides infrastructure so that properly configured web services serialize results into JSON, which in turn is immediately usable in client code. We will use a LINQ to SQL item that will automatically generate the classes we need to easily retrieve the data with a few lines of code.

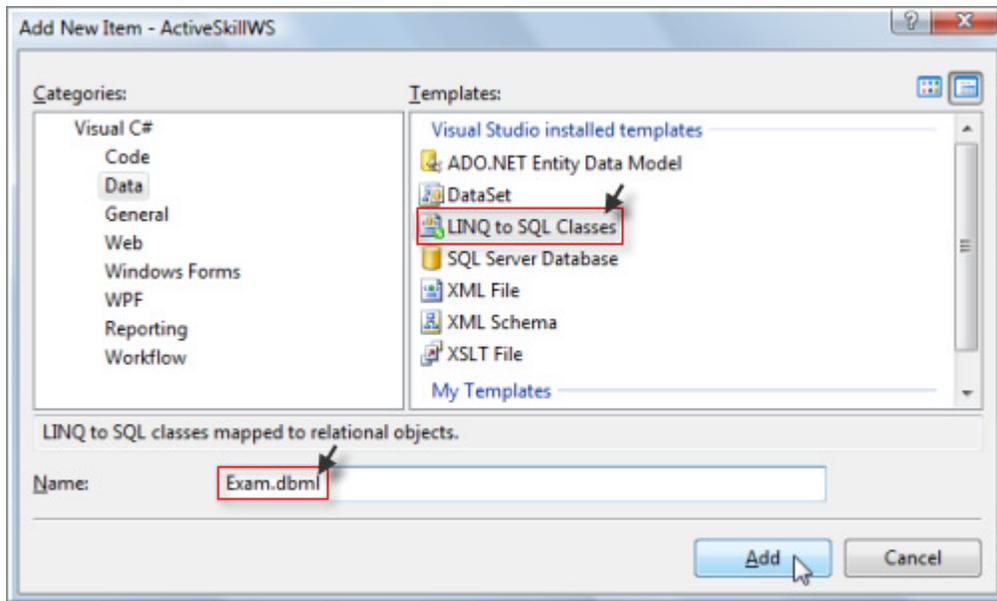
When we first set up the ActiveSkill solution, we created an ActiveSkillWS project that we will use as a starting point.

1. In the ActiveSkillWS web.config file, add the ActiveSkill connection string. If the connectionStrings element already exists be sure to replace it.

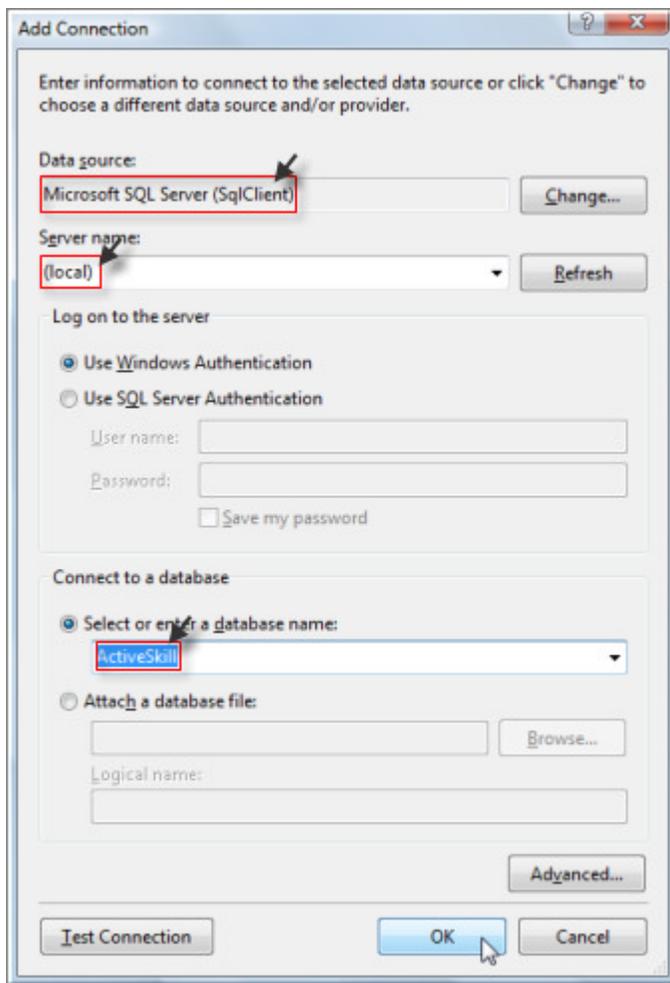
[XML] Adding the Connection String

```
<connectionStrings>
  <add name="ActiveSkillConnectionString"
        connectionString="Data Source=localhost;Initial Catalog=ActiveSkill;Integrated
Security=True"
        providerName="System.Data.SqlClient"/>
</connectionStrings>
```

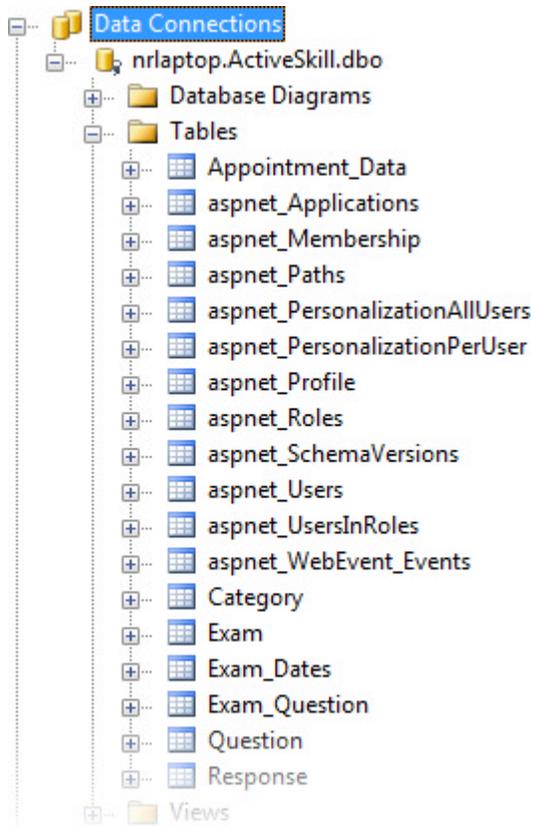
- Right-click the ActiveSkillWS project and select **Add | New Item** from the context menu. In the Add New Item dialog that displays, choose "LINQ to SQL Classes" and name the item "Exam.dbml".



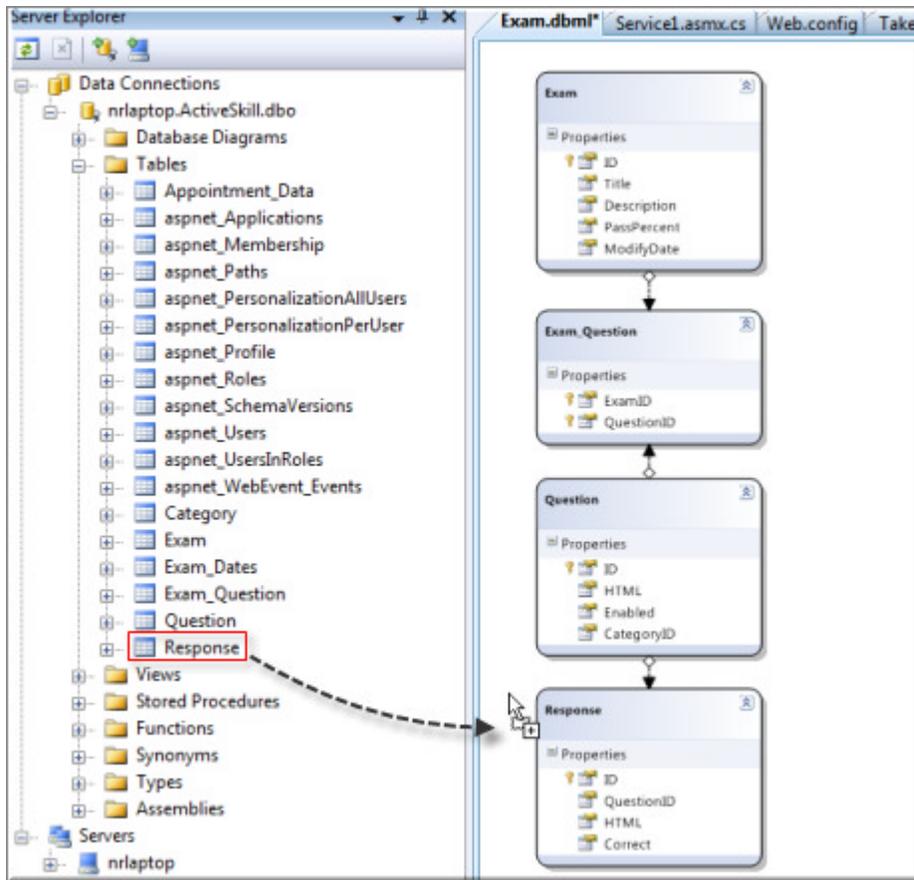
- A design surface will display to allow the new item to be configured. From the View menu in Visual Studio, select **Server Explorer**. Right-click the Data Connections node and select **Add Connection**. In the Add Connection dialog that displays, click the **Change** button to display the Change Data Source dialog, select "SQL Server" and click **OK** to close the dialog. Back in the Add Connection dialog, enter the name of the server where the ActiveSkill database resides. For example, this might be "(local)" for your local SQL server or "(local)SQLEXPRESS" if you have only SQLEXPRESS installed. Click **OK** to close the Add Connection dialog.



4. Now you should be able to open the list of tables that will include the ActiveSkill tables for Exam, Exam_Question, Question and Response.

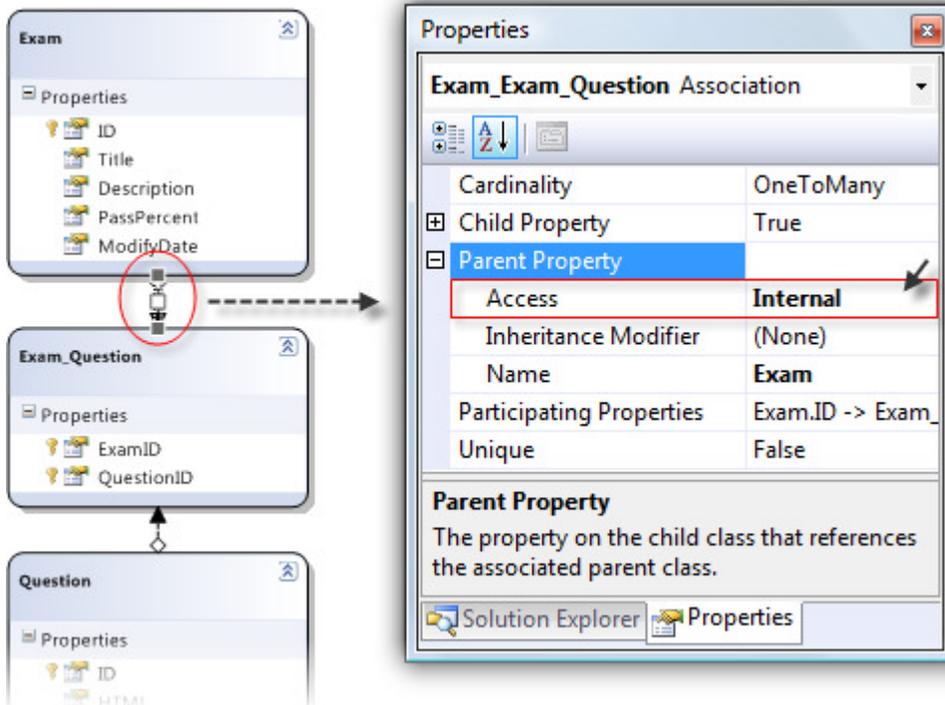


5. Drag each of the tables from the server explorer to the design surface: Exam, Exam_Questions, Question and Response. Notice that the relationships are already defined between these tables and denoted by the arrows pointing between tables.



✎ The relationships between tables allow much more interaction on the server than when they are serialized. When serialized the rules become stricter and the possibility of "Circular reference" errors are more likely.

6. Click on the relationship arrow between the Exam and Exam_Questions. In the properties window set the Parent Property Access to "Internal"



7. Click the Exam_Question to Question relationship arrow. Set the **Child Property** to "False".
8. Click the Question to Response relationship arrow. Set the **Parent Property Access** to "Internal".

We will be looking at the data output by these entities in XML form as we test the web service. You may want to play with these relationships and see how they affect data.

9. Open Service1.asmx (added by default when we first created the web service project) and add the code below. Notice the "ScriptService" attribute. ScriptService lets the web service be consumed by a client script. This happens automatically, just by adding the attribute. We will see in a moment how a JavaScript proxy class is generated for us. Also notice how the LINQ code in the GetExam() method is brief and to the point. Older versions of this same code without LINQ were quite wordy in comparison. We end up with an Exam object that was defined for us when we created the Exam.DBML.

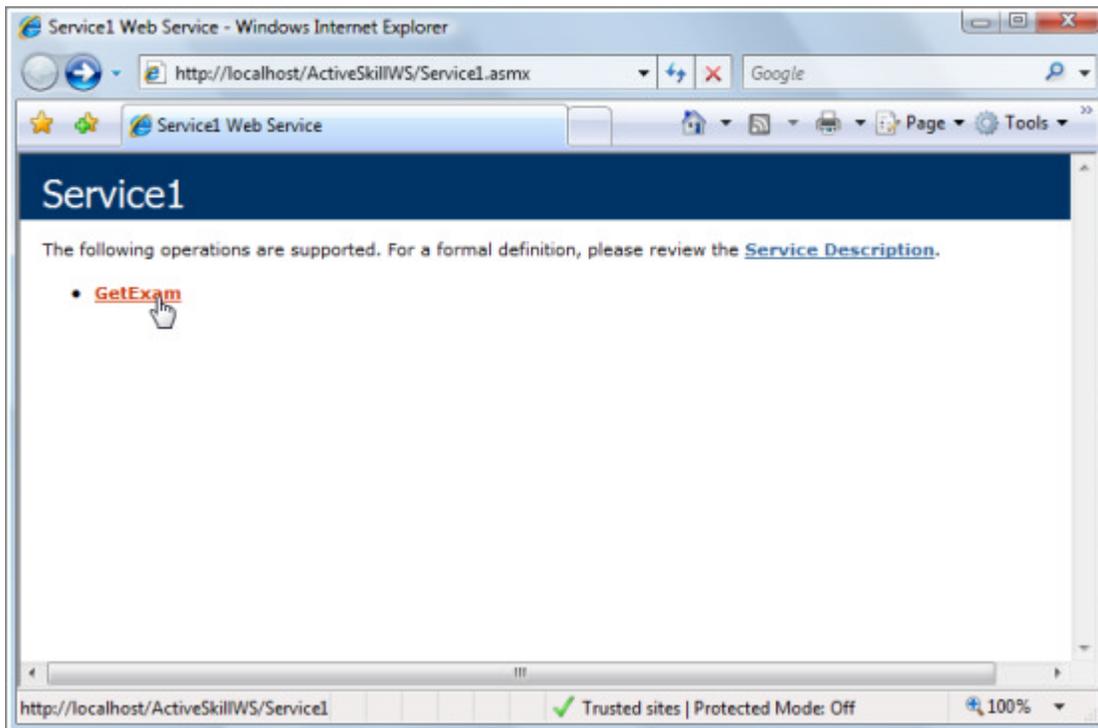
[VB] Defining the Web Service

[C#] Defining the Web Service

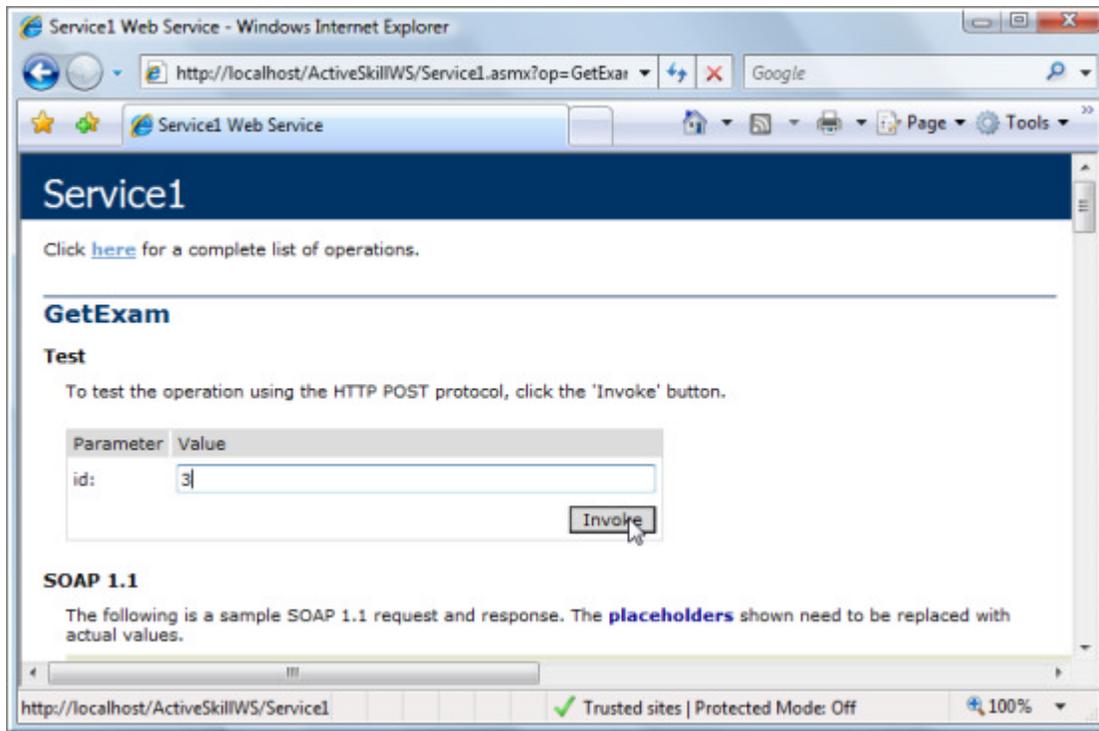
```
using System.Configuration;
using System.Data.Linq;
using System.Linq;
using System.Web.Services;
using System.Web.UI.WebControls;
namespace ActiveSkillWS
{
    [WebService(Namespace = "http://www.telerik.com (http://www.telerik.com/)")]
    [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
    [System.ComponentModel.ToolboxItem(false)]
    [System.Web.Script.Services.ScriptService]
    public class Service1 : System.Web.Services.WebService
    {
        private string connectionString =
            ConfigurationManager.ConnectionStrings["ActiveSkillConnectionString"].ConnectionStr;
        [WebMethod]
```

```
public Exam GetExam(int id)
{
    // Create an instance of the DataContext class automatically
    // produced by defining Exam.dbml. Pass the ActiveSkill connection
    // string in the constructor.
    ExamDataContext examContext =
        new ExamDataContext(connectionString);
    // retrieve all exams
    Table<Exam> exams = examContext.Exams;
    // retrieve just the exam that matches the ID passed in.
    return exams.Single(e => e.ID == id) as Exam;
}
}
```

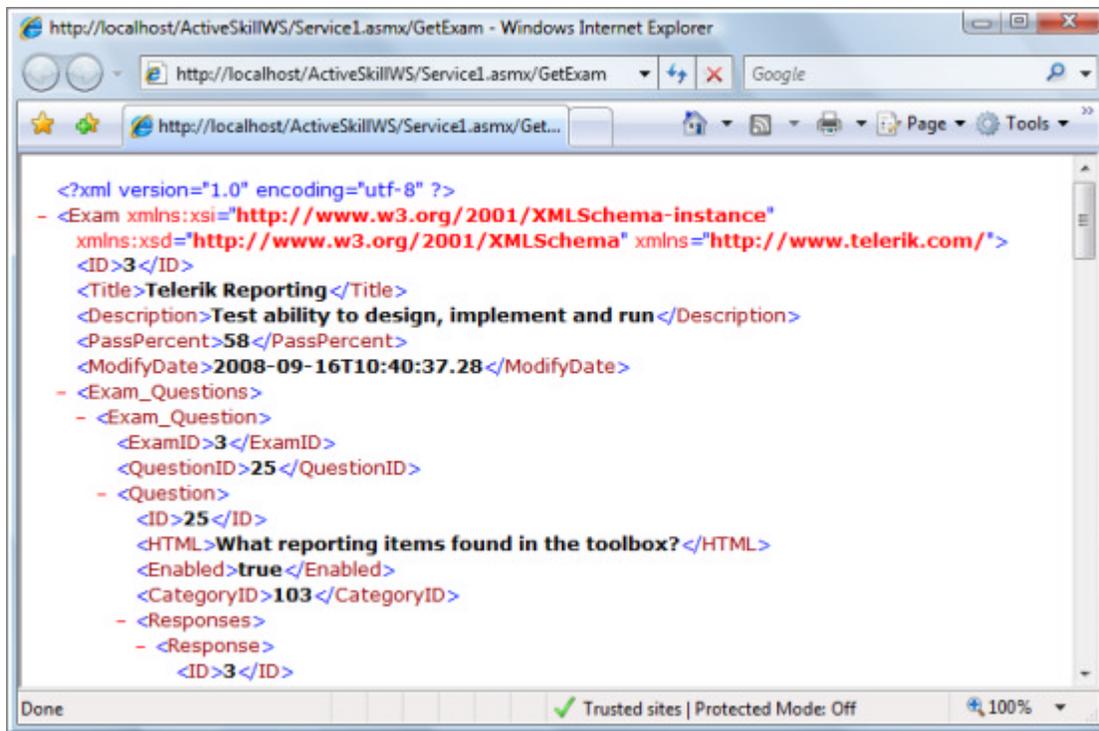
10. In the Solution Explorer, right-click the web service and set it to be the startup project. Press **F5** to run the service in the browser. This will display a default page listing our service methods, where we have only the one, "GetExam()".



11. Click GetExam() to display a page that allows us to test the service manually. Enter a "3" (a sample exam that should be available in the database) and click the **Invoke** button.



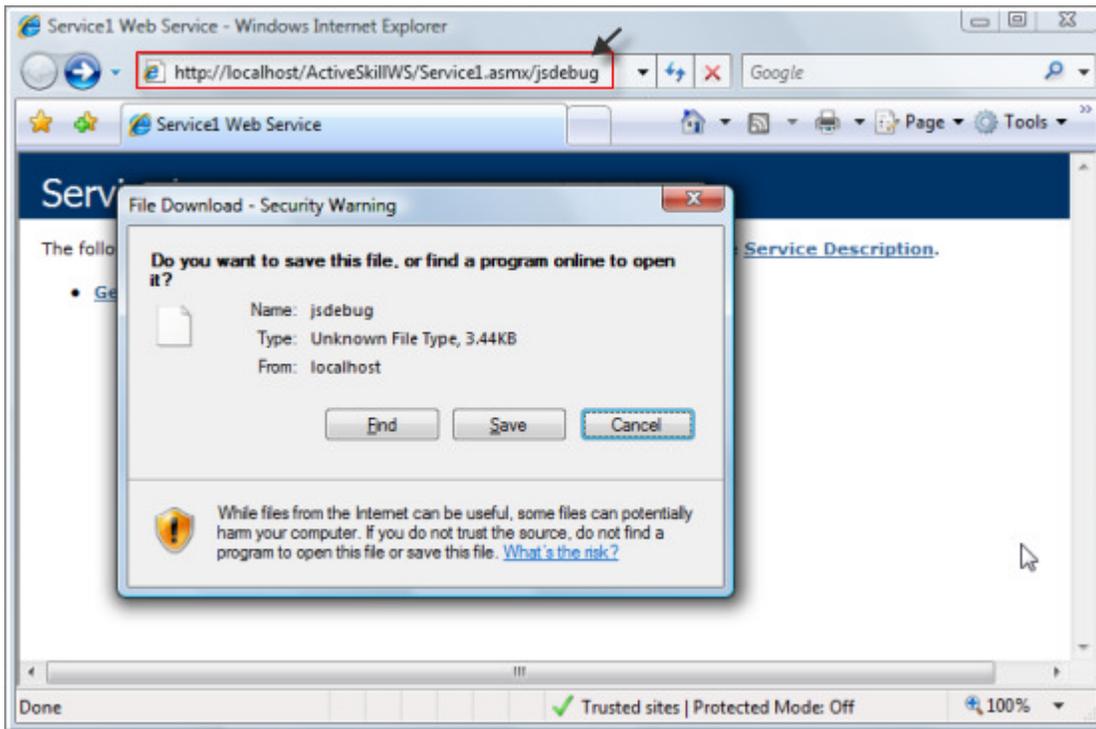
If our web service is working properly, the data should display as an XML/SOAP response where we can see the Exam information, followed by Exam_Questions that in turn contain Responses. All of this information will be available to us on the client.



The JavaScript Proxy

UI for ASP.NET AJAX

Before leaving the web service, look at the JavaScript proxy that's created automatically by the ScriptManager. You can see the file by using a special path to the web service. Add `\js` to the web service, or `\jsdebug` to return a commented version of the same code.



If the JavaScript is not displayed in the browser, save it locally for viewing. The commented version will look something like the code sample below. You can see that it registers a "ActiveSkillWS" namespace used to qualify the name of its one object "Service1". Service1 has some inherited behavior for defining the succeeded and failed callback functions, as well as path, timeout and user context properties. The GetExam() function is defined for us so that we simply need to call it from our JavaScript. Also be aware that if you don't manually include the JavaScript file in your application, ScriptManager will retrieve the file behind the scenes.

[JavaScript] /jsdebug Output from the Web Service

```
Type.registerNamespace('ActiveSkillWS');
ActiveSkillWS.Service1=function() {
ActiveSkillWS.Service1.initializeBase(this);
this._timeout = 0;
this._userContext = null;
this._succeeded = null;
this._failed = null;
}
ActiveSkillWS.Service1.prototype={
_get_path:function() {
var p = this.get_path();
if (p) return p;
else return ActiveSkillWS.Service1._staticInstance.get_path();},
GetExam:function(id,succeededCallback, failedCallback, userContext) {
/// <param name="id" type="Number">System.Int32</param>
/// <param name="succeededCallback" type="Function" optional="true"
mayBeNull="true"></param>
/// <param name="failedCallback" type="Function" optional="true" mayBeNull="true"></param>
/// <param name="userContext" optional="true" mayBeNull="true"></param>
```

```

return this._invoke(this._get_path(), 'GetExam',false,
{id:id},succeededCallback,failedCallback,userContext); }}
ActiveSkillWS.Service1.registerClass('ActiveSkillWS.Service1',Sys.Net.WebServiceProxy);
ActiveSkillWS.Service1._staticInstance = new ActiveSkillWS.Service1();
ActiveSkillWS.Service1.set_path = function(value) {
ActiveSkillWS.Service1._staticInstance.set_path(value); }
ActiveSkillWS.Service1.get_path = function() {
/// <value type="String" mayBeNull="true">The service url.</value>
return ActiveSkillWS.Service1._staticInstance.get_path();}
ActiveSkillWS.Service1.set_timeout = function(value) {
ActiveSkillWS.Service1._staticInstance.set_timeout(value); }
ActiveSkillWS.Service1.get_timeout = function() {
/// <value type="Number">The service timeout.</value>
return ActiveSkillWS.Service1._staticInstance.get_timeout(); }
ActiveSkillWS.Service1.set_defaultUserContext = function(value) {
ActiveSkillWS.Service1._staticInstance.set_defaultUserContext(value); }
ActiveSkillWS.Service1.get_defaultUserContext = function() {
/// <value mayBeNull="true">The service default user context.</value>
return ActiveSkillWS.Service1._staticInstance.get_defaultUserContext(); }
ActiveSkillWS.Service1.set_defaultSucceededCallback = function(value) {
ActiveSkillWS.Service1._staticInstance.set_defaultSucceededCallback(value); }
ActiveSkillWS.Service1.get_defaultSucceededCallback = function() {
/// <value type="Function" mayBeNull="true">The service default succeeded callback.</value>
return ActiveSkillWS.Service1._staticInstance.get_defaultSucceededCallback(); }
ActiveSkillWS.Service1.set_defaultFailedCallback = function(value) {
ActiveSkillWS.Service1._staticInstance.set_defaultFailedCallback(value); }
ActiveSkillWS.Service1.get_defaultFailedCallback = function() {
/// <value type="Function" mayBeNull="true">The service default failed callback.</value>
return ActiveSkillWS.Service1._staticInstance.get_defaultFailedCallback(); }
ActiveSkillWS.Service1.set_path("/ActiveSkillWS/Service1.asmx");
ActiveSkillWS.Service1.GetExam= function(id,onSuccess,onFailed,userContext) {
/// <param name="id" type="Number">System.Int32</param>
/// <param name="succeededCallback" type="Function" optional="true"
mayBeNull="true"></param>
/// <param name="failedCallback" type="Function" optional="true" mayBeNull="true"></param>
/// <param name="userContext" optional="true" mayBeNull="true"></param>
ActiveSkillWS.Service1._staticInstance.GetExam(id,onSuccess,onFailed,userContext); }
var gtc = Sys.Net.WebServiceProxy._generateTypedConstructor;
if (typeof(ActiveSkillWS.Exam) === 'undefined') {
ActiveSkillWS.Exam=gtc("ActiveSkillWS.Exam");
ActiveSkillWS.Exam.registerClass('ActiveSkillWS.Exam');
}

```

Define Exam Question Control Markup.

The TakeExamQuestion.ascx control will consist of a FieldSet with the exam title in the Legend. Within the FieldSet will be a Label showing the text for the current question. Below that a RadGrid will display the possible responses to the question. At the bottom of the FieldSet will be two standard HTML buttons that navigate forward and back through the exam. When the user is on the last question and clicks "Next", the user navigates to the "TakeExamFinish.ascx" control.

1. Back in the \User folder of the ActiveSkillUI project, locate the TakeExamQuestion.ascx file and open the markup source for editing.
2. At the top of the file, add two hidden fields with id's "examIDField" and "examIDFieldSave". *We will use these two fields to detect when the user has changed exams.*

[ASP.NET] Adding the Hidden Fields

```
<!--hidden fields-->
<input id="examIDField" type="hidden" runat="server"
value="0" />
<input id="examIDFieldSave" type="hidden" runat="server"
value="0" />
```

3. Add the FieldSet and Legend definition below to the markup. Notice that the FieldSet has some styling for positioning its contents. The legend has some temporary text "Exam title" that will be replaced with the actual exam title using data from the web service.

[ASP.NET] Adding the FieldSet

```
<fieldset id="Fieldset1" style="border-color: #444; margin-right: 10px;
margin-bottom: 10px; padding: 5px" runat="server">
<legend id="questionLegend" cssclass="skillTitle" runat="server">
Exam title</legend>

<!-- Grid will go here -->

</fieldset>
```

4. In the designer, add a RadGrid inside the FieldSet tag with ID property "gridQuestion", AutoGenerateColumns set to "false", EnableViewState set to "false" and GridLines set to "None".
5. Set the MasterTableView properties NoMasterRecordsText to "", TableLayout to "Fixed" and ShowHeader to "false".
6. Add the following columns using the Columns collection editor or directly in the markup:
 - A GridBoundColumn with DataField "ID" and Visible equal to "false".
 - A GridCheckBoxColumn with DataField "Correct" and Visible equal to "false".
 - A GridTemplateColumn with DataField "UserChoice" and UniqueName "UserChoice". In the ItemTemplate for the column add a single standard ASP.NET Checkbox control with ID "cbUserChoice". Set the columns HeaderStyle.Width to "60px".
 - A GridBoundColumn with DataField "HTML".
7. Set the ClientSettings.ClientEvents to JavaScript functions to be defined later:
 - OnCommand to "ClientCommand".
 - OnRowDataBound to "RowDataBound".

[ASP.NET] Markup for the Grid

```
<telerik:RadGrid ID="gridQuestion" runat="server" AutoGenerateColumns="False"
EnableViewState="False" GridLines="None">
<MasterTableView NoMasterRecordsText="" ShowHeader="false"
TableLayout="Fixed">
<Columns>
<telerik:GridBoundColumn DataField="ID" UniqueName="ID"
Visible="False">
</telerik:GridBoundColumn>
<telerik:GridCheckBoxColumn DataField="Correct" UniqueName="Correct"
Visible="False">
</telerik:GridCheckBoxColumn>
<telerik:GridTemplateColumn DataField="UserChoice"
UniqueName="UserChoice">
```

```

        <ItemTemplate>
            <asp:CheckBox ID="cbUserChoice" runat="server" />
        </ItemTemplate>
        <HeaderStyle Width="60px" />
    </telerik:GridTemplateColumn>
    <telerik:GridBoundColumn DataField="HTML" UniqueName="HTML">
    </telerik:GridBoundColumn>
</Columns>
</MasterTableView>
<ClientSettings>
    <ClientEvents OnCommand="ClientCommand" OnRowDataBound="RowDataBound" />
</ClientSettings>
</telerik:RadGrid>

```

8. Below the grid, add two HTML buttons. Notice that the buttons are contained with divs that are styled to position the buttons. The onclick event points to functions we will code later, `goNext()` and `goBack()`. The `onmouseover` and `onmouseout` events change the source image of the buttons when the mouse passes over.

[ASP.NET] Adding Next and Back Buttons

```

<br />
<div style="float: right; padding-right: 10px; padding-bottom: 5px">
    
</div>
<div style="float: right; padding-right: 10px; padding-bottom: 5px">
    
</div>

```

Create the ExamManager Client Object.

The exam question control will need client code to navigate through the exam data and to keep track of the current question within the exam. The client code should also store the user responses to the questions and when the exam completes, summarize the results. Rather than scatter this code directly in procedure JavaScript code within the markup, this step creates an ExamManager JavaScript object that wraps all this functionality and is consumed by client code in the markup.

1. In the ActiveSkillUI project \Scripts directory, add a new JScript item "ExamManager.js".
2. Add the code below to define the ExamManager object.

Inside the code for ExamManager are a few details to take notice of:

- The constructor passes in the examID. The exam id originates from the TakeExamChoose.ascx control, was passed to the FirstLoad of the TakeExamQuestion.ascx control and placed to the "examIDField" hidden field. The hidden field value is retrieved just before ExamManager is constructed.
- The constructor also passes functions to handle the OnLoaded and OnFinish events of this object. OnLoaded fires just after the web service returns with its data. OnFinish fires when we run out of questions and the user tries to navigate to the next question.
- This object descends from "Sys.Component" (see the registerClass() method at the end of this code sample) and so has access to the event dispatching mechanism we use here to handle OnLoaded and OnFinish events. Once the call to initializeBase() in the constructor is performed, the Sys.Component functionality is available.
- In the constructor fires the web service method "ActiveSkillWS.Service1.GetExam()", passing functions

that will run if the web service succeeds or fails. `GetExam()` also passes a context object "this", i.e. the `ExamManager` itself.

- The prototype where methods and properties are defined for `ExamManager` has some simple properties for tracking the exam itself, the current question within the exam, the number of questions in the exam, and if we are currently on the last question.
- Navigation methods in the prototype, `next()` and `back()`, increment and decrement the current question index and check for the upper and lower bounds of the number of questions in the exam. The `next()` method has the additional responsibility of firing the `OnFinish` event if we're trying to navigate past the last question.
- The prototype `finish()` method iterates the questions and tallies the total and incorrect responses. This method also uses a `ExamResults` object (to be defined next) to store the results by question category. The `finish()` method serializes the results and builds a string of arguments that are ultimately sent to the `TakeExamFinish.ascx` control for display.

[JavaScript] Defining the ExamManager

```
Type.registerNamespace("ActiveSkill");
/* -- Exam Manager -- */
ActiveSkill.ExamManager = function(examID, onLoadedHandler, onFinishHandler)
{
    // Sys.Component supports event dispatching mechanism
    ActiveSkill.ExamManager.initializeBase(this);
    this._exam;
    this._index = 0;
    this._onLoadedHandler = onLoadedHandler;
    this._onFinishHandler = onFinishHandler;
    ActiveSkillWS.Service1.GetExam(examID, this.serviceSuccess, this.serviceFail, this);
}
ActiveSkill.ExamManager.prototype = {
    /* properties */
    get_exam: function()
    {
        return this._exam;
    },
    get_index: function()
    {
        return this._index;
    },
    get_question: function()
    {
        return this._exam.Exam_Questions[this._index].Question;
    },
    get_count: function()
    {
        return this._exam.Exam_Questions.length;
    },
    get_isLastQuestion: function()
    {
        return this._index == this.get_count() - 1;
    },
    /* navigation methods */
    next: function()
    {
        if (this.get_isLastQuestion())
        {
```

```

        this.finish();
    }
    else
    {
        this._index++;
    }
},
back: function() {
    if (this._index > 0) {
        this._index--;
    }
},
// Iterate the exam results, comparing the user responses
// with the responses marked as "correct".
// Tally the total and incorrect responses, keeping
// track of the responses by category.
finish: function()
{
    // create an instance of
    var results = new ExamResults();
    // iterate the questions
    for (var q in this._exam.Exam_Questions)
    {
        // get the category for the current question and add it to
        // a list of categories if not already present
        var categoryID = this._exam.Exam_Questions[q].Question.CategoryID;
        var category = results.find(categoryID);
        if (typeof (category) == 'undefined')
        {
            category = results.add(categoryID);
        }
        // Add to the total responses for this category
        category.Total++;
        // Iterate the responses and add to incorrect responses
        // tally for this category.
        for (var r in this._exam.Exam_Questions[q].Question.Responses)
        {
            var response = this._exam.Exam_Questions[q].Question.Responses[r];
            if (response.UserChoice != response.Correct)
            {
                category.Incorrect++;
                break;
            }
        }
    }
}

// convert the ExamResults object instance to a string
var resultsString = Sys.Serialization.JavaScriptSerializer.serialize(results);
// format an arguments string
var args = "&passPercent=" + this._exam.PassPercent;
args += "&title=" + this._exam.Title;
args += "&examResults=" + resultsString;
// add a OnFinish event and fire the event
this.add_examFinish(this._onFinishHandler);
var handler = this.get_events().getHandler("examFinish");

```

```

        if (handler != null) handler(this, args);
    },
    /* web service callbacks */
    serviceSuccess: function(exam, sender)
    {
        // retrieve the exam object passed from
        // the web service. Iterate all the responses
        // and add a "UserChoice" column.
        sender._exam = exam;

        for (var q in exam.Exam_Questions)
        {
            for (var r in exam.Exam_Questions[q].Question.Responses)
            {
                var response = exam.Exam_Questions[q].Question.Responses[r];
                response.UserChoice = false;
            }
        }
        // notify the client that the web service
        // callback has finished, so data should be available
        sender.add_examLoaded(sender._onLoadedHandler);
        sender.raise_examLoaded("examLoaded");
    },
    serviceFail: function(error)
    {
        alert("Web service failed with error: " +
            error.get_message() + " " +
            error.get_stackTrace());
    },
    /* examLoaded event */
    add_examLoaded: function(handler)
    {
        this.get_events().addHandler("examLoaded", handler);
    },
    raise_examLoaded: function()
    {
        var handler = this.get_events().getHandler("examLoaded");
        if (handler != null) handler(this, Sys.EventArgs.Empty);
    },
    /* examFinish event */
    add_examFinish: function(handler)
    {
        this.get_events().addHandler("examFinish", handler);
    },
    raise_examFinish: function(args)
    {
        var handler = this.get_events().getHandler("examFinish");
        if (handler != null) handler(this, args);
    }
}
ActiveSkill.ExamManager.registerClass("ActiveSkill.ExamManager", Sys.Component);

```

3. In the same file, add JavaScript objects that encapsulate categories and the exam results. The ExamResults object contains an array of Category objects where each Category tracks its own category ID, the total questions for the category and the total incorrect questions. ExamResults has a find() method to locate

categories that already exist in the array and an add() method to include new categories to the array.

[JavaScript] Defining the Category and ExamResults JavaScript Objects

```
// category object template
function Category(categoryID)
{
  this.CategoryID = categoryID;
  this.Total = 0;
  this.Incorrect = 0;
}
// examResults
function ExamResults()
{
  this.Categories = new Array();
  this.find = find;
  this.add = add;
  function find(id)
  {
    for (c in this.Categories)
    {
      if (this.Categories[c].CategoryID == id)
      {
        return this.Categories[c];
      }
    }
  }
  function add(categoryID)
  {
    var newCategory = new Category(categoryID, 0);
    this.Categories.push(newCategory);
    return newCategory;
  }
}
```

- At the end of the ExamManager.js file, add a call that notifies the ScriptManager that the JavaScript is loaded.

[JavaScript] Notify the ScriptManager

```
// notify script manager that this js is loaded
if (typeof (Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();
```

Create the UIManager Client Object.

A second object stores the UI elements of the page and handles updating the page with ExamManager data and saving ExamManager data from the page elements.

- Add a new UIManager.js file to the \scripts folder of the ActiveSkillUI project.
- Add the code below to define the UIManager JavaScript object.
 - The parameters passed to the constructor are simply references to page elements, e.g. "\$get("<%= btnNext.ClientID %>")".*
 - Most of the action happens in two methods, refresh() and save(). refresh() updates the legend, label displaying the question and grid containing questions. Notice that we are binding the grid on the client side to our Exam object that was retrieved from the web service. The binding only takes three lines of code: retrieving the MasterTableView client object, setting the MasterTableView dataSource property and calling the MasterTableView dataBind() method. Sweet!*

- The `save()` method reverses the process described for `refresh()`. The tricky part here is that we need to get the template column check box that stores the user choices for each response. The key pieces for getting the value from a templated control are to first get the `MasterTableView` data items, get the data item for a given row, get the table cell that contains our check box element by calling `MasterTableView.getCellByColumnUniqueName()`, and finally setting properties in our data source to match the check box element checked property. Remember that for the call to `getCellByColumnUniqueName` to work, it must match the `UniqueName` property for the grid column. Also note the call to `getElementsByTagName()` that retrieves the check box element is a safer approach than assuming the check box will be present in a particular position. As it turns out, browsers will present these elements differently so `checkboxCell[0]` may contain the check box or it may be in `checkboxCell[1]`.

[JavaScript] Defining the UIManager JavaScript Object

```
Type.registerNamespace("ActiveSkill");
/* -- UI Manager -- */
// The constructor accepts and stores references to each page
// element.
ActiveSkill.UIManager = function(nextButton, backButton, questionLegend, questionLabel,
grid)
{
    ActiveSkill.UIManager.initializeBase(this);
    this._nextButton = nextButton;
    this._backButton = backButton;
    this._questionLegend = questionLegend;
    this._questionLabel = questionLabel;
    this._grid = grid;
}
ActiveSkill.UIManager.prototype = {
    /* -- properties -- */
    get_nextButton: function()
    {
        return this._nextButton;
    },
    get_backButton: function()
    {
        return this._backButton;
    },
    get_questionLegend: function()
    {
        return this._questionLegend;
    },
    get_questionLabel: function()
    {
        return this._questionLabel;
    },
    get_grid: function()
    {
        return this._grid;
    },
    /* -- methods -- */
    // update page elements using the
    // data stored in the ExamManager object.
    refresh: function(examManager)
    {
        this.get_questionLegend().innerHTML = examManager.get_exam().Title;
    }
}
```

```

    this.get_questionLabel().innerHTML = examManager.get_question().HTML;
    // bind the exam responses for the current question to the grid
    var tableView = this.get_grid().get_masterTableView();
    tableView.set_dataSource(examManager.get_question().Responses);
    tableView.dataBind();
},
// retrieve the user choices from the grid.
save: function(examManager)
{
    var masterTableView = this.get_grid().get_masterTableView();
    var dataItems = masterTableView.get_dataItems();
    var responses = examManager.get_question().Responses;

    // iterate the grid rows
    for (var i = 0; i < responses.length; i++)
    {
        var dataItem = dataItems[i];
        // find the table cell that holds our UserChoice checkbox
        var checkBoxCell = masterTableView.getCellByColumnUniqueName(dataItem, "UserChoi
        if (checkBoxCell != null)
        {
            // retrieve the checkbox element
            var cbUserChoice = checkBoxCell.getElementsByTagName("INPUT")[0];
            // set the datasource response object UserChoice according to the checkbox
            responses[i].UserChoice = cbUserChoice.checked;
        }
    }
}
}
ActiveSkill.UIManager.registerClass("ActiveSkill.UIManager", Sys.Component);
// notify script manager that this js is loaded
if (typeof (Sys) !== 'undefined') Sys.Application.notifyScriptLoaded();

```

Add Client Code to Consume New Client Objects.

Now that all of the markup is in place and the client JavaScript objects have been defined, we need to use the objects to handle navigation through the exam and to sync the user interface with the state of the data.

1. Add a block of JavaScript just above the FieldSet element.

[JavaScript] Adding JavaScript

```

<telerik:RadScriptBlock ID="RadScriptBlock1" runat="server">
  <script type="text/javascript">
    /* -- Client Event Handlers go here-- */

    </script>
  </telerik:RadScriptBlock>

```

2. Add a pageLoad event handler inside the <script> tag. *This event will fire when the TakeExamQuestion.ascx control first loads on the client. Here we create a UIManager for use throughout this page. The UIManager constructor passes references to each page element using \$get() or \$find().*



Gotcha! Make sure you use `$find` when you access a control (vs a simple HTML element). `$get` will get you the control as an HTML element but it will be missing control functionality. If you're debugging client code and are looking for expected methods that don't show up, go back and check if you used `$get` instead of `$find` to retrieve the control reference. `pageLoad()` also retrieves references for the hidden fields that store the exam ID. `examIDField` gets the

current value placed there by the control's `FirstLoad()` server call (`FirstLoad()` will be handled in the next section of this chapter).

`pageLoad()` initializes an instance of `ExamManager` if this is the first time we have loaded the `TakeExamQuestion.ascx` control or if the user has selected a different exam. We derive a boolean variable `needReload` that is true if the `ExamManager` hasn't been created or if the exam ID is different from the saved exam ID. If `needReload` is true, the `ExamManager` instance is created, passing the current exam ID and functions that will handle the `OnExamLoaded` and `OnExamFinish` events. Finally, `pageLoad()` saves the exam id hidden field value to the second hidden field for later comparison.

[JavaScript] Handling the `pageLoad()` Client Event

```
function pageLoad()
{
    window.uiManager = new ActiveSkill.UIManager(
        $get("<%= btnNext.ClientID %>"),
        $get("<%= btnBack.ClientID %>"),
        $get("<%= questionLegend.ClientID %>"),
        $get("<%= questionLabel.ClientID %>"),
        $find("<%= gridQuestion.ClientID %>"));
    var examIDField = $get("<%= examIDField.ClientID %>");
    var examIDFieldSave = $get("<%= examIDFieldSave.ClientID %>");
    if (examIDField != null)
    {
        var examID = examIDField.value;
        var examIDSave = examIDFieldSave.value;
        var needReload = typeof (window.examManager) == "undefined" ||
            examID != examIDSave;
        if (needReload)
        {
            window.examManager =
                new ActiveSkill.ExamManager(examID, onExamLoaded, onExamFinish);
        }
        examIDFieldSave.value = examID;
    }
}
```

3. Add two handlers for the `ExamManager` `OnExamLoaded` and `OnExamFinish` events.

These consume the JavaScript objects we previously defined.

The `OnExamLoaded` sender is the `ExamManager` instance. When the exam data is loaded from the web service, it's safe to load the data to the user interface. The `UIManager` `refresh()` method handles updating the legend, question label and populating the grid, all using the `ExamManager` data. "args" for `OnExamLoaded` are empty.

The `OnExamFinish` event fires automatically when a call to `ExamManager` `next()` attempts to navigate past the last question. Here we use the `DynamicControl` object to load the `TakeExamFinish.ascx` control. "args" in this event handler are the serialized exam results passed from the `ExamManager` `finish()` method. The "args" string is passed onto the `TakeExamFinish.ascx` control where the `FirstLoad` will convert the argument to a `Dictionary` and use the results to display the user exam status and score.

[JavaScript] Handling the `OnExamLoaded` and `OnExamFinish` Events

```
function onExamLoaded(sender, args)
{
    window.uiManager.refresh(sender);
}
```

```

}
function onExamFinish(sender, args)
{
    window.DynamicControl.load("TakeExamFinish.ascx", args);
}

```

4. Add functions to handle grid events defined earlier in the markup.



Gotcha! As of this writing, when binding on the client side only, the grid ClientCommand expects a handler (even if empty) to avoid a "null object" error.

For each row in the grid, the RowDataBound event sets the "UserChoice" checkbox check based on the "UserChoice" value in the response.

[JavaScript] Handling Grid Client Events

```

function ClientCommand(sender, args) {
    // must assign this empty event handler to avoid "null object" error
}
function RowDataBound(sender, args) {
    // get the "UserChoice" checkbox
    var checkBoxCell = args.get_item().get_cell("UserChoice");
    var cbUserChoice = checkBoxCell.getElementsByTagName("INPUT")[0];
    // Set the checked property for the checkbox to
    // the underlying data value.
    if (cbUserChoice != null) {
        cbUserChoice.checked = args.get_dataItem()["UserChoice"];
    }
}

```

5. Add client functions to handle the next and back button clicks.

Both functions have essentially the same structure where the UIManager save() function is called to retrieve the current user choices, the ExamManager next() or back() methods are called to perform the navigation through the exam and the UIManager refresh() method is called to bind to the new question data.

[JavaScript] Responding to Next and Back Button Clicks

```

function goNext() {
    window.uiManager.save(window.examManager);
    window.examManager.next();
    window.uiManager.refresh(window.examManager);
}
function goBack() {
    window.uiManager.save(window.examManager);
    window.examManager.back();
    window.uiManager.refresh(window.examManager);
}

```

Code the FirstLoad() Method

Add the assignment of the examIDField Value property to the FirstLoad() method.

The `FirstLoad()` method of `TakeExamQuestion.ascx` retrieves the `id` value passed in from the `TakeExamChoose.ascx` client code and populates the `examIDField` hidden field. Later, when the `TakeExamQuestion` control loads on the client, `examIDField` will be retrieved and used to populate the `ExamManager` client object.

[VB] Loading the Hidden Field

```
Imports System.Collections.Generic
Imports Telerik.ActiveSkill.Common
Namespace Telerik.ActiveSkill.UI.User
    Public Partial Class TakeExamQuestion
        Inherits System.Web.UI.UserControl
        Implements IASControl
        #region IASControl Members
        Public Sub FirstLoad(ByVal args As Dictionary(Of String, String))
            examIDField.Value = args("id")
        End Sub
    #End Region
End Class
End Namespace
```

[C#] Loading the Hidden Field

```
using System.Collections.Generic;
using Telerik.ActiveSkill.Common;
namespace Telerik.ActiveSkill.UI.User
{
    public partial class TakeExamQuestion : System.Web.UI.UserControl, IASControl
    {
        #region IASControl Members
        public void FirstLoad(Dictionary<string, string> args)
        {
            examIDField.Value = args["id"];
        }
        #endregion
    }
}
```

Test Taking an Exam

1. If necessary, set the startup project to be `ActiveSkillUI` and the startup page to be `UserHome.aspx`. Press F5 to run the application.
2. Select an exam, respond to the questions and navigate through to the end.
3. Also test that when you check next to a response that the response is persisted. Navigate forward and back through the exam.
4. Put a break point on the `FirstLoad()` method and check the contents of "args". It should contain the control name "TakeExamFinish.ascx", the "passPercent", the exam title, and a JSON string that expresses the `ExamResults` client object. *In the upcoming chapter that implements the Finish page, we will deserialize the JSON string on the server side.*

```
#region IASControl Members
```

```
public void FirstLoad(Dictionary<string, string> args)
```

```
{
```

```
}
```

```
#endregion
```

		args	Count = 4
+	[0]	{[ControlName, TakeExamFinish.aspx]}	
+	[1]	{[passPercent, 58]}	
+	[2]	{[title, Telerik Reporting]}	
+	[3]	{[examResults, {"Categories": [{"Category":	
+	Raw View		



Gotcha!

If there are errors indicating the web service is not found, first verify that you can access the web service manually using the browser. Then verify this same path is used in a ScriptManager Reference element. The ScriptManager is kept on the UserHome.aspx page and will look something like this:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
  <Services>
    <asp:ServiceReference Path="http://localhost/ActiveSkillWS/service1.asmx" />
  </Services>
  <Scripts>
    <asp:ScriptReference Path="-/scripts/DynamicControl.js" />
    <asp:ScriptReference Path="-/scripts/ExamManager.js" />
    <asp:ScriptReference Path="-/scripts/UIManager.js" />
  </Scripts>
</asp:ScriptManager>
```

39.5 Summary

In this chapter you built functionality for the central purpose of the application: the taking of exams. The work was heavily weighted to the client where you consumed a web service to bring back the exam data, used your own JavaScript objects to encapsulate the exam, navigation through the exam and to summarize the exam results. You bound exam responses directly to the RadGrid using client code only. You also used LINQ to SQL within the web service to consume the Exam database data.

40 RadChart

40.1 Objectives

- Become familiar with RadChart by building a simple chart with static items and another basic chart using bound data.
- Take a tour of the basic elements of each RadChart and the available types of charts.
- Learn how designer interface tools help organize RadChart capabilities.
- Learn about some of the latest RadChart features.
- Create chart series and chart series items programmatically.
- Learn the specifics of data binding in RadChart.
- Learn how to handle RadChart server-side events.
- Learn how zooming and scrolling is performed in RadChart. Also learn how to perform zooming and scrolling in client-side code.
- Learn how image maps are created in RadChart and how an image map can be used to create a drill-down chart.

40.2 Introduction

RadChart is a powerful business data presentation tool that can show your data off with striking impact. RadChart comes with many customizable chart types and skins to tailor the behavior and look of each chart.

You can choose fine-tune control over all aspects of your chart or use the automatic layout, automatic text wrapping and intelligent labeling functions to handle the details. At design time you get quick access to critical properties with the Smart Tag, convenient groups of important properties in the RadChart wizard, or control all RadChart settings from the Properties Window.

The focus of this chapter will be in organizing the many capabilities and properties of this rich control so that you can get maximum use out of it from the outset.

40.3 Getting Started

Create a Chart with Static Data

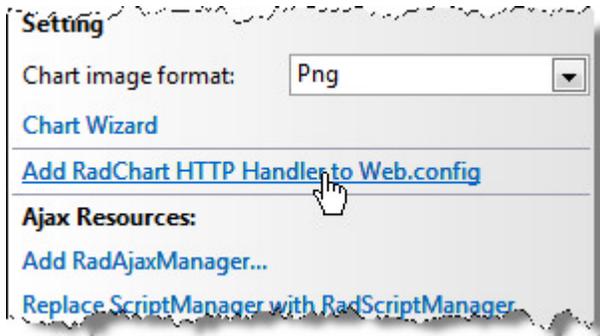
In this walk-through you will get up and running with a working RadChart application. You will create a chart populated with static data and modify several properties that affect chart appearance.



You can find the complete source for this project at:
\\VS Projects\Chart\GettingStarted1

Prepare the Project

1. Create an ASP.NET AJAX Web Application.
2. Create a new ASP.NET Web Application and drag a **ScriptManager** from the Tool Box onto the Web page.
3. From the Toolbox drag a **RadChart** component to the default web page.
4. Click the Smart Tag **Add RadChart HTTP Handler to Web.Config** link. *This adds a handler to the <configuration><system.web><httpHandlers> section of the web.config file.*



Gotcha!

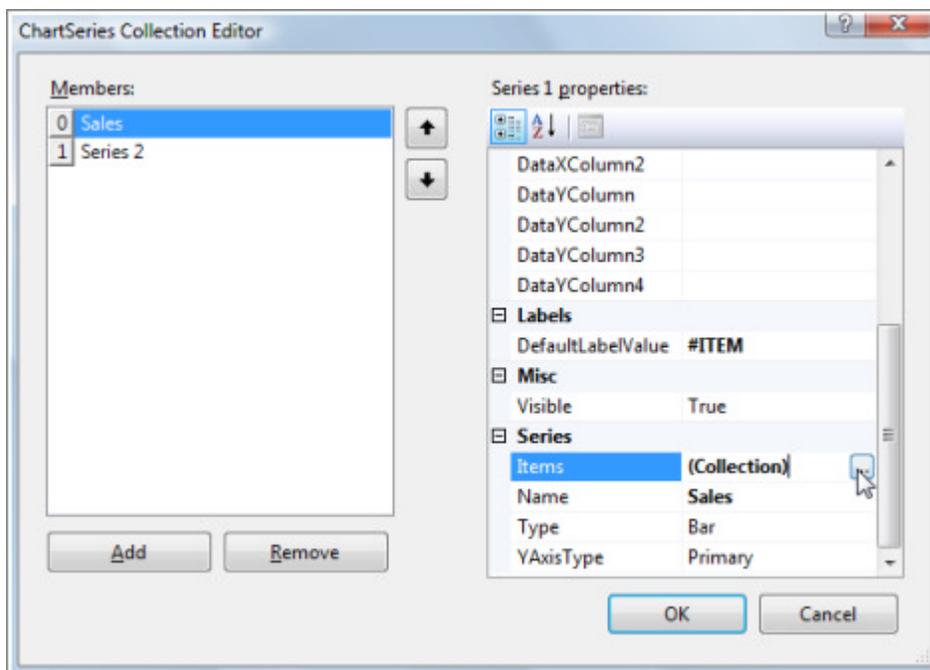
Gotcha! At the time of this writing, if you are using IIS7 Integrated Mode, you need to add a handler manually to the <system.webserver> "handlers" element of the web.config file:

```
<system.webServer>
  <handlers>
    <add name="ChartHandler" path="ChartImage.axd" verb="*"
      type="Telerik.Web.UI.ChartHttpHandler, Telerik.Web.UI" />
    ...
```

If you are using the internal web server, there shouldn't be a problem.

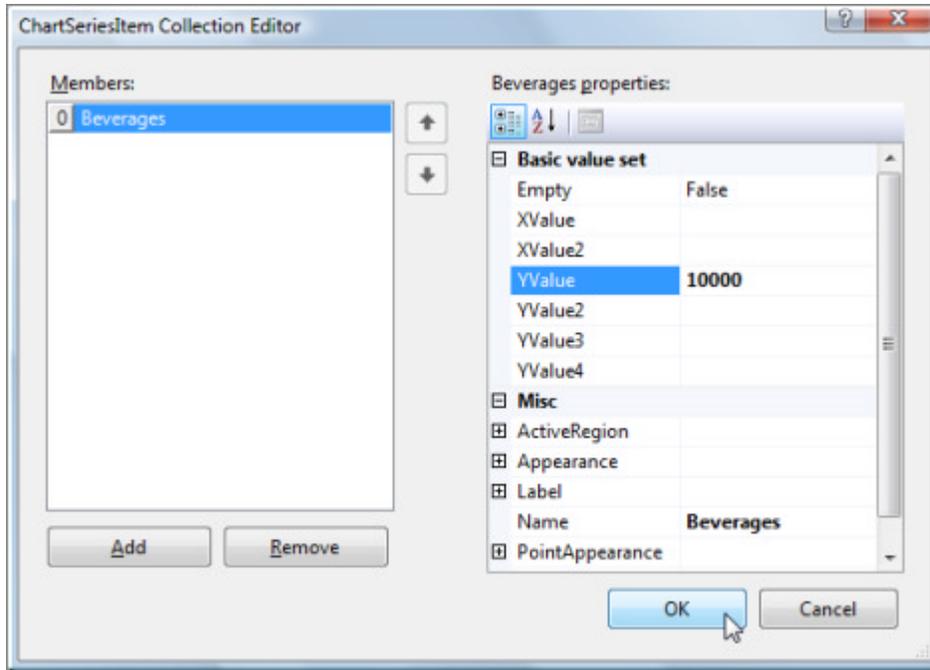
Populate Chart Data

1. Open the RadChart Smart Tag. From the Smart Tag "Data" section, click the ellipses for the Chart Series Collection.
2. Click "Series 1" in the members list on the left, then locate the **Name** property in the property window.
3. Change the **Name** property to "Sales" and the **DefaultLabelValue** to "#ITEM". *Name will be the series name that shows up in the legend. DefaultLabelValue will display the name of each item in the series instead of the item value.*
4. Locate the **Items** property in the property window.

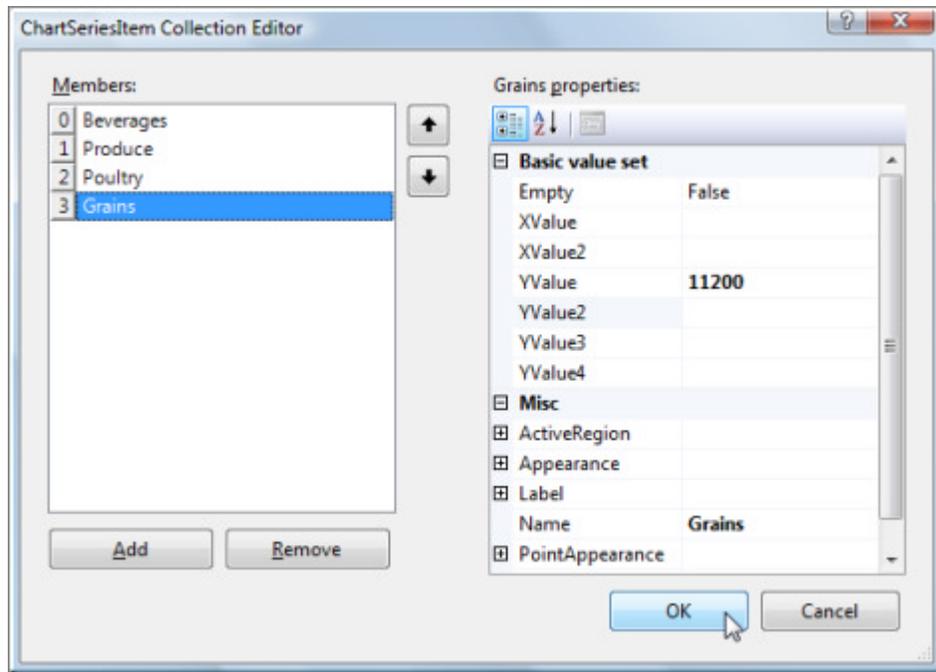


UI for ASP.NET AJAX

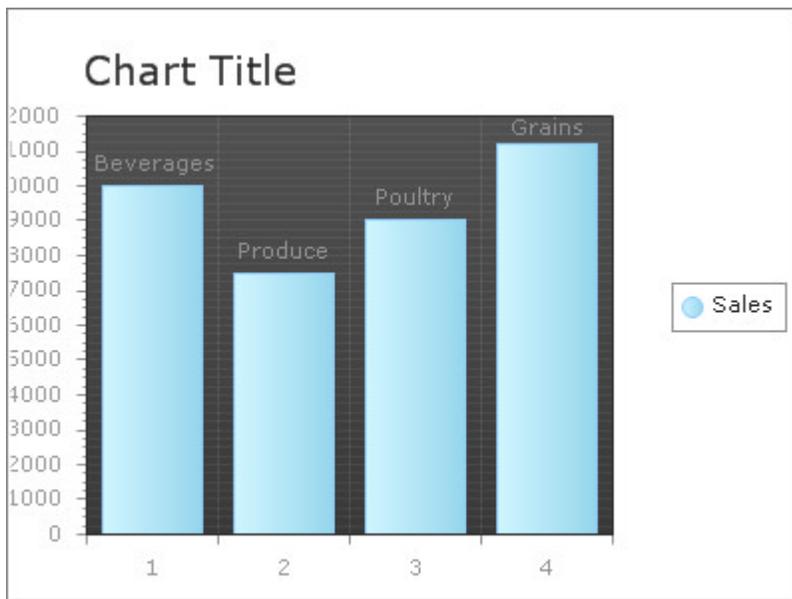
5. Click the ellipses button of the Items property to open the ChartSeriesItem Collection Editor.
6. Click the **Add** button to add a new Item.
7. In the property window for the new item, change the Name property to "Beverages".
8. Change the YValue property to "10000".



9. Repeat the Add Item steps to add 3 new items. Replace the properties for the three new items as follows:
 - o Label: Produce, YValue: 7500
 - o Label: Poultry, YValue: 9000
 - o Label: Grains, YValue: 11200



10. Click **OK** to close the ChartSeriesItem Collection Editor.
11. Click "Series 2" in the ChartSeries Collection Editor.
12. Click the **Remove** button to remove Series 2.
13. Click the **OK** button to close the ChartSeries Collection Editor.
14. The chart will display the new data using the default formatting.

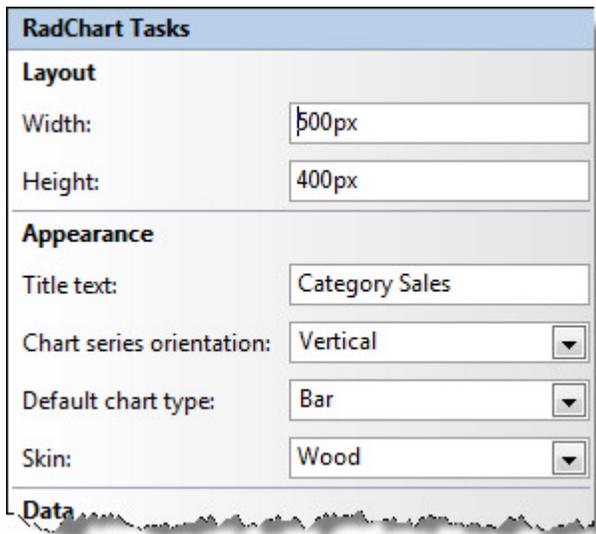


Format the Chart Using the SmartTag

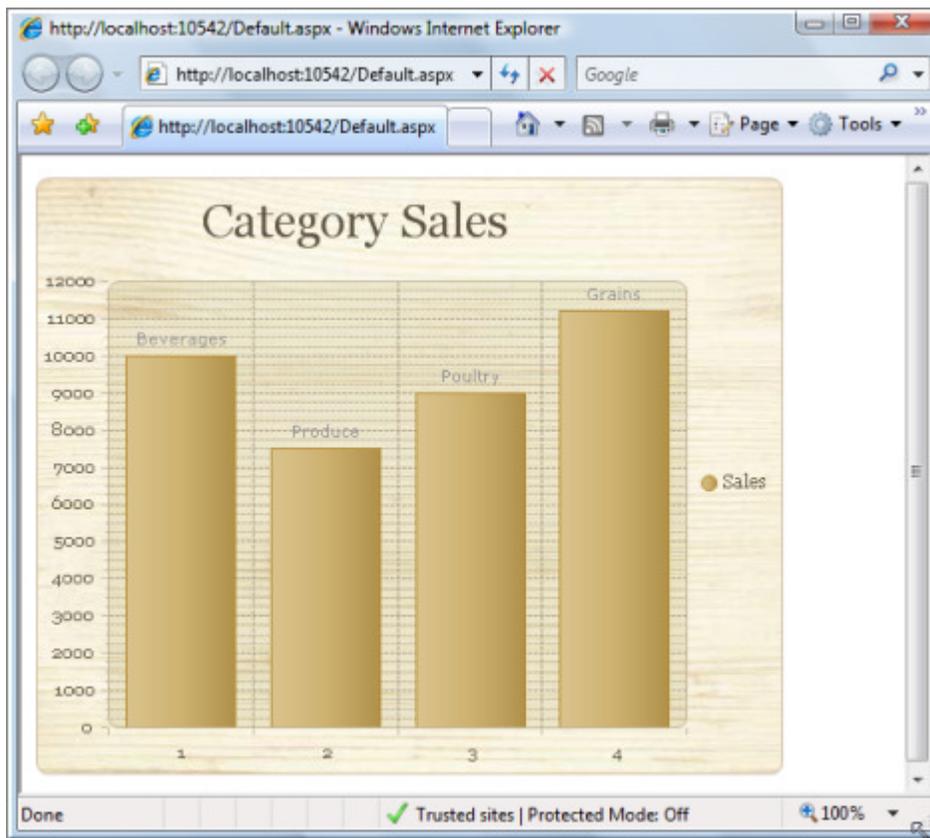
1. Click the RadChart's Smart Tag
2. Change the Layout section **Width** to "500px" and **Height** to "400px".

UI for ASP.NET AJAX

3. In the Appearance section, change the **Title Text** entry to "Category Sales" and the **Skin** to "Wood".



4. In the Properties window, set the **AutoLayout** property to "true".
5. Press **Ctrl-F5** to run the application. Notice that the AutoLayout feature of RadChart has positioned the labels on the Y axis (along the left side) and the legend ("Sales", on the right side of the chart). The chart title "Category Sales" should appear at the top of the chart.



Create a Chart with Bound Data

In this walk-through you will create a chart that consumes bound data. We will display the top ten product sales by category in a horizontal bar chart.



You can find the complete source for this project at:
 \VS Projects\Chart\GettingStarted2

Prepare the Project

1. Create an ASP.NET AJAX Web Application
2. Create a new ASP.NET Web Application and drag a **ScriptManager** from the Tool Box onto the Web page
3. From the Toolbox drag a **RadChart** component to the default web page.
4. Click the Smart Tag **Add RadChart HTTP Handler to Web.Config** link.
5. Locate the "Northwind.mdf" file in the "Live Demos\App_Data" folder under the folder where you installed RadControls for ASPNET AJAX. Drag this file into the "App_Data" folder of your project.
6. Open the "Web.config" file of your project. Add the standard Northwind connection string to your project by replacing the line


```
<connectionStrings />
```

 with


```
<connectionStrings>
    <add name="NorthwindConnectionString" connectionString="Data
    Source=.\SQLEXPRESS;AttachDbFilename=|DataDirectory|Northwind.mdf;Integrated
    Security=True;User Instance=True" providerName="System.Data.SqlClient" />
</connectionStrings>
```

Configure the RadChart Using the Wizard

1. From the Smart Tag select the **Chart Wizard** link from the Setting section.
2. In the Type tab of the wizard, select the **Horizontal** orientation and the **Bar** chart type.

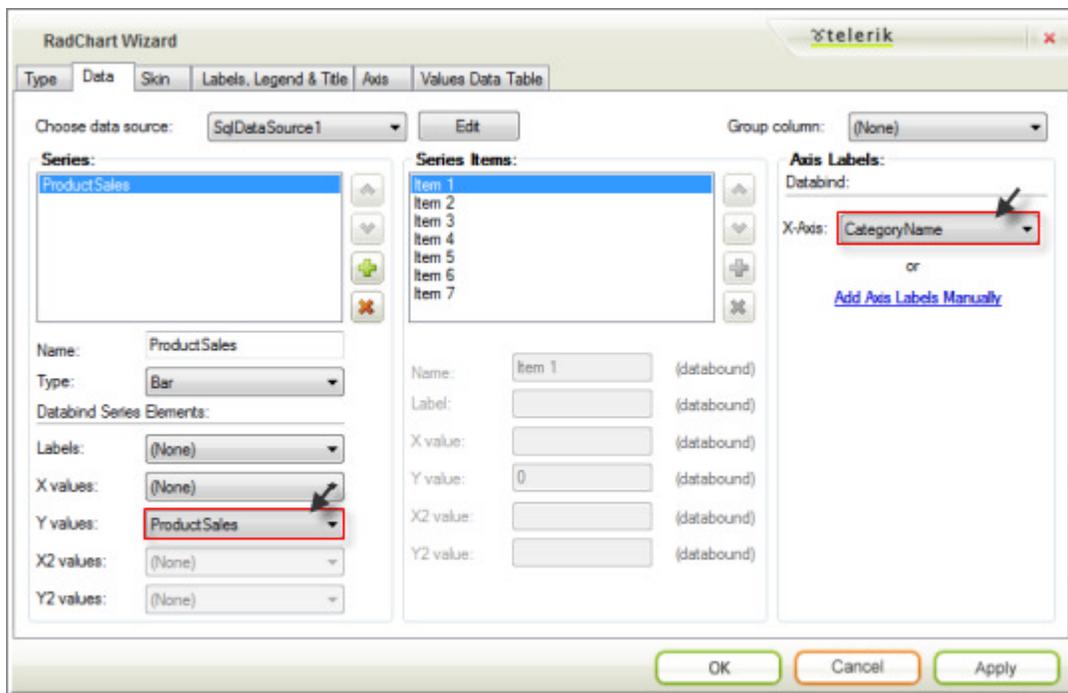


- In the Data tab of the wizard, select "<New Data Source...>" from the Choose Data Source drop down list. In the Data Source Configuration dialog:
 - In the **Choose a Data Source Type** page, choose "Database". Click the **Next** button to continue.
 - In the **Choose Your Data Connection** page select the NorthwindConnectionString from the drop down list. Click the **Next** button to continue.
 - In the **Configure the Select Statement** page, choose the "Specify a custom SQL Statement or stored procedure" radio button. Click the **Next** button to continue.
 - In the **Define Custom Statements or Stored Procedures** page, enter the following SQL to the SELECT tab and click **Next** to continue.

[T-SQL] Defining the Select

```
SELECT TOP (10) ProductName, ProductSales, CategoryName FROM [Sales by Category] order ProductSales desc
```

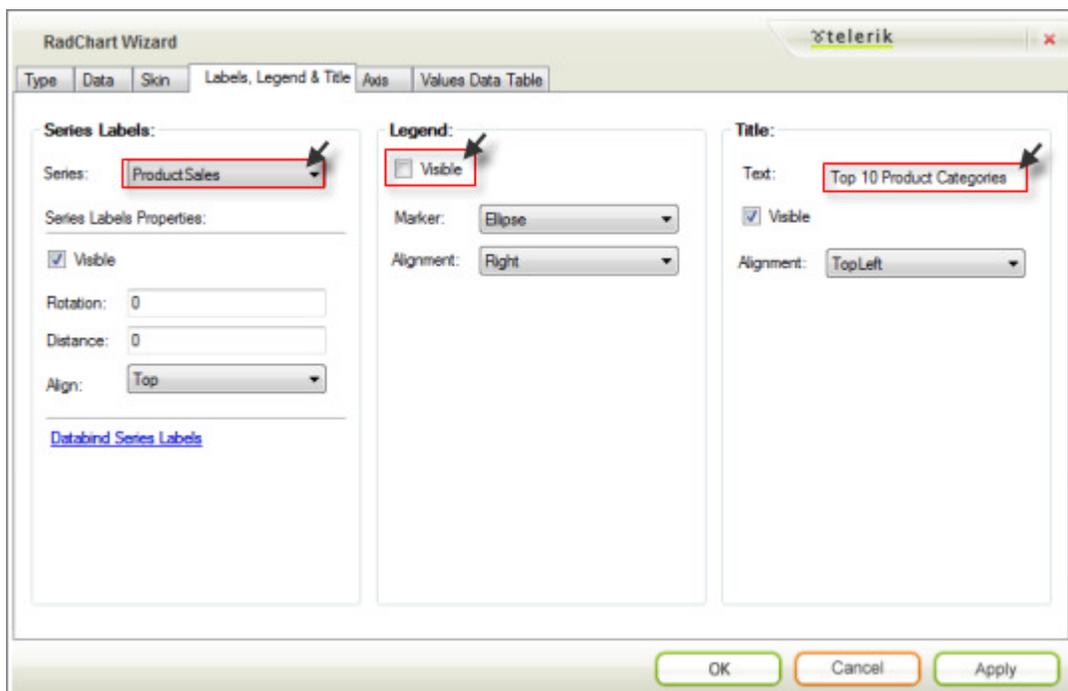
- Test the query if you wish, and then click **Finish**.
- Still in the Data tab of the wizard, set the **Y values** drop down list to "ProductSales" and the **X-Axis** to "CategoryName". *In a horizontal bar chart, the X-Axis will list the category names from top to bottom on the left hand side of the chart.*



- On the Skin tab of the wizard, select the DeepGreen skin.



6. In the Labels, Legend & Title tab, set the **Series Labels** to "ProductSales" from the drop down list, de-select the **Legend Visible** check box. Set the **Title Text** to "Top 10 Product Categories".



7. Click the **OK** button to close the wizard.
8. In the Properties window set the **AutoLayout** property to "true".
9. Press **Ctrl-F5** to run the application. Notice that the labels on the X-axis are arranged from top to bottom on the left hand side in this horizontal layout. If the layout were vertical, the labels would be listed along the

bottom.



RadChart Basics

Charts are composed of a hierarchy of elements. Many of the elements are common across all chart types. Take a look at the figure below to see some of the main chart elements, particularly the Plot Area, Chart Series, Chart Series Items and Axis.

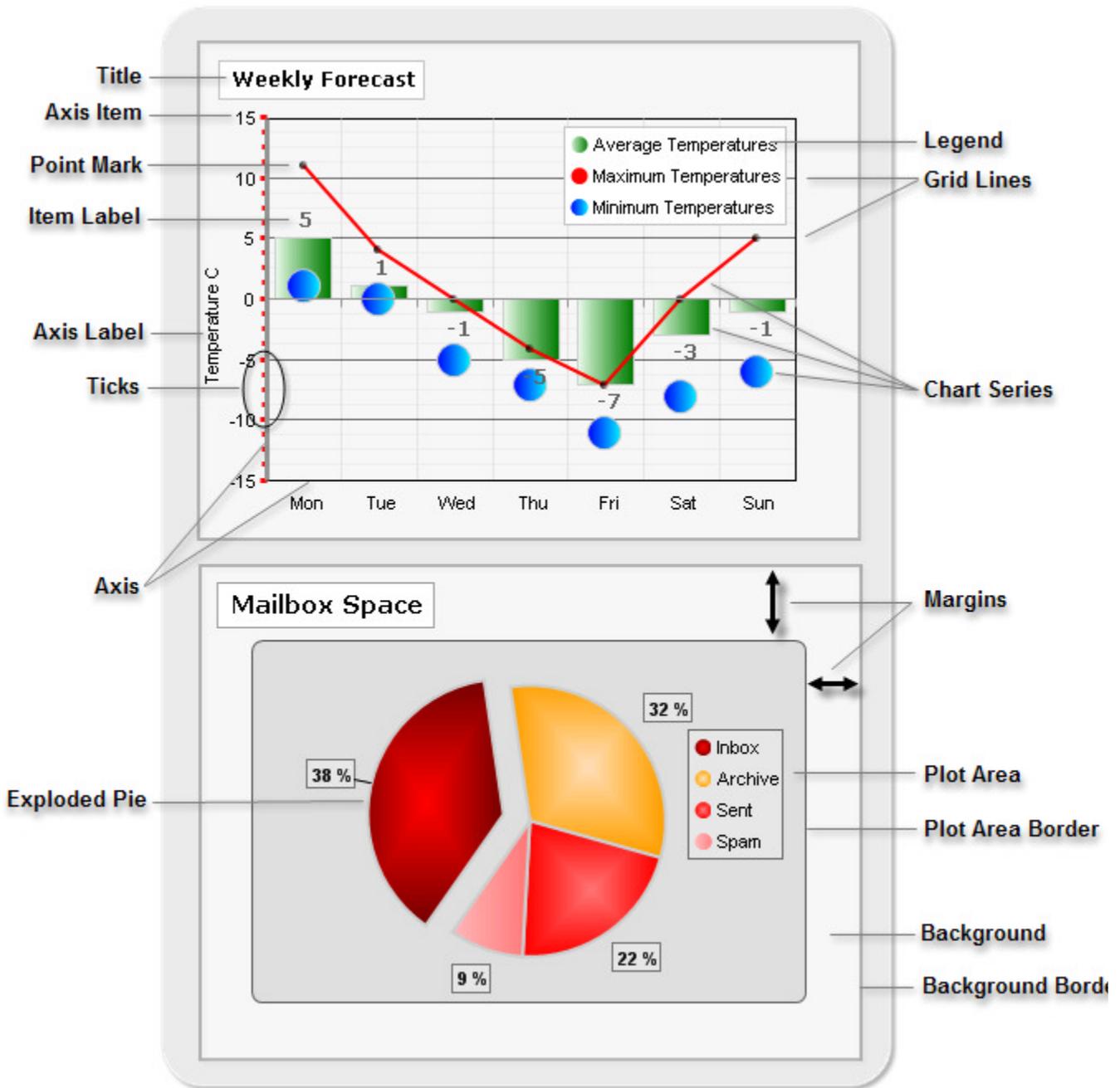


Chart Background

The background of the chart is the outermost rectangle that encloses all other elements of the chart. It stretches for the whole width and length of the output image of the chart.

Title, Legend and Labeling

These three chart elements let you apply meaningful labels to the chart, the data and to groupings of the data. The actual property you would be looking at for title is **ChartTitle**. The legend property is **Legend**. For axis labeling, you look for the axis properties within the PlotArea property: **PlotArea.XAxis.AxisLabel**, **PlotArea.YAxis.AxisLabel** and **PlotArea.YAxis2.AxisLabel**.

We will spend a little extra time on common sub-properties of the title, legend and label properties because

UI for ASP.NET AJAX

they show up in many aspects of the chart.

- **ActiveRegion**: contains properties for HTML Attributes, Tooltip and URL. The ActiveRegion property is found throughout the chart control and can be used to create links that can be clicked to navigate the page to web sites. The properties set as below would let the user click the chart title and navigate to the Wikipedia web site. Hovering the mouse over the title would display the tool tip.

[-] ChartTitle	
[-] ActiveRegion	
Attributes	
Tooltip	Click here to visit Wikipedia
Url	http://en.wikipedia.org/wiki/Products

- **Appearance**: This is an extensive property, also found attached to other properties throughout the chart. The exact makeup of Appearance changes depending on the context you find it in. Appearance lets you customize all the visual aspects of the chart element you're working with, such as layout, dimensioning, positioning, fill, background images, font colors and borders. The appearance properties for the ChartTitle are shown below. Here we're setting the RotationAngle to -20.

[-] Appearance	
[-] Border	
CompositionType	None
[-] Corners	Rectangle, Rectangle, Rectangle, Rectangle, 3
[-] Dimensions	
Figure	Rectangle
[-] FillStyle	
[-] Position	
RotationAngle	-20
[-] Shadow	
Visible	True

You can see the effect where the title is rotated 20 degrees to the left:



- **Marker**: Controls a small graphic for whatever area is being described, e.g. title, legend, etc. By default the marker is not visible. Notice that the Marker property has it's own ActiveRegion and Appearance properties nested within. In the example below we've set the Figure property to "Star3" and the Visible property to true.

[-] Marker	
[-] ActiveRegion	
[-] Appearance	
[-] Border	
[-] Corners	Rectangle, Rectangle, Rectangle, Rectangle, 3
[-] Dimensions	
Figure	Star3
[-] FillStyle	
[-] Position	
RotationAngle	0
[-] Shadow	
Visible	True
Visible	True

These property settings place a small rightward-pointing graphic to the left of the title.



- **TextBlock:** lets you fine-tune the appearance of the text, the visibility of the text and the text string itself. In the example below we add a border set to the AliceBlue color.

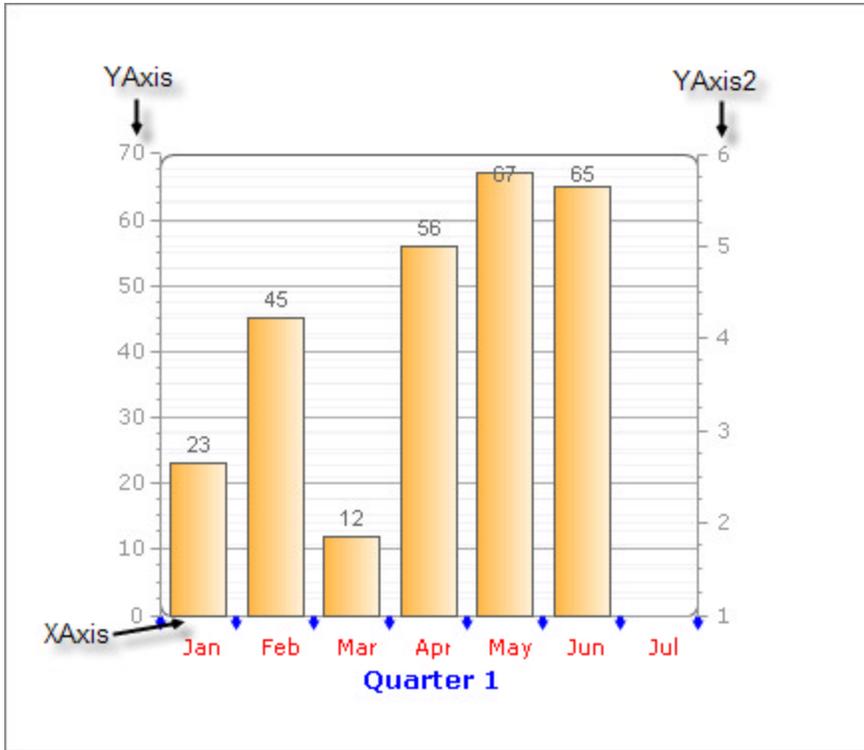
[-] TextBlock	
[-] Appearance	
AutoTextWrap	Auto
[-] Border	
Color	AliceBlue
PenStyle	Solid
Visible	True
Width	1
[-] Corners	Rectangle, Rectangle, Rectangle, Rectangle, 3
[-] Dimensions	
[-] FillStyle	
MaxLength	255
[-] Position	
[-] Shadow	
[-] TextProperties	
Visible	True
Text	Top 10 Product Categories
Visible	True
Visible	True

The TextBlock.Appearance.Border property setting was applied to the ChartTitle to get this appearance:



Axis

X and Y axes are included in all chart types except the Pie chart. Typically the YAxis displays values and the XAxis displays categories. For example, the YAxis might show "items sold" or "revenue", while the XAxis might show "Months" or "Products". The second Y axis lets you scale data on two criteria at once.

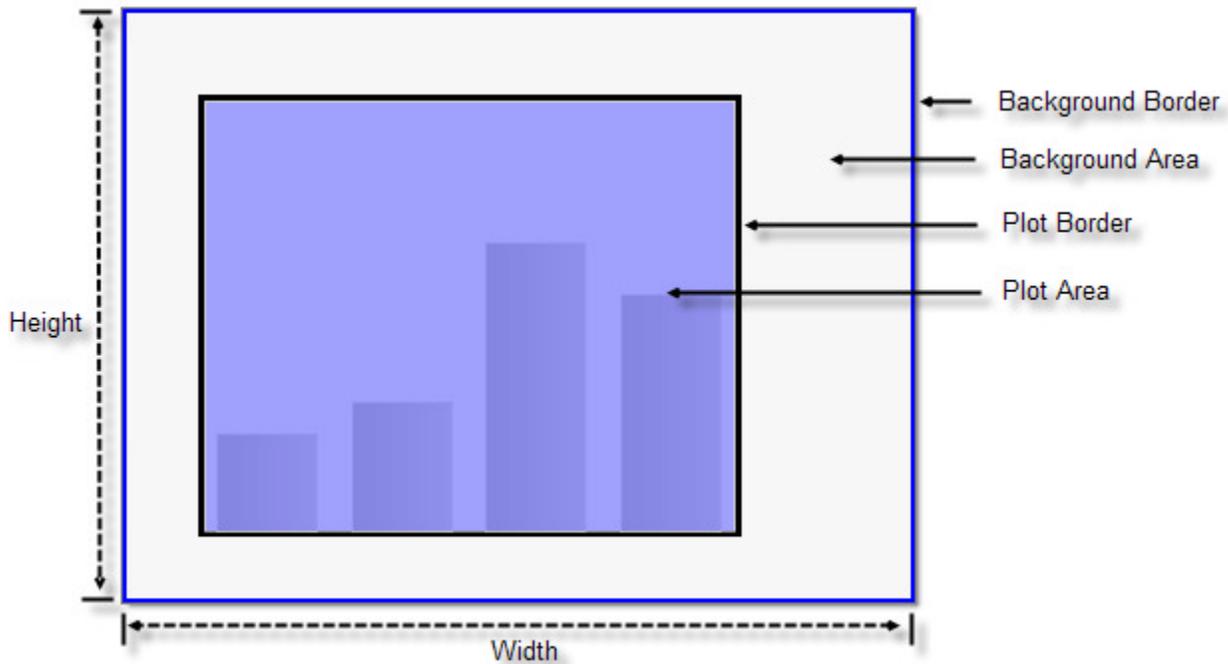


Plot Area

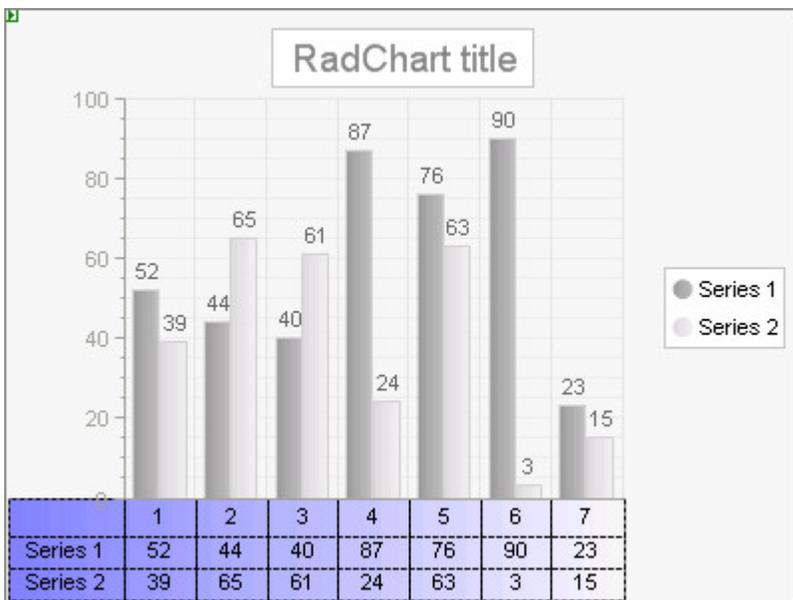
The plot area is the working rectangular area between X and Y axes where data is displayed. This property is a major jumping off point for configuring the axis of the chart.

[-] PlotArea	
[-] Appearance	
[-] DataTable	
[-] EmptySeriesMessage	
MarkedZones	(Collection)
Visible	True
[-] XAxis	
[-] YAxis	
[-] YAxis2	

The size of the plot depends on the chart background size and the chart margins, which define the distance between the border of the plot area and the border of the chart background.



The **PlotArea DataTable** displays a spreadsheet style table of the data in the chart, typically just below the chart itself. You can see in this screenshot that the data for both series is displayed in the table at the bottom of the chart.



The **PlotArea EmptySeriesMessage** is a predefined message that displays in the PlotArea when there is no series data defined for the chart.

MarkedZones are areas in the background of the chart that can be defined, labeled and filled. MarkedZones are used to highlight or group areas on the chart and by default display behind the chart series. You can create any number of members for the MarkedZones collection and each marked zone is defined by starting and ending X and Y value pairs. There are two marked zones displayed in the screenshot below that delineate extreme high and low temperatures.

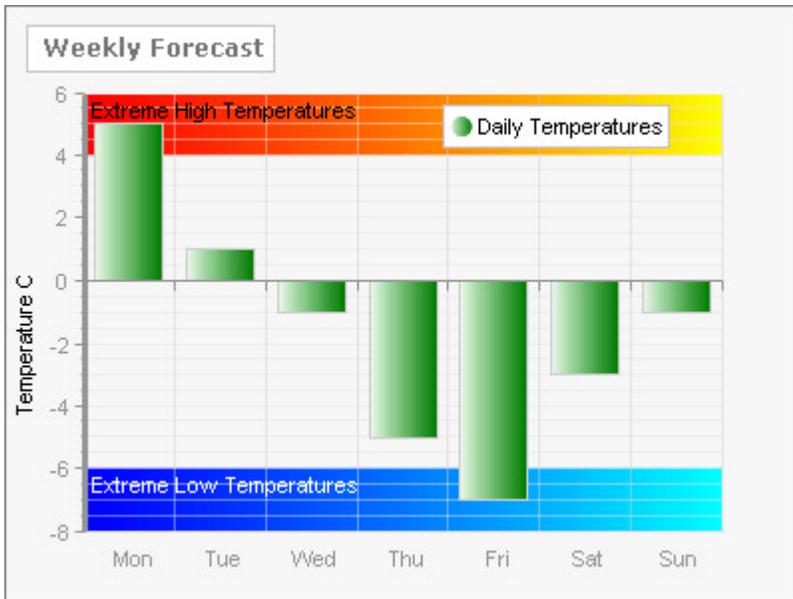
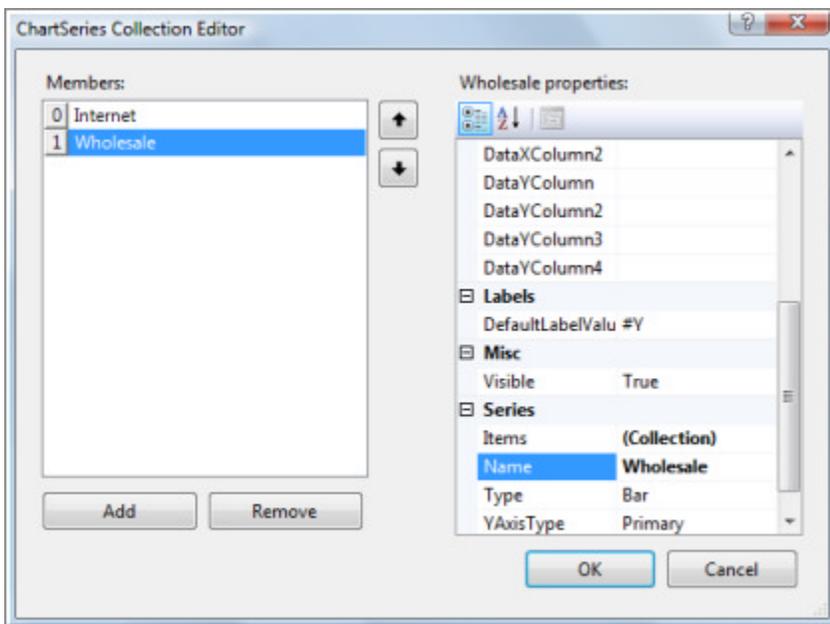


Chart Series

Series contains a set of data points to be drawn on the chart. This set of points contains related data. Each series can be represented by a chart type. Pie charts use only a single series. For other chart types there is no limitation to the number of series or items within each series. The screenshot below shows two series named "Internet" and "WholeSale" defined within the ChartSeries Collection Editor.



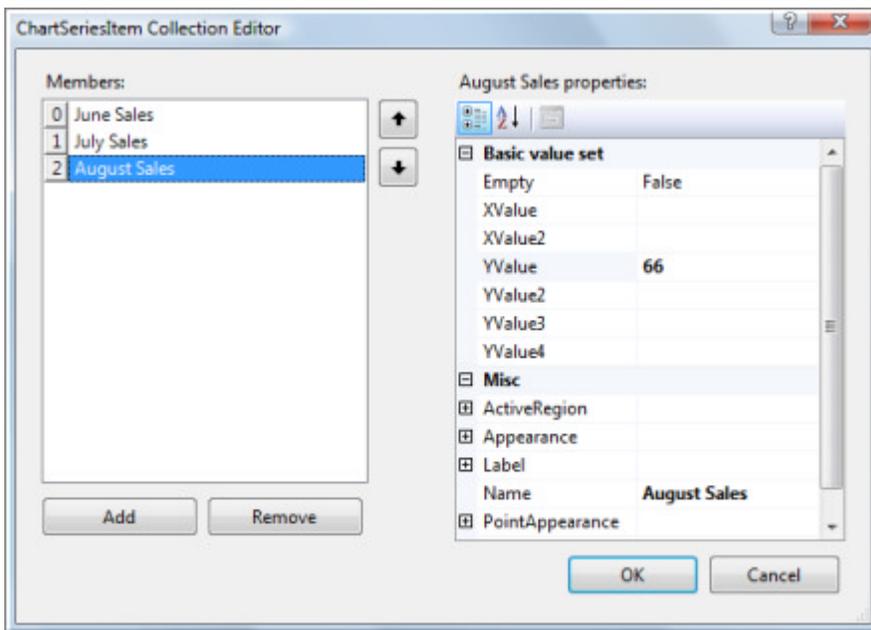
The **DefaultLabelValue** holds label formatting options for the series:

- Use "#Y" or "#X" to display numbers from the X or Y axis respectively
- Use "#%" for a percentage of the total sum (of all items).
- Use "#SUM" to display the total of all items.
- "#STSUM" displays the sum of a stacked series.

- "#SERIES" displays the series name.
- "#ITEM" displays the item name.
- You can also use standard numeric format strings. Use curly brackets to contain the formats. For example, you can display Y values as currency by setting DefaultLabelValue to "#Y{C}".

Series Items

Each chart series item encapsulates a single data point within a chart series. For simple charts along a single axis, you can populate the **YValue** property only. Use the **XValue** property to add a second data dimension. For example, the Y values could represent "Sales Volume" and the X values might show time periods or geographic regions. The meaning of the **XValue2** and **YValue2** properties vary depending on the type of chart. For example **XValue2** and **YValue2** are used by Gantt type to indicate a period of time and the Bubble chart type to show amplitude of data.



Tour of Chart Types

Here is a quick 1000 foot view of the available chart types and a few ideas on how you might use them.

Bar



Bar charts graphically display values in vertical and horizontal bars across categories. Bar charts are useful for comparing multiple series of data (i.e. providing snapshots of data at particular points in time).

Stacked Bar



Stacked Bar charts are used to compare contributions of values to a total across categories. Use the Stacked Bar chart when you need visibility to the combined values for each category.

Stacked Bar 100%



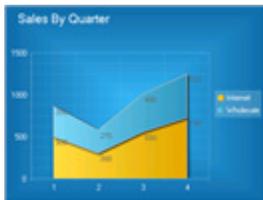
Stacked Bar 100% shows the combined contribution of values as percentages where the combined total for each category is 100 percent. Use when the relationship between values in a category is more significant than the amounts.

Area



The Area chart consists of a series of data points joined by a line and where the area below the line is filled. Area charts are appropriate for visualizing data that fluctuates over a period of time and can be useful for emphasizing trends.

Stacked Area



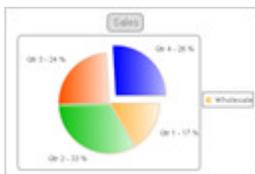
The Stacked Area chart is a variation of the Area chart that display trends of the contribution of each value over time (or across categories). The areas are stacked so that each series adjoins but does not overlap the preceding series. Area charts are appropriate for visualizing data that fluctuates over a period of time and where the entire area for all series data must be visible at one time.

Stacked Area 100%



Stacked Areas 100% charts are a variation of Stacked Area charts that present values for trends as percentages, totaling to 100% for each category. Use this chart type to visualize data that fluctuates over a period of time and where the relationship between values in a category is more significant than the amounts.

Pie



The Pie chart shows slices representing fractional parts of a whole.

Gantt



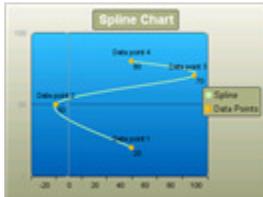
Gantt charts, also known as Time charts, display separate events as bars along a time scale. These charts are often used for project/time planning, where data can be plotted using a date-time scale or other numeric scale.

The Bezier chart displays a series of points on a curved line. Two "control points" determine the position and amount of curvature in the line between end points. The Bezier chart is often used for data modelling by taking a limited number of data points and interpolating or estimating the intervening values.

Bezier

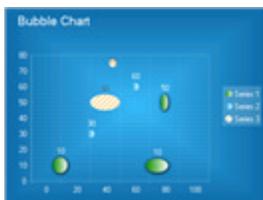


Spline



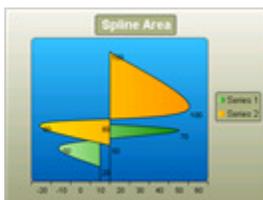
Spline charts allow you to take a limited set of known data points and approximate intervening values. The Spline chart, like the Bezier, is often used for data modelling by taking a limited number of data points and interpolating or estimating the intervening values.

Bubble



The Bubble chart is an extension of the Point chart but each point can be a circle or oval of any size or dimension. The bubble size may be used to convey larger values. The Bubble chart is often used for scientific data modeling or financial data.

Spline Area



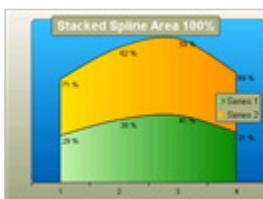
The Spline Area chart type defines one or more spline curves and fills in the area defined by the spline. This chart type can also be used for data modelling in that it takes a limited number of data points and interpolates the intervening values.

Stacked Spline Area



The Stacked Spline Area chart is a variation of the Spline Area chart. The areas are stacked so that each series adjoins but does not overlap the preceding series. Also can be used for data modelling in that it takes a limited number of data points and interpolates the intervening values. This chart type allows the entire surface area for all sequences to be displayed at one time.

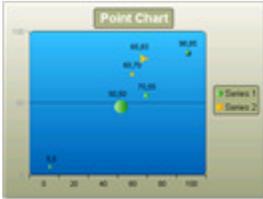
Stacked Spline Area 100%



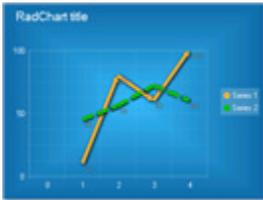
The Stacked Spline Area 100% chart is a variation of the Spline Area chart. The areas are stacked so that each series adjoins but does not overlap the preceding series and where the combined total for each category is 100 percent. The Stacked Spline Area 100% chart can also be used for data modelling in that it takes a limited number of data points and interpolates the intervening values. This chart type allows the entire surface area for all sequences to be displayed at one time. Use this chart type when the relationship between values in a category is more significant than the amounts.

Point or "Scatter" charts are used to show correlations between two sets of values. The Point chart is often used for scientific data modeling or financial data. The Point chart is typically not used with time dependent data where a Line chart is more suited.

Point



Line



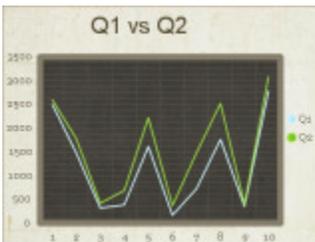
The Line chart type displays a set of data points connected by a line. A common use for the line chart is to show trends over a period of time.

CandleStick



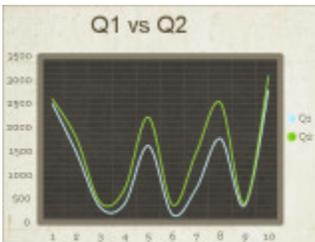
The CandleStick chart combines bar and line chart styles to show a range of value movement over time. Dark colored bars show downward trends, light colored bars show upward trends and the line through the center (the "wick") shows the extreme high and low values. Use this chart type to visualize price or currency fluctuations. Typically this chart is used to analyze stock prices or currency changes.

Stacked Line



The Stacked Line chart allows multiple series of Y values to be compared.

Stacked Spline



The Stacked Spline chart, like the Stacked Line, lets you have multiple series of Y values. It can take a limited number of data points and interpolate the intervening values.

40.4 Designer Interface

Smart Tag

The RadChart Smart Tag contains control-specific areas in addition to the standard Ajax Resources, Skin selection, and Learning center sections.

RadChart Tasks

Layout

Width: 400px

Height: 300px

Appearance

Title text: Q1 vs Q2

Chart series orientation: Vertical

Default chart type: StackedSpline

Skin: Marble

Data

Chart series collection: (Collection)

Choose data source: SqlDataSource1

[Configure Data Source](#)

[Refresh Schema](#)

Setting

Chart image format: Png

[Chart Wizard](#)

Ajax Resources:

[Add RadAjaxManager...](#)

[Replace ScriptManager with RadScriptManager](#)

[Add RadStyleSheetManager](#)

Learning center:

[Online RadChart for ASP.NET Ajax examples](#)

[Online RadChart for ASP.NET Ajax help](#)

[Online RadControls for ASP.NET Ajax Code Library](#)

Search [www.telerik.com](#) for:

[Search](#)

[Go to online Telerik support center](#)

Layout

At the top of the Smart Tag in the Layout section, you can set the **Width** and **Height** of the chart as a whole.

Appearance

Below the Layout area, you can use the Appearance section to quickly set the

- **Title Text**
- **Chart Series Orientation** to Horizontal or Vertical from the drop down list.
- **Default Chart Type** to one of the chart types in the drop down list, i.e. Bar, Pie, Line or any of the types we reviewed in the Getting Started section.
- **Skin** can be set from the drop down list to quickly style the entire look of the chart.

Data

You can bring up the Chart Series collection editor from the ellipses if you want to statically define series and items directly at design time. If you want to bind data, select a data source from the drop down list. If no data sources exist in the project yet, select "<New Data Source...>" from the drop down list. Once a data source is configured and selected, two additional links are displayed in this area, "Configure Data Source" and "Refresh

Schema".

Setting

From the Setting section you can choose a **Chart Image Format** from the drop down list. Most popular formats are supported including MemoryBmp, Bmp, Emf, Gif, Jpeg, Png, Tiff, Exif and Icon.

A link to the Chart Wizard lets you execute the wizard dialog for settings that are more detailed than the Smart Tag, but much smaller than the total number of properties available from the Properties window.

Chart Wizard

The RadChart Wizard helps you traverse the many properties of RadChart by providing the most commonly used properties in an intuitive way. The wizard can help you quickly set up the basic structure of your chart. The Wizard functions are arranged in tabs:

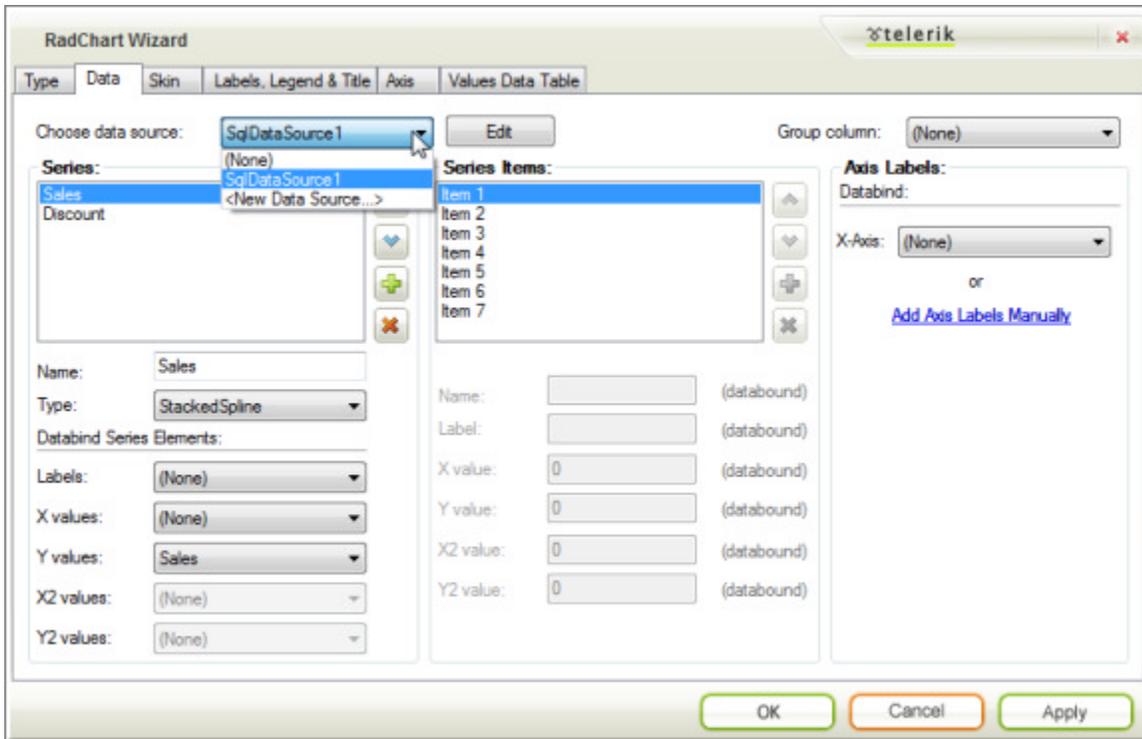
Type Tab

The Type tab lets you quickly choose the chart type by providing visual cues to what each type looks like. Here you can also choose the chart orientation.



Data Tab

The Data tab brings together the Series, Series Item, Axis labels and data binding to a single screen. Here you can add data points to your chart manually or by binding to data sources.



Choose Data Source

Choose Data Source appears on the upper left hand portion of the screen. Select from an existing data source or select "new data source" from the drop down list. If you have an existing data source selected, click the Edit button to reconfigure the data source in the Configure Data Source Wizard.

Group Column

The Group Column appears on the upper right side. Select a column name from a bound data source to group by that column data.

Series

Use the Series area of the tab to add, delete and reorder chart series elements using the list box provided. Use the plus and minus buttons to add or delete a series element. Use the up and down arrows to move a series element up or down in the list. For each selected series element in the list box you can provide a name and select from the list of chart types.

If you bind to a data source the Databind Series Elements portion will be enabled and allow you to choose column names for your labels and values from the drop down lists provided. When you bind to a data source the Series list box will be populated automatically with a series for each numeric column in the data source. If you need to fine tune the behavior or appearance of a series in more depth than the Data tab provides, use the RadChart Series property in the property window.

Series Items

For each series you select in the Series area list, you can add, edit, delete and reorder entries. Use the plus and minus buttons to add and delete series items. Use the up and down arrows to move series items up or down in the list. For each item you can set the Name, Label and X and Y Values. X2 and Y2 values are enabled for Gantt

and Bubble chart types.

Axis Labels

This section lets you choose between binding to a column in the data source and using the column data to populate the labels along an Axis. Click the **Add Labels Manually** link to navigate to the Axis tab.

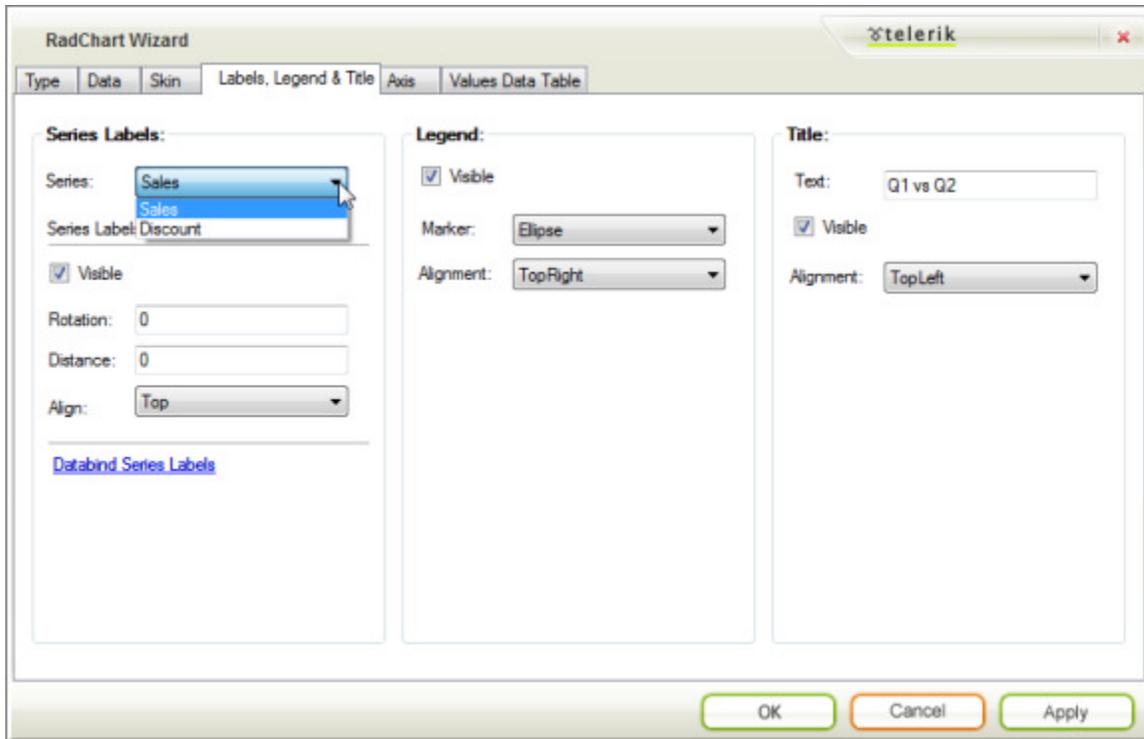
Skin Tab

The RadChart Skin property lets you apply a coordinated set of style changes to all the chart visual aspects at one time. The Skin tab lets you visually inspect how a chart might look with a given skin. The skins displayed reflect the current chart type.



Labels, Legend and Title Tab

Use this tab to tailor the principal labeling characteristics of the chart all at one time.



Series Labels

This section lets you set label properties for a series name selected in the Series drop down list. Uncheck the Visible box to hide series labels. Enter a value between 0 and 360 to the Rotation entry to rotate all series labels at one time. Positive Rotation values rotate the labels clockwise, negative values rotate the labels counter-clockwise. Positive Distance values move the labels away from the chart series items.

Legend

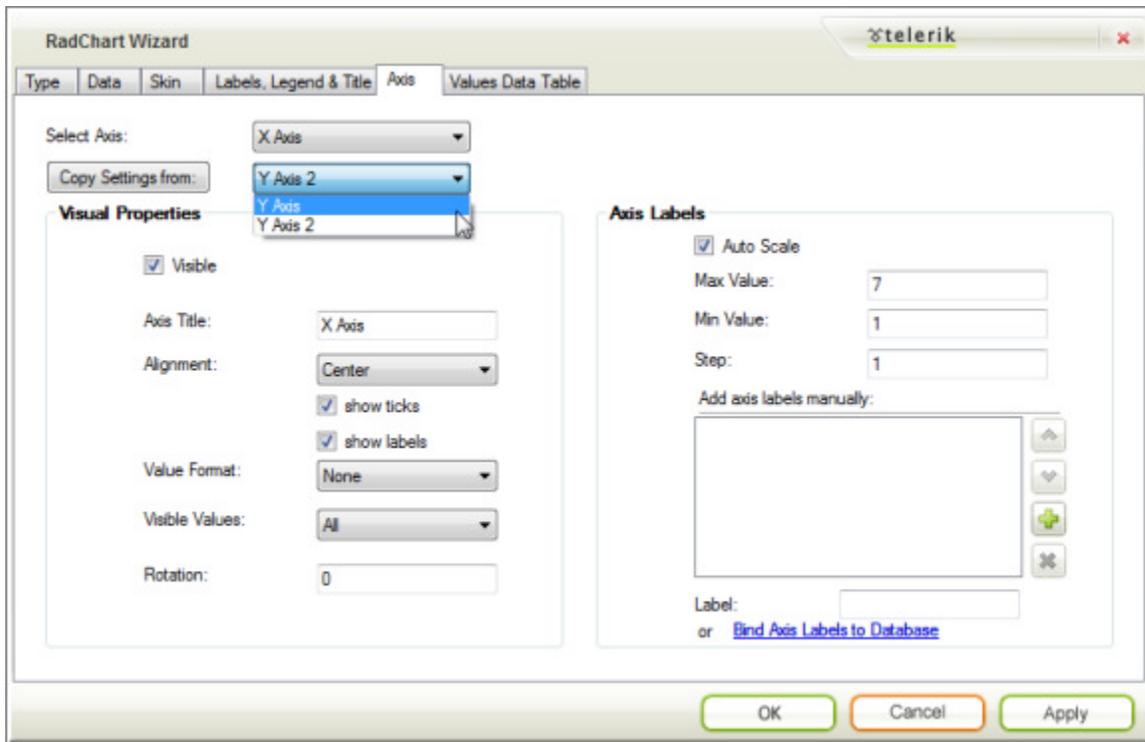
Un-select the Visible check box to hide the legend. Use the Marker drop down to select from a predefined list of shapes, e.g. Cross, Diamond, Ellipse, Rectangle, etc. Use the Alignment drop down to move the legend position between None, Left, Top, Bottom, Center, TopRight, TopLeft, BottomRight and BottomLeft.

Title

The Title section lets you set the text and toggle visibility of the chart title. Use the Alignment drop down to move the title position between None, Left, Top, Bottom, Center, TopRight, TopLeft, BottomRight and BottomLeft.

Axis Tab

From this tab you can select an axis from the drop down list at the top of the page. Properties you modify will be retained for the selected axis. Use the Copy Settings From button to replicate settings from another axis.



Visual Properties

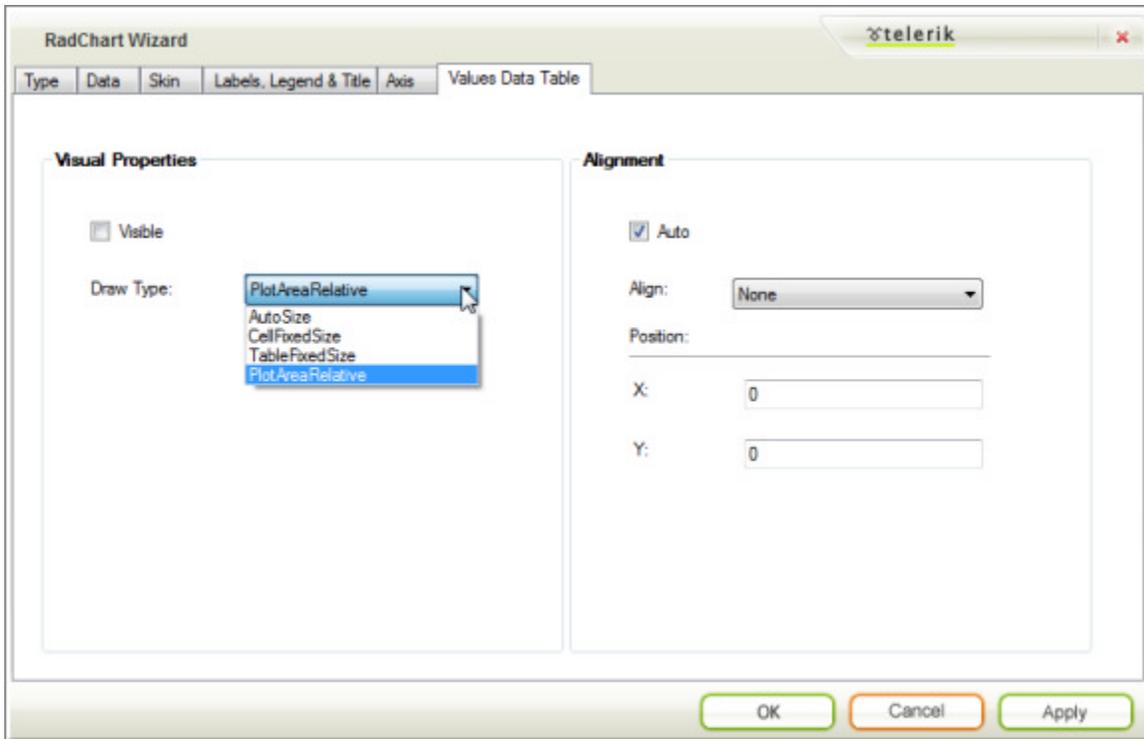
The Visual Properties section of the page controls properties for the axis as a whole. Uncheck the **Visible** checkbox to hide the entire axis (including labels and tick marks). The **Axis Title** text populates a single label that appears for the axis as a whole. Use the **Alignment** property to place the axis label in a predefined position, e.g. Left, Right, Top, Bottom, Center, TopRight, TopLeft, BottomRight, BottomLeft. Un-check **Show Ticks** to hide the axis tick marks. Un-check **Show Labels** to hide the axis labels (but not the Axis Title). The **Value Format** drop down list automatically formats axis labels as various kinds of dates, times, percentages, numbers and currency. **Visible Values** can be All, Positive or Negative values. **Rotation** is used to rotate the axis label text. Positive numbers spin the labels clockwise, negative numbers counter-clockwise.

Axis Labels

Turn off **Auto Scale** if you want to provide custom axis labels instead of the default numeric values. Turning off Auto Scale also lets you use the Min, Max and Step values. Enter **Min** and **Max** values to control the number of series items to be displayed along that axis. Enter a **Step** value to control the interval between axis labels. If Auto Scale is off you can use the provided list box to add, delete and reorder axis label items manually. By selecting any one of the axis label values in the listbox you can assign a text label. Click the **Bind Axis Labels to Database** link to navigate back to the Data tab.

Values Data Table Tab

The Values Data Table tab controls the general look and positioning of the chart data table.



Visual Properties

Check **Visible** to display the chart data table. By default this is unchecked. Select **Draw Type** from the drop down list to control the general size and positioning of the chart:

- Select **AutoSize** to have each cell size to the data inside of it.
- **PlotAreaRelative** places each cell just below the chart series item it represents.
- **CellFixedSize** and **TableFixedSize** fix the size of the cells or table irrespective of the data it contains.

Alignment

Use the **Align** drop down list to place the chart data table in a predefined position (e.g. Top, Bottom, BottomRight, etc.) To place the data table at exact coordinates, un-check **Auto** and enter values for X and Y.

Properties Window

At design time, you can use the Properties Window to configure almost every aspect of the chart. You will need to build a mental map of how the critical properties are arranged. At the top level the critical properties are **ChartTitle**, **DataSourceID**, **Legend**, **PlotArea** and the **Series** collection. Within the **Series** collection are **Items** collections that define the individual data points in the series. Other helpful properties:

IntelligentLabelsEnabled: For charts that have many data points or data points with values close to one another, labels tend to collide making readability a problem. The **Intelligent Labels** feature of **RadChart** automatically re-aligns labels making each labeled value stand out clearly.

UseSession and **TempImagesFolder**: If **UseSession** is true (default value), the chart is being streamed through the session. If **UseSession** is false, the images can be streamed to the path specified in **TempImagesFolder** ("~/Temp" by default).



Gotcha!

If you are using the chart in a web farm, make sure that the session state is either *StateServer* or *SQLServer*. If this is not so, the chart image can be generated on one server and the image request

served by another. If the Session is not common for all servers, the chart image will be lost.

AutoTextWrap when true causes text to be wrapped for all text blocks within the chart control.

SeriesOrientation can be Horizontal or Vertical.

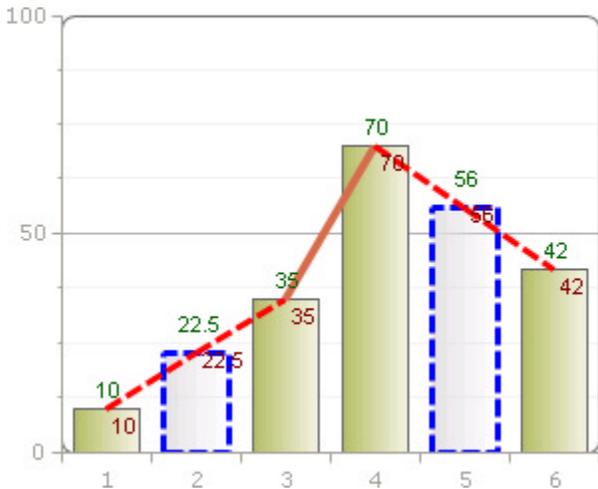
Skin sets the color scheme for the entire chart. **SeriesPalette** lets you use a color scheme for the series and series items that is different from the chart Skin. **SkinsOverrideStyles** when true (false by default) use the Skin only and ignore SeriesPalette and Appearance property settings.

40.5 Control Specifics

RadChart has some unique features that we haven't run into yet that you should be aware of:

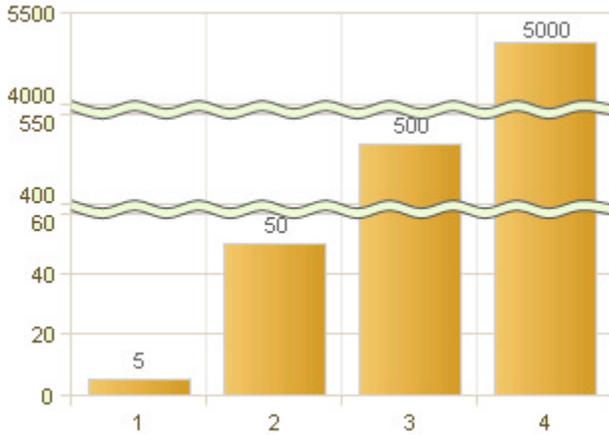
Empty Values

RadChart automatically approximates missing values between known data points, simply by setting the **Empty** property true on any chart series item. This works for bar, line and area based chart types. You also have complete control over the visual style of empty values. The empty value style can be articulated separately from the style for the main values.



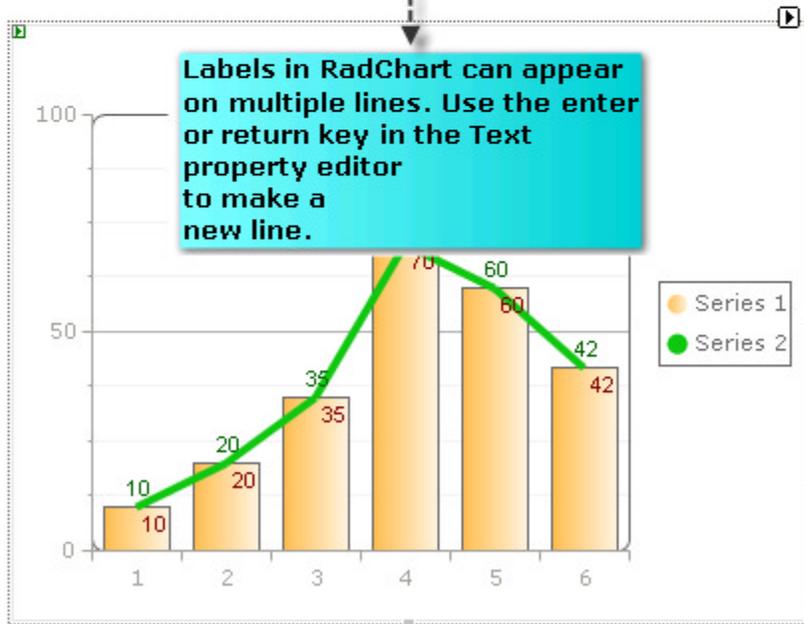
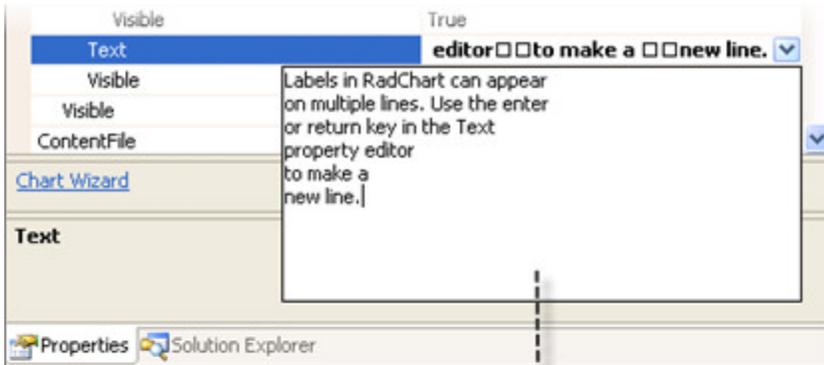
Scale Breaks

The **ScaleBreaks** feature allows you to "break off" large chunks of the axis so that graphs with large amplitude are easier to read. **ScaleBreaks** are available for both **YAxis** and **YAxis2** properties of the **PlotArea**. You can tailor the maximum number of breaks, the minimum interval between data points before a break can occur, the visual size of the breaks and the visual style of the breaks.



Multi-Line Labels

Labels in RadChart can appear on multiple lines. For example, the property editor for `TextBlock.Text` properties allows you to hit the enter key to start a new line. Press control-enter to accept the text and close the property editor.



Strict Mode

UI for ASP.NET AJAX

"Strict mode" is not a property or setting, but a behavior of bar chart series where X values are respected and bars are positioned according to their XValues. If there are no series items with XValues then RadChart resumes standard sequential ordering of each item.

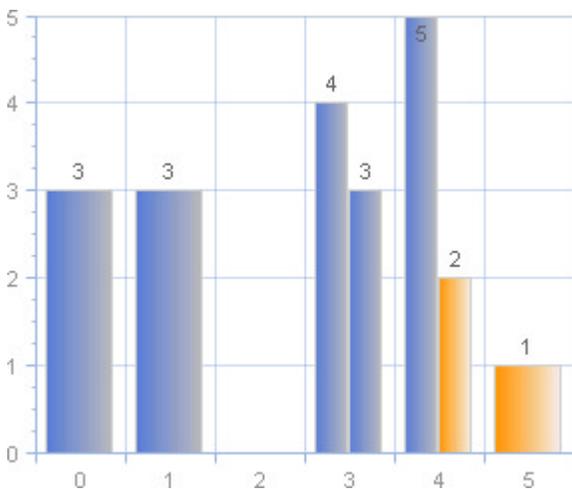
The screen shot below was produced using the X and Y values from this table:

Series 1 (Blue)

YValue	XValue
3	0
3	1
4	3
3	3
5	4

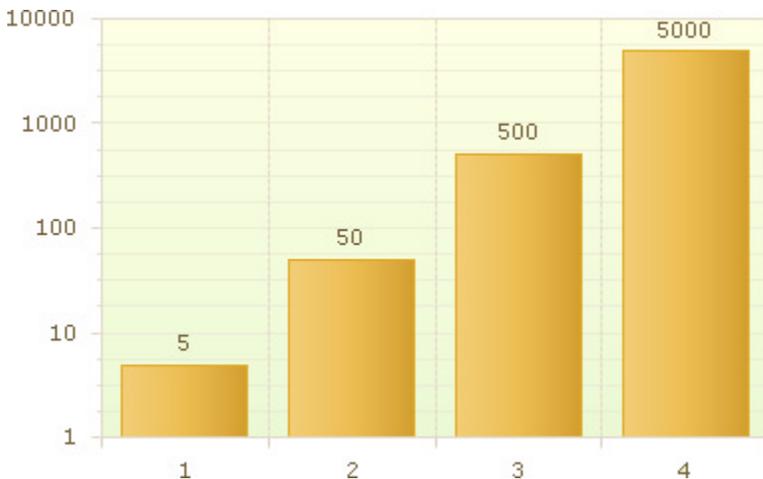
Series 2 (Orange)

YValue	XValue
1	5
2	4



Logarithmic Y-Axis

RadChart's Y-Axis now supports logarithmic mode. This is convenient when you would like to display rapidly increasing values. Set the YAxis or YAxis2 **IsLogarithmic** property (false by default) to true to enable this behavior. The **LogarithmBase** property (10 by default) can be increased to further compress the presentation of values.



40.6 Server-Side Programming

Create a RadChart Series Programmatically

You can create and configure all aspects of the chart programmatically, from the chart itself, right down to the smallest data point or tick mark. A typical task would be to create series and series items at runtime. To create the series object use one of the many constructor overloads. The example below passes in the chart series name and chart type. You will need to add a **Telerik.Charting** reference to your "Imports" (VB) or "uses" (C#) clause to support the **ChartSeriesType** used here.

[VB] Adding a Chart Series

```
Dim chartSeries As New ChartSeries("Average Temperatures", ChartSeriesType.Bar)
RadChart1.Series.Add(chartSeries)
```

[C#] Adding a Chart Series

```
ChartSeries chartSeries =
    new ChartSeries("Average Temperatures", ChartSeriesType.Bar);
RadChart1.Series.Add(chartSeries);
```

To add items to the new series, call the **ChartSeries AddItem()** method. **AddItem()** also has several overloads. Two versions of the method are shown below. The first is a quick way of getting started with adding data by simply defining a Y value. The second creates a **ChartSeriesItem** and passes a boolean value. The boolean value overload is interpreted as an empty value item if true.

[VB] Adding a Chart Series Item

```
' add an item with a Y value
chartSeries.AddItem(5)
' add an empty item
Dim isEmpty As Boolean = True
Dim item As New ChartSeriesItem(isEmpty)
chartSeries.AddItem(item)
```

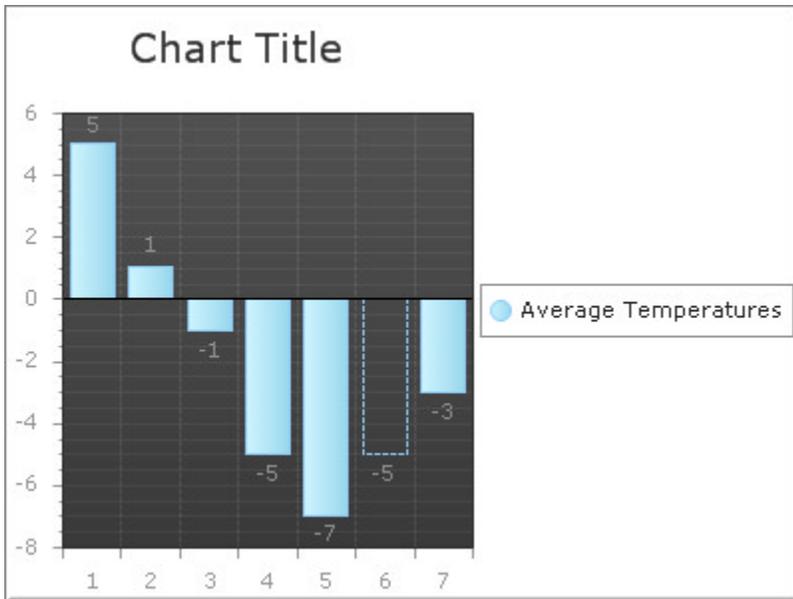
[C#] Adding a Chart Series Item

```
// add an item with a Y value
chartSeries.AddItem(5);
// add an empty item
bool isEmpty = true;
ChartSeriesItem item = new ChartSeriesItem(isEmpty);
chartSeries.AddItem(item);
```

Let's put both of these together in a quick example that plots a series of Y data points along a single series. This example assumes the RadChart has already been added to the page, the HTTP Handler has been added to the web.config.



You can find the complete source for this project at:
 \VS Projects\Chart\ServerSide2



[VB] Adding a Chart Series and Items

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    RadChart1.AutoLayout = True
    ' Create the series and assign the ChartSeriesType
    RadChart1.Series.Clear()
    Dim chartSeries As New ChartSeries("Average Temperatures", ChartSeriesType.Bar)
    ' Define the items in the series
    chartSeries.AddItem(5)
    chartSeries.AddItem(1)
    chartSeries.AddItem(-1)
    chartSeries.AddItem(-5)
    chartSeries.AddItem(-7)
    ' add an empty item
    Dim isEmpty As Boolean = True
    Dim item As New ChartSeriesItem(isEmpty)
    chartSeries.AddItem(item)
    chartSeries.AddItem(-3)
    ' Add the series to the chart, chart to page.
    RadChart1.Series.Add(chartSeries)
End Sub
```

[C#] Adding a Chart Series and Items

```
protected void Page_Load(object sender, EventArgs e)
{
    RadChart1.AutoLayout = true;
    // Create the series and assign the ChartSeriesType
    RadChart1.Series.Clear();
    ChartSeries chartSeries =
        new ChartSeries("Average Temperatures", ChartSeriesType.Bar);
    // Define the items in the series
    chartSeries.AddItem(5);
    chartSeries.AddItem(1);
    chartSeries.AddItem(-1);
    chartSeries.AddItem(-5);
    chartSeries.AddItem(-7);
}
```

```

// add an empty item
bool isEmpty = true;
ChartSeriesItem item = new ChartSeriesItem(isEmpty);
chartSeries.AddItem(item);

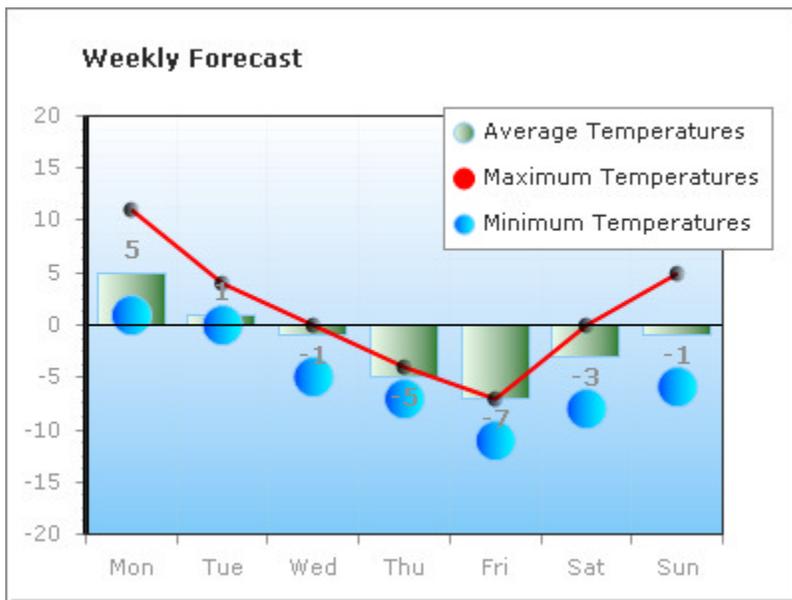
chartSeries.AddItem(-3);
// Add the series to the chart, chart to page.
RadChart1.Series.Add(chartSeries);
}

```

Let's extend our example that creates a chart series and items, to include three different series and chart types. Let's also configure the chart title, legend and axis labels.

One frequently asked question about RadChart is "how do I explicitly label one of the axis?". You do that by turning off the **AutoScale** property and adding your own axis array members. In this example we will add the short day names along the bottom of the chart.

A second question is "how do I stop all the values from scrunching up too high on the chart?". The remedy here is to set the **YXis AxisMode** to "Extended" for a little more headroom.



You can find the complete source for this project at:
 \VS Projects\Chart\ServerSide1

1. Create an ASP.NET AJAX Web Application.
2. Create a new ASP.NET Web Application and drag a **ScriptManager** from the Tool Box onto the Web page.
3. From the Toolbox drag a **RadChart** component to the default web page.
4. Click the Smart Tag **Add RadChart HTTP Handler to Web.Config** link.
5. Begin coding the **Page_Load** handler by setting up the label and chart title.

*For the legend, you need to shut off the **Appearance.Position.Auto** so that you can explicitly position the legend exactly where you want it. You could also have used one of the predefined positions, hidden the legend or set the **Appearance fill** to a transparent color so you could see through to the data points*

beneath.

The *ChartTitle* is positioned to the upper left and the *Text* is "Weekly Forecast".

[VB] Defining the Chart Legend and Title

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Configure the Legend and Chart Title labeling
    RadChart1.Legend.Appearance.Position.Auto = False
    RadChart1.Legend.Appearance.Position.X = 220
    RadChart1.Legend.Appearance.Position.Y = 50
    RadChart1.ChartTitle.Appearance.Position.AlignedPosition =
    Telerik.Charting.Styles.AlignedPositions.TopLeft
    RadChart1.ChartTitle.TextBlock.Text = "Weekly Forecast"
    '...
End Sub
```

[C#] Defining the Chart Legend and Title

```
protected void Page_Load(object sender, EventArgs e)
{
    // Configure the Legend and Chart Title labeling
    RadChart1.Legend.Appearance.Position.Auto = false;
    RadChart1.Legend.Appearance.Position.X = 220;
    RadChart1.Legend.Appearance.Position.Y = 50;
    RadChart1.ChartTitle.Appearance.Position.AlignedPosition =
    Telerik.Charting.Styles.AlignedPositions.TopLeft;
    RadChart1.ChartTitle.TextBlock.Text = "Weekly Forecast";

    //...
}
```

- Next, add code to the *Page_Load* event handler below the *Legend* and *ChartTitle* configuration code. Reduce the right margin of the *PlotArea* to 10%. Set the main fill color to white and the secondary fill color to *LightSkyBlue*:

[VB] Configure the PlotArea

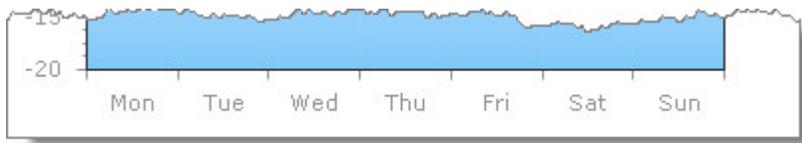
```
' Configure the PlotArea
RadChart1.PlotArea.Appearance.Dimensions.Margins.Right =
Telerik.Charting.Styles.Unit.Percentage(10)
RadChart1.PlotArea.Appearance.FillStyle.MainColor = System.Drawing.Color.White
RadChart1.PlotArea.Appearance.FillStyle.SecondColor = System.Drawing.Color.LightSkyBlue
```

[C#] Configure the PlotArea

```
// Configure the PlotArea
RadChart1.PlotArea.Appearance.Dimensions.Margins.Right =
    Telerik.Charting.Styles.Unit.Percentage(10);
RadChart1.PlotArea.Appearance.FillStyle.MainColor =
    System.Drawing.Color.White;
RadChart1.PlotArea.Appearance.FillStyle.SecondColor =
    System.Drawing.Color.LightSkyBlue;
```

- Configure the *XAxis*. Here we want to replace the default *X Axis* labeling that appears along the bottom of the chart with our own custom labels. To do this, set the *AutoScale* property to "false". If "true", you would see the numbers 1..7 along the bottom of the chart. Call the *XAxis AddRange()* method, passing the minimum value (1), maximum value (7) and the step (1). Now go back and manually populate the text for each *XAxis* element in the collection with the short names of the days. The image below shows the effect

on the XAxis labels.



[VB] Configuring the XAxis

```
' Configure the XAxis
RadChart1.PlotArea.XAxis.AutoScale = False
RadChart1.PlotArea.XAxis.AddRange(1, 7, 1)
RadChart1.PlotArea.XAxis(0).TextBlock.Text = "Mon"
RadChart1.PlotArea.XAxis(1).TextBlock.Text = "Tue"
RadChart1.PlotArea.XAxis(2).TextBlock.Text = "Wed"
RadChart1.PlotArea.XAxis(3).TextBlock.Text = "Thu"
RadChart1.PlotArea.XAxis(4).TextBlock.Text = "Fri"
RadChart1.PlotArea.XAxis(5).TextBlock.Text = "Sat"
RadChart1.PlotArea.XAxis(6).TextBlock.Text = "Sun"
```

8.

[C#] Configuring the XAxis

```
// Configure the XAxis
RadChart1.PlotArea.XAxis.AutoScale = false;
RadChart1.PlotArea.XAxis.AddRange(1, 7, 1);
RadChart1.PlotArea.XAxis[0].TextBlock.Text = "Mon";
RadChart1.PlotArea.XAxis[1].TextBlock.Text = "Tue";
RadChart1.PlotArea.XAxis[2].TextBlock.Text = "Wed";
RadChart1.PlotArea.XAxis[3].TextBlock.Text = "Thu";
RadChart1.PlotArea.XAxis[4].TextBlock.Text = "Fri";
RadChart1.PlotArea.XAxis[5].TextBlock.Text = "Sat";
RadChart1.PlotArea.XAxis[6].TextBlock.Text = "Sun";
```

9. Configure the YAxis AxisMode to Extended so that there is a little more room at the top of the chart. Set the Text for the AxisLabel.TextBlock to "Temperature C" and the Appearance.Width to "3".

[VB] Configure the YAxis

```
' Configure the YAxis
RadChart1.PlotArea.YAxis.AxisMode = ChartYAxisMode.Extended
RadChart1.PlotArea.YAxis.AxisLabel.TextBlock.Text = "Temperature C"
```

[C#] Configure the YAxis

```
// Configure the YAxis
RadChart1.PlotArea.YAxis.AxisMode = ChartYAxisMode.Extended;
RadChart1.PlotArea.YAxis.AxisLabel.TextBlock.Text = "Temperature C";
```

10. Clear the chart Series collection to remove the default two series that show up at design time when you add the chart to the page. Create a new ChartSeries with name "Average Temperatures" and type "Bar". Set the main color for the series Appearance FillStyle to "HoneyDew" and the secondary color to "Green".

[VB] Add the Chart Series

```
' Create the series and assign the ChartSeriesType
RadChart1.Series.Clear()
Dim chartSeries As New ChartSeries("Average Temperatures", ChartSeriesType.Bar)
chartSeries.Appearance.FillStyle.MainColor = System.Drawing.Color.Honeydew
chartSeries.Appearance.FillStyle.SecondColor = System.Drawing.Color.Green
```

[C#] Add the Chart Series

```
// Create the series and assign the ChartSeriesType
RadChart1.Series.Clear();
ChartSeries chartSeries =
    new ChartSeries("Average Temperatures", ChartSeriesType.Bar);
chartSeries.Appearance.FillStyle.MainColor =
    System.Drawing.Color.Honeydew;
chartSeries.Appearance.FillStyle.SecondColor =
    System.Drawing.Color.Green;
```

11. Add the code below to the end of the Page_Load event handler: Add the data points to the first series by using the **AddItem()** method of the chart series and passing Y values.

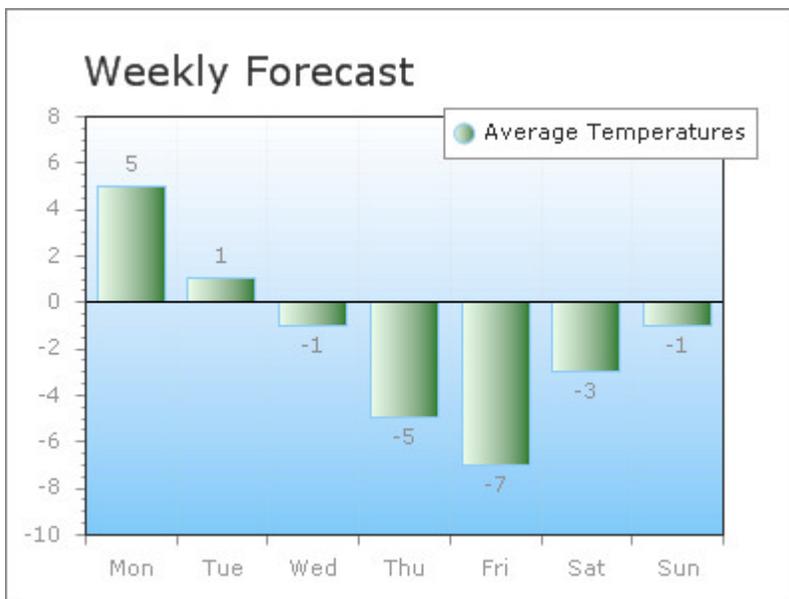
[C#] Adding Chart Series Items

```
' Define the items in the series
chartSeries.AddItem(5)
chartSeries.AddItem(1)
chartSeries.AddItem(-1)
chartSeries.AddItem(-5)
chartSeries.AddItem(-7)
chartSeries.AddItem(-3)
chartSeries.AddItem(-1)
```

[C#] Adding Chart Series Items

```
// Define the items in the series
chartSeries.AddItem(5);
chartSeries.AddItem(1);
chartSeries.AddItem(-1);
chartSeries.AddItem(-5);
chartSeries.AddItem(-7);
chartSeries.AddItem(-3);
chartSeries.AddItem(-1);
```

The chart should now look something like the screenshot below:



12. Add the code below to the end of the Page_Load event handler: Add a second series with name "Maximum Temperatures" and type "Line". Hide the labels by setting the series **Appearance.LabelAppearance.Visible**

to "false". Set the `LineSeriesAppearance` Color to "Red".

[C#] Create and Configure Line Series

```
' Create a second series and assign the ChartSeriesType
Dim chartSeries2 As New ChartSeries("Maximum Temperatures", ChartSeriesType.Line)
chartSeries2.Appearance.LabelAppearance.Visible = False
chartSeries2.Appearance.LineSeriesAppearance.Color = System.Drawing.Color.Red
```

[C#] Create and Configure Line Series

```
// Create a second series and assign the ChartSeriesType
ChartSeries chartSeries2 =
    new ChartSeries("Maximum Temperatures", ChartSeriesType.Line);
chartSeries2.Appearance.LabelAppearance.Visible = false;
chartSeries2.Appearance.LineSeriesAppearance.Color =
    System.Drawing.Color.Red;
```

13. Add the code below to the end of the `Page_Load` event handler. Again, chart series items are added to the second series by calling `AddItem()` with Y values as parameters.

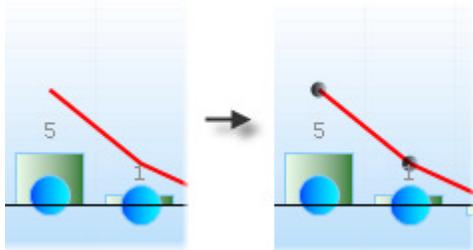
[VB] Add Items to the Second Series

```
' Define the items in the series
chartSeries2.AddItem(11)
chartSeries2.AddItem(4)
chartSeries2.AddItem(0)
chartSeries2.AddItem(-4)
chartSeries2.AddItem(-7)
chartSeries2.AddItem(0)
chartSeries2.AddItem(5)
```

[VB] Add Items to the Second Series

```
// Define the items in the series
chartSeries2.AddItem(11);
chartSeries2.AddItem(4);
chartSeries2.AddItem(0);
chartSeries2.AddItem(-4);
chartSeries2.AddItem(-7);
chartSeries2.AddItem(0);
chartSeries2.AddItem(5);
```

14. Add the code below to the end of the `Page_Load` event handler. Instead of displaying a red line only to represent "Maximum Temperatures", turn on the series `Appearance.PointMark` to make a black 5 x 5 pixel mark at each data point. You can see the before-and-after effect in the image below.



[VB] Enhance the "Maximum Temperature" Data Points

```
' visually enhance the data points
chartSeries2.Appearance.PointMark.Dimensions.Width = 5
chartSeries2.Appearance.PointMark.Dimensions.Height = 5
```

```
chartSeries2.Appearance.PointMark.FillStyle.MainColor = System.Drawing.Color.Black  
chartSeries2.Appearance.PointMark.Visible = True
```

[C#] Enhance the "Maximum Temperature" Data Points

```
// visually enhance the data points  
chartSeries2.Appearance.PointMark.Dimensions.Width = 5;  
chartSeries2.Appearance.PointMark.Dimensions.Height = 5;  
chartSeries2.Appearance.PointMark.FillStyle.MainColor =  
    System.Drawing.Color.Black;  
chartSeries2.Appearance.PointMark.Visible = true;
```

15. Add a third "Minimum Temperatures" series, add items and set the **PointMark** appearance for the series. This code is very similar to the code for the second "Maximum Temperatures" series except that the colors and Y values are different.

[VB] Add the "Minimum Temperatures" Series

```
' Create a third series and assign the ChartSeriesType  
Dim chartSeries3 As New ChartSeries("Minimum Temperatures", ChartSeriesType.Bubble)  
chartSeries3.Appearance.LabelAppearance.Visible = False  
chartSeries3.Appearance.FillStyle.MainColor = System.Drawing.Color.Blue  
chartSeries3.Appearance.FillStyle.SecondColor = System.Drawing.Color.Aqua  
' Define the items in the series  
chartSeries3.AddItem(1)  
chartSeries3.AddItem(0)  
chartSeries3.AddItem(-5)  
chartSeries3.AddItem(-7)  
chartSeries3.AddItem(-11)  
chartSeries3.AddItem(-8)  
chartSeries3.AddItem(-6)  
' visually enhance the data points  
chartSeries3.Appearance.PointMark.Dimensions.Width = 5  
chartSeries3.Appearance.PointMark.Dimensions.Height = 5  
chartSeries3.Appearance.PointMark.FillStyle.MainColor = System.Drawing.Color.Black  
chartSeries3.Appearance.PointMark.Visible = True
```

[C#] Add the "Minimum Temperatures" Series

```
// Create a third series and assign the ChartSeriesType  
ChartSeries chartSeries3 =  
    new ChartSeries("Minimum Temperatures", ChartSeriesType.Bubble);  
chartSeries3.Appearance.LabelAppearance.Visible = false;  
chartSeries3.Appearance.FillStyle.MainColor =  
    System.Drawing.Color.Blue;  
chartSeries3.Appearance.FillStyle.SecondColor =  
    System.Drawing.Color.Aqua;  
// Define the items in the series  
chartSeries3.AddItem(1);  
chartSeries3.AddItem(0);  
chartSeries3.AddItem(-5);  
chartSeries3.AddItem(-7);  
chartSeries3.AddItem(-11);  
chartSeries3.AddItem(-8);  
chartSeries3.AddItem(-6);  
// visually enhance the data points  
chartSeries3.Appearance.PointMark.Dimensions.Width = 5;  
chartSeries3.Appearance.PointMark.Dimensions.Height = 5;  
chartSeries3.Appearance.PointMark.FillStyle.MainColor =
```

```
System.Drawing.Color.Black;
chartSeries3.Appearance.PointMark.Visible = true;
```

16. Add all three series to the RadChart **Series** collection.

[VB] Add to the RadChart Series Collection

```
' Add the series to the chart.
RadChart1.Series.Add(chartSeries)
RadChart1.Series.Add(chartSeries2)
RadChart1.Series.Add(chartSeries3)
```

[C#] Add to the RadChart Series Collection

```
// Add the series to the chart.
RadChart1.Series.Add(chartSeries);
RadChart1.Series.Add(chartSeries2);
RadChart1.Series.Add(chartSeries3);
```

17. Press **Ctrl-F5** to run the application.

Data Binding

RadChart data binding works similarly to other RadControls in that you can bind the same basic types, consume the same data source controls and can assign either **DataSourceID** declaratively or **DataSource** (and call **DataBind()**) at runtime. The control-specific differences are in the properties used to specify what columns are bound to particular displays and behaviors in the chart.

Data Binding Properties

ChartSeries comes with properties for **DataXColumn**, **DataXColumn2**, **DataYColumn** and **DataYColumn2**. At minimum you need to bind the ChartSeries **DataYColumn** to populate any chart type.

The Pie chart type only pays attention to the **DataYColumn**, but most other chart types also can bind to the **DataXColumn**. For example, the Point chart type can plot individual point marks where X and Y values intersect. The Bubble chart is an extension of the Point chart but each point can be a circle or oval of any size or dimension. Instead of using just the **XValue** and **YValue**, the Bubble chart uses **XValue/XValue2**, and **YValue/YValue2** pairs to define the dimensions of each bubble.

There are two other ChartSeries properties **DataYColumn3** and **DataYColumn4**. The CandleStick chart type uses all four Y column value properties where their meaning is:

- YValue = Open
- YValue2 = Close
- YValue 3 = Max
- YValue 4 = Min

The ChartSeries has a **DataLabelsColumn** property to define a column that will supply the text that displays next to each X Axis item. The XAxis also has this **DataLabelsColumn** property.

Data Binding Events

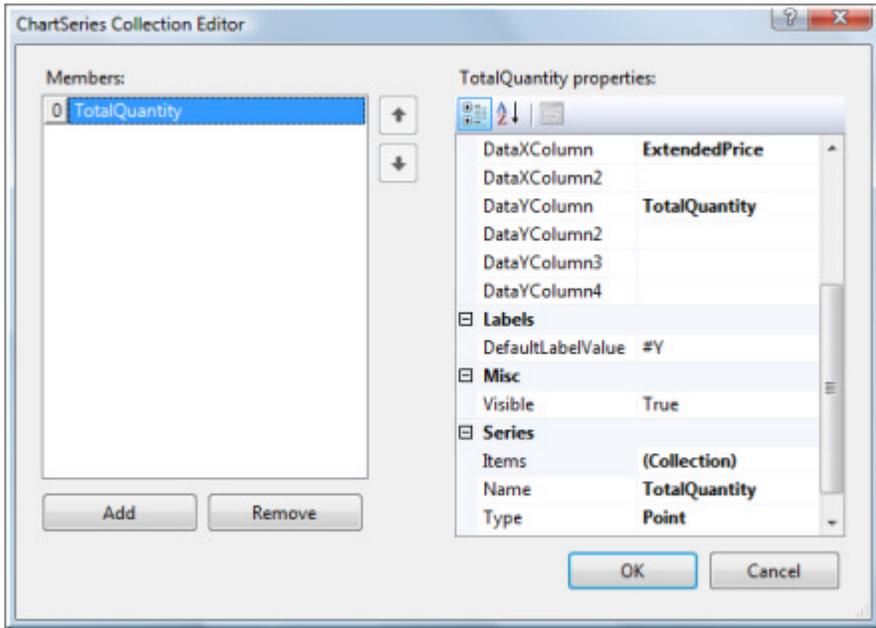
RadChart has a **OnItemDataBound** event that you can use to individually tailor **ChartSeriesItems** based on what's happening in the data item. The event handler takes a **ChartItemDataBoundEventArgs** parameter that brings **DataItem**, **ChartSeries** and **SeriesItem** properties along for the ride. You can use any of the columns in the data source for a particular data point (i.e. row) to make very specific changes to your **SeriesItem**.

Here's an example where we declaratively bind to the Telerik.mdf file (found in the Telerik RadControls installation directory under Live Demos\App_Data) and queries from the Products table:

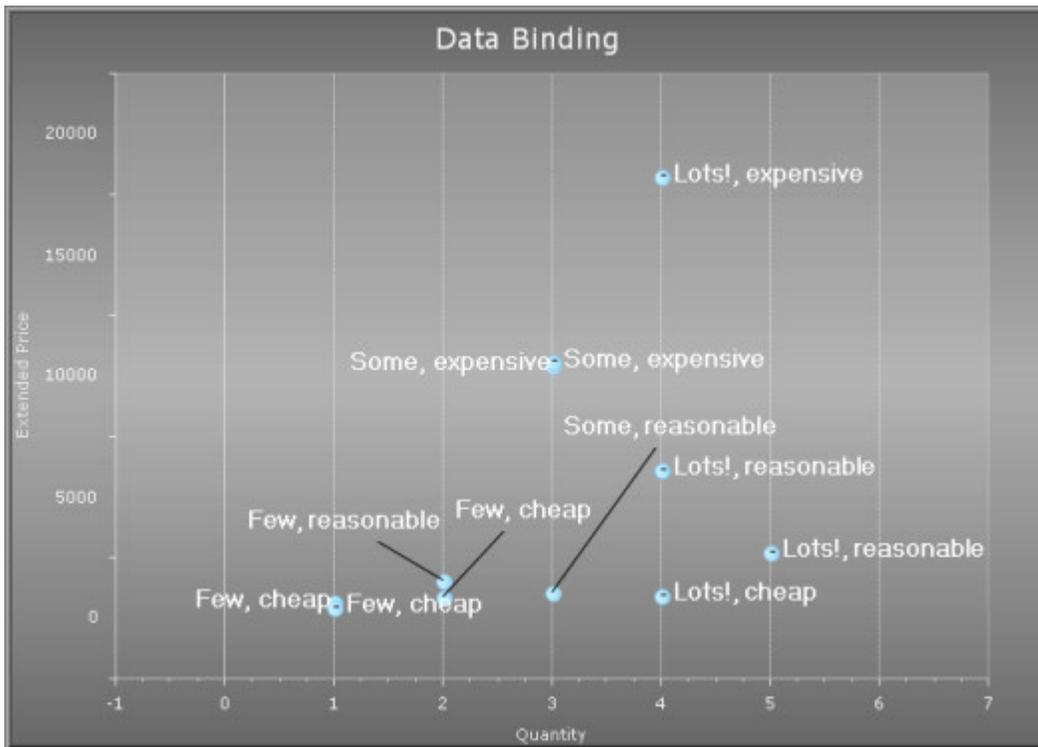
```
SELECT SalesRepresentative, SUM(Quantity) AS TotalQuantity, SUM(Quantity * Price) AS ExtendedPrice
FROM Products GROUP BY SalesRepresentative
```

UI for ASP.NET AJAX

This point chart plots each point where the "ExtendedPrice" appears along the X axis and the "TotalQuantity" along the Y axis.



The resulting chart looks something like the example below:



As the items are bound, the labels are formatted based on the ranges of values the data points fall within.

[C#] Handling the ItemDataBound Event

```
Protected Sub RadChart1_ItemDataBound(ByVal sender As Object, ByVal e As Telerik.Charting.ChartItemDataBoundEventArgs)
```

```

Dim qty As Integer = Convert.ToInt32((TryCast(e.DataItem, DataRowView))("TotalQuantity"))
Dim quantityLabel As String = [String].Empty
Select Case qty
    Case 1, 2
        quantityLabel = "Few"
        Exit Select
    Case 3
        quantityLabel = "Some"
        Exit Select
    Case 4, 5
        quantityLabel = "Lots!"
        Exit Select
End Select
Dim price As Double = Convert.ToDouble((TryCast(e.DataItem, DataRowView))("ExtendedPrice"))
Dim priceLabel As String = [String].Empty
If price < 1000 Then
    priceLabel = "cheap"
ElseIf price < 7500 Then
    priceLabel = "reasonable"
Else
    priceLabel = "expensive"
End If
e.SeriesItem.Label.TextBlock.Appearance.TextProperties.Font = New System.Drawing.Font
("Ariel", 12, System.Drawing.FontStyle.Bold)
e.SeriesItem.Label.TextBlock.Text = quantityLabel + ", " + priceLabel
End Sub

```

[C#] Handling the ItemDataBound Event

```

protected void RadChart1_ItemDataBound(object sender,
Telerik.Charting.ChartItemDataBoundEventArgs e)
{
    int qty = Convert.ToInt32((e.DataItem as DataRowView)["TotalQuantity"]);
    string quantityLabel = String.Empty;
    switch (qty)
    {
        case 1:
        case 2:
            quantityLabel = "Few";
            break;
        case 3:
            quantityLabel = "Some";
            break;
        case 4:
        case 5:
            quantityLabel = "Lots!";
            break;
    }
    double price = Convert.ToDouble((e.DataItem as DataRowView)["ExtendedPrice"]);
    string priceLabel = String.Empty;
    if (price < 1000)
    {
        priceLabel = "cheap";
    }
    else if (price < 7500)
    {

```

```
    priceLabel = "reasonable";
}
else
{
    priceLabel = "expensive";
}
e.SeriesItem.Label.TextBlock.Appearance.TextProperties.Font =
    new System.Drawing.Font("Arial", 12, System.Drawing.FontStyle.Bold);
e.SeriesItem.Label.TextBlock.Text = quantityLabel + ", " + priceLabel;
}
```

💡 Did you see the lines going from the labels to some of the data points? The problem came about because `IntelligentLabelsEnabled` was set to true. This moved the labels too far away from their respective data points, making the chart harder to interpret. Setting the `ChartSeries.Appearance.ShowLabelConnectors` property to true displays the lines between the labels and the data points.



You can find the complete source for this project at:
\\VS Projects\Chart\Databinding

Grouping Data Bound Items

You can group your data automatically by defining a column that

The `DataGroupColumn` property defines the column name in the datasource that is the criteria for grouping the chart series items. There will be as many series as the number of distinct values in this column. If we have these settings:

- Data with columns "Year", "Quarter" and "Value"
- "Year" contains multiple rows for "2007" and "2008".
- The `DataGroupColumn` property is "Year".

...then there will be two series, one for "2007" and the second for "2008".

A second `RadChart` property, `GroupNameFormat`, defines a format for the legend item. The format can have free text and can include two special words:

- **#NAME**: denotes the group column name.
- **#VALUE**: denotes the group column value (it is the same for all the records shown in the same series).

The SQL below gets a sampling of Invoice data and brings back the `CustomerID`, `ExtendedPrice` and `Quantity`.

[T-SQL] Selecting Invoice Data

```
SELECT TOP (25) CustomerID, ExtendedPrice, Quantity FROM Invoices ORDER BY CustomerID
```

💡 The "ORDER BY" clause counts for group queries. If the data in the example above was unordered, you would get a group for the first few records of customer "ALFKI", then a few records for "ANATR", then perhaps another bar for the next few "ALFKI" customer again. In typical cases adding the ORDER BY clause will give you the results you expect.

The screenshot below shows the `DataGroupColumn` set to "CustomerID". No series is set and the `DataYColumn` property of the series is not set. The actual values that shown in the bar are derived from the last numeric column in the datasource. In the figure below the "Quantity" data shows in the chart.



Gotcha! Don't define the series `DataYColumn` as it will take precedence over the group property settings.

Using the Year/Quarter/Value data mentioned above and if we set the `GroupNameFormat` to "#NAME: #VALUE", the legend will be "Year: 2007" and "Year: 2008". We can build this example by first creating a class to contain the Year/Quarter/Value, populating a generic list of these objects, setting the group properties and finally binding to the grid.



You can find the complete source for this project at:
 \VS Projects\Chart\Grouping

1. Create an ASP.NET AJAX Web Application
2. Create a new ASP.NET Web Application and drag a **ScriptManager** from the Tool Box onto the Web page
3. From the Toolbox drag a **RadChart** component to the default web page
4. Click the Smart Tag **Add RadChart HTTP Handler to Web.Config** link.
5. In the `Page_Load`, populate a generic List of Sales objects:

[VB] Populate and Group Chart Data

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    ' Populate the generic list of Sales
    Dim sales As New List(Of Sales)()
    sales.Add(New Sales(2007, 1, 5))
    sales.Add(New Sales(2007, 2, 2))
    sales.Add(New Sales(2007, 3, 3))
    sales.Add(New Sales(2007, 4, 1))
    sales.Add(New Sales(2008, 1, 4))
    sales.Add(New Sales(2008, 2, 3))
    sales.Add(New Sales(2008, 3, 8))
    sales.Add(New Sales(2008, 4, 2))
    ' Setup the chart appearance and title
    RadChart1.Skin = "DeepBlue"
    RadChart1.ChartTitle.TextBlock.Text = "Sales Grouped by Quarter"
    ' Remove the default series
    RadChart1.Series.Clear()
    ' Create and add a new Bar series type
    Dim chartSeries As New ChartSeries("Sales", ChartSeriesType.Bar)
    RadChart1.Series.Add(chartSeries)
    ' Set the grouping properties
    RadChart1.DataGroupColumn = "Year"
    RadChart1.Legend.Appearance.GroupNameFormat = "#NAME: #VALUE"
    ' bind the chart last to include the preceding property
    ' settings.
    RadChart1.DataSource = sales
    RadChart1.DataBind()
End Sub
```

[C#] Populate and Group Chart Data

```
protected void Page_Load(object sender, EventArgs e)
{
    // Populate the generic list of Sales
    List<Sales> sales = new List<Sales>();
    sales.Add(new Sales(2007, 1, 5));
    sales.Add(new Sales(2007, 2, 2));
    sales.Add(new Sales(2007, 3, 3));
    sales.Add(new Sales(2007, 4, 1));
    sales.Add(new Sales(2008, 1, 4));
    sales.Add(new Sales(2008, 2, 3));
    sales.Add(new Sales(2008, 3, 8));
    sales.Add(new Sales(2008, 4, 2));
    // Setup the chart appearance and title
    RadChart1.Skin = "DeepBlue";
    RadChart1.ChartTitle.TextBlock.Text =
        "Sales Grouped by Quarter";
    // Remove the default series
    RadChart1.Series.Clear();
    // Create and add a new Bar series type
    ChartSeries chartSeries =
        new ChartSeries("Sales", ChartSeriesType.Bar);
    RadChart1.Series.Add(chartSeries);
    // Set the grouping properties
    RadChart1.DataGroupColumn = "Year";
    RadChart1.Legend.Appearance.GroupNameFormat = "#NAME: #VALUE";
}
```

```

// bind the chart last to include the preceding property
// settings.
RadChart1.DataSource = sales;
RadChart1.DataBind();
}

```

- Press **Ctrl-F5** to run the application. Notice the two series, one for each year defined by the `DataGroupColumn`. Each year has four data points:



Use the axis `DataLabelColumn` property to add meaningful labels to the data across the bottom of this chart. If we had a property/column "QuarterDescription" with values "Qtr 1", "Qtr 2"... , these could be used in place of the number 1, 2...

Server Events

Use the RadChart **OnClick** event to handle server postbacks caused by clicking on areas of the chart. The event handler returns "sender", i.e. the RadChart itself and **ChartClickEventArgs**. `ChartClickEventArgs` contains `Element`, that is, the chart element that was clicked. For instance, you can test if `Element` is `Telerik.Charting.ChartTitle` to see if the `ChartTitle` was clicked. You can also use the `Element`'s `ActiveRegion` property to access the `ToolTip` and `Url` properties if you want to navigate based off the click.

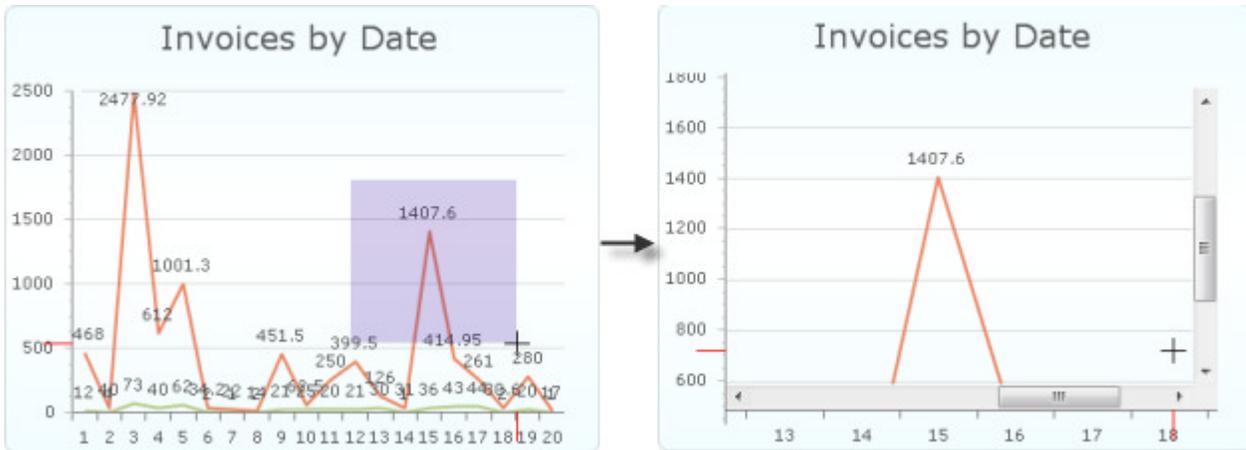
40.7 Client-Side Programming

Zooming and Scrolling Basics

The zooming and scrolling feature enables the user to zoom into an area of the chart so the data is shown in greater detail. For performance reasons, the visible image chunk is requested from the server-side. The user can scroll into view other parts of the chart data and the requested image chunks are automatically loaded via callback requests on the fly.

Manual Zooming and Scrolling

Manual zoom is performed by dragging a rectangle area over the chart with the mouse. This rectangle is exactly the area that will be shown in the Plot Area. Zooming, by default, performs regular postbacks but also works seamlessly using AJAX calls. In the screenshots below the area around "1407.6" is selected by the user. The second image shows the zoomed-in state of the chart and displays vertical and horizontal scroll bars for scrolling.

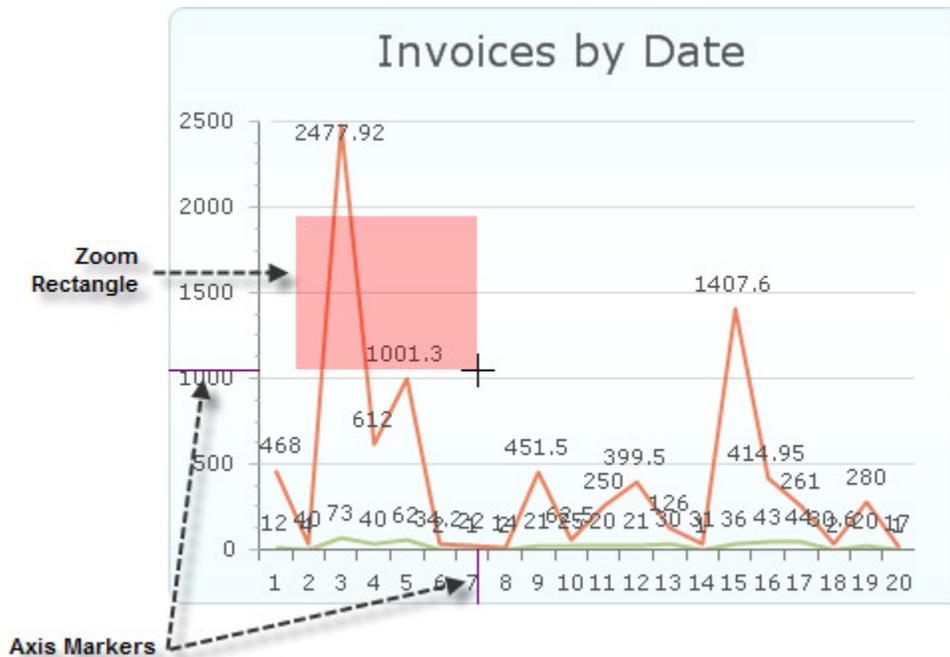


ClientSettings

RadChart has a **ClientSettings** property with sub properties that control zooming and scrolling. Zooming and Scrolling are disabled by default. Enable zooming and scrolling by setting the **ScrollMode** property to a value other than **None**. The available **ScrollMode** values are **None**, **XOnly**, **YOnly** and **Both**.

ClientSettings	
AxisMarkersColor	Red
AxisMarkersSize	20
EnableAxisMarkers	True
EnableZoom	True
ScrollMode	None
XScale	1
XScrollOffset	0
YScale	1
YScrollOffset	0
ZoomRectangleColor	51, 0, 153
ZoomRectangleOpacity	0.2

The image below shows the zoom rectangle and the axis markers that help the user know what area is to be zoomed.



You can customize the zoom rectangle using the **ZoomRectangleColor** and **ZoomRectangleOpacity** properties. The screenshot above sets the **ZoomRectangleColor** property to "Red" and the **ZoomRectangleOpacity** to .3 (making it slightly more opaque than the default .2). You can set **ZoomRectangleOpacity** from 0 (transparent) to 1 (completely opaque).

The axis markers are controlled by **AxisMarkersColor**, set to "Purple" in the screenshot above, and **AxisMarkersSize** that controls the length of the axis marker line in pixels. You can hide axis markers by setting **EnableAxisMarkers** to "false".

Scroll Only

You can also use scrolling alone by explicitly disabling manual client-side zooming (`RadChart.ClientSettings.EnableZoom = False`). You can still provide **XScale** and **YScale** values on the server-side. For example, the markup below allows the user to see a chart that is scaled by 4, can scroll along the X Axis, but cannot zoom.

Client-Side API

Zooming and scrolling can be controlled completely on the client side so you can make your chart interact with other elements on your web page. After getting a reference to the RadChart client object you can call the following methods:

- **scroll()**: programmatically scroll along both axis at one time or only along X or Y axis. Use the `get_xScrollOffset()` or `get_yScrollOffset()` methods to preserve the status quo. In the example code below, `get_xScrollOffset()` is used to preserve the current X offset so that the scroll only occurs along the Y axis.

[JavaScript] Using the scroll() method

```
var chart = $find("<%= RadChart1.ClientID %>");
chart.scroll(0.2, 0.3);
// scroll to top-left corner
chart.scroll(0, 0);
// scroll to bottom-right corner
chart.scroll(1, 1);
// scroll only by XAxis
chart.scroll(0.4);
```

UI for ASP.NET AJAX

```
// scroll only by YAxis  
chart.scroll(chart.get_xScrollOffset(), 0.4);
```

- **zoom():** Zooming can be done along X and Y axis combined or only along X or Y axis. Similar to the scrolling example above, use the `get_xScale()` and `get_yScale()` methods to preserve the existing offsets. You can also pass additional parameters to `zoom()` so that after zooming you can scroll at the same time.

[JavaScript] Using the zoom() method

```
var chart = $find("<%= RadChart1.ClientID %>");  
//scale XAxis by factor 3  
chart.zoom(3);  
//scale XAxis by factor 3 and YAxis by factor 2  
chart.zoom(3, 2);  
//scale only YAxis by factor 2  
chart.zoom(chart.get_xScale(), 2);  
//scale XAxis by factor 3 and YAxis by factor 2  
//scroll to bottom-right corner of the plotArea  
chart.zoom(3, 2, 1, 1);
```

- **zoomOut():** RadChart keeps a history of zooming actions. This method zooms out the current chart view to the previous scaling step and also restores the scrolled position.
- **resetZoom():** Resets the scaling factors so that no zoom is applied.

Client API Example

This example shows a RadToolBar executing RadChart zoom and scroll methods, all on the client. The Scroll button scrolls to the lower right of the chart. The Zoom button zooms to a factor entered in a RadNumericTextbox. Reset Zoom restores the original view of the chart. The chart is bound to a relatively long series of data that can't be easily viewed in detail at one time.





You can find the complete source for this project at:
 \VS Projects\Chart\ClientSide

The RadToolBar has a single **OnClientButtonClicked** event handler that interprets which button was clicked and executes RadChart client API methods in response.

[JavaScript] Using scroll() and zoom() methods

```
function buttonClicked(sender, args) {
  // get the clicked button value
  var item = args.get_item();
  var toolBarValue = item.get_value();
  // get the chart client reference
  var chart = $find("<%= RadChart1.ClientID %>");
  // get the textbox value
  var zoomValue = $find("<%= tbZoom.ClientID %>").get_value();
  // based on the clicked button, scroll, zoom or reset.
  switch (toolBarValue) {
    case 'scroll':
      {
        chart.scroll(1, 1);
        break
      }
    case 'zoom':
      {
        chart.zoom(zoomValue);
        break
      }
    case 'reset':
      {
        chart.resetZoom();
        break
      }
  }
}
```

40.8 How To

Creating Image Maps and Drill-Down

Image Maps

Image maps are visual areas within the chart that display tool tips. Clicking these areas automatically navigates the user to a URL. Image maps are implemented with the help of the **ActiveRegion** property that contains **URL** and **ToolTip** properties. You can assign the ActiveRegion URL property directly or use the ActiveRegion Click event and respond in server code.

The ActiveRegion resolves to a standard HTML "<map>" tag that defines the area within the chart image that will respond to the mouse:

[HTML] An HTML Fragment from the Rendered Chart

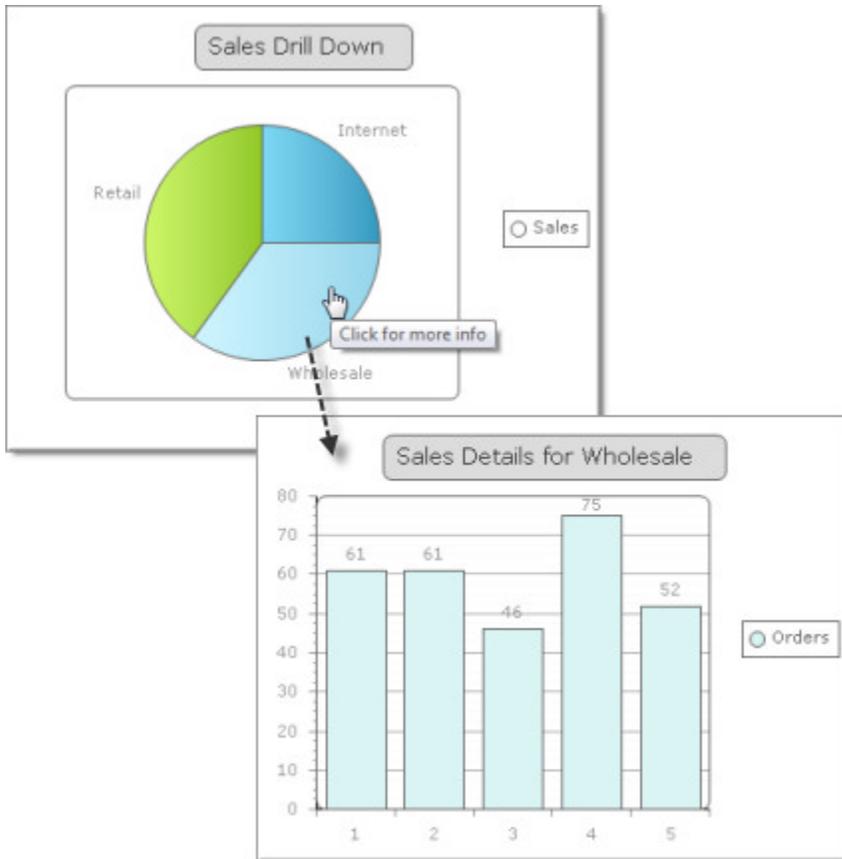
```
<map id='imRadChart1' name='imRadChart1'>
  <area
    shape="poly"
    href="http://www.telerik.com (http://www.telerik.com/)"
    coords="176,168,247,167,248,199,227,226,198,236"
    alt="Sales"
```

```
title="Sales"  
</>  
</map>
```

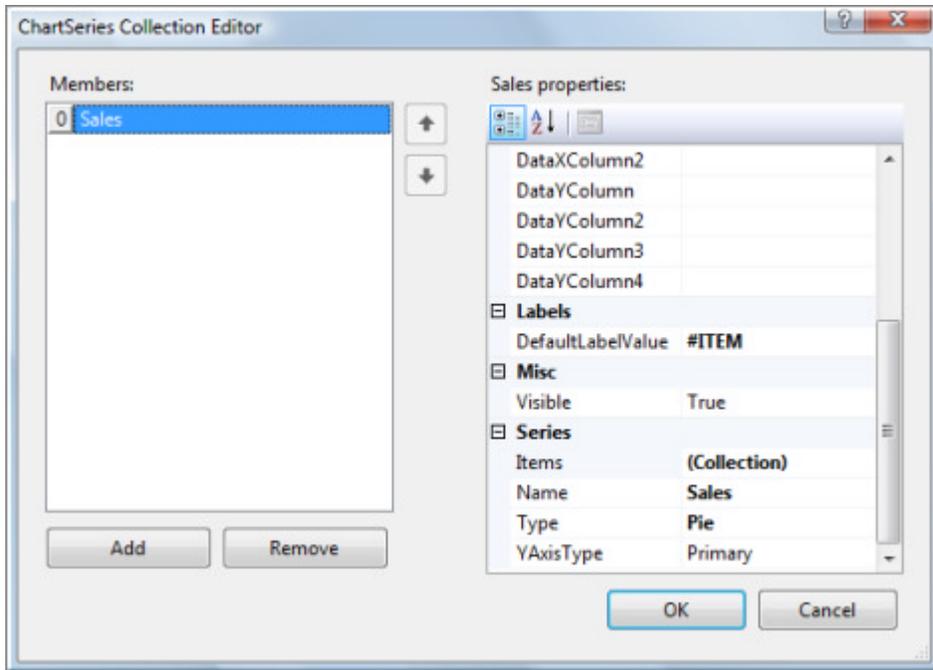
Drill Down Interfaces

Using an image map we can implement a "Drill Down" interface where the user clicks on an element of the chart, e.g. one of the pie slices in a pie chart, and navigates or displays a more detailed view of that slice.

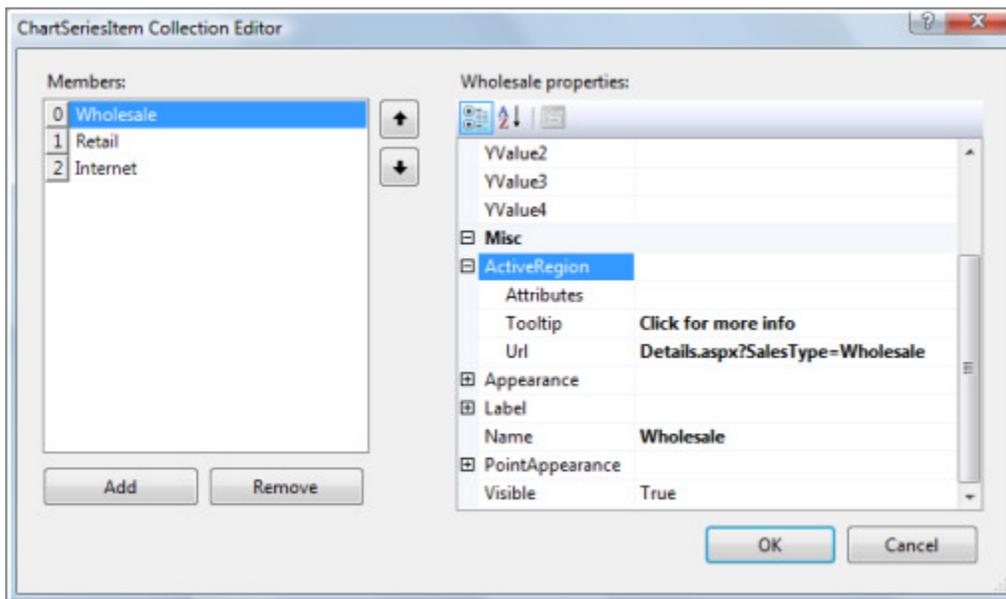
In this example we start with a pie chart of sales categories slices "Retail", "Wholesale" and "Internet". For each slice we set the ActiveRegion property to point to a "Details.aspx" page with a query string that defines the "SalesType".



The default page is setup where the chart Type is "Pie". The **DefaultLabelValue** property is "#ITEM" so that each slice is labeled with its name.



Inside the Items collection for this series are three ChartSeriesItems, each with the ActiveRegion configured in a similar way. The Tooltip reads "Click for more info" and the Url is "Details.aspx?SalesType=Wholesale". The SalesType query string is specific to each chart series item.



The second page in the project (other than default.aspx) is Details.aspx. This page contains the usual ScriptManager and RadChart. In the Page_Load event handler, the query string is received and used to build the chart title. Some dummy items are created, but you can adapt this code to use the passed in query string in a "WHERE" clause that filters a data source.

[VB] Configuring the Details RadChart

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        ' Get the sales type passed from the default page
```

```
' and set the chart title to reflect this
Dim salesType As String = Request.QueryString("SalesType")
RadChart1.ChartTitle.TextBlock.Text = "Sales Details for " + salesType
' Prepare some sample numbers
Dim random As New Random()
' Clear and populate the series with 5 dummy orders
RadChart1.Series.Clear()
Dim series As New ChartSeries("Orders")
RadChart1.Series.Add(series)
Dim i As Integer = 0
While i < 5
    series.AddItem(random.[Next](1, 100))
    System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
End While
End If
End Sub
```

[C#] Configuring the Details RadChart

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // Get the sales type passed from the default page
        // and set the chart title to reflect this
        string salesType = Request.QueryString["SalesType"];
        RadChart1.ChartTitle.TextBlock.Text = "Sales Details for " + salesType;

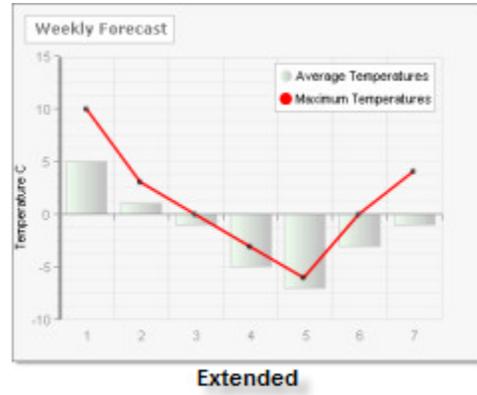
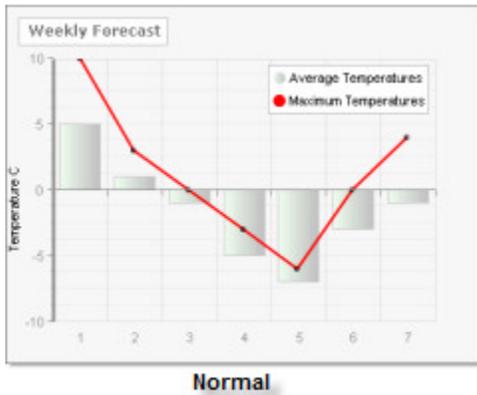
        // Prepare some sample numbers
        Random random = new Random();
        // Clear and populate the series with 5 dummy orders
        RadChart1.Series.Clear();
        ChartSeries series = new ChartSeries("Orders");
        RadChart1.Series.Add(series);
        for (int i = 0; i < 5; i++)
        {
            series.AddItem(random.Next(1, 100));
        }
    }
}
```



You can find the complete source for this project at:
\\VS Projects\chart\DrillDown

Extend the Displayable Area

By default, the data along the YAxis fills the available space. You may want some additional room to make the data easier to see. Use the YAxis **AxisMode** property in the case and set it to "Extended" to get a bit more headroom.



40.9 Summary

In this chapter you built a simple chart with static items and also learned how to bind data to the chart. You took a tour of the basic RadChart elements as well as the types of charts that are available. You learned how to use the tools in the designer to help navigate the many RadChart capabilities. You learned some of the latest RadChart features, including zooming and scrolling. You created and configured many of the chart elements programmatically, including the chart series, items, legend and chart title. You learned how to bind to database data and respond to events on the server side.

41 RadHtmlChart

41.1 Objectives

- Explore the main features of the RadHtmlChart control
- Getting started by running a simple example
- Get familiar with the control's visual and code structure
- See the available types of charts
- Review the basic ways to databind the RadHtmlChart
- See the use of a lightweight callback to load the data

41.2 Introduction

The **RadHtmlChart** was added to the RadControls for ASP.NET AJAX suite in Q2 2012. It provides powerful charting mechanism based on SVG when shown in modern browsers and VML in older browsers. The main features the control boasts are:

- pure client-side rendering through JavaScript which reduces the amount of work the server has to do - only serialized data is sent to the client instead of rendering the entire image and sending markup
- the ability to load its data after the rest of the page has loaded to allow a faster initial load when large amounts of data need to be serialized. This happens with a very light callback (not even an AJAX request)
- a variety of different charts:
 - BarChart
 - ColumnChart
 - LineChart
 - PieChart
 - ScatterChart
 - ScatterLineChart
 - Stacked BarCharts and ColumnCharts
- support for various server datasources
- animation effects when it is being rendered
- intuitive markup structure to make configuration easier

41.3 Getting Started

The following tutorial demonstrates how to add a **RadHtmlChart** to a page:
In a new AJAX-Enabled Web Site drop a **RadHtmlChart** from the ToolBox to the default web page:

1. Add at least one series in the **Series** collection of the **PlotArea** inner tag. For this example these can be **ColumnSeries**
2. Add items to the **Items** collection of the **Series**. They only need their **YValue** property set so that they can be placed according to the Y-axis
3. Add items to the **Items** collection of the **XAxis** tag that is also a child of the **PlotArea** tag. Their number must match the number of items declared for the series as they will be shown on the X-axis below each column
4. Optionally you may also set other properties for the control by using the inner tags most of its elements

provide:

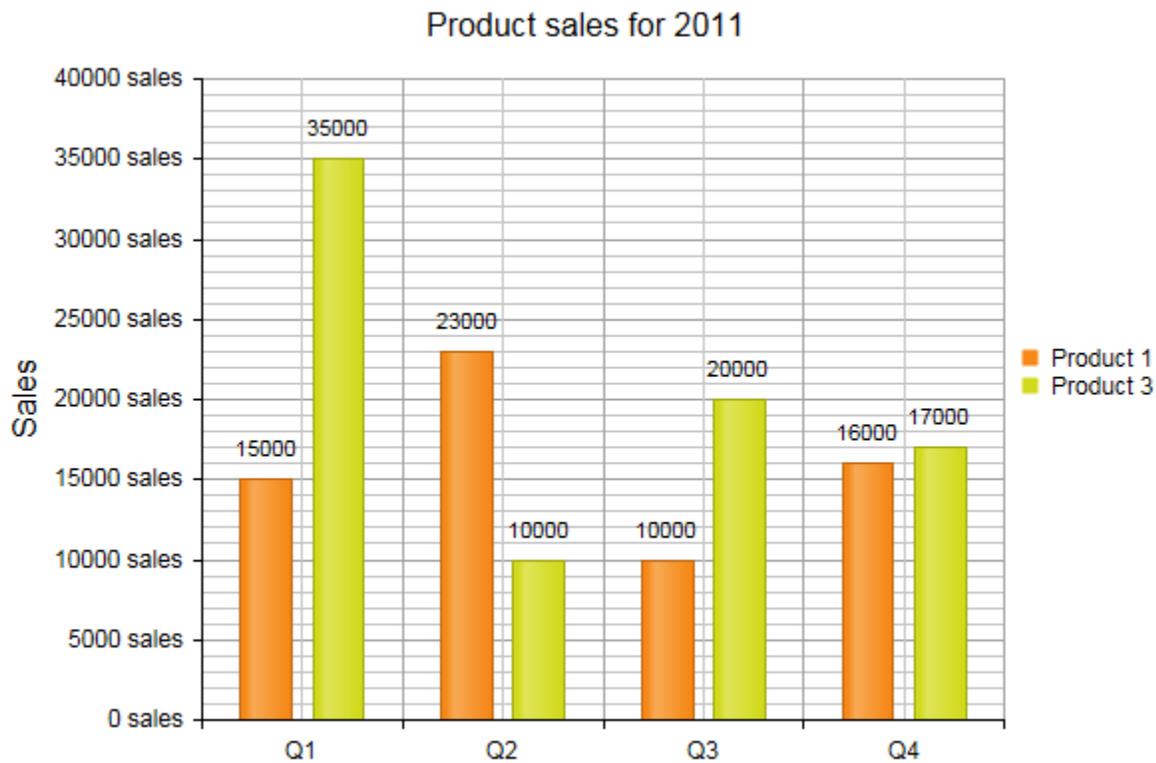
1. Set the **Title** property of the entire chart and/or for each axis.
 2. Customize the **tooltips** and/or **labels** for the series via the **TooltipsAppearance** and **LabelsAppearance** inner tags.
 3. Customize the y-axis by changing the minimum and maximum values, grid lines, and/or labels.
5. Press F5 to run the page. You will see the **RadHtmlChart**.

Below is some example markup that can be created with the steps above:

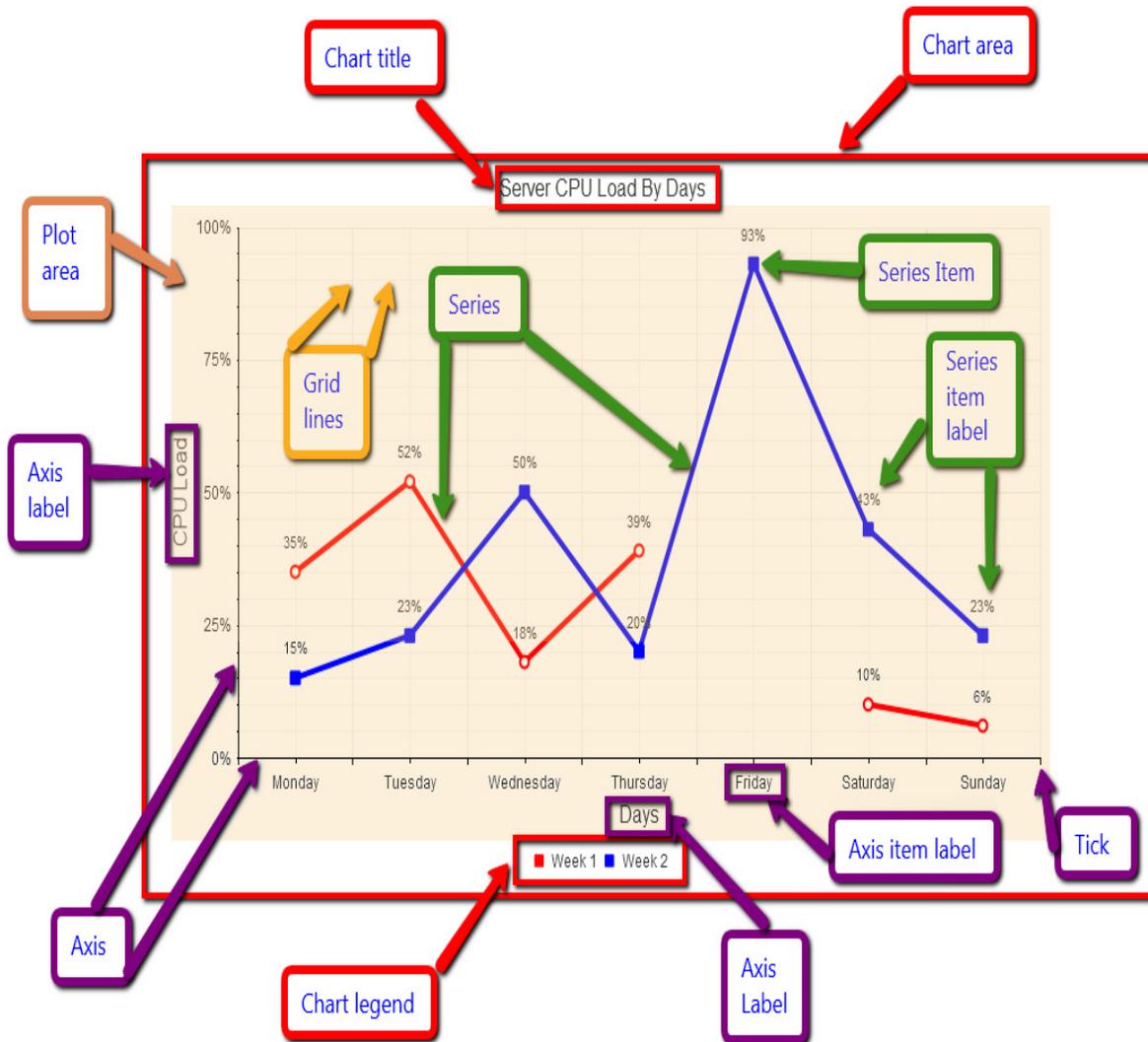
ASPX

```
<telerik:RadHtmlChart runat="server" ID="ColumnChart1" Width="600px" Height="400px">
  <PlotArea>
    <Series>
      <telerik:ColumnSeries Name="Product 1">
        <Items>
          <telerik:SeriesItem YValue="15000" />
          <telerik:SeriesItem YValue="23000" />
          <telerik:SeriesItem YValue="10000" />
          <telerik:SeriesItem YValue="16000" />
        </Items>
        <LabelsAppearance Position="OutsideEnd" />
        <TooltipsAppearance Visible="false" />
      </telerik:ColumnSeries>
      <telerik:ColumnSeries Name="Product 3">
        <Items>
          <telerik:SeriesItem YValue="35000" />
          <telerik:SeriesItem YValue="10000" />
          <telerik:SeriesItem YValue="20000" />
          <telerik:SeriesItem YValue="17000" />
        </Items>
        <LabelsAppearance Position="OutsideEnd" />
        <TooltipsAppearance Visible="false" />
      </telerik:ColumnSeries>
    </Series>
    <XAxis>
      <Items>
        <telerik:AxisItem LabelText="1" />
        <telerik:AxisItem LabelText="2" />
        <telerik:AxisItem LabelText="3" />
        <telerik:AxisItem LabelText="4" />
      </Items>
      <LabelsAppearance DataFormatString="Q{0}" RotationAngle="0" />
    </XAxis>
    <YAxis>
      <LabelsAppearance DataFormatString="{0} sales" RotationAngle="0" />
      <TitleAppearance Position="Center" RotationAngle="0" Text="Sales" />
    </YAxis>
  </PlotArea>
  <ChartTitle Text="Product sales for 2011">
  </ChartTitle>
</telerik:RadHtmlChart>
```

This will result in the following chart:



The **RadHtmlChart** has a complex structure that consists of many elements that are outlined in the image below:



The main parts of the chart that can be controlled outside of the specific series are:

- **Chart area** - the main wrapper of the chart. This is the background on which everything else is placed, including the PlotArea with the series and axes, chart title and legend. It is controlled via the Appearance inner tag from the main tag of the control. Currently the background color of the entire chart can be set there.
- **Chart title** - this is the global title of the chart. It is configured by the ChartTitle inner tag. There the string that will be shown is set and it also provides the Appearance inner tag where the position (bottom or top), alignment (left, right or centered), background color and visibility can be set.
- **Legend** - this is the list with series names or item names in the case of a PieChart along with a symbol that indicates their color in the actual chart. Its appearance can be customized via the Appearance tag inside the Legend tag that is a direct child of the main tag. The available properties control background color, position (bottom, left, right or top) and visibility.
- **Plot Area** - this is the part where the actual chart is rendered. It includes the series with their labels, the axes along with their labels and titles. PlotArea is also the name of the inner tag of the main chart tag where the axes and series are defined.

The series are added to the Series tag inside the PlotArea tag. Their inner tags contain further properties that can be used to control their appearance and databinding. Regardless of their configuration all series have the

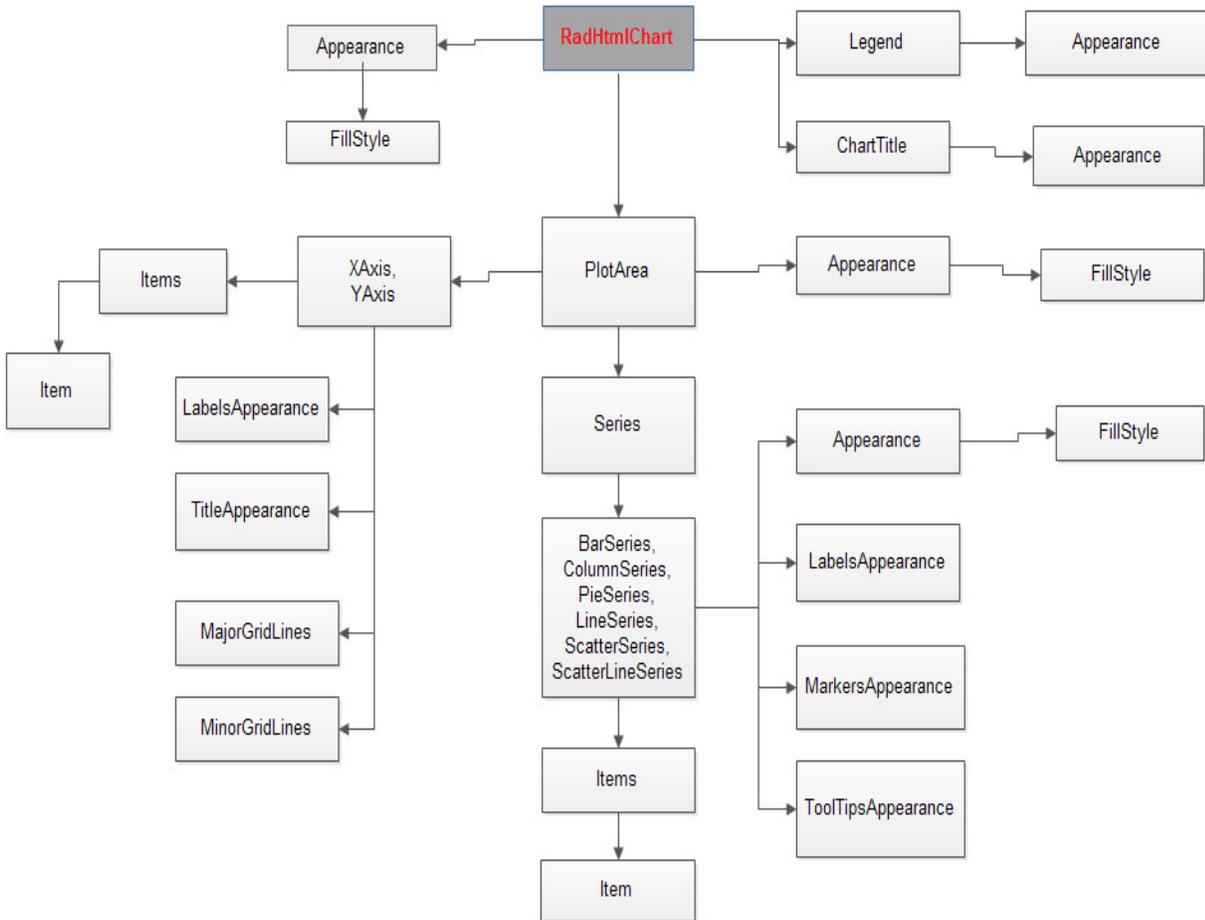
same common set of elements:

- **Series itself** - this is the shape that is defined by the type of the series. In the above image this is a line that connects the points defined as items for the series. For bar and column charts the series consists of several rectangles that correspond to the series items, for the scatter chart it is the points themselves, etc.
- **Series item** - this is the unit of data that is passed to the series. It defines the value of the chart at the given point/for the given x-axis item. For line type charts these points define the spots through which the line passes.
- **Series item label** - this is the text next to each item that shows the value it holds. It can be modified with a format string to show a pattern related to this value.
- **Series item tooltip** - this is a tooltip that is shown only when the mouse hovers over a series item, which is why it is not present in the above diagram. It consists of a rectangle with the series' color (or the color predefined by the developer) and the item's value (plus format string) inside.

The axes are two perpendicular lines that define the scale of the chart and also show the reference values/items. They can be translated to form grid lines inside the chart to aid the visual estimation of the series' values. The axes are direct children of the plot area and this is also the place where they are defined in the markup of the control via their own inner tags.

- **Axis** - the actual axis of the chart - it is a single line whose color and width can be changed if the default values do not match the needs of the developer.
- **Ticks** - small marks on the axis that define axis values (or items) and are also starting points for the grid lines.
- **Grid lines** - lines that are parallel to the axes to aid readability of the values. There are two types of grid lines - major (usually thicker and spaced further off from each other) and minor (usually thinner and with lighter color and closer together).
- **Axis item label** - text that corresponds to each item on an x-axis that requires items. It shows a string defined in the code. In the case of a numerical axis the values either calculated by the chart or set by the developer and can take a format string to show a template.
- **Axis label** - this is the title of the entire axis. It is usually used to show what the axis corresponds to or the unit of measurement.

Each of them corresponds to a certain tag/class in the control's code so that they can easily be found and configured:



41.4 Chart Types

The **RadHtmlChart** offers a number of chart types to fit different scenarios and data. The type of each series is controlled via its tag name: **BarSeries**, **ColumnSeries**, **LineSeries**, **PieSeries**, **ScatterSeries** and **ScatterLineSeries**.

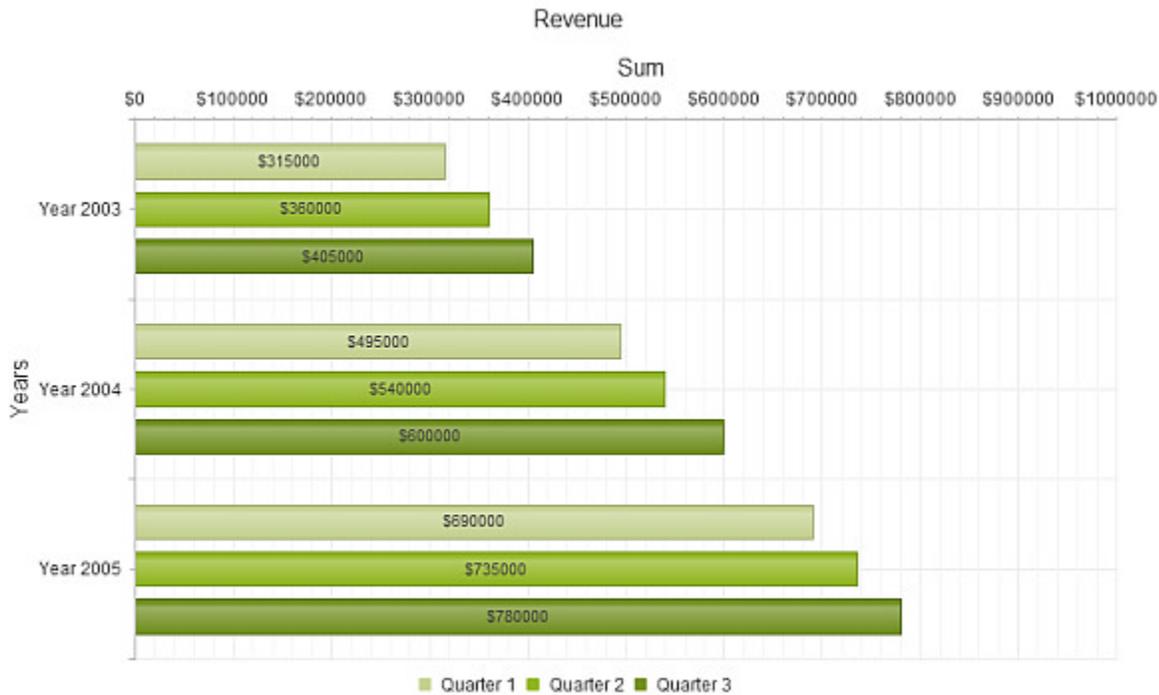
The **BarSeries**, **ColumnSeries** and **LineSeries** have a numerical Y-axis where the values of their items (their **YValue** property) are distributed while each bar column or line point lies above a given item from the X-axis (the **Items** collection of the **XAxis**).

The **ScatterSeries** and **ScatterLineSeries** have numerical X-axes as well and thus their items have one more property - **XValue** to determine their position according to this axis as well and thus they do not require items for the x-axis.

Series with the same type of axes can be combined in the same chart. The **PieSeries** is an exception, because it does not have axes at all and only one can be present in the chart. In this sense it shows the names of its items in the legend instead of the names of the series.

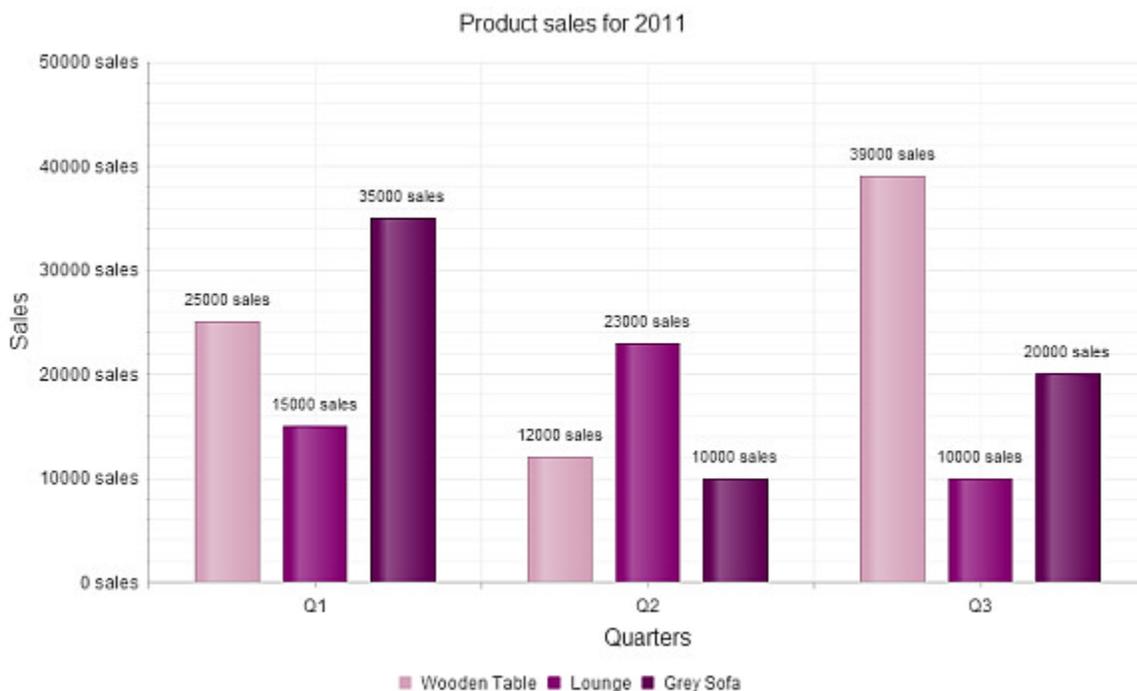
Below follow examples that show how each type of charts looks like:

BarSeries



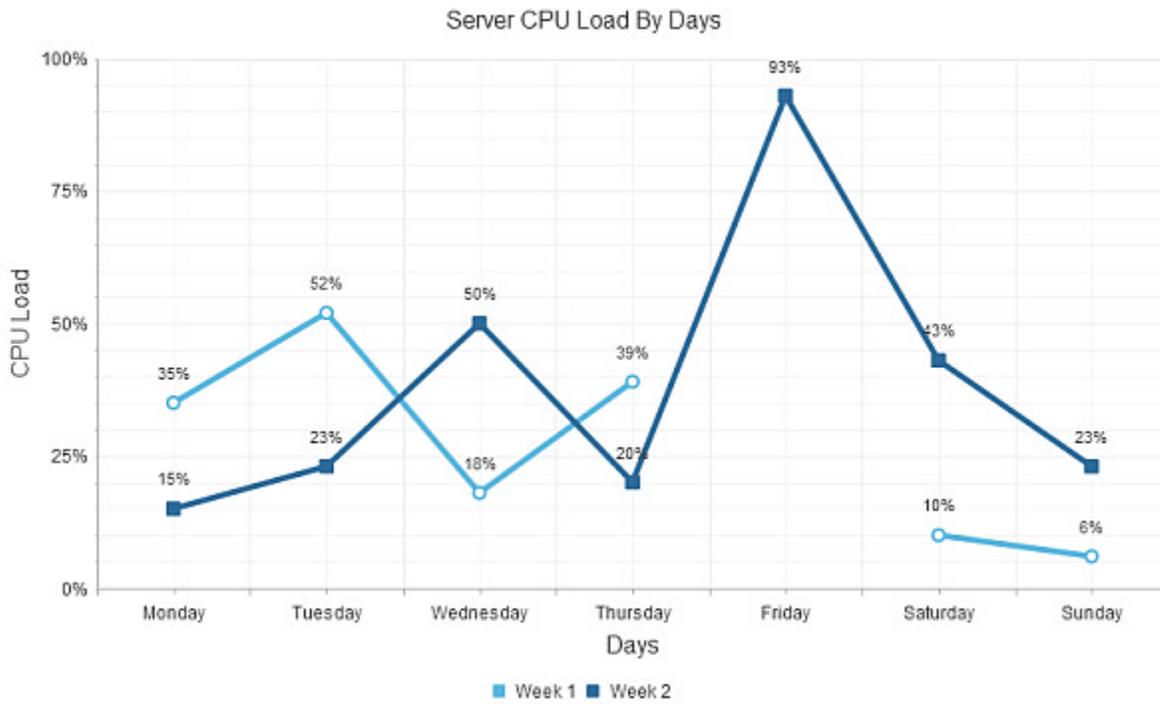
The Y-axis is rotated 90 degrees clockwise and is horizontal, yet this is where the YValues are located. This is done so that the series have a common way of setting their values.

ColumnSeries



The main difference between the Column and BarSeries is that the latter is horizontal while the former is vertical.

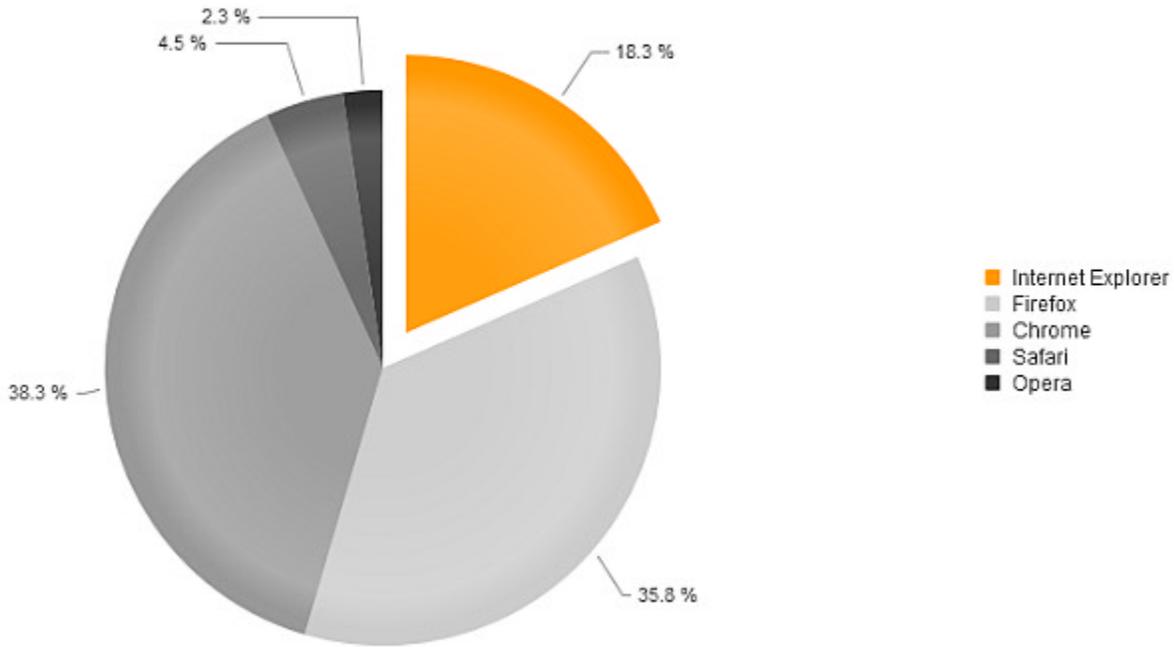
LineSeries



This is a line connecting the items declared for the series. In case a value is missing the RadHtmlChart can interpolate it or leave it blank.

PieSeries

Browser Usage for April 2012



A pie can have a number of sectors with a specified color and some of them can be separated from the rest to emphasize their importance. This is done via the item's Exploded property.

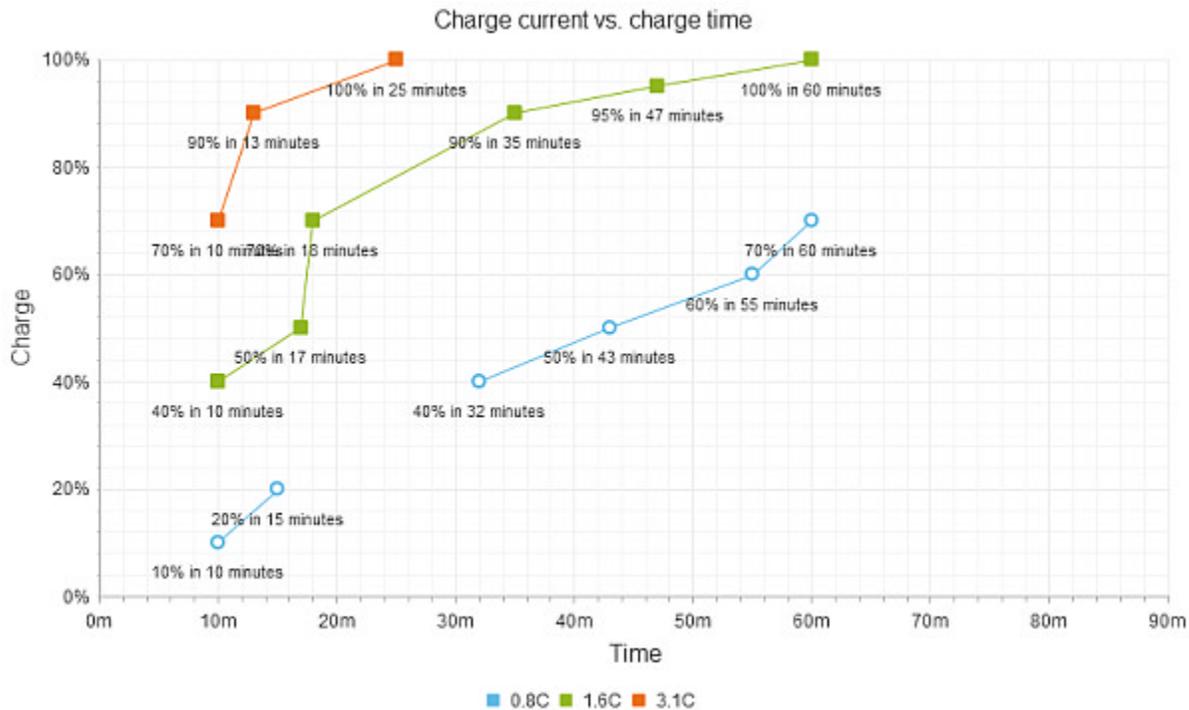
ScatterSeries

Performance / Price for CPUs



This series type is just a set of points in the plane and is useful for showing experimental data.

ScatterLineSeries



This is very similar to the LineSeries in the sense that it shows a trend over time by connecting the items with lines, but the main difference is that the X-axis is numerical.

41.5 Databinding

The **RadHtmlChart** can be bound to various server datasources, regardless of its client-side rendering, which makes it suitable for the regular scenarios while keeping the performance benefit of the client-side rendering.

To make things even better the data is only serialized to be sent to the client to take as little volume as possible. To make things better the data itself can even be loaded through a callback after the page has loaded or on demand when the developer needs it. This is controlled via the **InvokeLoadData** property of the chart. In case it is configured to **FromCode** a call to the JavaScript **loadData()** method will quickly get the serialized datasource from the server.

Other than that setting a datasource is quite easy - an **SqlDataSource**, an **EntityDataSource**, a **LinqDataSource**, **XmlDataSource**, or even simple **DataTables**, **arrays** or **lists** can be used. You only need to configure the datasource to return the needed data and feed it to the **RadHtmlChart**'s **DataSource** (for programmatic data, then call **DataBind()**) or **DataSourceID** (for declarative sources) property like with any other databound control. The essential properties needed to pass the data to the series and axes are:

- **DataField** - set for the series to point it to the desired column of the datatable
- **DataLabelsField** - set for the x-axis labels to populate the items for the axis (it can also be applied to the y-axis)

Here follows a simple example:

ASPX

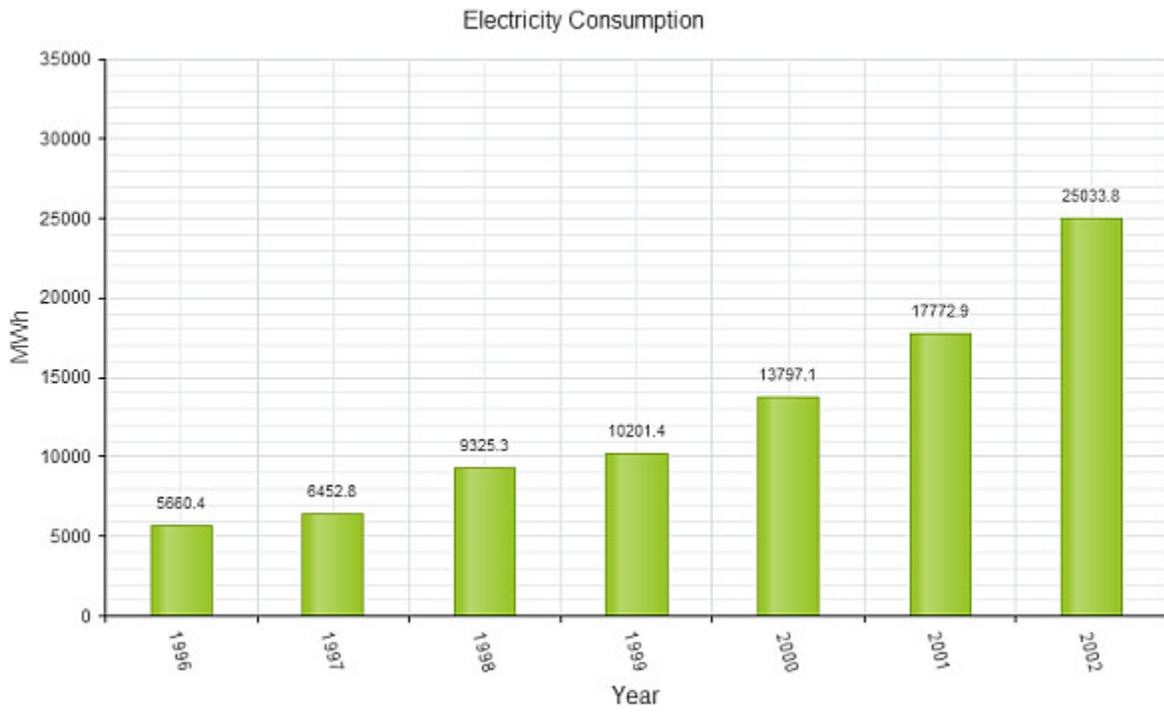
```
<telerik:RadHtmlChart runat="server" Width="800px" Height="500px" ID="RadHtmlChart1"
DataSourceID="SqlDataSource1">
  <PlotArea>
    <Series>
      <telerik:ColumnSeries DataField="Value" Name="Electricity Consumption">
      </telerik:ColumnSeries>
    </Series>
    <XAxis DataLabelsField="Year">
      <LabelsAppearance RotationAngle="75" />
      <TitleAppearance Text="Year" />
    </XAxis>
    <YAxis>
      <TitleAppearance Text="MWh" />
    </YAxis>
  </PlotArea>
  <Legend>
    <Appearance Visible="false" />
  </Legend>
  <ChartTitle Text="Electricity Consumption">
  </ChartTitle>
</telerik:RadHtmlChart>
```

And the datasource itself, which, of course, may need some tweaking to match your own database:

ASPX

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server" ConnectionString="<%=
ConnectionStrings:TelerikConnectionString %>"
SelectCommand="SELECT [Year], [Value] FROM [Data] WHERE ([Subcategory_Id] = 1)">
</asp:SqlDataSource>
```

That yields the following chart:



42 ActiveSkill: Building the Exam Finish Control

42.1 Objectives

- Build the TakeExamFinish.ascx User Control.
- Deserialize a JSON string into a server-side object.
- Configure a RadChart control and bind it to a generic List object.

42.2 Building the Exam Finish Page

In this chapter we will add the full functionality for the TakeExamFinish.ascx control. The "page" will work off of the ExamResults serialized and sent from the client, deserialize this object and work with it in server code. The "page" will display exam results and also bind results-by-category data to a RadChart.



You can find the complete source for this project at:
\\VS Projects\ActiveSkill Add Finish Page

Add DataSource

Add a SqlDataSource to the control with ID "dsCategory", use the ActiveSkillConnectionString for the **ConnectionString** property, set **SelectCommand** to "Skill_Category_SelectWhere" and **SelectCommandType** to "StoredProcedure". In the **SelectParameters** collection add a single parameter with Name "ID" and Type "Int32". The markup should look like the example below.

[ASP.NET] Adding the DataSource

```
<!--Data sources-->
<asp:SqlDataSource ID="dsCategory" runat="server"
  ConnectionString="<%= $ConnectionStrings.ActiveSkillConnectionString %>"
  SelectCommand="Skill_Category_SelectWhere"
  SelectCommandType="StoredProcedure">
  <SelectParameters>
    <asp:Parameter Name="ID" Type="Int32" />
  </SelectParameters>
</asp:SqlDataSource>
```

Add Exam Summary Markup

1. Add a span with two images that show a happy/sad face based on the exam results. The style for the span tag positions the images on the page.

[ASP.NET] Adding the Pass/Fail Images

```
<span id="divImg" runat="server" style="position: absolute;
  left: 200px; top: 100px">
  
  
</span>
```

2. Add two more span tags that contain a pass/fail message and an appropriately labeled button to return back to the TakeExamChoose.ascx control. *The onclick event handlers use the DynamicControl JavaScript object load() method to navigate.*

[ASP.NET] Add Results and Buttons

```
<span id="divPass" runat="server" style="position: absolute;
left: 300px; top: 110px; line-height: 30px">
  <div class="skillHighlight" style="">
    YOU PASSED!</div>
  
    style="position: relative; left: -20px" />
</span>
<span id="divfailed" runat="server" style="position: absolute;
left: 300px; top: 110px; line-height: 30px">
  <div class="skillRed">
    YOU FAILED!</div>
  
    style="position: relative; left: -20px" />
</span>
```

3. Add a third span tag that contains labels that contain some literal text and the title and score information. *These labels will be populated in the TakeExamFinish.ascx FirstLoad() method.*

[ASP.NET] Add the Exam Summary Information

```
<span id="divSummary" runat="server" style="position: absolute;
left: 500px; top: 120px; line-height: 20px">

  <div id="divScore" runat="Server" class="skillHighlight">
    Your results for exam
    <asp:Label ID="lblTitle" runat="server" CssClass="skillGreen"></asp:Label>
  </div>

  <span id="divScore2" runat="Server">
    <asp:Label ID="lblSummary" runat="server" CssClass="skillSummary"
      Text="You scored "></asp:Label>
    <asp:Label ID="lblScore" runat="server" CssClass="skillGreen"></asp:Label>
    <asp:Label ID="lblSummary2" runat="server" CssClass="skillSummary"
      Text=" with "></asp:Label>
    <asp:Label ID="lblTotal" runat="server" CssClass="skillGreen"></asp:Label>
    <asp:Label ID="Label1" runat="server" CssClass="skillSummary"
      Text=" required to pass"></asp:Label>
  </span>
</span>
```

4. Add a div to contain a RadChart to be added later.

[ASP.NET]

```
<div id="chartDiv" style="position: absolute; left: 200px; top: 190px">

  <!-- RadChart goes here-->

</div>
```

Add and Configure RadChart

1. Add a RadChart to the TakeExamFinish.ascx control.
2. In the RadChart Smart Tag, click the **Add RadChart HTTP Handler to Web.config** link.



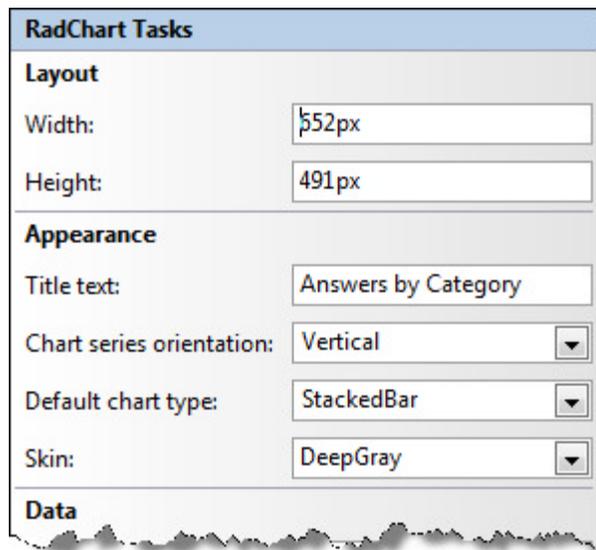
Gotcha!

At the time of this writing, if you are using IIS7 Integrated Mode, you need to add a handler manually to the <system.webserver> "handlers" element of the web.config file:

```
<system.webServer>
  <handlers>
    <add name="ChartHandler" path="ChartImage.axd" verb="*"
    type="Telerik.Web.UI.ChartHttpHandler, Telerik.Web.UI" />
  ...
```

3. Also in the Smart Tag, enter these settings:

- **Width:** 652
- **Height:** 491
- **Title Text:** Answers by category
- **Chart Series Orientation:** Vertical
- **Default chart type:** Stacked Bar
- **Skin:** DeepGray



4. In the Properties Window set the **IntelligentLabelsEnabled** property to "true".
5. In the **Series** collection editor, set the name of the first series to "Correct Answers" and the second series to "Incorrect Answers".

The completed markup should look something like this example:

[ASP.NET] RadChart Definition

```
<telerik:RadChart ID="RadChart1" runat="server" DefaultType="StackedBar" Height="491px"
Skin="DeepGray" Width="652px" IntelligentLabelsEnabled="true">
  <Series>
    <telerik:ChartSeries Name="Correct Answers" Type="StackedBar" DataLabelsColumn>
    </telerik:ChartSeries>
    <telerik:ChartSeries Name="Incorrect Answers" Type="StackedBar">
    </telerik:ChartSeries>
  </Series>
  <PlotArea>
    <XAxis></XAxis>
    <YAxis AxisMode="Extended"></YAxis>
  </PlotArea>
</telerik:RadChart>
```

6. Check the ActiveSkillUI references and verify that Telerik.Charting is in the list. If not, add it now.

Add ExamResults Server-Side Object

We can move objects back and forth between server and client. The MS AJAX Library includes functions to serialize objects, e.g.

```
System.Web.Script.Serialization.JavaScriptSerializer.serialize(myJsonObject);
```

...and we can also serialize and deserialize objects on the server using the `JavaScriptSerializer` object from the `System.Web.Script.Serialization` namespace. We will need a server side version of the `Category` and `ExamResults` objects to deserialize into when we receive the results argument in the `TakeExamFinish.ascx` `FirstLoad()` method.

1. Add a new class file "ExamResults.cs" to the ActiveSkillBO project.
2. Add the `Category` and `ExamResults` code to the file.

[VB] Defining the Server-Side Category and ExamResults Objects

```
Imports System.Collections.Generic
Namespace Telerik.ActiveSkill.Common
  #region Category
  ' Category stores a tally of total and incorrect responses
  ' for a single category.
  Public Class Category
    Private _categoryID As Integer
    Private _total As Integer
    Private _incorrect As Integer
    Private _title As String
    Public Property CategoryID() As Integer
    Get
      Return _categoryID
    End Get
    Set
      _categoryID = value
    End Set
  End Property
  Public Property Title() As String
  Get
    Return _title
  End Get
  Set
    _title = value
  End Set
End Class
```

```
End Set
End Property
Public Property Total() As Integer
Get
    Return _total
End Get
Set
    _total = value
End Set
End Property
Public Property Incorrect() As Integer
Get
    Return _incorrect
End Get
Set
    _incorrect = value
End Set
End Property
Public ReadOnly Property Correct() As Integer
Get
    Return _total - _incorrect
End Get
End Property
Public ReadOnly Property Score() As Double
Get
    Dim total As Double = Me.Total
    Dim incorrect As Double = Me.Incorrect
    Return ((total - incorrect) / total) * 100
End Get
End Property
End Class
#End Region Category
#region ExamResults
' ExamResults summarizes the scores for all categories
Public Class ExamResults
Private _categories As New List(Of Category)()
Public Property Categories() As List(Of Category)
Get
    Return _categories
End Get
Set
    _categories = value
End Set
End Property
' Returns the total of all questions for all categories.
Public ReadOnly Property Total() As Integer
Get
    Dim result As Integer = 0
    For Each category As Category In _categories
        result += category.Total
    Next
    Return result
End Get
End Property
' Returns the total of incorrect questions for all categories
```

```

Public ReadOnly Property Incorrect() As Integer
    Get
        Dim result As Integer = 0
        For Each category As Category In _categories
            result += category.Incorrect
        Next
        Return result
    End Get
End Property
' Returns the total score accross all categories
Public ReadOnly Property Score() As Double
    Get
        Dim total As Double = Me.Total
        Dim incorrect As Double = Me.Incorrect
        Return ((total - incorrect) / total) * 100
    End Get
End Property
End Class
#End Region ExamResults
End Namespace

```

[C#] Defining the Server-Side Category and ExamResults Objects

```

using System.Collections.Generic;
namespace Telerik.ActiveSkill.Common
{
    #region Category
    // Category stores a tally of total and incorrect responses
    // for a single category.
    public class Category
    {
        private int _categoryID;
        private int _total;
        private int _incorrect;
        private string _title;
        public int CategoryID
        {
            get { return _categoryID; }
            set { _categoryID = value; }
        }
        public string Title
        {
            get { return _title; }
            set { _title = value; }
        }
        public int Total
        {
            get { return _total; }
            set { _total = value; }
        }
        public int Incorrect
        {
            get { return _incorrect; }
            set { _incorrect = value; }
        }
    }
}

```

```

public int Correct
{
    get { return _total - _incorrect; }
}
public double Score
{
    get
    {
        double total = this.Total;
        double incorrect = this.Incorrect;
        return ((total - incorrect) / total) * 100;
    }
}
}
#endregion Category
#region ExamResults

// ExamResults summarizes the scores for all categories
public class ExamResults
{
    private List<Category> _categories = new List<Category>();
    public List<Category> Categories
    {
        get { return _categories; }
        set { _categories = value; }
    }
    // Returns the total of all questions for all categories.
    public int Total
    {
        get
        {
            int result = 0;
            foreach (Category category in _categories)
            {
                result += category.Total;
            }
            return result;
        }
    }
    // Returns the total of incorrect questions for all categories
    public int Incorrect
    {
        get
        {
            int result = 0;
            foreach (Category category in _categories)
            {
                result += category.Incorrect;
            }
            return result;
        }
    }
    // Returns the total score accross all categories
    public double Score
    {

```

```

    get
    {
        double total = this.Total;
        double incorrect = this.Incorrect;
        return ((total - incorrect) / total) * 100;
    }
}
}
#endregion ExamResults
}

```

Implement the FirstLoad() IASControl Method

The FirstLoad() method of the TakeExamFinish.ascx control gets the collection of scores by category and summarizes them.

1. The TakeExamFinish.ascx page code-behind should have the following references to the "Imports" (VB) or "uses" (C#) clauses:
 - o System
 - o System.Collections.Generic
 - o System.Data
 - o System.Web.Script.Serialization
 - o System.Web.UI
 - o Telerik.ActiveSkill.Common
2. In the FirstLoad() method add code to store the the incoming arguments. *The passPercent and title arguments are simple assignments to the "passPercent" numeric variable and the "title" string variable. The examResults is a serialized JSON string that must be Deserialized into object form before we can use it. The JavaScriptSerializer Deserialize() method takes care of this by converting the JSON string into a server-side ExamResults object.*

[VB] Storing Incoming Arguments

```

Public Sub FirstLoad(ByVal args As Dictionary(Of String, String))
    ' Store args information
    Dim passPercent As Double = Convert.ToDouble(args("passPercent"))
    Dim title As String = args("title")
    ' Retrieve the serialized JSON ExamResults client object and
    ' Deserialize into the server ExamResults object.
    Dim jss As New JavaScriptSerializer()
    Dim examResults As ExamResults = jss.Deserialize(Of ExamResults)(args("examResults"))
    ' . . .
End Sub

```

[C#] Storing Incoming Arguments

```

public void FirstLoad(Dictionary<string, string> args)
{
    // Store args information
    double passPercent = Convert.ToDouble(args["passPercent"]);
    string title = args["title"];
    // Retrieve the serialized JSON ExamResults client object and
    // Deserialize into the server ExamResults object.
    JavaScriptSerializer jss = new JavaScriptSerializer();
    ExamResults examResults = jss.Deserialize<ExamResults>(args["examResults"]);
}

```

```
// . . .  
}
```

3. Below the arguments assignment, add code to determine if the user passed or failed by comparing the ExamResults Score property against the "passPercent" variable. The code assigns the properties for the title, score and total. Finally, the boolean "pass" is used to show or hide parts of the UI.

[VB] Calculate the Pass/Fail Status and Update the UI

```
' Calculate if the user passed the exam.  
Dim pass As Boolean = examResults.Score >= passPercent  
' Set the page element properties: set the title,  
' Score and percentage needed to pass.  
lblTitle.Text = title  
lblScore.Text = [String].Format("{0:0}%", examResults.Score)  
lblTotal.Text = [String].Format("{0:0}%", passPercent)  
' Hide and show areas of the page according to the  
' pass/fail status  
divPass.Visible = pass  
divfailed.Visible = Not pass  
imgPass.Visible = pass  
imgFail.Visible = Not pass
```

[C#] Calculate the Pass/Fail Status and Update the UI

```
// Calculate if the user passed the exam.  
bool pass = examResults.Score >= passPercent;  
// Set the page element properties: set the title,  
// Score and percentage needed to pass.  
lblTitle.Text = title;  
lblScore.Text = String.Format("{0:0}%", examResults.Score);  
lblTotal.Text = String.Format("{0:0}%", passPercent);  
// Hide and show areas of the page according to the  
// pass/fail status  
divPass.Visible = pass;  
divfailed.Visible = !pass;  
imgPass.Visible = pass;  
imgFail.Visible = !pass;
```

4. Next add code to retrieve the title for each category and store it in our ExamResults array of categories.

[VB] Load Category Titles

```
' Retrieve the category title for each category and add it  
' to each server Category object.  
For Each category As Category In examResults.Categories  
    dsCategory.SelectParameters("ID").DefaultValue = category.CategoryID.ToString()  
    Dim dvCategory As DataView = DirectCast(dsCategory.[Select]  
(DataSourceSelectArguments.Empty), DataView)  
    For Each row As DataRow In dvCategory.Table.Rows  
        category.Title = row("Title").ToString()  
    Next  
Next
```

[C#] Load Category Titles

```
// Retrieve the category title for each category and add it  
// to each server Category object.  
foreach (Category category in examResults.Categories)  
{  
    dsCategory.SelectParameters["ID"].DefaultValue = category.CategoryID.ToString();  
}
```

```

DataView dvCategory = (DataView)dsCategory.Select(DataSourceSelectArguments.Empty);
foreach (DataRow row in dvCategory.Table.Rows)
{
    category.Title = row["Title"].ToString();
}
}
}

```

5. Finally, bind the chart to the generic list of Categories in the ExamResults object. Set the first series of data to the "Correct" column and the second series to the "Incorrect" column. Set the labels to use the category titles.

[VB] Binding the Chart

```

' Bind the RadChart to our server ExamResults.Categories array.
' Set the first series to use the Correct property and the
' second series to use the Incorrect property of the Category object.
RadChart1.DataSource = examResults.Categories
RadChart1.Series(0).DataYColumn = "Correct"
RadChart1.Series(0).DefaultLabelValue = "Correct:#Y"
RadChart1.Series(1).DataYColumn = "Incorrect"
RadChart1.Series(1).DefaultLabelValue = "Incorrect:#Y"
RadChart1.PlotArea.XAxis.DataLabelsColumn = "Title"
RadChart1.DataBind()

```

[C#] Binding the Chart

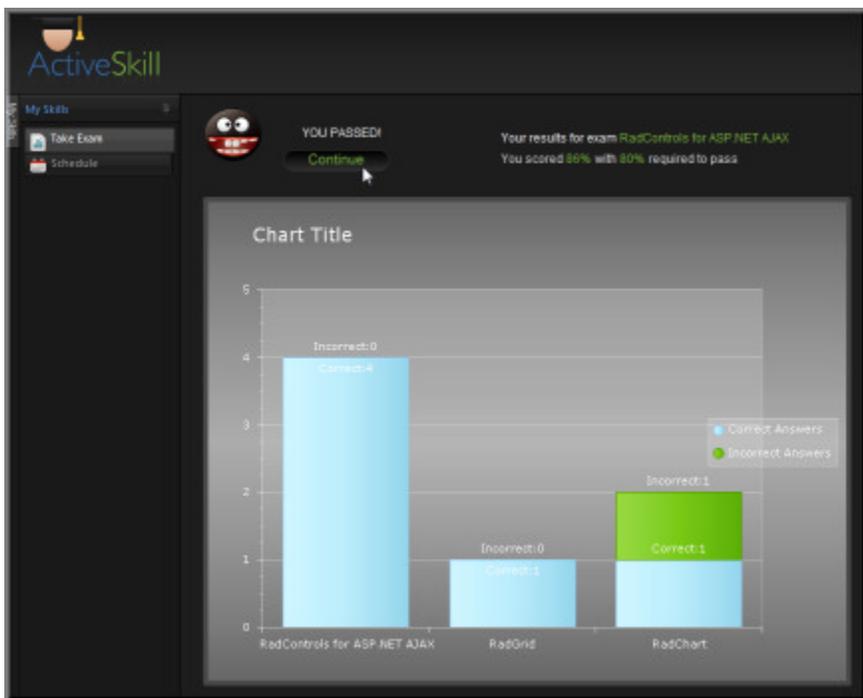
```

// Bind the RadChart to our server ExamResults.Categories array.
// Set the first series to use the Correct property and the
// second series to use the Incorrect property of the Category object.
RadChart1.DataSource = examResults.Categories;
RadChart1.Series[0].DataYColumn = "Correct";
RadChart1.Series[0].DefaultLabelValue = "Correct:#Y";
RadChart1.Series[1].DataYColumn = "Incorrect";
RadChart1.Series[1].DefaultLabelValue = "Incorrect:#Y";
RadChart1.PlotArea.XAxis.DataLabelsColumn = "Title";
RadChart1.DataBind();

```

Test Exam

Press Ctl-F5 to run the application. Take a short exam multiple times to verify the output on the final page when the score is passing and failing. Check that the "Continue" or "Try Again" button navigates back to the TakeExamChoose.ascx control.



42.3 Summary

In this chapter you implemented the TakeExamFinish.ascx control. You deserialized a JSON string passed from the client into a server ExamResults object. You added HTML controls to display exam results and also added and configured a RadChart. You bound the RadChart to a generic List of objects and displayed the data in a stacked bar format with two series of data.

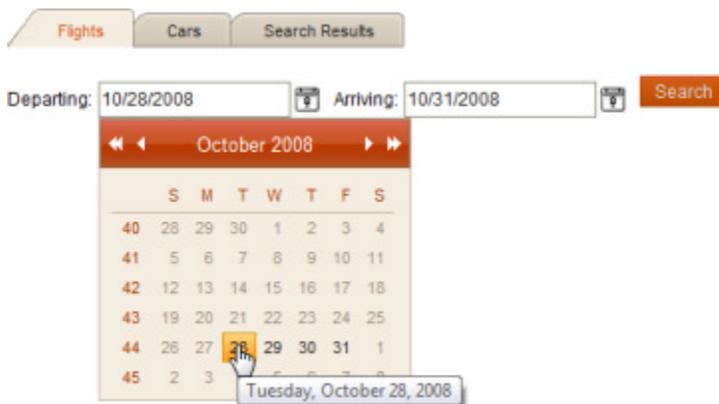
43 Date, Time, Calendar and Scheduling

43.1 Objectives

- Explore the features of the date, time, calendar and scheduler controls.
- Create simple applications to get familiar with the basic controls.
- Explore the design time interfaces, including the Smart Tag, Properties Window and Template Design surface.
- Explore principal properties and groups of properties where most of the functionality is found.
- Learn server-side coding techniques including the major server side objects, setting calendar special days, adding scheduler appointments, adding scheduler resources, scheduling recurrence and handling server-side events.
- Explore some of the client-side methods of the date, time, calendar client-side objects. Includes drilling down to the objects that make up the picker controls, controlling popups, and selected dates.
- Learn how to validate date and time entry.
- Learn to use scheduler templates.

43.2 Date-Time and Calendar Controls Getting Started

This walk-through will use the RadDatePicker, RadDateTimePicker and the RadCalendar.



1. Open the web.config file and add the following application setting to set the Skin for all RadControls in the application to "Sunset".

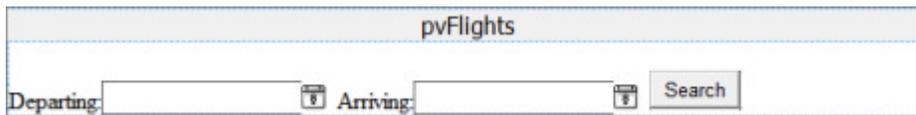
[ASP.NET] Add the Telerik Skin Setting

```
<appSettings>
  <!-- Sets the skin for all RadControls -->
  <add key="Telerik.Skin" value="Sunset"/>
</appSettings>
```

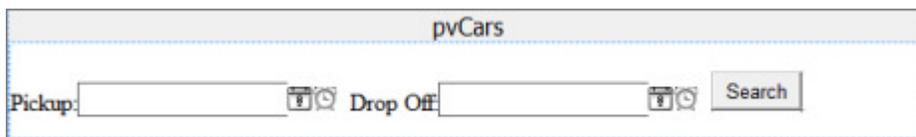
⚠ Be sure to *not* set the **Skin** properties for any of the controls. That includes making sure that tags like `Skin=""` and `Skin="Default"` do not exist in the markup. If the "Sunset" skin does not appear for one of the controls, check the markup for any instances of `"Skin=xxx"` and remove them.

2. In a new web application add a **RadFormDecorator** to the default page.
3. Open the Smart Tag and select **Add ScriptManager** from the context menu.

4. Add a **RadTabStrip** to the page and configure it:
 - Open the Smart Tag and select **Build RadTabStrip...**
 - Add three items to the tab strip with **Text** properties set to "Flights", "Cars" and "Search Results".
5. Drop a **RadMultiPage** control below the tab strip and configure the multi page:
 - Open the Smart Tag and click the **Add PageView** link two times for a total of three pages (there should be a default page there when the RadMultiPage is first added).
 - Click on the page views and set their ID properties in the Properties window to "pvFlights", "pvCars" and "pvSearchResults".
6. Go back to the RadTabStrip and set the **MultiPageID** property to the RadMultiPage control's ID.
7. Add controls to the "pvFlights" page view:
 - Add a standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "Departing:". Note: We're hijacking "radInput_Sunset" which is a CSS style that exists in the Sunset skin and suits our purposes.
 - Add a **RadDatePicker** and set the **ID** property to "dpDepart".
 - Add another standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "Arriving:".
 - Add a second **RadDatePicker** and set the **ID** property to "dpArrive".
 - Add a standard ASP **Button** control and set the **ID** property to "btnFlights". Note: functionality for the buttons will be implemented in the upcoming section on server-side code.

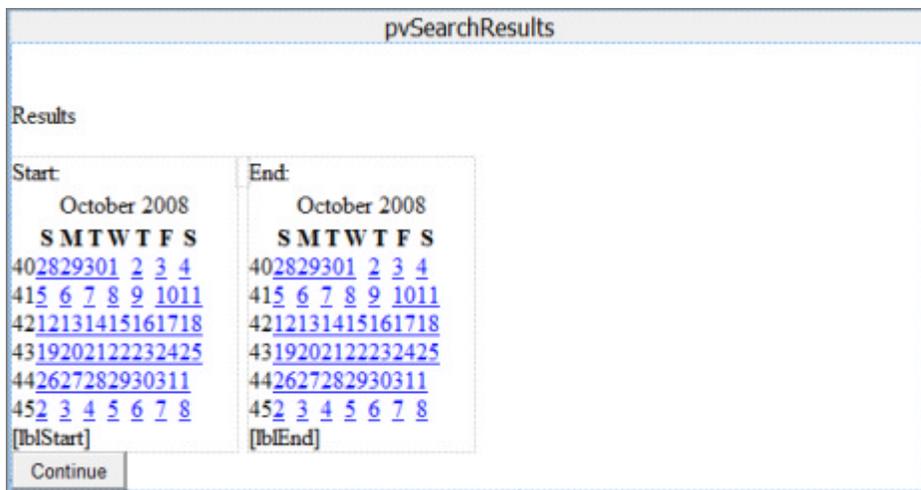


8. Add controls to the "pvCars" page view:
 - Add a standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "Pickup:".
 - Add a **RadDateTimePicker** and set the **ID** property to "dtpPicker".
 - Add another standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "Drop Off:".
 - Add a second **RadDateTimePicker** and set the **ID** property to "dtpDropoff".
 - Add a standard ASP **Button** control and set the **ID** property to "btnCars".



9. Add controls to the "pvSearchResults" page view:
 - Add a standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "Results".
 - From the HTML tab of the Toolbox add a **Div** tag. Set the **Style** property to "float: left; margin:5px".
 - In the Div tag add a standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "Start:".
 - Add a **RadCalendar** and set its **ID** property to "calStart" and the **Enabled** property to false. Also set the

- **PresentationType** property to "Preview". Open the **SelectedDayStyle** property and set **BorderColor** to "LightBlue" and **BorderStyle** to "Dotted".
- Add a standard ASP **Label** control and set the **ID** property to "lblStart". Set the **CssClass** property to "radInput_Sunset" and the **Text** to "".
- From the HTML tab of the Toolbox add a **Div** tag. Set the **Style** property to "float: left; margin:5px"".
- In the Div tag add a standard ASP **Label** control. Set the **CssClass** property to "radInput_Sunset" and the **Text** to "End:".
- Add a **RadCalendar** and set its **ID** property to "calEnd" and the **Enabled** property to false. Also set the **PresentationType** property to "Preview". Open the **SelectedDayStyle** property to set **BorderColor** to "LightBlue" and **BorderStyle** to "Dotted".
- Add a standard ASP **Label** control and set the **ID** property to "lblEnd". Set the **CssClass** property to "radInput_Sunset" and the **Text** to "".
- From the HTML tab of the Toolbox add a **Div** tag. Set the **Style** property to "clear: both".
- ✎ The "clear:both" style setting lets us move the elements that come later to the following line. Failing to clear will display the next elements to the right of the calendars, rather than below the calendars.
- In the Div tag, add a standard ASP **Button** control with **ID** property set to "btnContinue" and **Text** as "Continue".



10. In the code-behind for the page add the code below to the Page_Load event handler. This code sets the minimum and maximum dates for both the departure and arrival **RadDatePicker** controls.

[VB] Setting the Min and Max Dates

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
        ' set both start and end date limits
        dpDepart.MinDate = DateTime.Today
        dpArrive.MinDate = DateTime.Today
        dpDepart.MaxDate = DateTime.Today.AddDays(21)
        dpArrive.MaxDate = DateTime.Today.AddDays(21)
    End If
End Sub
```

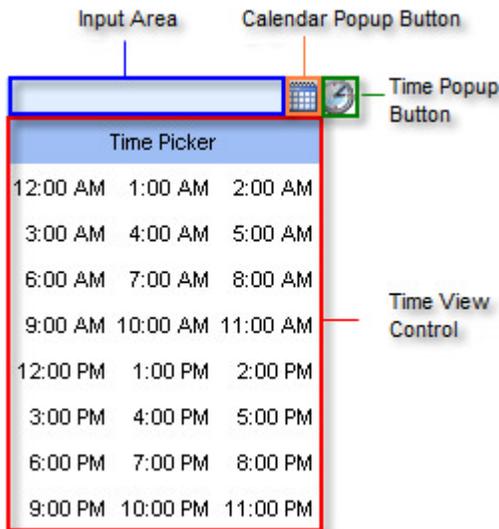
[C#] Setting the Min and Max Dates

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        // set both start and end date limits
        dpDepart.MinDate = DateTime.Today;
        dpArrive.MinDate = DateTime.Today;
        dpDepart.MaxDate = DateTime.Today.AddDays(21);
        dpArrive.MaxDate = DateTime.Today.AddDays(21);
    }
}
```

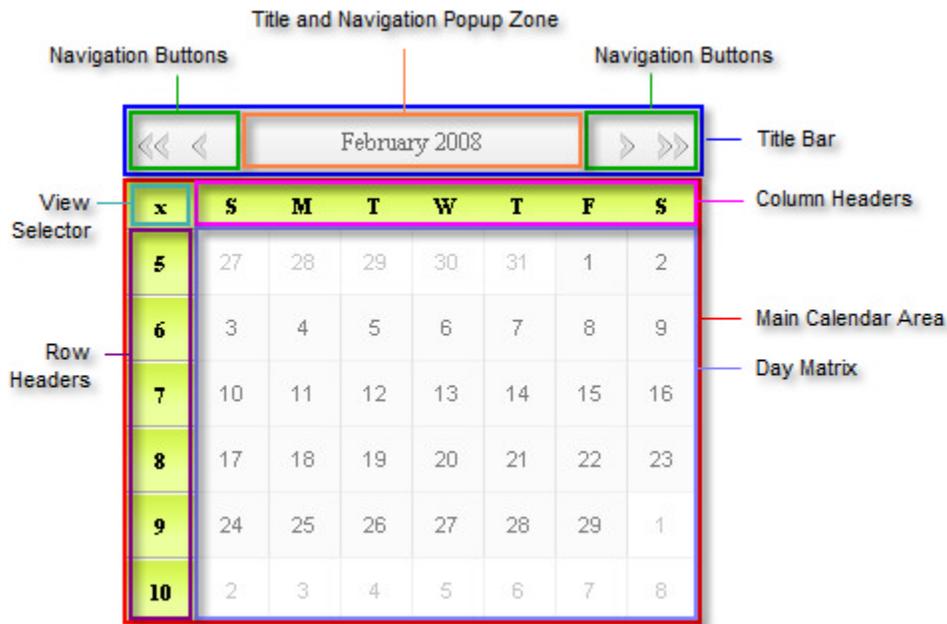
11. Press **Ctrl-F5** to run the application. Notice that difference in how the calendars are handled where the "Flights" RadDatePicker controls are limited to 21 days from the current date. Notice that you not only can't select a date out of that range, but you can't navigate to a month past the month that contains the last valid day. Also notice that the "Sunset" skin set in the web configuration file is propagated throughout the web page and also that the use of the "radInput_Sunset" style makes the labels conform to the other elements on the page.

43.3 Tour of Date-Time and Calendar Controls

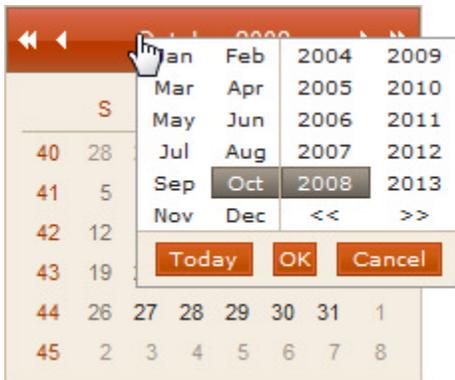
You can get an idea of the layout for RadDatePicker and RadTimePicker by looking at the RadDateTimePicker. The Input Area lets the user directly enter date and time values as text. The Calendar Popup Button displays a RadCalendar popup where the user can select a date to populate the Input Area. The Time Popup Button displays a Time View as a popup where the user can select a time value from a list. You can use the built-in calendar and time view controls or specify a shared control.



RadCalendar has a set of navigation controls and titling along the top. Row and column headers are displayed along the left side and top, with a View Selector button showing at the apex of row and column headers. The View Selector lets the user select all days in the view at once.



The user can also click the Title and Navigation Popup area to display the "Fast Navigation" popup.



43.4 Date-Time and Calendar Controls Designer Interface

In the Visual Studio designer, you can configure the date and time controls using the Smart Tag or the Properties Window. In addition, you can add templates using the Template Design surface for time and calendar controls.

Smart Tag

The RadTimePicker, RadDateTimePicker, RadDatePicker and MonthYearPicker have similar Smart Tags with some small differences. All have the standard Ajax Resources, Skin selection, and Learning center sections.

RadTimePicker Tasks

General Features:

Enable AutoPostBack

Ajax Resources:

[Add RadAjaxManager...](#)
[Add RadScriptManager](#)
[Add RadStyleSheetManager](#)
[Add ScriptManager](#)

Skin

Learning center:

[Online RadCalendar for ASP.NET Ajax examples](#)
[Online RadCalendar for ASP.NET Ajax help](#)
[Online RadControls for ASP.NET Ajax Code Library](#)
Search www.telerik.com for:

[Search](#)
[Go to online Telerik support center](#)
[Edit Templates](#)

RadDateTimePicker Tasks

General Features:

AutoPostBack Controls:

Ajax Resources:

[Add RadAjaxManager...](#)
[Add RadScriptManager](#)
[Add RadStyleSheetManager](#)
[Add ScriptManager](#)

Skin

Learning center:

[Online RadCalendar for ASP.NET Ajax examples](#)
[Online RadCalendar for ASP.NET Ajax help](#)
[Online RadControls for ASP.NET Ajax Code Library](#)
Search www.telerik.com for:

[Search](#)
[Go to online Telerik support center](#)
[Edit Templates](#)

RadDatePicker Tasks

General Features:

Enable AutoPostBack

SelectedDate:

Ajax Resources:

[Add RadAjaxManager...](#)
[Add RadScriptManager](#)
[Add RadStyleSheetManager](#)
[Add ScriptManager](#)

Skin

Learning center:

[Online RadCalendar for ASP.NET Ajax examples](#)
[Online RadCalendar for ASP.NET Ajax help](#)
[Online RadControls for ASP.NET Ajax Code Library](#)
Search www.telerik.com for:

[Search](#)
[Go to online Telerik support center](#)

RadMonthYearPicker Tasks

General Features:

Enable AutoPostBack

SelectedDate:

[Add RadStyleSheetManager](#)

Skin

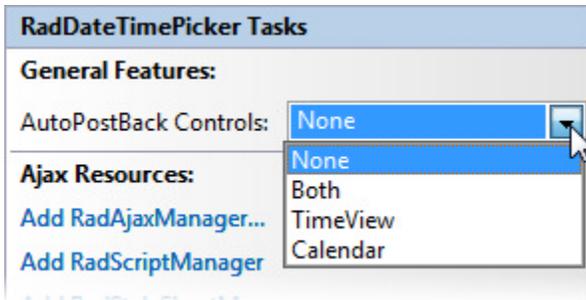
Learning center:

[Online RadCalendar for ASP.NET Ajax examples](#)
[Online RadCalendar for ASP.NET Ajax help](#)
[Online RadControls for ASP.NET Ajax Code Library](#)
Search www.telerik.com for:

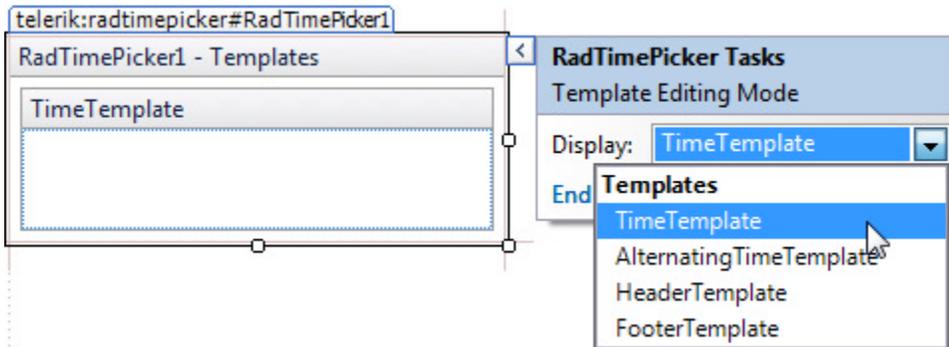
[Search](#)
[Go to online Telerik support center](#)

All contain the ability to enable AutoPostBack, but RadDateTimePicker allows you to choose which controls trigger the post back:

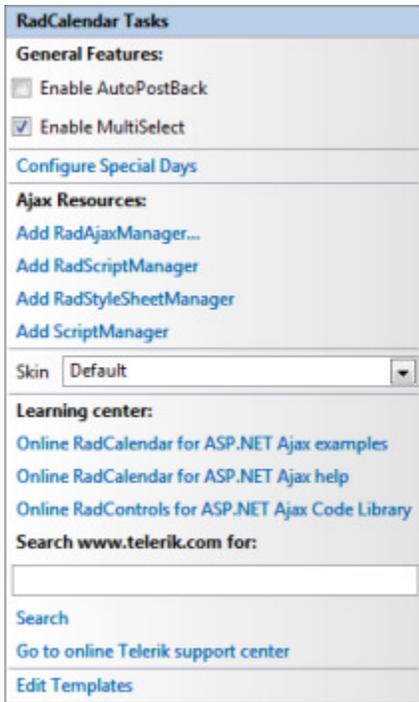
UI for ASP.NET AJAX



The RadDateTimePicker and RadTimePicker controls also let you Edit Templates to control the exact look and feel of each cell that displays the time, an alternating template so you can visually differentiate closely packed times for a clearer user interface, header and footer templates.



The RadCalendar Smart Tag lets you toggle the **AutoPostBack** and the ability to **MultiSelect** days on the calendar. The **Configure Special Days** link jumps you to the RadCalendarDay Collection Editor dialog. The **Edit Templates** link lets you create customized headers and footers for the calendar.



Properties Window For Date and Time Picker Controls

At design time, you can use the Properties Window to configure almost every aspect of the date, time and

calendar controls. (A notable exception is the creation of templates.) As with the other controls we have seen, we'll take a look at the most significant properties.

Date and Time Picker Behavior

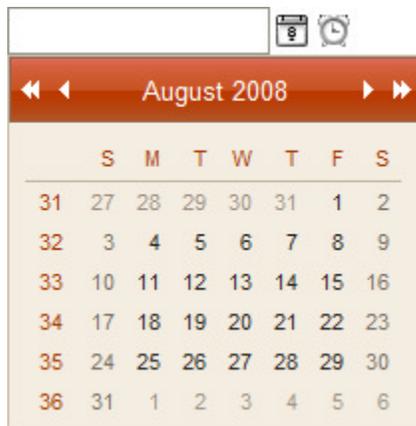
RadDatePicker, RadTimePicker and RadDateTimePicker controls are all made up of sub-components such as the calendar, date and time input controls, date and time popup buttons and time views. Go to the Behavior group of properties to find these controls. You can configure these controls by opening their sub-properties.

Behavior	
AutoPostBackControl	None
Calendar	dtpPickup.Calendar
Culture	English (United States)
DateInput	dtpPickup.DateInput
DatePopupButton	dtpPickup.DatePopupButton
Enabled	True
EnableTheming	True
EnableTyping	True
EnableViewState	True
SharedCalendarID	
SharedTimeViewID	
SkinID	
TimePopupButton	dtpPickup.TimePopupButton
TimeView	dtpPickup.TimeView
ToolTip	
Visible	True

You can also share controls to be used by multiple pickers by setting the SharedCalendarID and SharedTimeViewID properties. The SharedCalendarID points to the ID of a RadCalendar control which may also be used stand-alone, but the SharedTimeViewID points to a special RadTimeView control available on the ToolBox that cannot be used stand-alone.

Controlling the Date Selection

Within the calendar portion of a picker control you can set the **FocusedDate** and the **SelectedDate**. The component uses FocusedDate to focus the calendar on a given month when the input is empty, but doesn't show the FocusedDate as selected. The screenshot below was taken after the FocusedDate was set to 8/1/2008 and the SelectedDate was left blank.



You can limit the user selection within a range of dates by setting the **MinDate** and **MaxDate** properties. These

UI for ASP.NET AJAX

are all DateTime types so you can populate them using the usual .NET DateTime methods and properties, e.g. Today, AddDays(), AddMonths(), etc.

Date Selection	
FocusedDate	1/1/1980
MaxDate	12/31/2099
MinDate	1/1/1980
SelectedDate	

Properties Window for RadCalendar

Controlling Calendar Appearance

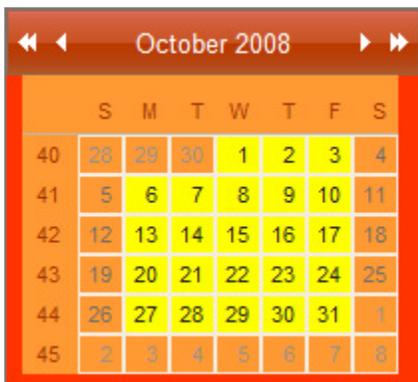
The Appearance group of calendar properties controls layout, styling and has properties to enumerate and control "Special days".

Appearance	
BackColor	
BorderColor	
BorderStyle	NotSet
BorderWidth	
CalendarDayTemplates	(Collection)
CalendarTableStyle	
CssClass	
DayOverStyle	
DayStyle	
DefaultCellPadding	0
DefaultCellSpacing	0
DisabledDayStyle	
EnableAjaxSkinRendering	True
EnableEmbeddedBaseStylesheet	True
EnableEmbeddedScripts	True
EnableEmbeddedSkins	True
FastNavigationStyle	
Font	
ForeColor	
HeaderStyle	
ImagesPath	
OtherMonthDayStyle	
OutOfRangeDayStyle	
PresentationType	Interactive
SelectedDayStyle	
Skin	Default
SpecialDays	(Collection)
TitleStyle	
ViewSelectorStyle	
WeekendDayStyle	

The Appearance group of properties include:

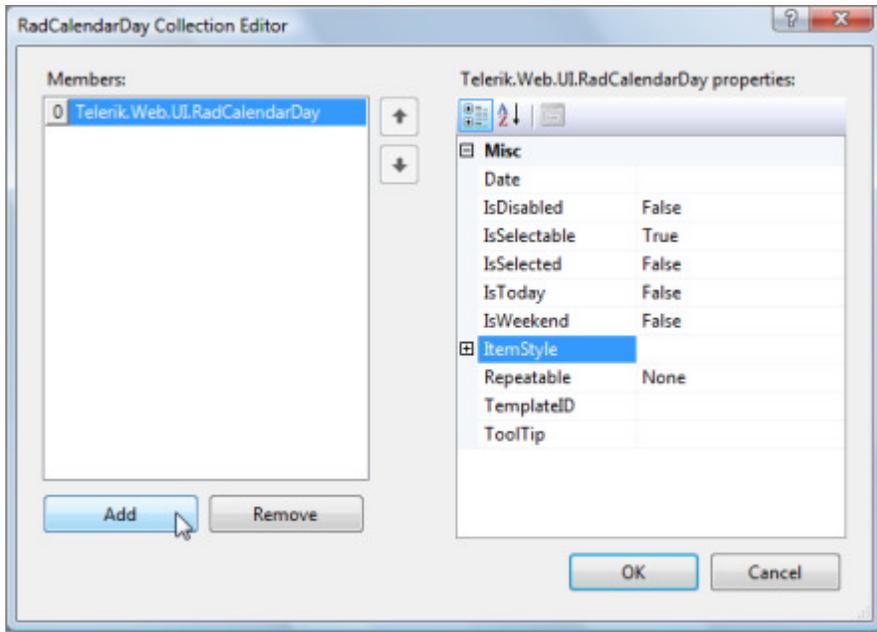
- Layout, e.g. **DefaultCellPadding**, **DefaultCellSpacing**
- Specific appearance properties for the calendar as a whole, e.g. **BackColor**, **BorderStyle**, etc., as well as a **CssClass** property to allow the calendar to be styled.
- Properties suffixed with "Style" (e.g. **CalendarTableStyle**, **FastNavigationStyle**, etc). Each of these "Style" properties has sub-properties that define specific appearance (i..e. **BackColor**, **BorderStyle**, etc) or let you define a **CssClass**.
- The **Skin** and skin related properties.

How do these appearance properties for the calendar as a whole, "Style" properties and Skins work together? To get an idea, take a look at the screenshot below where the calendar Skin is set to "Sunset", the BackColor to "Red", CalendarTableStyle BackColor to "Orange" and the DayStyle BackColor to "Yellow". You can see that the calendar heading retains the "Sunset" skin, but the other visual aspects of the calendar have been overridden by the more specific appearance properties.



You may have noticed the **PresentationType** property in the Appearance group. Setting the **PresentationType** property from "Interactive" to "Preview" determines how the Calendar will handle layout and user interaction. Using the "Preview" setting prevents the user from selecting a date, but the user can still navigate in the calendar. In the past you might have set **Enabled** to false, but this prevents the user from navigating.

The **SpecialDays** property is a collection of **RadCalendarDay** objects that can be styled separately from the other calendar days. In fact, each special day can be styled individually by using the **ItemStyle** property to set **BackColor**, **CssClass** etc., or can use its **TemplateID** to point at a template.



Templates for special days are contained in the **CalendarDayTemplates** collection property. You can populate these templates directly within the markup.

Calendar Behavior

Behavior properties include an **Orientation** property that can be **RenderInRows** (the default) or **RenderInColumns**. You should also note the **EnableMultiSelect** that defaults to true. The screenshot below was taken after Orientation was set to RenderInColumns and EnableMultiSelect was left at its default of true.



Managing Dates

This group of properties controls the layout of days, what dates are visible, the valid dates that can be selected and the actual dates that are selected.

Dates Management	
FirstDayOfWeek	Default
FocusedDate	1/1/1980
FocusedDateColumn	0
FocusedDateRow	0
RangeMaxDate	12/30/2099
RangeMinDate	1/1/1980
SelectedDate	
SelectedDates	(Collection)

FirstDayOfWeek is an enumeration of Sunday through Saturday that controls the layout of days in the calendar. **FocusedDate** makes the specified date visible in the calendar when **SelectedDate** has not been set. Note that **FocusedDate** does not actually select the date or make any appearance changes to the date.

At the time of this writing, **FocusedDateColumn** and **FocusedDateRow** are marked to be obsolete in future versions.

RangeMinDate and **RangeMaxDate** set the lower and upper boundaries of dates that may be selected. Dates outside this range will be disabled and the user will not even be able to navigate outside views in the range.

View Settings

The calendar is presented as a "single view" by default where only a single month shows. Instead you can create a "multiple view" showing multiple months. The General View Settings and MonthView Specific Settings control the layout and appearance for these views.

General View Settings	
CellAlign	Center
CellVAlign	Middle
EnableNavigationAnimation	False
MonthLayout	Layout_7columns_x_6rows
MultiViewColumns	1
MultiViewRows	1
SingleViewColumns	7
SingleViewHeight	0px
SingleViewRows	6
SingleViewWidth	0px

MonthView Specific Settings	
CellDayFormat	%d
ShowOtherMonthsDays	True
UseColumnHeadersAsSelectors	True
UseRowHeadersAsSelectors	True

To get multiple months shown within the calendar, either set the **MonthLayout** property to any of a number of pre-defined dimensions (e.g. 7 columns x 6 rows, 14 x 3, 21 x 2, 7 x 6...) or you can specify your own by setting the **MultiViewColumns** and **MultiViewRows** properties to values larger than the default of "1". The screenshot below shows a whole year of months where **MultiViewColumns** is set to "4" and **MultiViewRows** to "3".

UI for ASP.NET AJAX



Likewise you can control the dimensions of a single month view using the **SingleViewColumns** and **SingleViewRows** properties. In the screenshot below, **SingleViewColumns** is set to "14". Notice that only the January days are highlighted even though the days extend into February and March.



EnableNavigationAnimation is false by default but when enabled produces a very cool effect that "slides" between months when the navigation buttons are clicked.

In the Monthview Specific Settings you can use a standard format character in the **CellDayFormat** property. By default this value is "%d", i.e. it outputs an integer number with no leading zeros. The screenshot below uses the format string "dd" to include a leading zero:



ShowOtherMonthDays when set to false hides any days that are not part of the month being viewed. **UseColumnHeadersAsSelectors** and **UseRowHeadersAsSelectors** are both true by default and let your user select entire columns or rows with a single click.

Navigation

The Navigation properties control the behavior, visibility, layout and text for the navigation buttons along the top of the calendar.

Navigation Management	
EnableMonthYearFastNavigation	True
EnableNavigation	True
FastNavigationNextImage	fastNavRight.gif
FastNavigationNextText	>>
FastNavigationNextToolTip	>>
FastNavigationPrevImage	fastNavLeft.gif
FastNavigationPrevText	<<
FastNavigationPrevToolTip	<<
FastNavigationSettings	
CancelButtonCaption	Cancel
DatesOutOfRangeMessage	Date is out of range.
EnableTodayButtonSelection	False
OkButtonCaption	OK
TodayButtonCaption	Today
FastNavigationStep	3
NavigationCellPadding	3
NavigationCellSpacing	0
NavigationNextImage	arrowRight.gif
NavigationNextText	>
NavigationNextToolTip	>
NavigationPrevImage	arrowLeft.gif
NavigationPrevText	<
NavigationPrevToolTip	<

Titling

Titling properties handle the formatting and layout of the title area at the top center of the calendar.

Title Settings	
DateRangeSeparator	-
DayCellToolTipFormat	dddd, MMMM dd, yyyy
TitleAlign	Center
TitleFormat	MMMM yyyy

43.5 Date-Time and Calendar Controls Server-Side Programming

Date-Time Picker Controls Server Objects

You can of course directly access the RadDatePicker, RadTimePicker, RadDateTimePicker and RadMonthYearPicker controls, but you can also access the objects that make up each control:

- **RadDateInput** is the class for the input area of RadDatePicker, RadTimePicker, and RadDateTimePicker. It handles the formatting and parsing of date and time strings, and has a number of its own properties, methods, and events. RadDateInput is one of the standard RadInput controls.
- **CalendarPopupButton** and **TimePopupButton** are the classes for the popup buttons that display the calendar on RadDatePicker and RadDateTimePicker and the time view popup in RadTimePicker and RadDateTimePicker.
- **RadMonthYearPopUpButton** is the class for the button that displays the MonthYearView popup in the MonthYearPicker control.
- **RadTimeView** is the class for the popup time view used by RadTimePicker and RadDateTimePicker. RadTimeView is also available in the toolbox so that a single instance can be shared among multiple RadTimePicker and RadDateTimePicker controls.
- **MonthYearView** is the class for the popup monthyear view used by the RadMonthyearPicker.
- **DataListItem** is the standard System.Web.UI.WebControls.DataListItem class. The RadTimeView control uses this class for each of the items it displays. It can be accessed through the DataList property, and is also available as the Item property of the EventArgs of the ItemCreated and ItemDataBound events. DataListItem descends from WebControl.

Calendar Server Objects

In addition to the RadCalendar class,

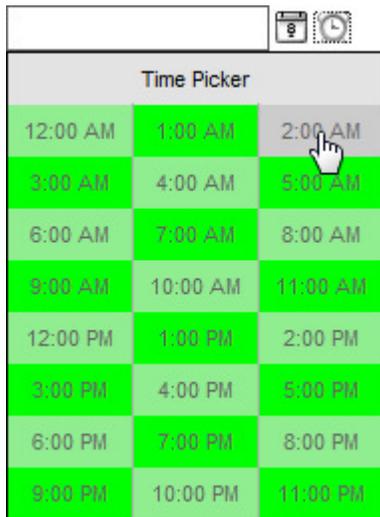
- **RadCalendarDay** maps a date value to its corresponding visual settings and a number of boolean properties that represent its status (weekend date, disabled, selected, and so on). This class is used in the Day property of the EventArgs of the DayRender event, and for elements in the SpecialDays collection.
- **CalendarView** represents the current view of the calendar. It can include links to child views if the calendar is in multi-view mode. This class is also used in the OldView and NewView properties of the EventArgs of the DefaultViewChanged event.
- **MonthView** is a descendant of CalendarView that represents the view information for a single month. The View property of the EventArgs in the DayRender event is of MonthView type. It is also the type for the CalendarView property when the calendar is in single-view mode.
- **TableCell** is the control class for a cell in the day matrix. The Cell property within the DayRender event args is of this type.
- **RadDate** is a wrapper for System.DateTime. It is used for persisting DateTime values in collections such as the SelectedDates property.
- **DayTemplate** is the type for each element of the CalendarDayTemplates property. DayTemplate implements the ITemplate interface.

Date-Time Picker Controls Server Events

Events of special interest to date-time picker controls are:

- **SelectedDateChanged:** RadDatePicker and RadDateTimePicker surface a SelectedDateChanged event that occurs when the user changes the value of the control, either when the input area loses focus after the user has typed a new value, or when the user selects a new value in the popup calendar or time view control. The event passes **SelectedDateChangedEventArgs** as a parameter that contains two DateTime properties **NewDate** and **OldDate**. Don't forget that this event will not fire unless there's a postback, so you'll need to set the AutoPostBack, AutoPostBackControl (when using RadDateTimePicker) or trigger a postback through some other means.
- **ItemCreated** and **ItemDataBound:** Controls that use a RadTimeView, i.e. RadTimePicker and

RadDateTimePicker also surface ItemCreated and ItemDataBound events. These two events pass **TimePickerEventArgs** which has a single property of interest, **Item**, which is of type **DataListItem**. The example below catches the ItemCreated event and sets the Item CssClass based on the ItemType.



[CSS] Alternating Row Styles

```
<style type="text/css">
  .FirstRowCss
  {
    background-color: LightGreen;
  }
  .FirstAlternatingRowCss
  {
    background-color: Lime;
  }
</style>
```

[VB] Handling the ItemCreated Event

```
Protected Sub RadDateTimePicker1_ItemCreated(ByVal sender As Object, ByVal e As
Telerik.Web.UI.Calendar.TimePickerEventArgs)
  If e.Item.ItemType = ListItemType.Item Then
    e.Item.CssClass = "FirstRowCss"
  End If
  If e.Item.ItemType = ListItemType.AlternatingItem Then
    e.Item.CssClass = "FirstAlternatingRowCss"
  End If
End Sub
```

[C#] Handling the ItemCreated Event

```
protected void RadDateTimePicker1_ItemCreated(object sender,
Telerik.Web.UI.Calendar.TimePickerEventArgs e)
{
  if (e.Item.ItemType == ListItemType.Item)
  {
    e.Item.CssClass = "FirstRowCss";
  }
  if (e.Item.ItemType == ListItemType.AlternatingItem)
```

```
{  
    e.Item.CssClass = "FirstAlternatingRowCss";  
}
```

- **ChildrenCreated** occurs when the child controls (the input area, popup buttons, and embedded calendar or time view controls) are created. The example below shows how to add a link right next to the calendar popup button of a RadDatePicker. When the button is clicked the picker's `clear()` client method is called.



[VB] Handling the ChildrenCreated Event

```
Protected Sub RadDatePicker1_ChildrenCreated(ByVal sender As Object, ByVal e As EventArgs)  
    Dim picker As RadDatePicker = DirectCast(sender, RadDatePicker)  
    Dim clearLink As New HyperLink()  
    clearLink.NavigateUrl = String.Format("javascript:$find('{0}').clear()", picker.ClientID)  
    clearLink.Text = "Clear"  
    clearLink.ToolTip = "Clear the date picker"  
    picker.Controls.Add(clearLink)  
End Sub
```

[C#] Handling the ChildrenCreated Event

```
protected void RadDatePicker1_ChildrenCreated(object sender, EventArgs e)  
{  
    RadDatePicker picker = (RadDatePicker)sender;  
    HyperLink clearLink = new HyperLink();  
    clearLink.NavigateUrl =  
        string.Format("javascript:$find('{0}').clear()", picker.ClientID);  
    clearLink.Text = "Clear";  
    clearLink.ToolTip = "Clear the date picker";  
    picker.Controls.Add(clearLink);  
}
```

Calendar Server Events

- **DayRender** occurs immediately before the calendar renders the cell for a single day in the day matrix.
- **HeaderCellRender** occurs immediately before the calendar renders a cell in the column or row headers (or the view selector). The `HeaderCellRenderEventArgs` passed in as a parameter has two properties: `Cell` and `HeaderType`. You can use the `Cell` property to configure the appearance of the cell and the `HeaderType` to determine where a given cell will be placed. This next example sets the `Cell BackColor` based on the `HeaderType`. Note that the "View" `HeaderType` denotes the View Selector button in the upper left hand corner of the calendar. Also be aware that you should set `EnableViewSelector` to true for this example.

x	S	M	T	W	T	F	S
40	28	29	30	1	2	3	4
41	5	6	7	8	9	10	11
42	12	13	14	15	16	17	18
43	19	20	21	22	23	24	25
44	26	27	28	29	30	31	1
45	2	3	4	5	6	7	8

[VB] Handling the HeaderCellRender Event

```
Protected Sub RadCalendar1_HeaderCellRender(ByVal sender As Object, ByVal e As
Telerik.Web.UI.Calendar.HeaderCellRenderEventArgs)
    Select Case e.HeaderType
        Case Telerik.Web.UI.Calendar.HeaderType.Column
            e.Cell.BackColor = System.Drawing.Color.AliceBlue
            Exit Select
        Case Telerik.Web.UI.Calendar.HeaderType.Row
            e.Cell.BackColor = System.Drawing.Color.PaleGreen
            Exit Select
        Case Telerik.Web.UI.Calendar.HeaderType.View
            e.Cell.BackColor = System.Drawing.Color.PaleTurquoise
            Exit Select
    End Select
End Sub
```

[C#] Handling the HeaderCellRender Event

```
protected void RadCalendar1_HeaderCellRender(object sender,
Telerik.Web.UI.Calendar.HeaderCellRenderEventArgs e)
{
    switch (e.HeaderType)
    {
        case Telerik.Web.UI.Calendar.HeaderType.Column:
            e.Cell.BackColor = System.Drawing.Color.AliceBlue;
            break;
        case Telerik.Web.UI.Calendar.HeaderType.Row:
            e.Cell.BackColor = System.Drawing.Color.PaleGreen;
            break;
        case Telerik.Web.UI.Calendar.HeaderType.View:
            e.Cell.BackColor = System.Drawing.Color.PaleTurquoise;
            break;
    }
}
```

- **SelectionChanged** occurs when the user changes the current selection in the calendar. This event does not fire unless the `AutoPostBack` property is `True`. This event passes a **SelectedDatesEventArgs** that has a single property **SelectedDates**. `SelectedDates` is a `DateTimeCollection`.
- **DefaultViewChanged** occurs when the user changes the current view using the navigation controls in the title bar. This event does not fire unless the `AutoPostBack` property is `True`. This event passes **DefaultViewChangedEventArgs** as a parameter. This object has two properties, `OldView` and `NewView`. Both are `CalendarView` objects that contain extensive properties describing the view before and

after navigation.

[VB] Handling the DefaultViewChanged Event

```
Protected Sub RadCalendar1_DefaultViewChanged(ByVal sender As Object, ByVal e As Telerik.Web.UI.Calendar.DefaultViewChangedEventArgs)
    If e.OldView.ViewStartDate < e.NewView.ViewStartDate Then
        Label1.Text = e.OldView.TitleContent + " -> " + e.NewView.TitleContent
    Else
        Label1.Text = e.NewView.TitleContent + " <- " + e.OldView.TitleContent
    End If
End Sub
```

[C#] Handling the DefaultViewChanged Event

```
protected void RadCalendar1_DefaultViewChanged(object sender,
Telerik.Web.UI.Calendar.DefaultViewChangedEventArgs e)
{
    if (e.OldView.ViewStartDate < e.NewView.ViewStartDate)
        Label1.Text = e.OldView.TitleContent + " -> " + e.NewView.TitleContent;
    else
        Label1.Text = e.NewView.TitleContent + " <- " + e.OldView.TitleContent;
}
```



You can find the complete source for this project at:
\\VS Projects\DateTimeSchedule\DateTimeServerSide

43.6 Date-Time and Calendar Controls Server-Side Walk-through

Starting with the "Getting Started Project" or a copy of it, this walk-through will add functionality to move date and time information between the controls.

1. Remove the RadTabStrip **MultiPageID** value so that property is left blank. *We will navigate the tab strip and multi-page in code from now on.*
2. Open the RadTabStrip **Tabs** collection in the Property window, select the last tab "Search Results" and set its **Visible** property to true.
3. In the code-behind, add a helper method GetSpecialDays() that returns an array of RadCalendarDay populated with days that fall on a specific day of the week and that fall between two given dates.

[VB] Get Special Days

```
Private Function GetSpecialDays(ByVal startDate As DateTime, ByVal endDate As DateTime,
ByVal dayOfWeek As DayOfWeek) As RadCalendarDay()
    Dim specialDays As New List(Of RadCalendarDay)()
    Dim endMonth As Integer = endDate.Month
    ' set up a temporary DateTime variable "date" with the first day
    ' from the startDate passed in.
    Dim [date] As DateTime = startDate
    ' walk through the date range, one day at a time
    While [date].Month <= endMonth
        ' when we encounter the passed in "dayOfWeek", create a
        ' new RadCalendarDay object and populate it with the
        ' date, style the day, make it read-only and add it
        ' to the collection.
        If [date].DayOfWeek = dayOfWeek Then
```

```

Dim specialDay As New RadCalendarDay()
specialDay.[Date] = [date]
specialDay.ItemStyle.BackColor = Color.LightGray
specialDay.ToolTip = "Not Available"
specialDays.Add(specialDay)
End If
[date] = [date].AddDays(1)
End While
Return specialDays.ToArray()
End Function

```

[C#] Get Special Days

```

private RadCalendarDay[] GetSpecialDays(
    DateTime startDate, DateTime endDate, DayOfWeek dayOfWeek)
{
    List<RadCalendarDay> specialDays = new List<RadCalendarDay>();
    int endMonth = endDate.Month;
    // set up a temporary DateTime variable "date" with the first day
    // from the startDate passed in.
    DateTime date = startDate;
    // walk through the date range, one day at a time
    while (date.Month <= endMonth)
    {
        // when we encounter the passed in "dayOfWeek", create a
        // new RadCalendarDay object and populate it with the
        // date, style the day, make it read-only and add it
        // to the collection.
        if (date.DayOfWeek == dayOfWeek)
        {
            RadCalendarDay specialDay = new RadCalendarDay();
            specialDay.Date = date;
            specialDay.ItemStyle.BackColor = Color.LightGray;
            specialDay.ToolTip = "Not Available";
            specialDays.Add(specialDay);
        }
        date = date.AddDays(1);
    }
    return specialDays.ToArray();
}

```

4. In the designer, double-click the "btnFlights" search button in the pvFlights page to create a Click event handler. This event handler moves the dates from the picker controls to the corresponding calendars.



Gotcha! When you assign from the RadDatePicker SelectedDate property to the RadCalendar SelectedDate, be sure to cast to a DateTime type. If both types are DateTime, why the cast? The RadDatePicker SelectedDate is a nullable type where the type is denoted by "DateTime?", not "DateTime".

[VB] Handling the Flight Search Button Click

```

Protected Sub btnFlights_Click(ByVal sender As Object, ByVal e As EventArgs)
    lblTitle.Text = "Flight Departure and Arrival"
    ' move the date picker dates over to the corresponding calendar dates.
    ' Note: be sure to cast this to DateTime because the date picker SelectedDate
    ' is a nullable type, i.e. "DateTime?".

```

```
calStart.SelectedDate = DirectCast(dpDepart.SelectedDate, DateTime)
calEnd.SelectedDate = DirectCast(dpArrive.SelectedDate, DateTime)
' Add informational message
lblStart.Text = [String].Format("Depart on {0:d}", dpDepart.SelectedDate)
lblEnd.Text = [String].Format("Arrive on {0:d}", dpArrive.SelectedDate)
' Navigate to the "results page" and hide tab strip
RadMultiPage1.SelectedIndex = RadMultiPage1.PageViews.Count - 1
RadTabStrip1.Visible = False
End Sub
```

[C#] Handling the Flight Search Button Click

```
protected void btnFlights_Click(object sender, EventArgs e)
{
    lblTitle.Text = "Flight Departure and Arrival";
    // move the date picker dates over to the corresponding calendar dates.
    // Note: be sure to cast this to DateTime because the date picker SelectedDate
    // is a nullable type, i.e. "DateTime?".
    calStart.SelectedDate = (DateTime)dpDepart.SelectedDate;
    calEnd.SelectedDate = (DateTime)dpArrive.SelectedDate;
    // Add informational message
    lblStart.Text = String.Format("Depart on {0:d}", dpDepart.SelectedDate);
    lblEnd.Text = String.Format("Arrive on {0:d}", dpArrive.SelectedDate);
    // Navigate to the "results page" and hide tab strip
    RadMultiPage1.SelectedIndex = RadMultiPage1.PageViews.Count - 1;
    RadTabStrip1.Visible = false;
}
```

5. Create another button click event handler for "btnCars" and add the code below. The logic parallels the "btnFlights" version.

[VB] Handling the Car Search Button Click

```
Protected Sub btnCars_Click(ByVal sender As Object, ByVal e As EventArgs)
    lblTitle.Text = "Car Rental Pickup and Dropoff"
    ' move the date picker dates over to the corresponding calendar dates.
    calStart.SelectedDate = DirectCast(dtpPickup.SelectedDate, DateTime)
    calEnd.SelectedDate = DirectCast(dtpDropoff.SelectedDate, DateTime)
    ' Add informational message
    lblStart.Text = [String].Format("Pickup at {0:t}", dtpPickup.SelectedDate)
    lblEnd.Text = [String].Format("Drop off at {0:t}", dtpDropoff.SelectedDate)
    ' Navigate to the "results page" and hide tab strip
    RadMultiPage1.SelectedIndex = RadMultiPage1.PageViews.Count - 1
    RadTabStrip1.Visible = False
End Sub
```

[C#] Handling the Car Search Button Click

```
protected void btnCars_Click(object sender, EventArgs e)
{
    lblTitle.Text = "Car Rental Pickup and Dropoff";
    // move the date picker dates over to the corresponding calendar dates.
    calStart.SelectedDate = (DateTime)dtpPickup.SelectedDate;
    calEnd.SelectedDate = (DateTime)dtpDropoff.SelectedDate;
    // Add informational message
    lblStart.Text = String.Format("Pickup at {0:t}", dtpPickup.SelectedDate);
    lblEnd.Text = String.Format("Drop off at {0:t}", dtpDropoff.SelectedDate);
    // Navigate to the "results page" and hide tab strip
}
```

```
RadMultiPage1.SelectedIndex = RadMultiPage1.PageViews.Count - 1;
RadTabStrip1.Visible = false;
}
```

6. Add the code below to the "Continue" button. *This event handler simply makes the tab strip visible again and navigates back to the first tab.*

[VB] Handling the Continue Button Click

```
Protected Sub btnContinue_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' Navigate back to the "flights" page view and show the tabstrip
    RadMultiPage1.SelectedIndex = 0
    RadTabStrip1.SelectedIndex = 0
    RadTabStrip1.Visible = True
End Sub
```

[C#] Handling the Continue Button Click

```
protected void btnContinue_Click(object sender, EventArgs e)
{
    // Navigate back to the "flights" page view and show the tabstrip
    RadMultiPage1.SelectedIndex = 0;
    RadTabStrip1.SelectedIndex = 0;
    RadTabStrip1.Visible = true;
}
```

7. Press **Ctrl-F5** to run the application. Experiment with setting dates and times, then navigating to and from the "search results" page. Notice that there is no checking for blank dates (will cause an error) or start dates that occur after ending dates. These checks will be added next.



You can find the complete source for this project at:
 \VS Projects\DateTimeSchedule\DateTime_ServerSide2

43.7 Date-Time Picker Validation

Starting with the previous server-side walk-through we will add standard validation controls to verify that the date and time values exist and that starting dates do not exceed ending dates.

1. Set the RadTabStrip **CausesValidation** property to false.
2. Just after the dpDepart date picker add a ASP.NET **RequiredFieldValidator** with the following properties:
 - o **ErrorMessage:** "Departure date required"
 - o **ControlToValidate:** "dpDepart"
 - o **ValidationGroup:** "FlightsGroup"
 - o **Text:** "*"

3. Just after the dpArrive date picker add a ASP.NET **RequiredFieldValidator** with the following properties:
 - **ErrorMessage:** "Arrival date required"
 - **ControlToValidate:** "dpArrive"
 - **ValidationGroup:** "FlightsGroup"
 - **Text:** "*"
4. Add a CompareValidator with these properties:
 - **ErrorMessage:** "The arrival date must be after the departure date"
 - **ControlToValidate:** "dpArrive"
 - **ControlToCompare:** "dpDepart"
 - **ValidationGroup:** "FlightsGroup"
 - **Text:** "*"
 - **Operator:** "GreaterThan"
 - **Type:** "Date"
5. Set the btnFlights **ValidationGroup** property to "FlightsGroup". *Now when you click btnFlights the validation will fire for both date inputs.*
6. Just after the dtpPickup date picker add a ASP.NET **RequiredFieldValidator** with the following properties:
 - **ErrorMessage:** "Pick-up date required"
 - **ControlToValidate:** "dtpPickup"
 - **ValidationGroup:** "CarsGroup"
 - **Text:** "*"
7. Just after the dtpDropoff date picker add a ASP.NET **RequiredFieldValidator** with the following properties:
 - **ErrorMessage:** "Drop-off date required"
 - **ControlToValidate:** "dtpDropoff"
 - **ValidationGroup:** "CarsGroup"
 - **Text:** "*"
8. Add a CustomValidator:
 - **ClientValidationFunction:** "CompareCarsDates"
 - **ErrorMessage:** "Pick-up must be less than drop-off date and time".
 - **ValidationGroup:** "CarsGroup"
9. Set the btnCars **ValidationGroup** property to "CarsGroup".
10. Inside the <form> tag add the JavaScript below.

*The JavaScript method must match the **ClientValidationFunction** property value in the CustomValidator. The method takes a sender and event args. The "args" parameter contains an IsValid property that is set depending on the functions custom criteria. In this case the criteria is that the pick-up is less than the drop-off date and time.*

F[JavaScript] CustomValidator Compare Function

```
<script type="text/javascript">
function CompareCarsDates(sender, args) {
    var dtpPickup = $find("<%= dtpPickup.ClientID %>");
```

```

var dtpDropoff = $find("<%= dtpDropoff.ClientID %>")
args.IsValid = dtpPickup.get_selectedDate() < dtpDropoff.get_selectedDate();
}
</script>

```

11. Underneath the multi-page add two **ValidationSummary** controls, the first with ValidationGroup property "FlightsGroup" and the second with ValidationGroup "CarsGroup".
12. Press **Ctrl-F5** to run the application.



You can find the complete source for this project at:
 \VS Projects\DateTimeSchedule\DateTime_Validation



Gotcha! If clicking the button doesn't perform the validation, check that the ValidationGroup property of both the button that initiates the validation, the controls to be validated and the validation controls themselves. These must all be present and match.

43.8 Date-Time and Calendar Controls Client-Side Programming

Date and Time Picker Client API

You can access the date-time picker controls and calendar using the usual \$find() method against the control's ClientID property. As with the date and time picker server-side API, you can also access child controls:

[JavaScript] Getting References to Date and Time Picker Objects and Child Elements

```

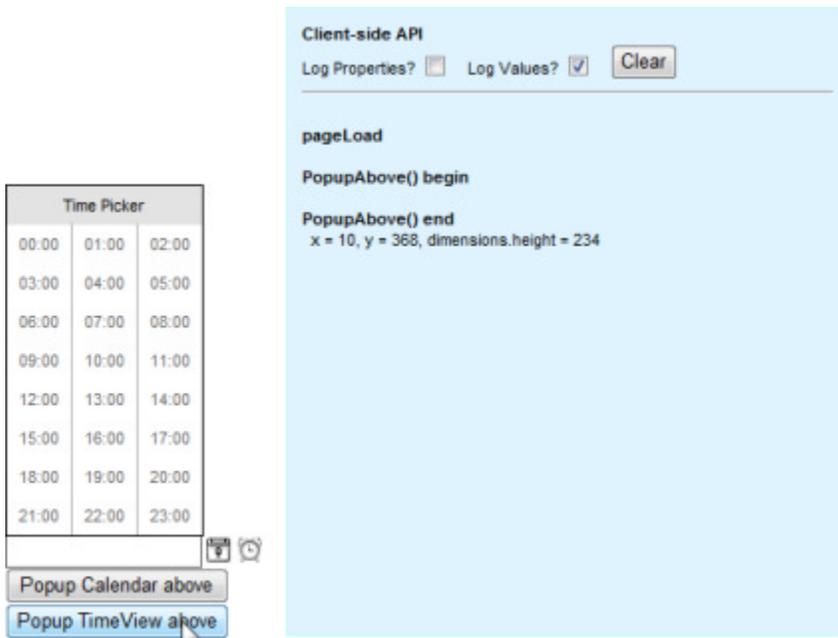
var datepicker = $find("<%= RadDatePicker1.ClientID %>");
var timepicker = $find("<%= RadTimePicker1.ClientID %>");
var datetimetypepicker = $find("<%= RadDateTimePicker1.ClientID %>");
var input = RadDateTimePicker1.get_dateInput();
var calendar = RadDateTimePicker1.get_calendar();
var timeview = RadDateTimePicker1.get_timeView();

```

To get the DOM element for any of the picker controls and the RadCalendar use the **get_element()** method.

You can drill down to the DOM elements for the major parts that make up the picker controls. To get the calendar or time popup buttons, use **get_popupButton()** or **get_timePopupButton()**. To get DOM elements for the popups use **get_popupContainer()** and **get_timePopupContainer()**. To get the textbox portion of the picker control use the **get_textBox()** method.

The example below displays the calendar or time view popup by clicking a button that in turn fires a **PopupAbove()** method. **PopupAbove()** gets a reference to the div DOM element for the calendar or time view and a reference to the textbox. The method then gets the position of the text box and calculates the offset where the popup will be displayed at. The popup is displayed using the **showPopup()** method of the picker. You can call **showPopup()** without parameters to display the popup just below the text box. If you have screen real estate issues you can pass the x and y pixel coordinates to **showPopup()**. Another option is to use *Telerik.Web.RadDatePickerPopupDirection* enumeration as shown in the example below where the popup is displayed just above the text box.



[ASP.NET] Using Picker DOM Elements

```
function popupAbove(showCalendar) {
    logTitle("PopupAbove() begin ");
    // get a client side reference to the date time picker
    var picker = $find("<%= RadDateTimePicker1.ClientID %>");
    // get a reference to the input textbox within the picker
    var textBox = picker.get_textBox();
    // get a reference to the div DOM element for the calendar
    // or time view. Get the position of the text box, the
    // dimensions for the popup, and display the popup
    // at Y coordinate at the position of the textbox subtracting
    // the height of the popup.
    if (showCalendar) {
        var popupElement = picker.get_popupContainer();
        var dimensions = picker.getElementDimensions(popupElement);
        var position = picker.getElementPosition(textBox);
        picker.set_popupDirection(Telerik.Web.RadDatePickerPopupDirection.TopLeft);
        picker.showPopup();
    }
    else {
        var popupElement = picker.get_timePopupContainer();
        var dimensions = picker.getElementDimensions(popupElement);
        var position = picker.getElementPosition(textBox);
        picker.set_popupDirection(Telerik.Web.RadDatePickerPopupDirection.TopLeft);
        picker.showTimePopup();
    }
    logTitle("PopupAbove() end ");
    log("x = " + position.x +
        ", y = " + position.y +
        ", dimensions.height = " + dimensions.height, 2);
}
//...
<telerik:RadDateTimePicker ID="RadDateTimePicker1"
```

```

runat="server" Culture="(Default)">
<TimeView Culture="(Default)" runat="server">
</TimeView>
<Calendar runat="server">
</Calendar>
</telerik:RadDateTimePicker>
<button onclick="javascript: popupAbove(true);return false;" style="width: 150px;">
  Popup Calendar above</button>
<button onclick="javascript: popupAbove(false);return false;" style="width: 150px;">
  Popup TimeView above</button>

```

Date and Time Picker Events

The date and time picker controls let you respond to events:

- **PopupOpening** and **PopupClosing** provide an opportunity to call `args.set_cancel()` to prevent the popup from displaying or closing. Get a reference to the time view or calendar client object using `args.get_popupControl()`. In the **PopupOpening** event you can use `args.set_cancelCalendarSynchronization()` to prevent the popup control from synchronizing its value to the value in the input area.
- **DateSelected** is called immediately after the value of the control's selection has changed. You have access to the old and new dates as both `Date` objects and strings using `get_oldDate()`, `get_newDate()`, `get_oldValue()` and `get_newValue()`.

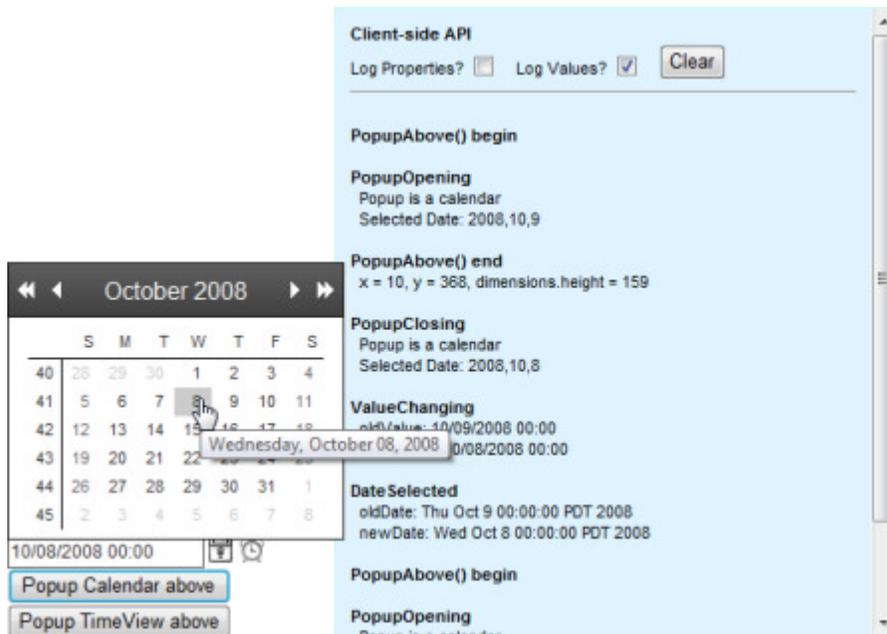
You can also get a reference to one of the child elements of the picker and attach events. For example, in the `pageLoad` event (available automatically whenever you have a `ScriptManager` on the page) you could get a reference to a picker's date input client object and attach a `ValueChanging` event:

[JavaScript] Assigning Events to Child Elements

```

function pageLoad(sender, args) {
  // get a reference to the date time picker's
  // internal date input control and assign the ValueChanging client event
  var RadDateTimePicker1 = $find("<%=RadDateTimePicker1.ClientID%>");
  var input = RadDateTimePicker1.get_dateInput();
  input.add_valueChanging(valueChanging);
}

```



Calendar Client API

The RadCalendar client object represents dates as triplets. Each triplet is an array of three integer values, which specify the year, month, and day of a date, in that order. To select a date, construct one of these triplets. You can populate the array using a Date object's `getFullYear()`, `getMonth()` and `getDate()` methods. The `selectDate()` method takes the triplet array as a parameter. The second parameter to `selectDate()` if true, navigates to the view containing the newly selected date.

[JavaScript] Selecting a Single Day

```
function SelectToday()
{
    var todaysDate = new Date();
    var todayTriplet =
        [todaysDate.getFullYear(), todaysDate.getMonth()+1, todaysDate.getDate()];
    var calendar = $find("<%=RadCalendar1.ClientID%>");
    calendar.selectDate(todayTriplet, true);
}
```

You can also navigate the view to a given date using the `navigateToDate()` method:

```
function GoToSummerOfLove()
{
    var triplet = [1968, 6, 1];
    var calendar = $find("<%=RadCalendar1.ClientID%>");
    calendar.set_rangeMinDate([1960, 1, 1]);
    calendar.navigateToDate(triplet);
}
```

To select multiple days, construct an array that contains triplet arrays. The next example creates two triplet arrays, then folds both of these into yet another array. The `selectDates()` method takes this last array as a parameter.

[JavaScript] Selecting Multiple Days

```
function selectTodayAndTomorrow() {
    var todaysDate = new Date();
    var todayTriplet =
```

```
[todaysDate.getFullYear(), todaysDate.getMonth() + 1, todaysDate.getDate()];
var tomorrowTriplet =
    [todaysDate.getFullYear(), todaysDate.getMonth() + 1, todaysDate.getDate() + 1];
var selectedDates = [todayTriplet, tomorrowTriplet];
var calendar = $find("<%=RadCalendar1.ClientID%>");
calendar.selectDates(selectedDates, true);
}
```

To get the selected date or dates on a RadCalendar control, use the `get_selectedDates()` method. This methods returns an array. If no date is selected, the array has length 0. If the calendar does not support multi-select mode, then the array has at most one element.

Here's an example that fires on the picker's client DateSelected event and de-selects any selections made in prior years to "today". The `get_selectedDates()` function gets an array of dates that are iterated and each date is decomposed to their year/month/day triplets and compared against the current year. The RadCalendar `unselectDate()` method takes a Date object as a parameter and removes the selection for that date.

[JavaScript] Getting Selected Dates

```
function dateSelected(sender, args) {
    var calendar = $find("<%= RadCalendar1.ClientID %>");
    var dates = calendar.get_selectedDates();
    var today = new Date();
    for (var i = 0; i < dates.length; i++) {
        var date = dates[i];
        var year = date[0];
        var month = date[1];
        var day = date[2];
        if (year < today.getYear())
            calendar.unselectDate(date);
    }
}
```

You can also unselect a number of dates at one time using the `unselectDates()` method. Here's a snippet that unselects all the dates in the calendar:

```
calendar.unselectDates(calendar.get_selectedDates());
```

Calendar Events

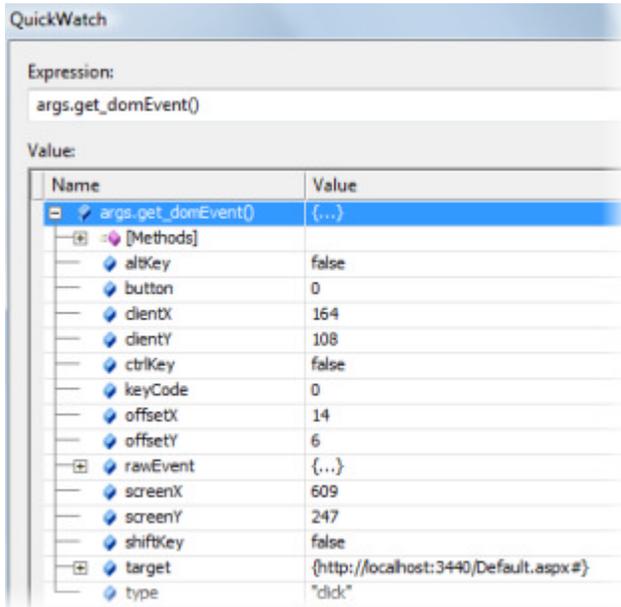
There are three events that fire when a date is clicked and fire in this order, `OnClick`, `OnDateSelecting` and `OnDateSelected`. Both `OnClick` and `OnDateSelecting` can be cancelled using the the `set_cancel()` method.

S	M	T	W	T	F	S
40	28	29	30	1	2	3
41	5	6	7	8	9	10
42	12	13	14	15	16	17
43	19	20	21	22	23	24
44	26	27	28	29	30	31
45	2	3	4	5	6	7

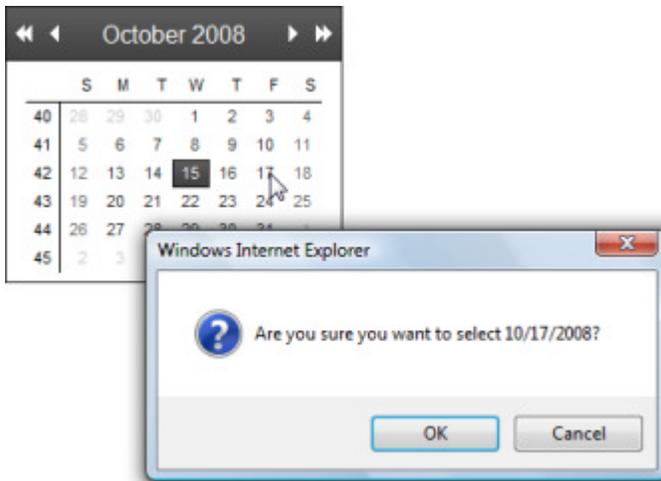
Event	Methods
DateClick	set_cancel get_cancel get_renderDay get_domEvent
DateSelecting	set_cancel get_cancel get_renderDay get_isSelecting
DateSelected	get_renderDay

UI for ASP.NET AJAX

- OnDateClick occurs when the user clicks on a date in the calendar (regardless of whether the date can be selected). The arguments surface the set_cancel() just mentioned, but also a **get_domEvent()** method that provides access to a lot of mouse information like the clientX and clientY coordinates:



OnDateClick arguments also include the **get_renderDay()** method returns the client that represents the clicked day. Here's an example that pops up a confirmation dialog when the date is clicked. Notice the **DateTimeFormatInfo** object being used here to format the date into its readable form:



[JavaScript] Handling the DateClick Event

```
function dateClick(sender, args) {  
    logEvent("DateClick", args);  
    var day = args.get_renderDay();  
    if (day.get_isSelectable()) {  
        var date = day.get_date();  
        var dfi = sender.DateTimeFormatInfo;  
        var formattedDate = dfi.FormatDate(day.get_date(), dfi.ShortDatePattern);  
        var confirmClick = confirm("Are you sure you want to " +
```

```

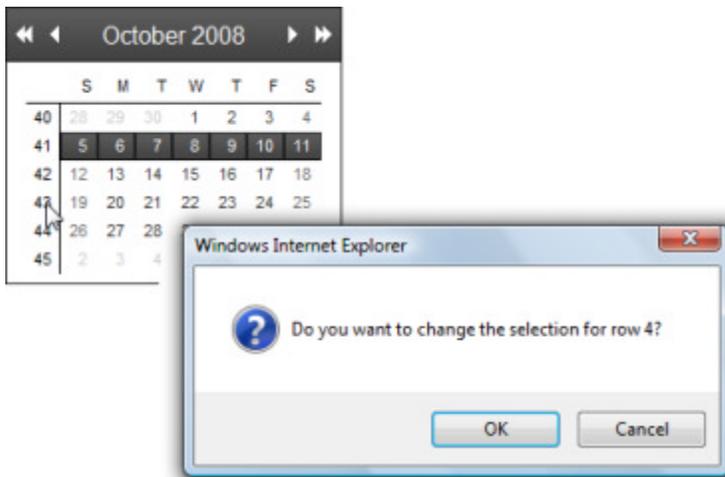
    (day.get_isSelected() ? "unselect " : "select ") + formattedDate + "?");
    args.set_cancel(!confirmClick);
}
}

```

- OnDateSelecting fires immediately before the selected dates collection is updated to reflect the selection or de-selection of a date. This event has arguments that include set_cancel(), get_renderDay() and one new method **get_isSelecting()** which is true if the day is in the process of being selected and false if the day is being de-selected.
- OnDateSelected fires immediately after the value of the control's selection has been changed. The arguments for this event only surface get_renderDay().

There are a few calendar methods that respond to selector, row and column clicks: **OnViewSelectorClick**, **OnRowHeaderClick** and **OnColumnHeaderClick**. All three events can be canceled with the args.set_Cancel(). All three events also surface the get_domElement(). OnRowHeaderClick and OnColumnHeaderClick have an additional method, get_index() that returns the 1-based index of the row or column clicked.

This next example shows an OnRowHeaderClick event handler implementation that displays a confirmation dialog:



[JavaScript] Handling the OnRowHeaderClick Event

```

function rowHeaderClick(sender, args) {
    var msg = "Do you want to change the selection for row " + args.get_index();
    var title = args.get_domElement().title;
    if (title != "")
        msg = msg + " (" + title + ")";
    msg = msg + "?";
    args.set_cancel(!confirm(msg));
}

```



You can find the complete source for this project at:
 \VS Projects\DateTimeSchedule\Calendar_ClientSide

43.9 Getting Started with RadScheduler

For RadScheduler to do its work, it must be bound to data. At a minimum, the data for the scheduler must include an **ID**, a **subject**, a **start time**, and an **end time**. Optionally, you can include fields for handling appointment recurrence. You can also include fields for custom resources and attributes. If you include custom resources, there must also be additional data binding to supply the scheduler with the possible values for each

custom resource type.

This walk-through will set up the scheduler with the minimum configuration to handle appointments. Later we will build on this to add resources and custom attributes.

1. In a new web application add a RadScheduler to the default page.
2. Open the Smart Tag and select **Add ScriptManager** from the context menu.
3. Also from the Smart Tag select **Choose Data Source...**

This step will display the Data Source Configuration Wizard dialog.

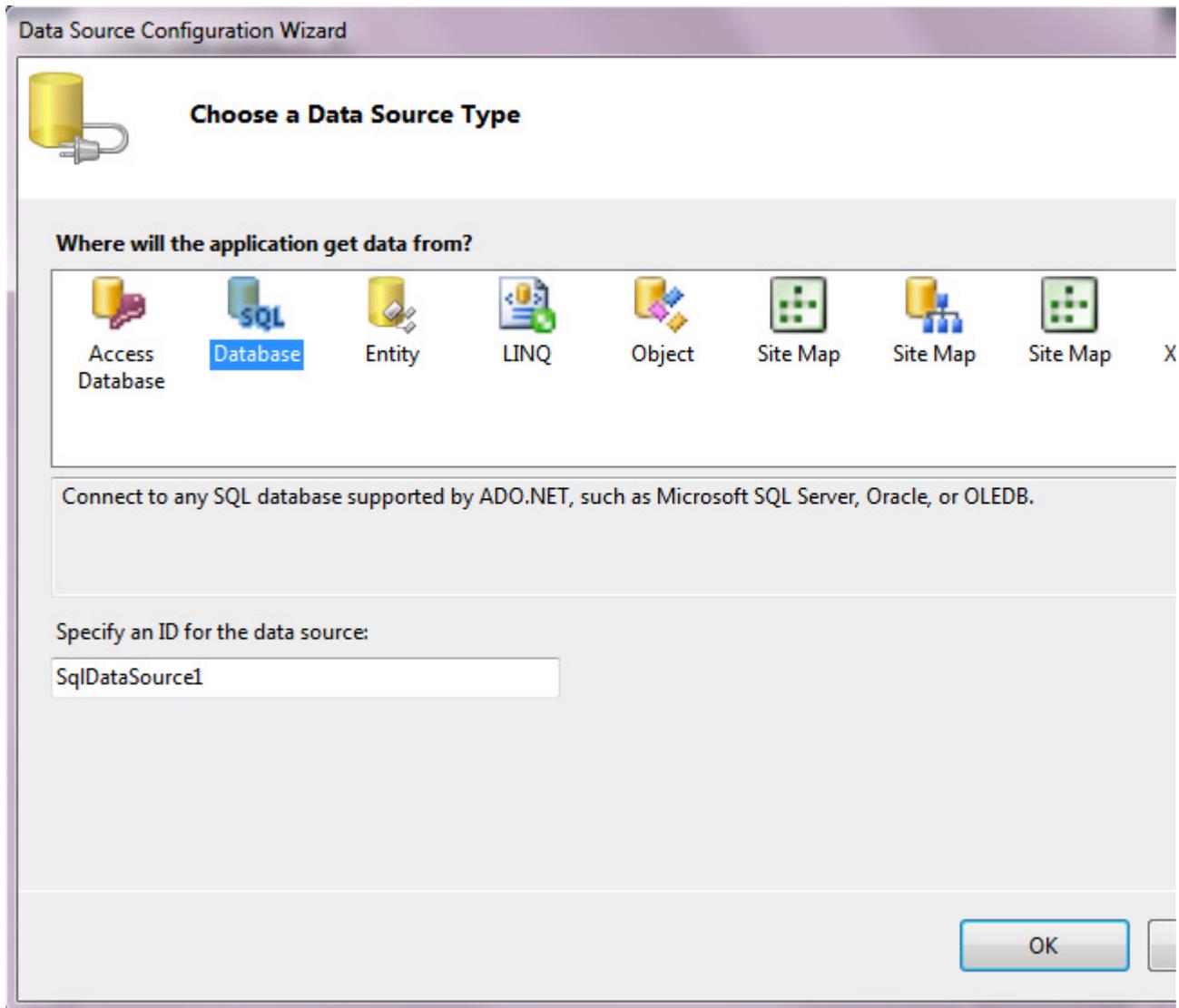
4. Drop down the **Select a datasource** and select **<New data source...>** from the list.

The screenshot shows the "Data Source Configuration Wizard" dialog box. It features a title bar with a close button (X). The main content area contains several sections, each with a dropdown menu and a label:

- Select a datasource:** The dropdown menu is open, showing options: "<New data source...>", "(None)", and "<New data source...>".
- Select a data field to be used as Start Time:** An empty dropdown menu.
- Select a data field to be used as End Time:** An empty dropdown menu.
- Select a data field to be used as Subject:** An empty dropdown menu.
- Select a data field to be used as Description:** An empty dropdown menu with the text "(optional)" to its right.
- Select a data field to be used for reminder storage:** An empty dropdown menu with the text "(optional)" to its right.
- Select a data field which will be used for recurrence storage:** An empty dropdown menu with the text "(optional)" to its right.
- Select a data field which will be used to store the recurrence parent key:** An empty dropdown menu with the text "(optional)" to its right.

At the bottom right of the dialog, there are two buttons: "OK" and "Cancel".

5. In the "Choose a Data Source Type" page of the wizard, select the **Database** icon. Click the **OK** button to continue.



*This step will display the **Configure Data Source** dialog.*

6. In the "Choose Your Data Connection" page of the wizard, click the **New Connection...** button.

This step will display the "Add Connection" dialog

7. In the *Add Connection* dialog, click the **Data Source Change...** button.

*This step will display the **Change Data Source** dialog.*

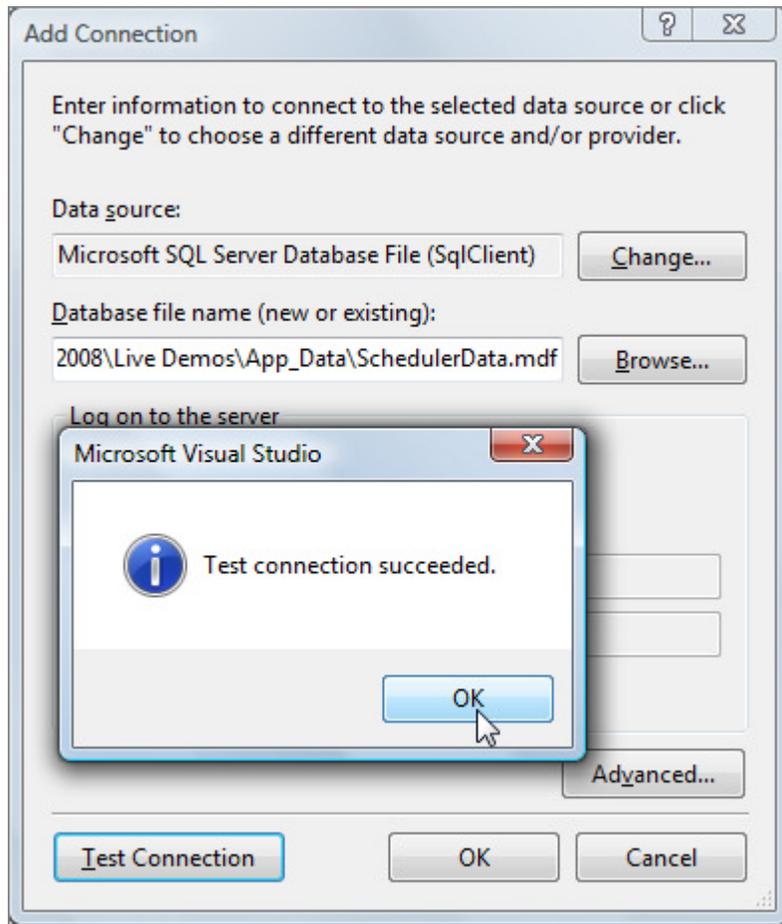
8. In the *Change Data Source* dialog, select the "Microsoft SQL Server Database File" data source option and click **OK** to close the dialog.

*This step will return you to the **Add Connection** dialog.*

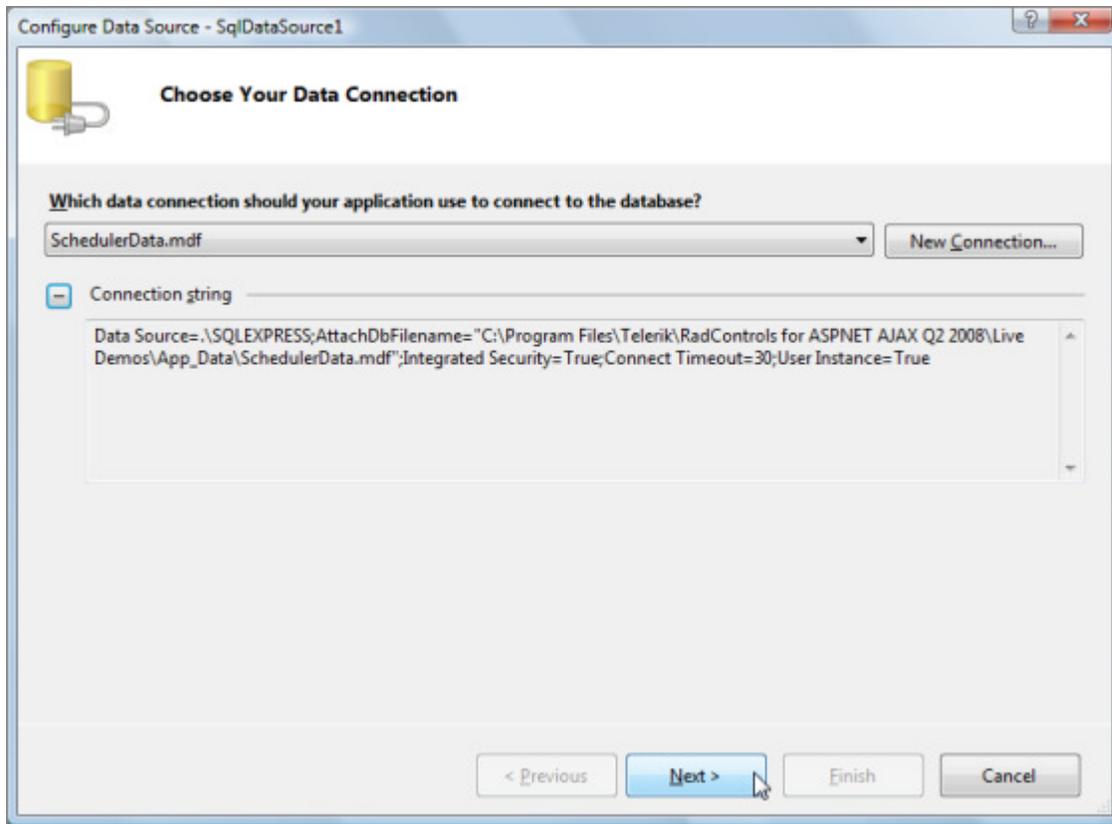
9. Configure the connection:
 - Click the **Browse...** button next to the "Database file name" entry.
 - Navigate to the RadControls installation directory and select "SchedulerData.mdb" from the \Live Demos\App_Data directory. In a typical installation the path will be \Program Files\Telerik\RadControls for ASPNET AJAX<version>\Live Demos\App_Data. This step should return you to the *Add Connection*

dialog. The dialog should look something like the example screenshot below.

- You can click the **Test Connection** button to verify that your connection string will be good.
- Click **OK** to close the Add Connection dialog.



10. Back in the Configure Data Source dialog you should see the Telerik.mdf connection. The dialog should look something like the screenshot below. Click the **Next** button to continue.

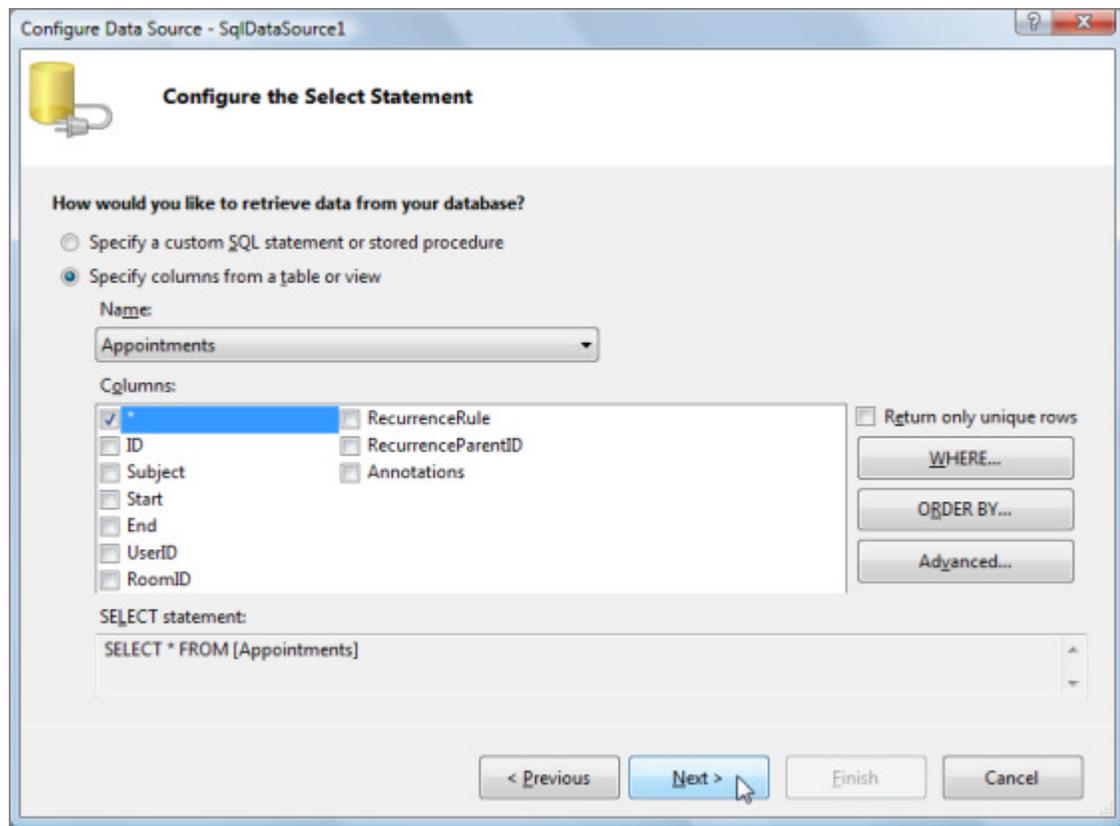


11. In the "Save the Connection String to the Application Configuration File" page of the wizard, click the **Next** button to continue.
12. In the "Configure the Select Statement" page of the wizard:
 - o The "Appointments" table should already be selected (if it's not, please do so now).
 - o Click the checkbox to select all columns of the Appointments table.
 - o Click the **Advanced** button, select the "Generate INSERT, UPDATE and DELETE" statements checkbox and click **OK**.

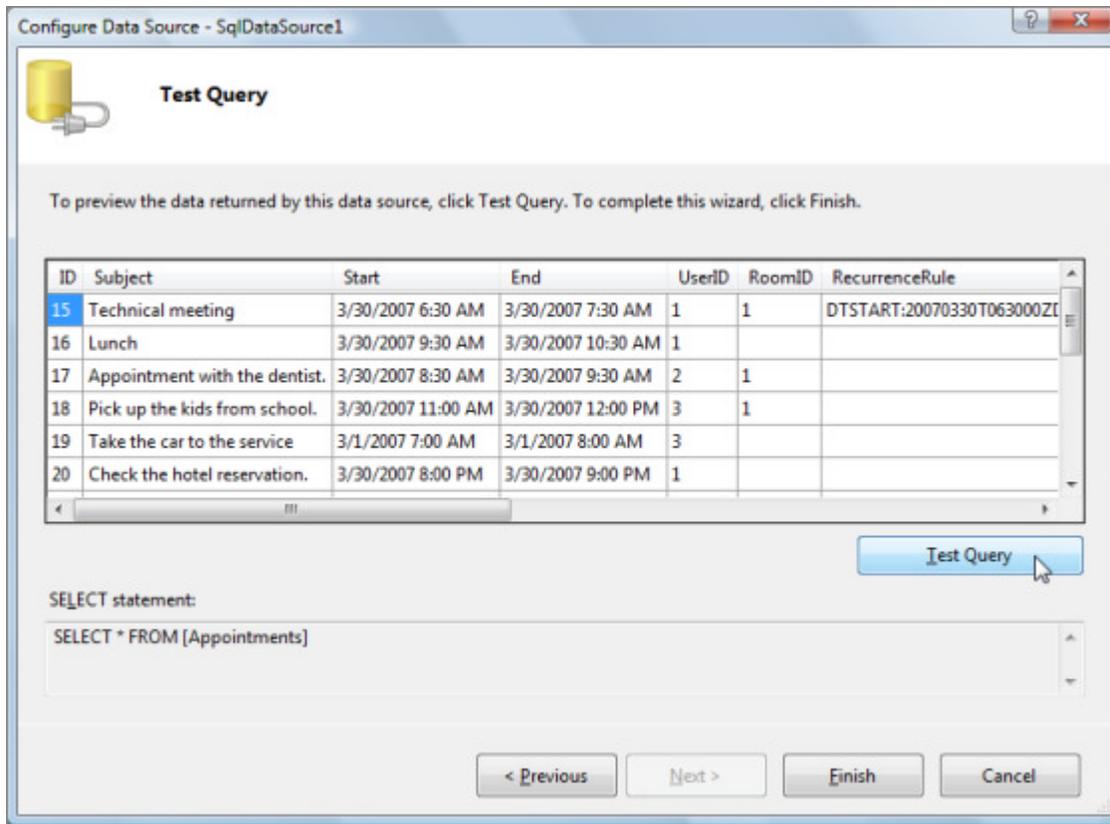


Gotcha! If you forget this last step where the CRUD (create-read-update-delete) statements are created, the SqlDataSource will not have SQL to handle any of the database operations other than simple selection. You would still see the data displayed, you could still double-click a day cell in the scheduler to insert a new appointment, but nothing would be saved.

- o Click the **Next** button to continue.



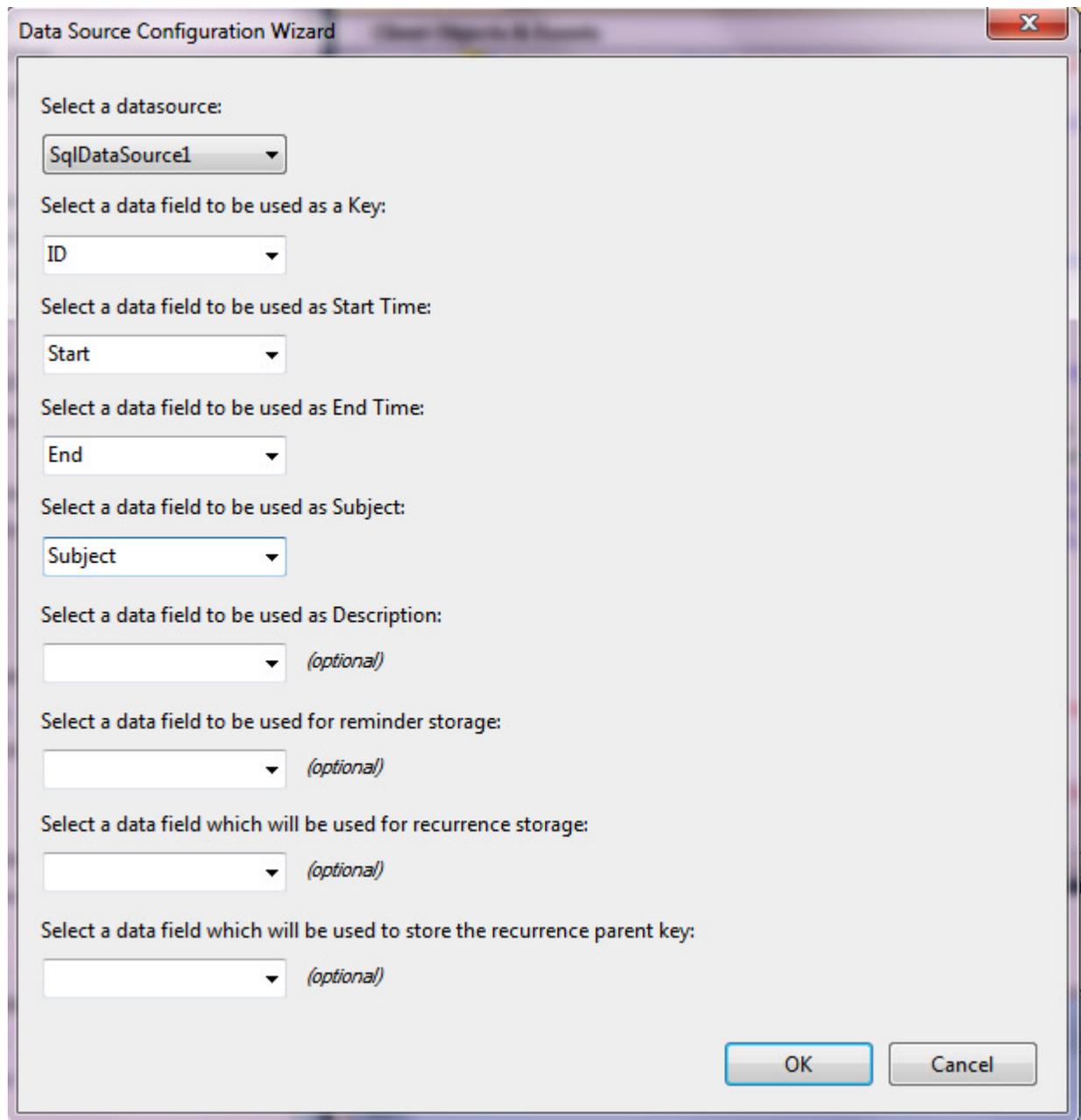
13. Click the **Test Query** button and check out the data we will be hooking up to the scheduler.



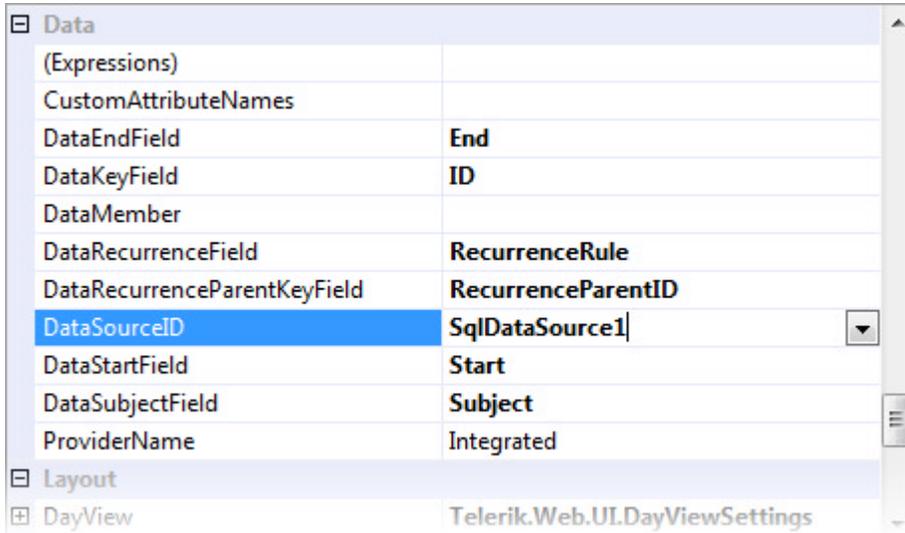
14. Click the **Finish** button to close the wizard.

This last step brings you back to the Data Source Configuration Wizard.

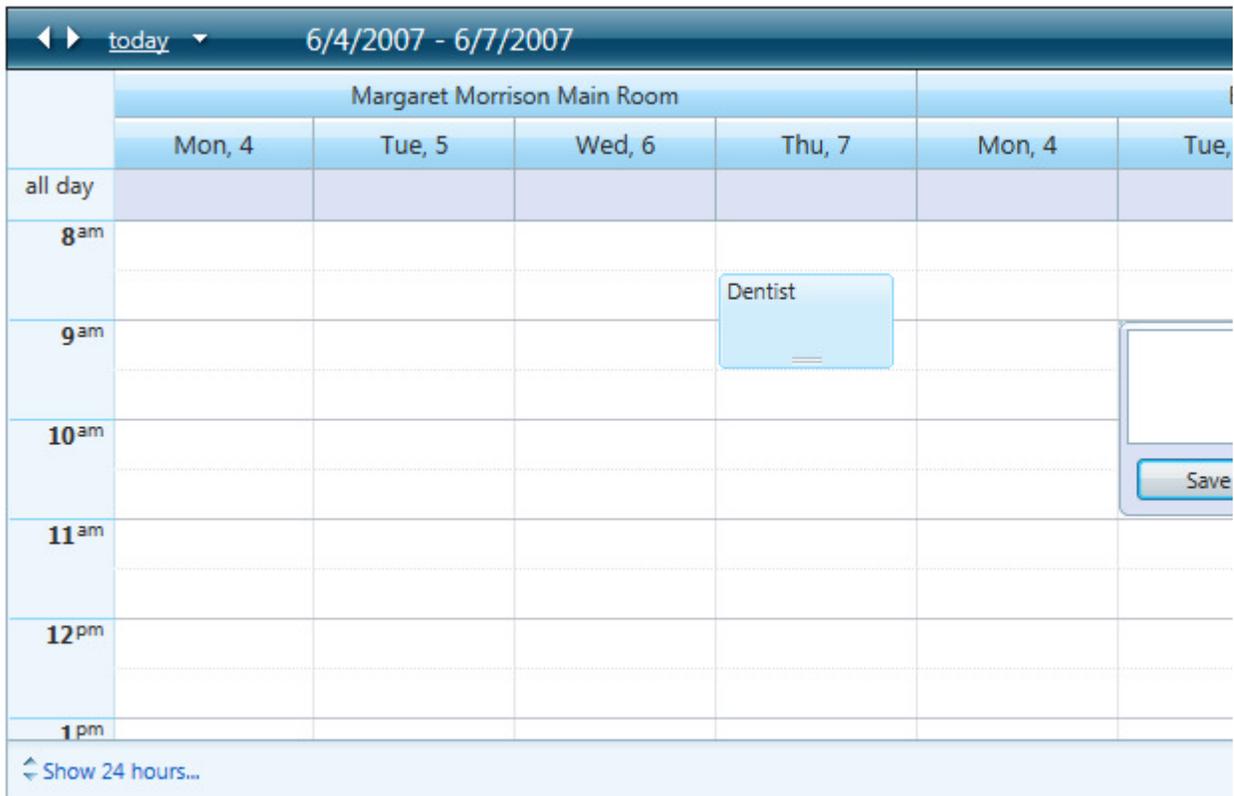
15. In the Data Source Configuration Wizard, use the drop down lists to match up data fields to the columns in the Appointments table. The key field should be set to the "ID" column. The remaining columns should be set to:
- Start
 - End
 - Subject
 - Description (optional)
 - Reminder (optional)
 - RecurrenceRule (optional)
 - RecurrenceParentID (optional)



16. Click the **OK** button to close the Data Source Configuration Wizard.
17. Check the Properties Window for the scheduler. Notice the Data category properties that have been populated by the wizard.



18. That's all there is to hooking up the basic properties of RadScheduler. We will take an extra step to position the scheduler to some data we know is already in the table. Set the **SelectedDate** property to "3/1/2007" and the **SelectedView** property to **MonthView**.
19. Press **Ctrl-F5** to run the application.
20. Experiment with the scheduler usability:
 - o Double-click a day cell to add a new appointment. Go to "Options" and then add the appointment.



New Appointment

Subject

Start time End time All day

Room:

Recurrence

Hourly
 Daily
 Weekly
 Monthly
 Yearly

No end date End after occurrences End by

- Double-click an existing appointment to update the appointment:

Edit Appointment

Subject:

Start time: End time: All day

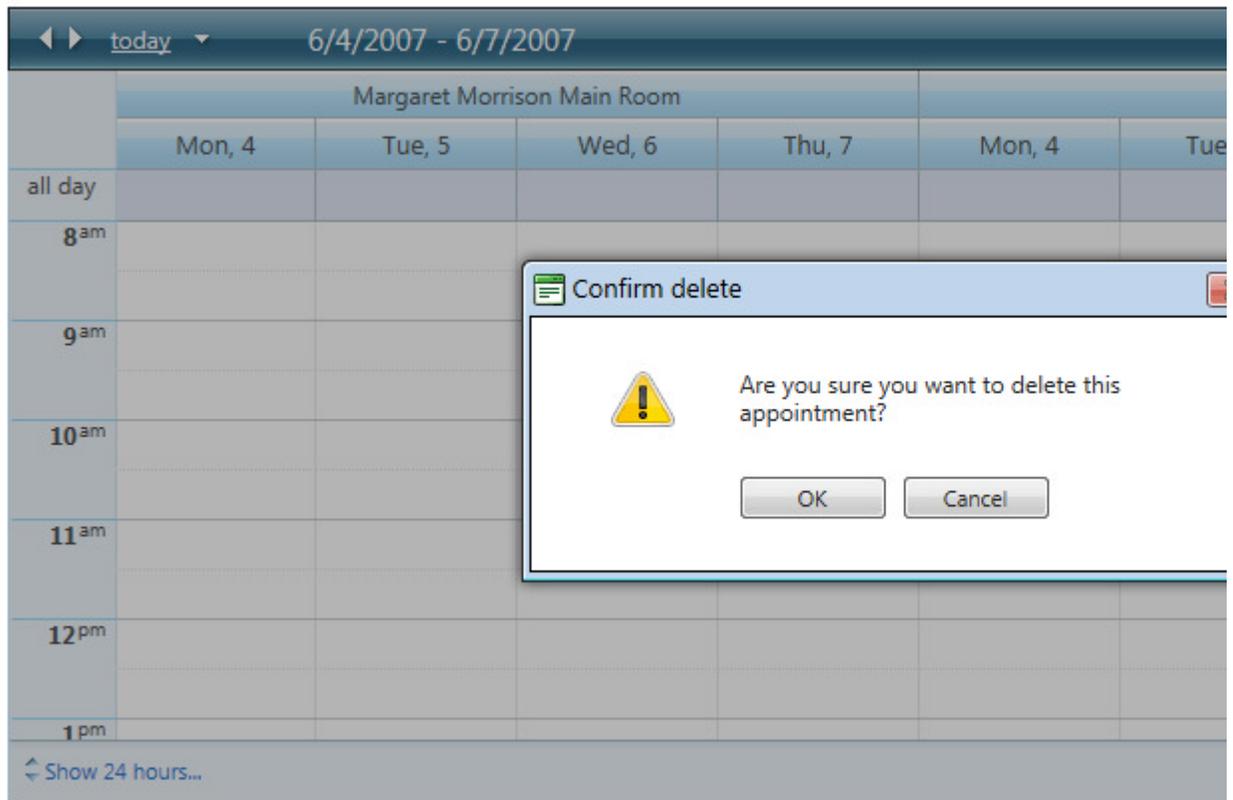
Room:

Recurrence

Hourly
 Daily
 Weekly
 Monthly
 Yearly

No end date End after occurrences End by

- Try clicking the red "X" to delete an existing appointment.



- Notice that you can drag existing appointments to any visible day.

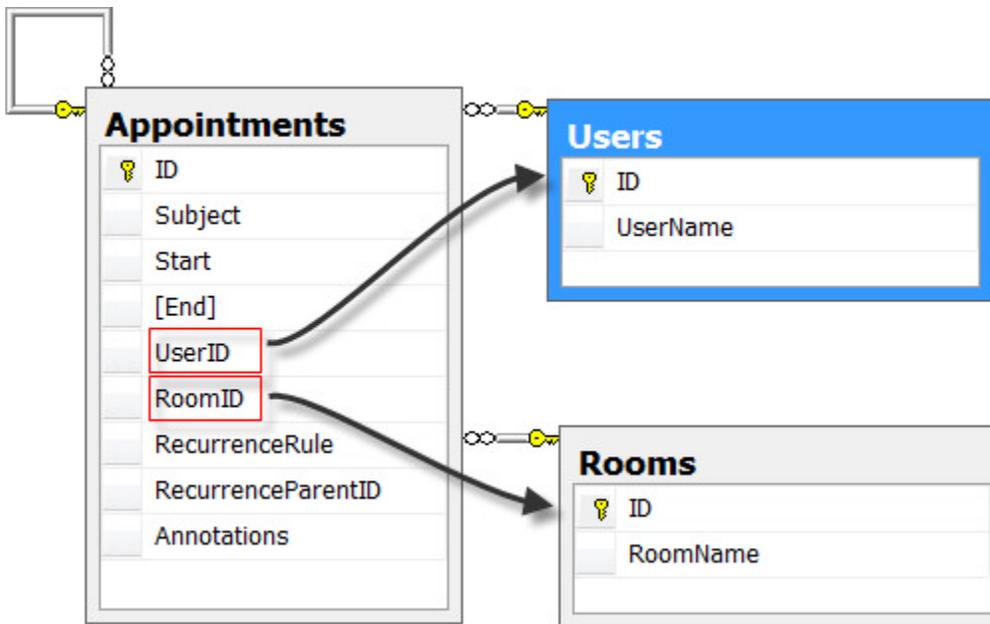


You can find the complete source for this project at:
\\VS Projects\DateTimeSchedule\GettingStarted_Scheduler

43.10 Scheduler Resources

Multiple arbitrary resources can be assigned to any appointment. For example, resources might be people, rooms, equipment or any random items you want to associate with an appointment. To make this work you need an ID for the resource in the appointment table (e.g. "UserID") and a separate master table with columns for an ID and a text value to describe the resource.

In this database diagram you can see the Appointments table from the Telerik.mdf demo file.



We can extend the scheduler "getting started" example by adding the users and rooms resources.

1. Add a SqlDataSource to the default page. Configure the data source to use the Users table:
 - Set the ID to "dsUsers".
 - Use the Smart Tag to configure the data source. In the Configure Data Source Wizard, re-use the SchedulerDataConnectionString.
 - Select both columns.
 - Test the query if you like and finish the wizard.
2. Add another SqlDataSource to the default page. Configure the data source to use the Rooms table:
 - Set the ID to "dsRooms".
 - In the Configure Data Source Wizard, re-use the SchedulerDataConnectionString.
 - Select both columns.
 - Test the query if you like and finish the wizard.
3. Using the RadScheduler Smart Tag, click the Resource Types ellipses.

RadScheduler Tasks

[Choose Data Source ...](#)

Resource Types: (Collection) [...]

Skin: Default [v]

Learning center:

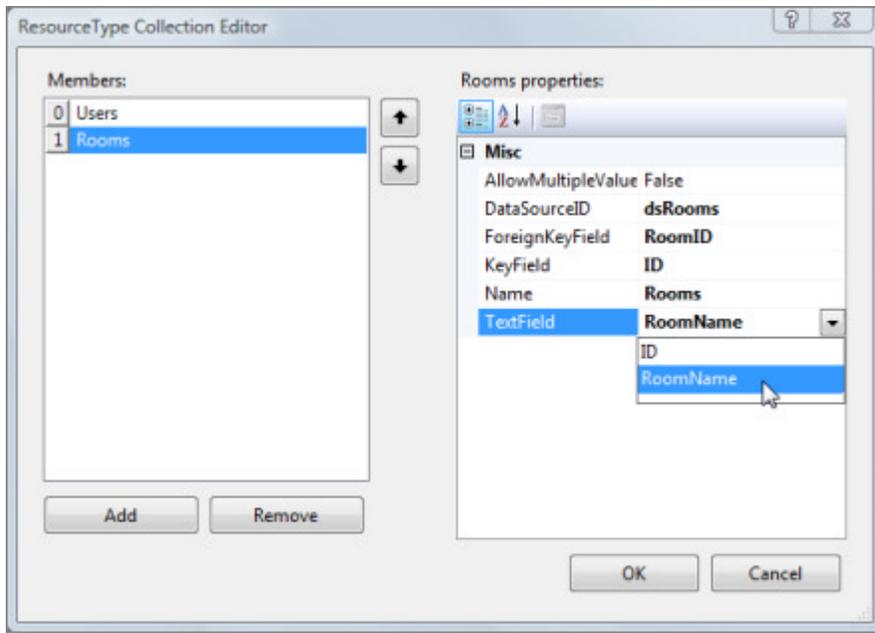
- [Online RadScheduler for ASP.NET Ajax examples](#)
- [Online RadScheduler for ASP.NET Ajax help](#)
- [Online RadControls for ASP.NET Ajax Code Library](#)

Search [www.telerik.com](#) for:

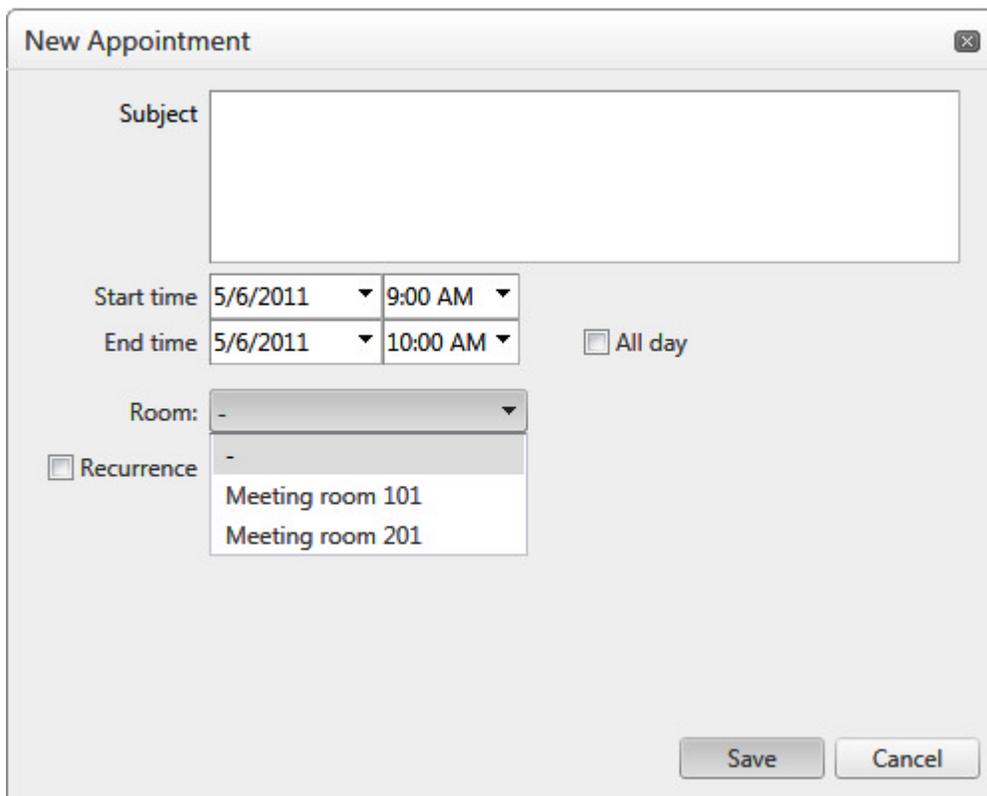
[Search](#)

[Go to online Telerik support center](#)

4. Click the **Add** button to create a new resource. Set the properties:
 - o **Name:** "Users"
 - o **DataSourceID:** "dsUsers"
 - o **ForeignKeyField:** "UserID"
 - o **KeyField:** "ID"
 - o **TextField:** "UserName"
5. Click the **Add** button to create a second resource. Set the properties:
 - o **Name:** "Rooms"
 - o **DataSourceID:** "dsRooms"
 - o **ForeignKeyField:** "RoomID"
 - o **KeyField:** "ID"
 - o **TextField:** "RoomName"



- Press **Ctrl-F5** to run the application. Try creating a new appointment and clicking the "more" link or double-click an existing appointment. The "More details" section of the update dialog now displays drop down lists for each resource.



You can find the complete source for this project at:
 \VS Projects\DateTimeSchedule\Scheduler_Resources



43.11 Custom Attributes

You can attach custom data from one or more columns in your appointment table. To do this add to the column names to the **CustomAttributeNames** collection property. Also set the **EnableCustomAttributeEditing** property to true. The Appointments table in the SchedulerData.mdb demonstration data file contains a "Annotation" column. The custom attributes get added to the "More details" section of edit and insert forms as shown in the screenshot below.

A screenshot of a web form titled "New Appointment". The form contains several input fields: a text box for "Subject", two date and time pickers for "Start time" (5/6/2011, 8:30 AM) and "End time" (5/6/2011, 9:30 AM), a checkbox for "All day", a dropdown menu for "Reminder" (set to "None"), a text area for "Annotations", a "Users" section with checkboxes for "Alex", "Bob", and "Charlie", a dropdown menu for "Rooms" (set to "-"), and a large text area for "Description".

1. Start with the Scheduler_Resources project.
2. In the Properties Window for the scheduler, click the **CustomAttributeNames** property ellipses. Add the attribute "Annotations" to the string collection editor and click **OK**.
3. Set the **EnableCustomAttributeEditing** property to "true".
4. Press **Ctrl-F5** to run the application. Insert an appointment and click the "more" link or edit an existing appointment. Notice that the custom attribute "Annotation" in the "Details" portion of the edit form. Verify that you can save and re-display the value.

43.12 Scheduler Designer Interface

In the Visual Studio designer, you can configure the **RadScheduler** using the Smart Tag or the Properties Window.

Smart Tag

The RadScheduler Smart Tag contains a few control-specific entries in addition to the standard Ajax Resources, Skin selection, and Learning center sections. The **Choose Data Source...** lets you define the data source and map the appointment table columns to the scheduler. After the data source is assigned the **Resource Types** collection editor becomes available.

Properties

Position the initial date shown in the scheduler by setting the **SelectedDate** property. Likewise you can set the scheduler view; that is, have the scheduler display the day, week, month or a timeline view using the **SelectedView** property. You can also set the visible range of days and times using the properties **FirstDayOfWeek**, **LastDayOfWeek**, **DayStartTime**, **DayEndTime**, **WorkDayStartTime** and **WorkDayEndTime**.

Control how the scheduler fits into the page real-estate by setting the **OverflowBehavior** property to **Expand** or **Scroll** (default).

Toggle visibility for major user interface elements using **ShowAllDayRow**, **ShowDateHeaders**, **ShowFooter**, **ShowFullTime**, **ShowHeader**, **ShowHoursColumn**, **ShowMonthlyColumnHeader**, **ShowNavigationPane**, **ShowResourceHeaders**, **ShowViewTabs**, and **ShowWeeklyColumnHeader**.

By default you can add, edit and delete appointments but you can turn this ability off using the **AllowInsert**, **AllowEdit** and **AllowDelete** properties.

Data

As we saw earlier in the "Getting Started" section of this chapter, the Data properties let you define the data source and map all the appointment table columns to scheduler-specific data properties.

Data	
(Expressions)	
CustomAttributeNames	Annotation
DataEndField	End
DataKeyField	ID
DataMember	
DataRecurrenceField	RecurrenceRule
DataRecurrenceParentKeyField	RecurrenceParentID
DataSourceID	SqlDataSource1
DataStartField	Start
DataSubjectField	Subject
ProviderName	Integrated

We have used all of these properties so far except the **ProviderName**. By default **ProviderName** is "Integrated", but you can use one of the providers supplied in the Telerik.Web.UI assembly, or you can implement your own.

See the online help for examples of **creating custom providers** (http://www.telerik.com/help/aspnet-ajax/schedule_databindingusingadataprovider.html) and a reference implementation of an **exchange provider** (<http://www.telerik.com/help/aspnet-ajax/data-binding-exchange-provider.html>) to handle Outlook integration.

To bind RadScheduler to a provider, set its **Provider** or **ProviderName** property. Use the **ProviderName** property when binding declaratively in the designer, and the **Provider** property when binding to a provider instance at runtime. Because providers supply information about appointments using the Telerik.Web.UI.Appointment type, you do not need to set the scheduler's **DataKeyField**, **DataSubjectField**, **DataStartField**, **DataEndField**, **DataRecurrenceField**, **DataRecurrenceParentKeyField** or **ResourceTypes** properties.

If you need to assign multiple resources *of the same type* to an appointment you must use a provider.

Layout

You will find "View" properties in the Layout group of properties. Each view has common sub-properties.

Layout	
DayView	Telerik.Web.UI.DayViewSettings
DayEndTime	18:00:00
DayStartTime	08:00:00
GroupBy	
GroupingDirection	Horizontal
HeaderDateFormat	D
ReadOnly	False
ShowHoursColumn	True
ShowResourceHeaders	True
UserSelectable	True
WorkDayEndTime	17:00:00
WorkDayStartTime	08:00:00
GroupBy	
GroupingDirection	Horizontal
Height	
MonthView	Telerik.Web.UI.MonthViewSettings
MultiDayView	Telerik.Web.UI.MultiDayViewSettings
SelectedView	MultiDayView
TimelineView	Telerik.Web.UI.TimelineViewSettings
WeekView	Telerik.Web.UI.WeekViewSettings
Width	

Set the **GroupBy** property to the name of a resource type and **GroupingDirection** to either **Horizontal** or **Vertical**. This screenshot shows the **TimelineView** **GroupBy** set to "Rooms" and the **GroupingDirection** as **Horizontal**.

Meeting room 101		Meeting room 201				Day	W
5/6/2011	5/7/2011	5/8/2011	5/6/2011	5/7/2011	5/8/2011	5/6/2011	5/7/2011

ShowResourceHeaders can be disabled to hide the column titling "Meeting room 101", "Meeting room 201" but typically you would want to leave this at the default of "true". **UserSelectable** toggles the visibility of each view in the view selector located at the upper right of the scheduler.

There are also view-specific sub-properties that handle start and end times, the number of units of whatever time is being shown (days, hours, etc), heading formats and that toggle visibility of specific UI elements for each view.

Localization Properties

Localization works in a similar manner to RadEditor in that you can define a resource file with the naming convention `RadScheduler.Main.<Culture Identifier>.resx`. Typically you would copy the existing resource file from the RadControls installation directory (usually on this path: `"\Program Files\Telerik\RadControls for ASPNET AJAX<version>\Live Demos\App_GlobalResources"`) to your project. After making changes to the resource file you set the Culture property to the same culture identifier you used in naming the resource file, e.g. "fr-FR".

You can also use the **Localization** property. The Localization property has sub-properties that define the strings displayed in each UI element of the scheduler.

Localization	Telerik.Web.UI.SchedulerStrings
AdvancedAllDayEvent	All day event
AdvancedCalendarCancel	Cancel
AdvancedCalendarOK	OK
AdvancedCalendarToday	Today
AdvancedDaily	Daily
AdvancedDay	Day

43.13 Scheduler Server-Side Programming

To add an appointment programmatically, you can create an Appointment object with one of several overloaded constructors available. Then add the appointment object using the RadScheduler **InsertAppointment()** method.

Making the appointment recurring is only slightly more complex. You will need to populate the appointment object's **RecurrenceRule** property with a RecurrenceRule descendent, namely HourlyRecurrenceRule, DailyRecurrenceRule, MonthlyRecurrenceRule or YearlyRecurrenceRule.

In the example below the HourlyRecurrenceRule constructor requires an integer "interval" value and a RecurrenceRange object instance. In this example the RecurrenceRange **Start** value is set to match the appointment start and the **EventDuration** is set to the difference between the appointment end and start times. Note: You can also use the RecurrenceRange **MaxOccurrences** property to control the end point of the range.

Adding Appointments Walk-Through

1. Set up a web application with the RadScheduler in the same manner as the "Resources" example.
2. Add a standard ASP.NET button, a standard checkbox and a RadTextBox below the scheduler. Set the control ID's to "btnAdd", "cbRecur" and "tbSubject", respectively.
3. Set the Text of the button to "Add Appointment".
4. Set the Text for the checkbox to "Repeat every hour?".
5. If you like, set the Skin properties of the RadControls to a common skin. You can also add a FormDecorator to style the checkbox and button to match. The example project uses the "WebBlue" skin.
6. Double-click the button and add the following code to the Click event handler (and the declaration to the "_id" member).

 Notice that the DateTime is converted to universal time before being stored by the Scheduler. RadScheduler has two helper methods, **DisplayToUtc()** and **UtcToDisplay()** for easily converting between the two formats.

[VB] Inserting New Appointments

```

' stores the current appointment id
Private _id As Integer = 1
Protected Sub btnAdd_Click(ByVal sender As Object, ByVal e As EventArgs)
' get a DateTime with the year day and hour only
Dim now As DateTime = DateTime.Now
Dim nextHour As DateTime
nextHour = New DateTime(now.Year, now.Month, now.Day, now.Hour, 0, 0).ToUniversalTime()
' create a new appointment that starts an hour from now and lasts 10 minutes,
' use the subject entered by the user to a text box
Dim appointment As New Appointment(_id, nextHour, nextHour.AddMinutes(10), tbSubject.Te;
System.Math.Max(System.Threading.Interlocked.Increment(_id),_id - 1)
' If a checkbox is checked, make this a recurring appointment that
' starts an hour from now and recurs every hour
If cbRecur.Checked Then
Dim range As New RecurrenceRange()
range.Start = appointment.Start
range.EventDuration = appointment.[End] - appointment.Start
appointment.RecurrenceRule = New HourlyRecurrenceRule(1, range).ToString()
End If
' Insert the appointment into the schedulers list of appointments.
RadScheduler1.InsertAppointment(appointment)
End Sub

```

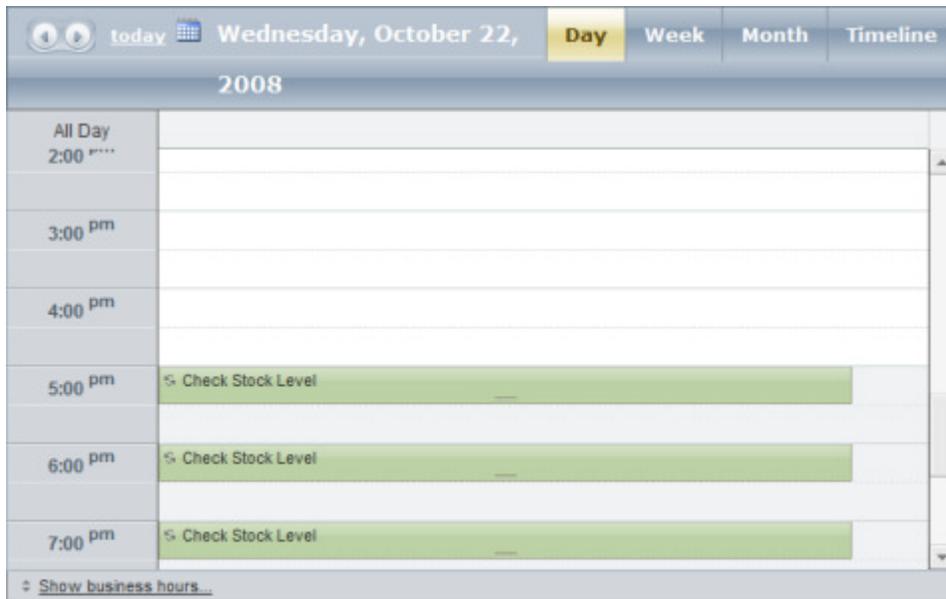
[C#] Inserting New Appointments

```

// stores the current appointment id
private int _id = 1;
protected void btnAdd_Click(object sender, EventArgs e)
{
// get a DateTime with the year day and hour only
DateTime now = DateTime.Now;
DateTime nextHour = new DateTime(now.Year, now.Month, now.Day, now.Hour, 0,
0).ToUniversalTime();
// create a new appointment that starts an hour from now and lasts 10 minutes,
// use the subject entered by the user to a text box
Appointment appointment =
new Appointment(_id, nextHour, nextHour.AddMinutes(10), tbSubject.Text);
_id++;
// If a checkbox is checked, make this a recurring appointment that
// starts an hour from now and recurs every hour
if (cbRecur.Checked)
{
RecurrenceRange range = new RecurrenceRange();
range.Start = appointment.Start;
range.EventDuration = appointment.End - appointment.Start;
appointment.RecurrenceRule = new HourlyRecurrenceRule(1, range).ToString();
}
// Insert the appointment into the schedulers list of appointments.
RadScheduler1.InsertAppointment(appointment);
}

```

7. Press **Ctrl-F5** to run the application. Add a subject line and click the "Add Appointment" button. Also try it with the "Repeat" checkbox selected.



Add Appointment Subject:
 Repeat every hour?



You can find the complete source for this project at:
\\VS Projects\DateTimeScheduler\Scheduler_Appointments



Recurrence Rules

When a user elects to make an appointment recurring, a record is set up in the database with a string representation of the recurrence data. The screenshot of the data below shows two records. The first is the master data for the recurring appointment. When the user first creates a recurring appointment, this is the record that is created. You can see the string representation of the recurrence including the start and end dates and the rule that defines when the appointment should recur. When the user decides to edit a single instance of the recurrence, an exception record is created with the `RecurrenceRule` set to `NULL` and the `RecurrenceParentID` pointing to the ID of the master record.

	Subject	RecurrenceRule	RecurrenceParentID
▶	Fly to Sofia	DTSTART:20081023T090000ZDTEND:20081023T100000ZRRULE:FRE...	NULL
	Fly to Sofia - exception	DTSTART:20081023T090000Z DTEND:20081023T100000Z RRULE:FREQ=MONTHLY;INTERVAL=1;BYMONTHDAY=1 EXDATE:20090201T090000Z	61

The `RecurrenceRule` also has two methods: `ToString()` to convert the rule to readable text as it will be stored in a database and `TryParse()` that reconstitutes the string back to a `RecurrenceRule` object.

Adding Resources Programmatically

You can expand the previous example to create and assign resources on-the-fly. First you need to add a `ResourceType` object when the page first loads. You can assign the `ResourceType` `Name` property in the constructor. Assign the other properties with the same values as when you created a `ResourceType` at design time. Finally, add the `ResourceType` object to the `RadScheduler` `ResourceTypes` collection.

[VB] Add a Resource Type

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)  
    If Not IsPostBack Then
```

```

Dim resourceType As New ResourceType("Rooms")
resourceType.DataSource = dsRooms
resourceType.ForeignKeyField = "RoomID"
resourceType.KeyField = "ID"
resourceType.TextField = "RoomName"
RadScheduler1.ResourceTypes.Add(resourceType)
End If
End Sub

```

[C#] Add a Resource Type

```

protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        ResourceType resourceType = new ResourceType("Rooms");
        resourceType.DataSource = dsRooms;
        resourceType.ForeignKeyField = "RoomID";
        resourceType.KeyField = "ID";
        resourceType.TextField = "RoomName";
        RadScheduler1.ResourceTypes.Add(resourceType);
    }
}

```

Then create and add a Resource object to the Appointment Resources collection before you insert the appointment. The Resource constructor takes the name of the ResourceType, a resource key and resource text.

[VB] Adding a Resource

```
appointment.Resources.Add(New Resource("Rooms", 1, "Resource text"))
```

[C#] Adding a Resource

```
appointment.Resources.Add(new Resource("Rooms", 1, "Resource text"));
```

43.14 Scheduler Server-Side Events

On the server-side, RadScheduler comes with a full set of events for responding to appointment changes, creation of advanced forms and time slots. Here are some of the significant groups of events surfaced by RadScheduler:

Responding to Appointment Events

If you need a simple click event to get started, use **AppointmentClick**. The SchedulerEventArgs passed to the handler include an Appointment object.

[VB] Handling the AppointmentClick Event

```

Protected Sub RadScheduler1_AppointmentClick(ByVal sender As Object, ByVal e As
Telerik.Web.UI.SchedulerEventArgs)
    RadAjaxManager1.Alert("You clicked: " + e.Appointment.Subject)
End Sub

```

[C#] Handling the AppointmentClick Event

```

protected void RadScheduler1_AppointmentClick(object
sender, Telerik.Web.UI.SchedulerEventArgs e)
{
    RadAjaxManager1.Alert("You clicked: " + e.Appointment.Subject);
}

```

When the scheduler first displays, the **AppointmentDataBound** events fire for each appointment, then the **TimeSlotCreated** events, once for each time slot. The number of times a time slot is created varies on the view

you have selected. For example, if you're looking at the Timeline view and it has only three cells, the TimeSlotCreated event only fires three times.

In the example below the AppointmentDataBound event uses a RoundToNearestHour() method (not shown here, but available in the demo Scheduler_Events project) that removes minutes and seconds from the appointment DateTime.

The TimeSlotCreated event handler looks for time slots with no appointments and changes the style for the time slot.

[VB] Handling the AppointmentDataBound and TimeSlotCreated Events

```
Protected Sub RadScheduler1_AppointmentDataBound(ByVal sender As Object, ByVal e As SchedulerEventArgs)
    Log("AppointmentDataBound")
    e.Appointment.Start = RoundToNearestHour(e.Appointment.Start)
End Sub
Protected Sub RadScheduler1_TimeSlotCreated(ByVal sender As Object, ByVal e As TimeSlotCreatedEventArgs)
    Log("TimeSlotCreated")
    If e.TimeSlot.Appointments.Count = 0 Then
        e.TimeSlot.CssClass = "OpenTimeSlots"
    End If
End Sub
```

[C#] Handling the AppointmentDataBound and TimeSlotCreated Events

```
protected void RadScheduler1_AppointmentDataBound(object sender, SchedulerEventArgs e)
{
    Log("AppointmentDataBound");
    e.Appointment.Start = RoundToNearestHour(e.Appointment.Start);
}
protected void RadScheduler1_TimeSlotCreated(object sender, TimeSlotCreatedEventArgs e)
{
    Log("TimeSlotCreated");
    if (e.TimeSlot.Appointments.Count == 0)
    {
        e.TimeSlot.CssClass = "OpenTimeSlots";
    }
}
```

You can handle your own appointment Insert, Update and Delete scheduler events. All three handlers have argument properties for the **Appointment** and **Cancel**. The AppointmentUpdate event handler arguments also has a **ModifiedAppointment** object so you get the "before" and "after" picture for the appointment. The example below puts some arbitrary limits on when these operations can occur and use the RadAjaxManager.Alert() method to report to the user.

[VB] Handling the AppointmentInsert, AppointmentUpdate and AppointmentDelete Events

```
Protected Sub RadScheduler1_AppointmentDelete(ByVal sender As Object, ByVal e As SchedulerCancelEventArgs)
    ' do not allow deleting past appointments
    e.Cancel = e.Appointment.Start < DateTime.Today
    If e.Cancel Then
        RadAjaxManager1.Alert("Cannot delete past appointments")
    End If
End Sub
Protected Sub RadScheduler1_AppointmentInsert(ByVal sender As Object, ByVal e As SchedulerCancelEventArgs)
```

```

' don't insert appointments with empty subjects
e.Cancel = e.Appointment.Subject = ""
If e.Cancel Then
    RadAjaxManager1.Alert("Cannot insert appointments with no subject")
End If
End Sub
Protected Sub RadScheduler1_AppointmentUpdate(ByVal sender As Object, ByVal e As
AppointmentUpdateEventArgs)
' don't allow an appointment with a start date in the future to be
' moved to the past
e.Cancel = (e.Appointment.Start >= DateTime.Today) AndAlso (e.ModifiedAppointment.Start <
DateTime.Today)
If e.Cancel Then
    RadAjaxManager1.Alert("Cannot move an appointment to the past")
End If
End Sub

```

[C#] Handling the AppointmentInsert, AppointmentUpdate and AppointmentDelete Events

```

protected void RadScheduler1_AppointmentDelete(object sender,
SchedulerCancelEventArgs e)
{
// do not allow deleting past appointments
e.Cancel = e.Appointment.Start < DateTime.Today;
if (e.Cancel)
{
    RadAjaxManager1.Alert("Cannot delete past appointments");
}
}
protected void RadScheduler1_AppointmentInsert(object sender,
SchedulerCancelEventArgs e)
{
// don't insert appointments with empty subjects
e.Cancel = e.Appointment.Subject == "";
if (e.Cancel)
{
    RadAjaxManager1.Alert("Cannot insert appointments with no subject");
}
}
protected void RadScheduler1_AppointmentUpdate(object sender,
AppointmentUpdateEventArgs e)
{
// don't allow an appointment with a start date in the future to be
// moved to the past
e.Cancel =
    (e.Appointment.Start >= DateTime.Today) &&
    (e.ModifiedAppointment.Start < DateTime.Today);
if (e.Cancel)
{
    RadAjaxManager1.Alert("Cannot move an appointment to the past");
}
}

```

Responding to Form Events

When one of the advanced forms is created to handle inserts and updates, two events fire. First the **FormCreating** event lets you cancel showing the advanced form and then **FormCreated** lets you make other

changes to the form and its controls.

In the example below the FormCreating event handler disallows advanced form editing for appointments with a start date less than today (i.e. past appointments). The `FormCreatingEventArgs` passed to FormCreating has properties for `Mode`, `Cancel` and `Appointment`. Mode is a `SchedulerFormMode` enumeration that lets you know if this is an Edit or Insert form that displays in-line within the scheduler or one of the advanced forms `AdvancedEdit` or `AdvancedInsert`.

The FormCreated event handler locates the "more" button on the "Insert" advanced form and alters the button appearance. The `FormCreatedEventArgs` has properties for `Appointment` and `Container`. Container represents the form. You can get at the controls using the `FindControl()` method.

[VB] Handling the FormCreating and FormCreated Events

```
Protected Sub RadScheduler1_FormCreating(ByVal sender As Object, ByVal e As SchedulerFormCreatingEventArgs)
    ' disallow advanced form editing for past appointments
    Dim preventEdit As Boolean = e.Mode = SchedulerFormMode.AdvancedEdit AndAlso e.Appointment.Start < DateTime.Today
    e.Cancel = preventEdit
    If preventEdit Then
        RadAjaxManager1.Alert("Advanced editing for past appointments not allowed")
    End If
End Sub
Protected Sub RadScheduler1_FormCreated(ByVal sender As Object, ByVal e As SchedulerFormCreatedEventArgs)
    ' When creating the insert form, locate and change the "more" button appearance.
    If e.Container.Mode = SchedulerFormMode.Insert Then
        Dim more As LinkButton = TryCast(e.Container.Controls(1).FindControl("more"), LinkButton)
        more.BorderStyle = BorderStyle.Ridge
        more.BorderWidth = 1
    End If
End Sub
```

[C#] Handling the FormCreating and FormCreated Events

```
protected void RadScheduler1_FormCreating(object sender, SchedulerFormCreatingEventArgs e)
{
    // disallow advanced form editing for past appointments
    bool preventEdit =
        e.Mode == SchedulerFormMode.AdvancedEdit
        && e.Appointment.Start < DateTime.Today;
    e.Cancel = preventEdit;
    if (preventEdit)
    {
        RadAjaxManager1.Alert("Advanced editing for past appointments not allowed");
    }
}
protected void RadScheduler1_FormCreated(object sender, SchedulerFormCreatedEventArgs e)
{
    // When creating the insert form, locate and change the "more" button appearance.
    if (e.Container.Mode == SchedulerFormMode.Insert)
    {
        LinkButton more = e.Container.Controls[1].FindControl("more") as LinkButton;
        more.BorderStyle = BorderStyle.Ridge;
        more.BorderWidth = 1;
    }
}
```

```
}

```

Responding to TimeSlot Events

Each time a time slot is created you get an opportunity to change the TimeSlot CssClass to style the slot. At the time of this writing, abilities are being added to this event to allow access to the controls within the time slot as well.

[VB] Handling the TimeSlotCreated Event

```
Protected Sub RadScheduler1_TimeSlotCreated(ByVal sender As Object, ByVal e As
TimeSlotCreatedEventArgs)
    Log("TimeSlotCreated")
    If e.TimeSlot.Appointments.Count = 0 Then
        e.TimeSlot.CssClass = "OpenTimeSlots"
    End If
End Sub

```

[C#] Handling the TimeSlotCreated Event

```
protected void RadScheduler1_TimeSlotCreated(object sender, TimeSlotCreatedEventArgs e)
{
    Log("TimeSlotCreated");
    if (e.TimeSlot.Appointments.Count == 0)
    {
        e.TimeSlot.CssClass = "OpenTimeSlots";
    }
}

```

Responding to Navigation Events

When you use any of the navigation buttons along the top of the scheduler to move between dates and within views, the **NavigationCommand** and **NavigationComplete** events fire. The NavigationCommand event lets you prevent navigation by setting the **Cancel** property to true. You also have access to the **Command** and the **SelectedDay**.

The NavigationComplete event fires after navigation has already occurred and lets you know the **Command** that got you to your current destination within the scheduler.

[VB] Handling the NavigationCommand and NavigationComplete Events

```
Protected Sub RadScheduler1_NavigationCommand(ByVal sender As Object, ByVal e As
SchedulerNavigationCommandEventArgs)
    ' disables the "today" button, when in "day" view
    e.Cancel = e.Command = SchedulerNavigationCommand.SwitchToSelectedDay AndAlso
RadScheduler1.SelectedView = SchedulerViewType.DayView
End Sub
Protected Sub RadScheduler1_NavigationComplete(ByVal sender As Object, ByVal e As
SchedulerNavigationCompleteEventArgs)
    ' display the current command
    RadAjaxManager1.Alert("Command: " + e.Command.ToString())
End Sub

```

[C#] Handling the NavigationCommand and NavigationComplete Events

```
protected void RadScheduler1_NavigationCommand(object sender,
SchedulerNavigationCommandEventArgs e)
{
    // disables the "today" button, when in "day" view
    e.Cancel =
        e.Command == SchedulerNavigationCommand.SwitchToSelectedDay &&

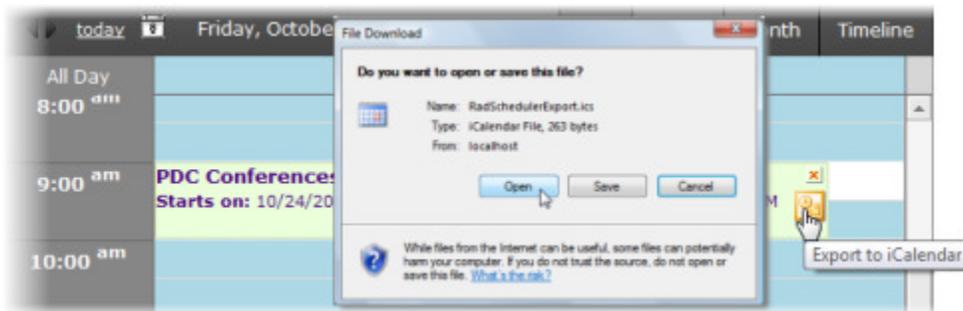
```

UI for ASP.NET AJAX

```
RadScheduler1.SelectedView == SchedulerViewType.DayView;
}
protected void RadScheduler1_NavigationComplete(object sender,
SchedulerNavigationCompleteEventArgs e)
{
    // display the current command
    RadAjaxManager1.Alert("Command: " + e.Command.ToString());
}
}
```

Custom Commands

If you want to add a button within a template that should fire some custom action, handle the **AppointmentCommand**. Just set the **CommandName** property of a button to the string you will look for in the AppointmentCommand arguments CommandName property. The screenshot below shows an ImageButton with an "Outlook" icon (you can find Outlook.gif in the \VS Projects\Images directory). The ImageButton CommandName property is set to "Export".



The example below exports an Outlook calendar file. The RadScheduler **ExportToICalendar()** method takes an Appointment object and returns a string.

[VB] Handling the AppointmentCommand

```
Protected Sub RadScheduler1_AppointmentCommand(ByVal sender As Object, ByVal e As
Telerik.Web.UI.AppointmentCommandEventArgs)
    If e.CommandName.Equals("Export") Then
        WriteCalendar(RadScheduler.ExportToICalendar(e.Container.Appointment))
    End If
End Sub
```

[C#] Handling the AppointmentCommand

```
protected void RadScheduler1_AppointmentCommand(object sender,
Telerik.Web.UI.AppointmentCommandEventArgs e)
{
    if (e.CommandName.Equals("Export"))
    {
        WriteCalendar(RadScheduler.ExportToICalendar(e.Container.Appointment));
    }
}
}
```

The private WriteCalendar() method takes the appointment string and writes it out to the Response stream.

[VB] Writing the Appointment String to the Response

```
Private Sub WriteCalendar(ByVal data As String)
    Dim response As HttpResponse = Page.Response
    response.Clear()
    response.Buffer = True
```

```

response.ContentType = "text/calendar"
response.ContentEncoding = System.Text.Encoding.UTF8
response.Charset = "utf-8"
response.AddHeader("Content-Disposition", "attachment;filename=""RadSchedulerExport.ics"")
response.Write(data)
response.[End]()
End Sub

```

[C#] Writing the Appointment String to the Response

```

private void WriteCalendar(string data)
{
    HttpResponse response = Page.Response;
    response.Clear();
    response.Buffer = true;
    response.ContentType = "text/calendar";
    response.ContentEncoding = System.Text.Encoding.UTF8;
    response.Charset = "utf-8";
    response.AddHeader("Content-Disposition",
        "attachment;filename=""RadSchedulerExport.ics"");
    response.Write(data);
    response.End();
}

```



You can find the complete source for this project at:

VS Projects\DateTimeScheduler\Scheduler_Events

You will need to verify that the SchedulerDataConnectionString in the web.config points to the correct path to RadControls on your system.

43.15 Scheduler Client-Side Programming

RadScheduler has a rich set of methods and events to do what can be done server-side, with some additional capabilities available only on the client. As with all the RadControls, use the \$find() method to get an object instance.

[JavaScript] Get Scheduler Client Reference

```
var scheduler = $find("<%= RadScheduler1.ClientID %>");
```

From there you can get collections from the scheduler for appointments and resources:

[JavaScript] Getting a Collection of Appointments and Resources

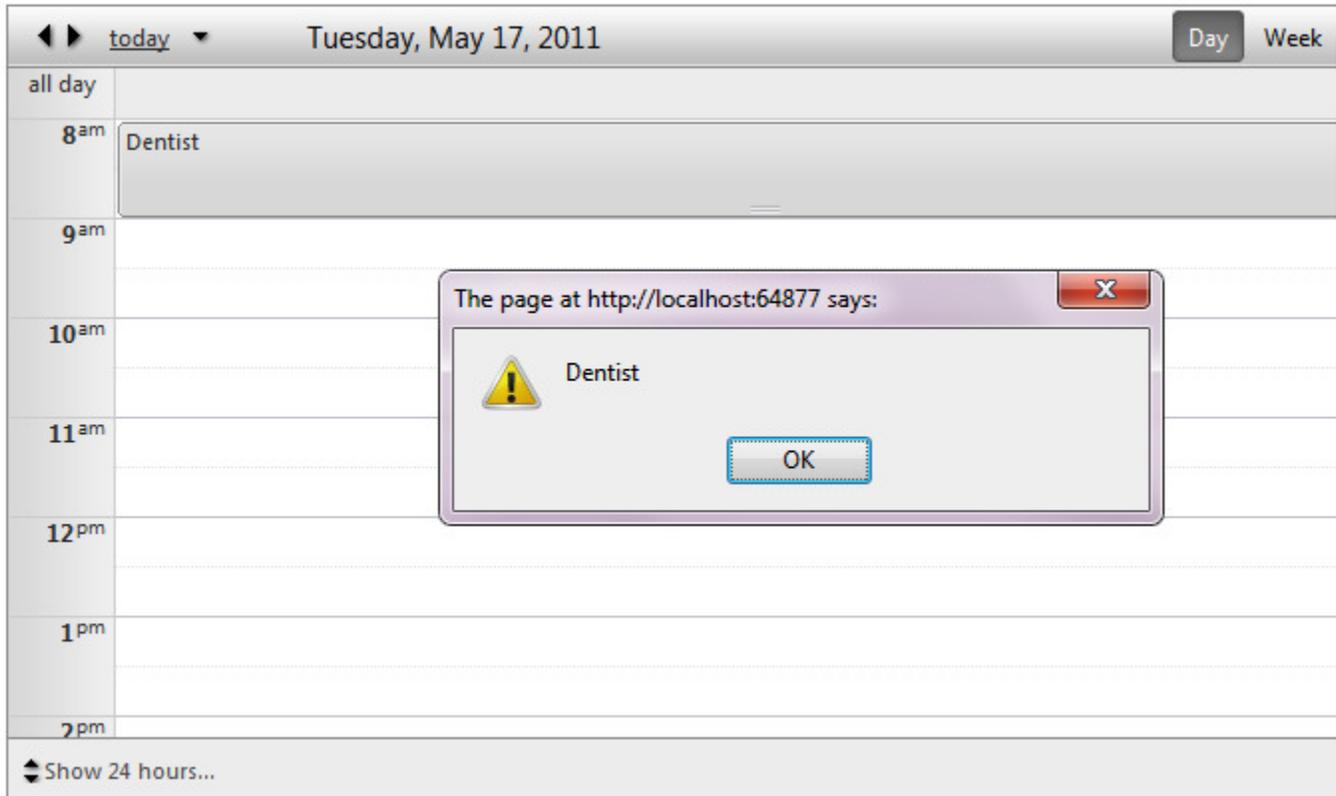
```

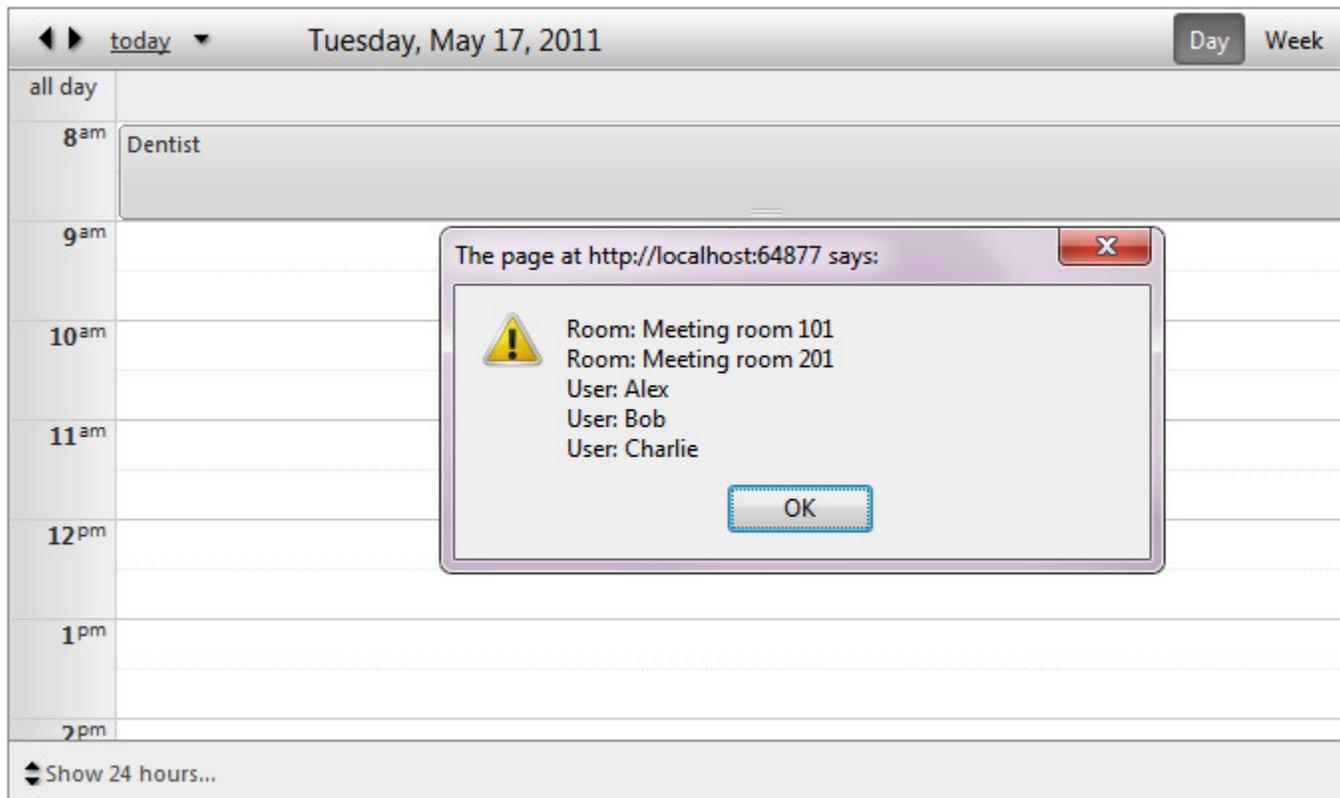
// display list of appointment subjects
var message = "";
var appointments = scheduler.get_appointments();
for (var index = 0; index < appointments.get_count(); index++) {
    var app = appointments.getAppointments(index);
    message += app.get_subject() + "\n";
}
alert(message);
//...
// display list of appointment resources
var message = "";
var resources = scheduler.get_resources();
for (var index = 0; index < resources.get_count(); index++) {
    var resource = resources.getResource(index);
    message += resource.get_type() + ": " + resource.get_text() + "\n";
}

```

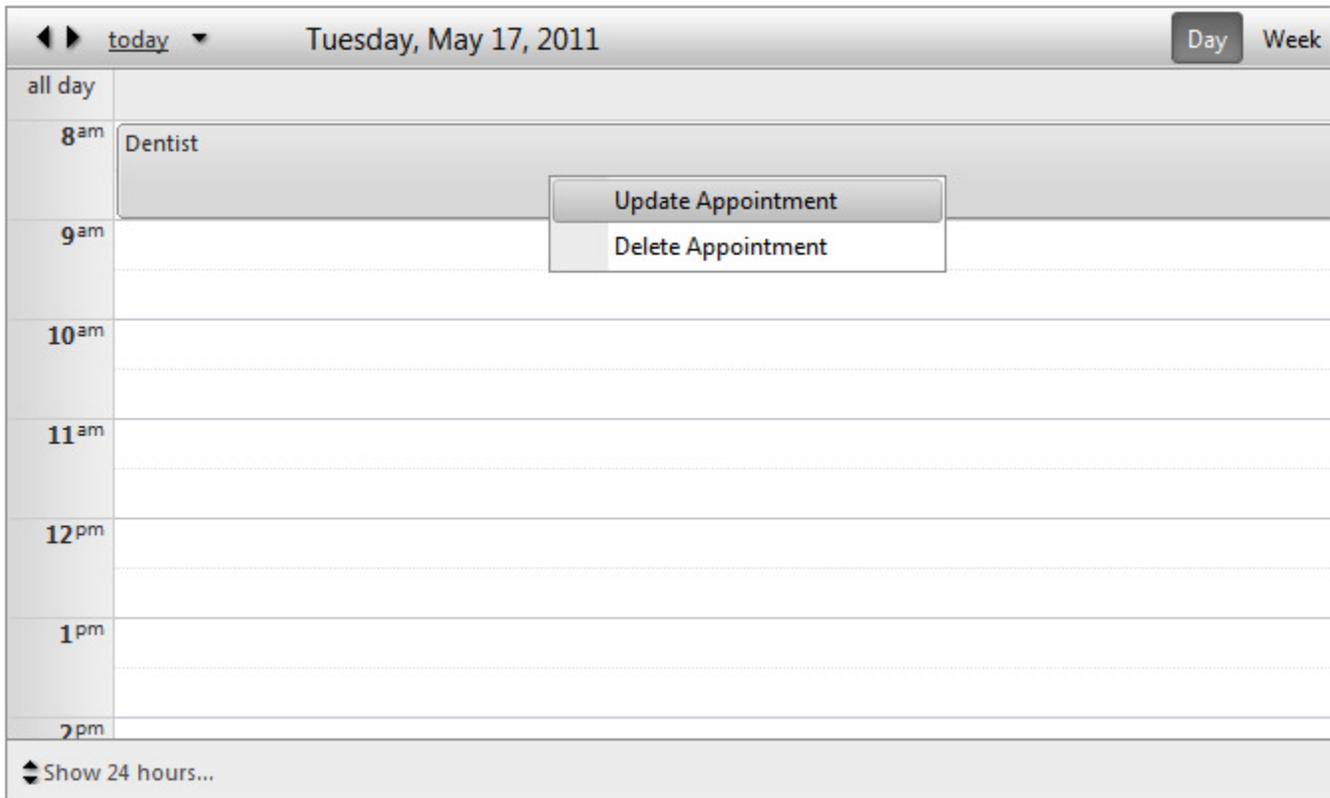
UI for ASP.NET AJAX

```
}  
alert(message);
```





You can also trigger the default Insert/Edit/Delete scheduler operations, much in the same way as if the user had initiated the action. For example, you could hook up to **OnClientAppointmentContextMenu** client event to grab the right-clicked appointment and display a context menu that will allow the user to edit or delete an appointment.



The RadContextMenu OnClientItemClicked event handler uses the saved appointment reference and calls the scheduler `editAppointment()` or `deleteAppointment()` methods. Both methods take a reference to the appointment as a parameter.

[JavaScript] Editing and Deleting Appointments

```
// holds the appointment reference
var selectedAppointment = null;
// Scheduler event responds to right-click of existing
// appointment. This event handler saves off the
// appointment that was right clicked as "selectedAppointment".
// selectedAppointment is used later in the context menu
// ClientItemClicked event handler.
function ClientAppointmentContextMenu(sender, args) {
    selectedAppointment = args.get_appointment();
    var menu = $find("<%= RadContextMenu1.ClientID %>");
    menu.show(args.get_domEvent());
}
function ClientItemClicked(sender, args) {
    var scheduler = $find("<%= RadScheduler1.ClientID %>");
    var item = args.get_item();
    switch (item.get_value()) {
        // displaying the advanced update form for editing
        case "edit":
            {
                scheduler.editAppointment(selectedAppointment);
                break;
            }
    }
}
```

```

// delete the appointment
case "delete":
{
    scheduler.deleteAppointment(selectedAppointment);
    break;
}
}
}

```

You can insert an appointment by displaying the inline insert form using the scheduler **showInsertFormAt(targetSlot)** method. A good place to call this is from the **OnClientTimeSlotClick** event which surfaces a **get_targetSlot()** method that can be used to feed the **showInsertFormAt()** method.

[JavaScript] Calling showInsertFormAt()

```

function ClientTimeSlotClick(sender, args) {
    var targetSlot = args.get_targetSlot();
    sender.showInsertFormAt(targetSlot);
}

```

A second route is to call the scheduler **insertAppointment()** method. The example below creates an appointment automatically when the user clicks a time slot.

The **ClientTimeSlotClick** event handler creates a new appointment object, populates the start and end times using the target slot's start time as a starting point and inserts the appointment.

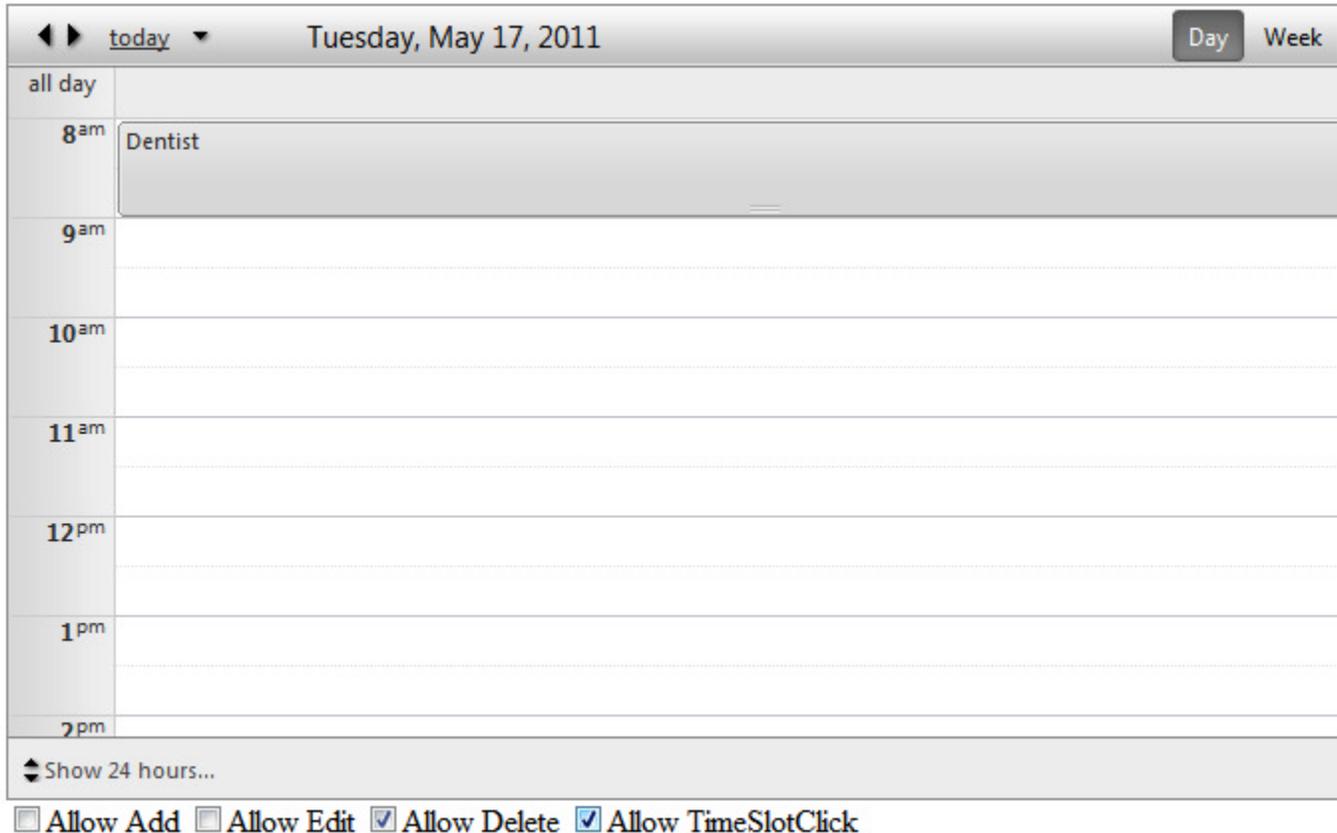
[JavaScript] Calling insertAppointment()

```

// A single left click to a time slot creates a new appointment with a
// default duration of 45 minutes..
function ClientTimeSlotClick(sender, args) {
    // create a new appointment client object
    var appointment = new Telerik.Web.UI.SchedulerAppointment();
    // get the time slot object for the cell that was clicked on
    var targetSlot = args.get_targetSlot();
    // get the start and end times for the target slot
    // and add 45 minutes to the end time
    var startTime = targetSlot.get_startTime();
    var endTime = new Date(startTime);
    endTime.setMinutes(endTime.getMinutes() + 45);
    // set the start time, end time and subject
    appointment.set_start(startTime);
    appointment.set_end(endTime);
    appointment.set_subject("Manually Inserted Appointment");
    // insert the appointment. This will trigger events on the
    // server side.
    sender.insertAppointment(appointment);
}

```

Toggle the ability to insert/edit/delete using the corresponding set of **RadScheduler** client properties. Here's an example that sets these properties based off a standard ASP **CheckBoxList** control.



The `setAllowSettings()` method in the example below does most of the work and is called when a checkbox is clicked or when the page first loads. The `CheckBoxList` check boxes are extracted by way of the `getElementsByTagName("input")` call. You can then index into the array of elements and use the "checked" property in your calls to `set_allowInsert()`, `set_allowEdit()` and `set_allowDelete()`. Also notice that you can suppress the delete confirmation using the `set_displayDeleteConfirmation()` method.

The last part of the `setAllowSettings()` method shows another client-side technique of controlling how the `RadScheduler` events are handled. Here the `OnClientTimeSlotClick` event is removed or added based on a checkbox value. Like the other `RadControls`, `RadScheduler` comes with a set of method pairs that add and remove event handlers. Also remember that multiple event handlers can be added to a single event.

[JavaScript] Setting the Allow Insert/Edit/Delete Properties

```

/* CheckBoxList Events */
function CheckItem(sender) {
    setAllowSettings(sender);
}
/* general MS AJAX Library Events */
function pageLoad() {
    setAllowSettings($get("<%= cbScheduler.ClientID %>"));
}
// toggles scheduler functionality based on checkbox selections
function setAllowSettings(checkBoxList) {
    var checkBoxes = checkBoxList.getElementsByTagName("input");
    var scheduler = $find("<%= RadScheduler1.ClientID %>");
    // toggle the add/edit/delete built-in functionality
    scheduler.set_allowInsert(checkBoxes[0].checked);
}

```

```

scheduler.set_allowEdit(checkBoxes[1].checked);
scheduler.set_allowDelete(checkBoxes[2].checked);
// always disable the delete confirmation dialog
scheduler.set_displayDeleteConfirmation(false);
// based on checkbox value, turn on handling for
// the OnClientTimeSlotClick event
if (checkBoxes[3].checked) {
    scheduler.add_timeSlotClick(ClientTimeSlotClick);
}
else {
    scheduler.remove_timeSlotClick(ClientTimeSlotClick);
}
}
}

```



You can also disable or change the appearance of the scheduler using styles. The styles below hide the delete and resize buttons respectively. Notice the "important" that follows the visibility property setting and that the semi-colon follows. You can use these same styles if you want to replace the default buttons.

[CSS] Hiding the Delete and Resize Buttons

```

<style type="text/css">
.RadScheduler_Hay .rsAptDelete
{
    visibility: hidden !important;
}
.rsAptWrap .rsAptResize
{
    visibility: hidden !important;
}
</style>

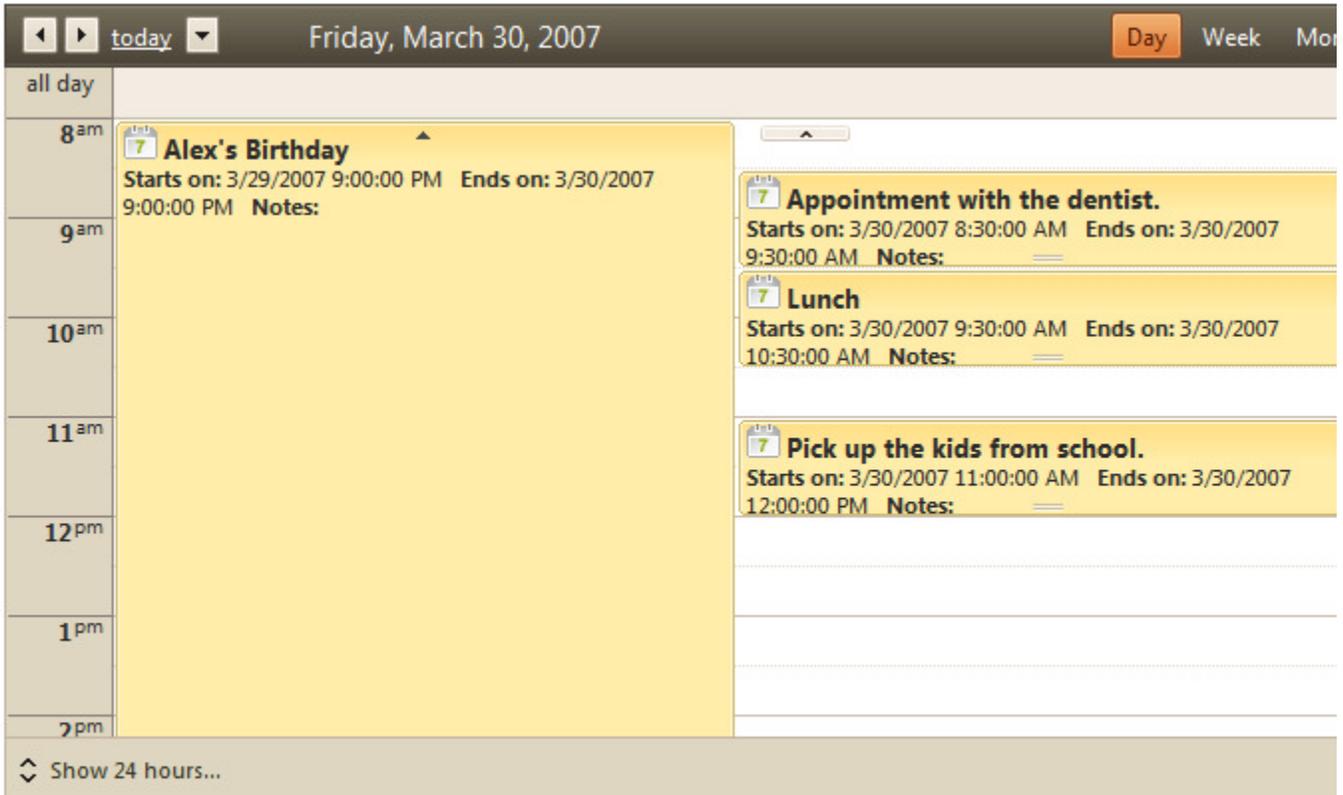
```



You can find the complete source for this project at:
 \VS Projects\DateTimeSchedule\Scheduler_ClientSide

43.16 Using Scheduler Templates

To extensively customize appearance and content use one of several templates that cover the scheduler user interface. The display of an appointment in any of the views can accept arbitrary HTML using the **AppointmentTemplate**. You can enter a template directly within the RadScheduler tag in the markup. Intellisense will give you a hand by showing the available templates. Just place your cursor inside the AppointmentTemplate tag, enter a less-than bracket ("**<**") and click ctrl-spacebar to initiate Intellisense.



If we take this a step farther and add another template, the `InlineInsertTemplate` to completely customize adding a new appointment. Here we use the `Bind()` method to get the two way data binding to work.

[ASP.NET] Adding the `InlineInsertTemplate`

```
<InlineInsertTemplate>
<!--Calendar image-->


<!--Subject-->
<telerik:RadTextBox ID="RadTextBox1" runat="server"
  Skin="Sunset" Text='<# Bind("Subject") %>'>
</telerik:RadTextBox>
<br />

<!--Start and End times-->
<b>Starts on:&nbsp;
  <telerik:RadDateTimePicker ID="RadDateTimePicker1"
    Skin="Sunset" runat="server" SelectedDate='<# Bind("Start")%>'>
  </telerik:RadDateTimePicker>
<b>Ends on:&nbsp;
  <telerik:RadDateTimePicker ID="RadDateTimePicker2"
    Skin="Sunset" runat="server" SelectedDate='<# Bind("End")%>'>
  </telerik:RadDateTimePicker>

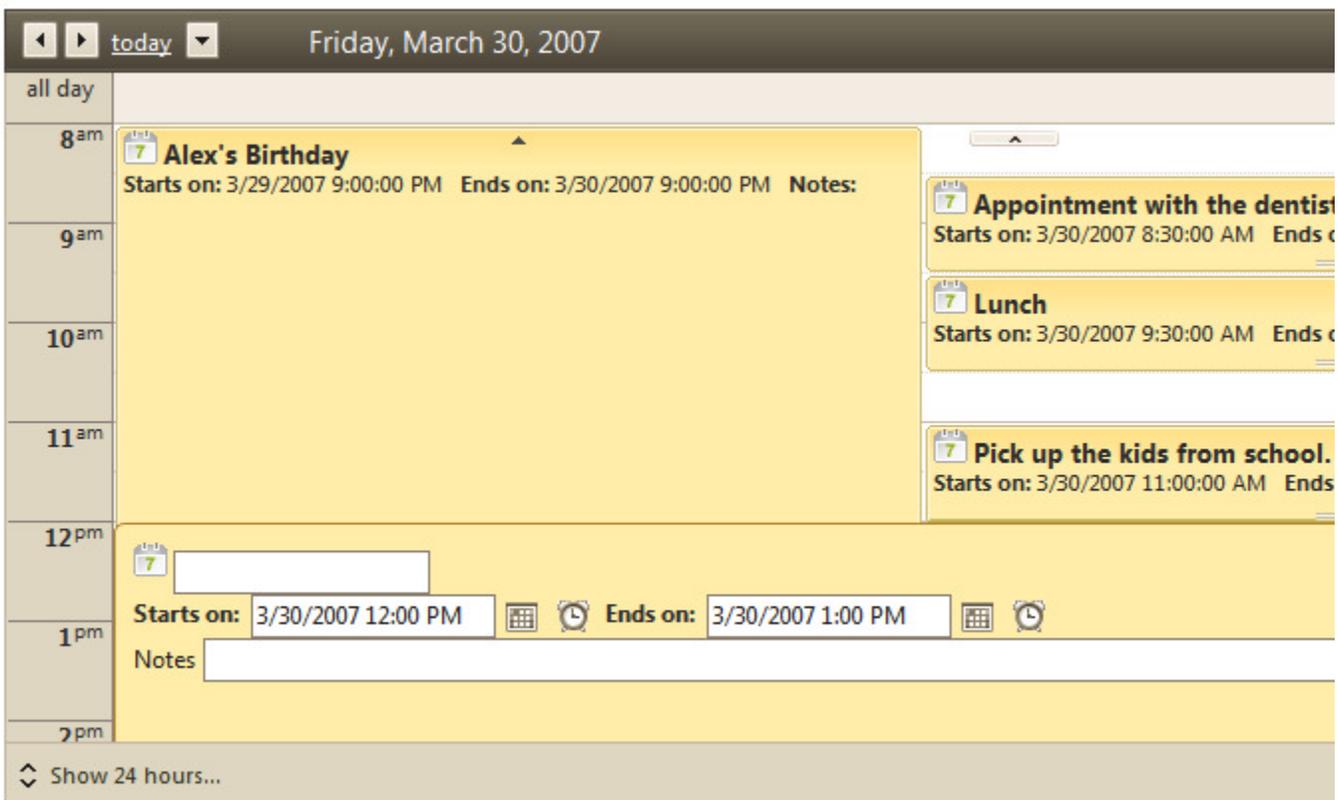
<!--Custom attributes "Annotations", i.e. "Notes" -->
<telerik:RadTextBox ID="RadTextBox2" runat="server"
```

```
TextMode="MultiLine" Width="100%" Label="Notes" Skin="Sunset"  
Text='<%= Bind("Annotations") %>'  
</telerik:RadTextBox>
```

```
<!--Ok and cancel buttons floated right-->  
<span style="float: right">  
  <asp:LinkButton ID="InsertButton" runat="server" CommandName="Insert">  
    <asp:Image runat="server" ID="insertImage" ImageUrl="Images/ok.gif"  
      AlternateText="insert" />  
  </asp:LinkButton>  
  <asp:LinkButton ID="InsertCancelButton" runat="server"  
    CausesValidation="False" CommandName="Cancel">  
    <asp:Image runat="server" ID="Image2" ImageUrl="Images/cancel.gif"  
      AlternateText="cancel" />  
  </asp:LinkButton>  
</span>  
</InlineInsertTemplate>
```

- Be sure to name the CommandName for the two buttons "Insert" and "Cancel". The control is expecting these two command names specifically and will not act correctly otherwise. Be aware that these commands are case sensitive.

The result is shown in the screenshot below.



43.17 Summary

In this chapter we explored the features of the date and time picker, calendar and scheduler controls. You created some simple applications to become familiar with the controls. We explored the design time interfaces of each of the controls. We worked with the server-side API to explore the major objects that make up each

control. In particular, we set calendar special days, added scheduler appointments, add scheduler resources, scheduler recurrence and handled client-side events. You learned how to validate date and time picker control entry. You also learned how to use scheduler templates.

44 ActiveSkill: Exam Scheduling

44.1 Objectives

- Build the ExamScheduler user control.
- Configure the RadTreeView and RadScheduler for drag and drop.
- Learn how to handle the scheduler events to create new appointments as tree nodes are dropped on the scheduler and "reserve" existing appointments when a button in the appointment template is clicked. Also learn to format appointments as they are created based on the logged in user role and attribute data of the appointment.
- Integrate the ExamScheduler to both the user and admin ScheduleExams.ascx controls.

44.2 Defining the Markup

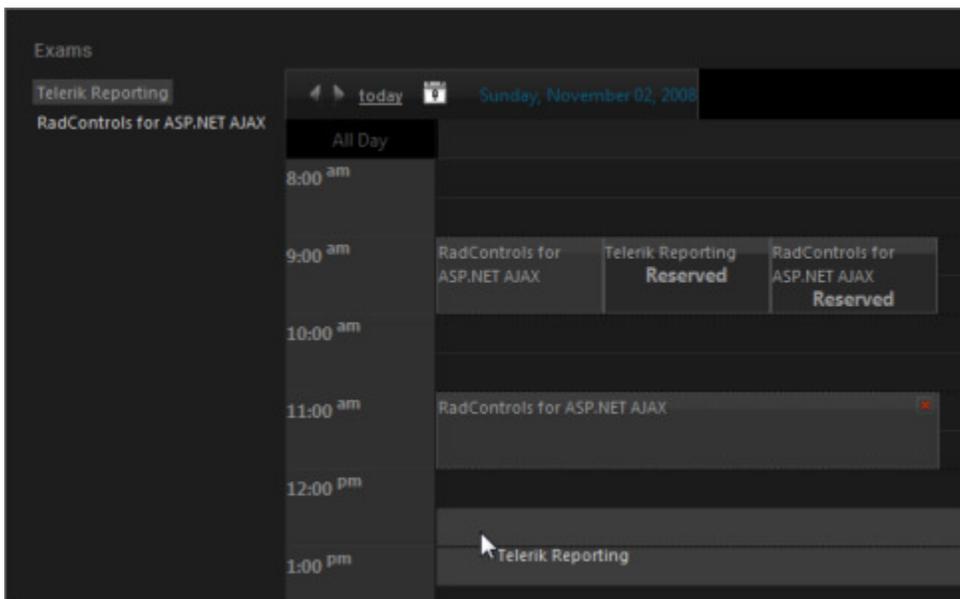


You can find the complete source for this project at:
\\VS Projects\ActiveSkill_Scheduling

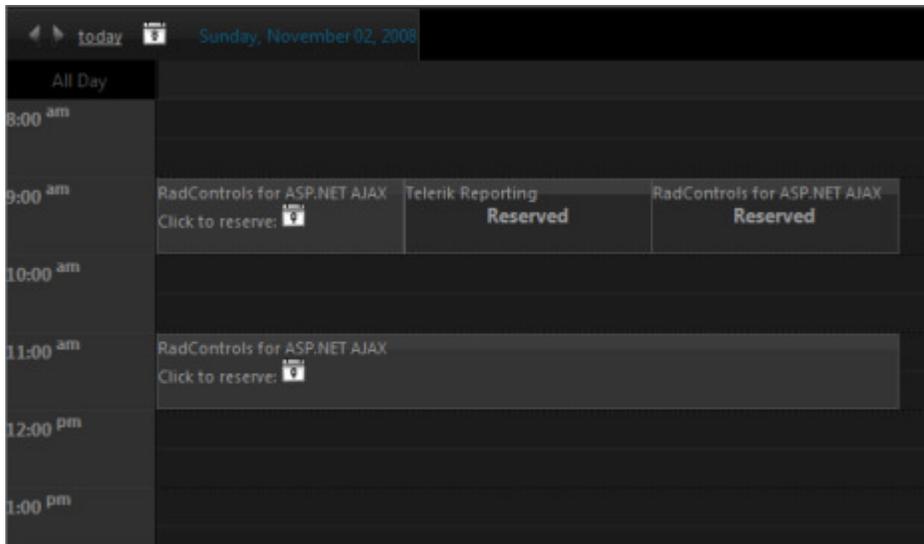
The exam scheduler user control displays a treeview with a flat list of available exams on the left and a scheduler control on the right. Exams from the treeview are dragged onto the scheduler and added to the scheduler.

The scheduler knows nothing about dragging and dropping. Even though the list of exams is a flat list, we use RadTreeView because it has the drag and drop capability already built-in.

This user control will display slightly differently depending on the ASP.NET Membership role of the logged-in user. The admin will see the scheduler *and* the list of exams, will be able to drag exams to the scheduler and be able to delete exams (if the exam hasn't already been reserved).



A "User" role login displays the scheduler only, and displays a calendar button and text so that the user can click to reserve an available appointment. Once the appointment is reserved, the user id and exam are recorded with the appointment and the appointment can no longer be deleted or reserved by any one else.



1. In the \ActiveSkillUI\Controls directory add a new **Web User Control** item and name it "ExamScheduler.ascx".
2. In the markup for the user control, add a **SqlDataSource**. *This data source will supply a RadTreeView with a list of available exams.*

[ASP.NET] Defining the Exams Data Source

```
<asp:SqlDataSource ID="dsExams" runat="server"
  ConnectionString="<%= $ ConnectionStrings.ActiveSkillConnectionString %>"
  SelectCommand="SELECT [ID], [Title] FROM [Exam]">
</asp:SqlDataSource>
```

3. Add a second SqlDataSource for the scheduler:

[ASP.NET] Defining the Scheduler Data Source

```
<asp:SqlDataSource ID="dsScheduler" runat="server"
  ConnectionString="<%= $ ConnectionStrings.ActiveSkillConnectionString %>"
  SelectCommand="SELECT [ID], [ExamID], [UserID], [Subject], [StartTime], [EndTime],
  [Recurrence], [RecurParentID] FROM [Appointment_Data]"
  DeleteCommand="DELETE FROM [Appointment_Data] WHERE [ID] = @ID"
  InsertCommand="INSERT INTO [Appointment_Data] ([ExamID], [UserID], [Subject], [StartTime],
  [EndTime], [Recurrence], [RecurParentID]) VALUES (@ExamID, @UserID, @Subject, @StartTime,
  @EndTime, @Recurrence, @RecurParentID)"
  UpdateCommand="UPDATE [Appointment_Data] SET [ExamID] = @ExamID, [UserID] = @UserID,
  [Subject] = @Subject, [StartTime] = @StartTime, [EndTime] = @EndTime, [Recurrence] =
  @Recurrence, [RecurParentID] = @RecurParentID WHERE [ID] = @ID">
  <DeleteParameters>
    <asp:Parameter Name="ID" Type="Int32" />
  </DeleteParameters>
  <UpdateParameters>
    <asp:Parameter Name="ExamID" Type="Int32" />
    <asp:Parameter Name="UserID" />
    <asp:Parameter Name="Subject" Type="String" />
    <asp:Parameter Name="StartTime" Type="DateTime" />
    <asp:Parameter Name="EndTime" Type="DateTime" />
  </UpdateParameters>
```

```

    <asp:Parameter Name="Recurrence" Type="String" />
    <asp:Parameter Name="RecurParentID" Type="Int32" />
    <asp:Parameter Name="ID" Type="Int32" />
</UpdateParameters>
<InsertParameters>
    <asp:Parameter Name="ExamID" Type="Int32" />
    <asp:Parameter Name="UserID" />
    <asp:Parameter Name="Subject" Type="String" />
    <asp:Parameter Name="StartTime" Type="DateTime" />
    <asp:Parameter Name="EndTime" Type="DateTime" />
    <asp:Parameter Name="Recurrence" Type="String" />
    <asp:Parameter Name="RecurParentID" Type="Int32" />
</InsertParameters>
</asp:SqlDataSource>

```



Gotcha! Notice the insert and update parameters named "UserID". If you build the parameters using the DataSource Configuration Wizard, the Type parameter will be "Object" automatically. The UserID in the table is actually a UNIQUEIDENTIFIER (GUID) which "Object" doesn't handle very well. You can get around this by deleting the Type from the markup by hand.

4. Add a hidden field to the markup with ID "TargetSlotHiddenField". *This hidden field will be populated later from client code and will contain the index of the timeslot that the user is dropping onto.*

[ASP.NET] Adding the TargetSlotHiddenField

```
<input type="hidden" id="TargetSlotHiddenField" runat="server" value="0" />
```

5. Add an "Exams" title and RadTreeView to the markup. *The treeview must have its EnableDragAndDrop property set to true. The OnClientNodeDropping event starts off the process by notifying us of when the scheduler has a node dropped on it. The client event handler also triggers the postback so the server-side OnNodeDrop event can fire.*

[ASP.NET] Adding the TreeView and TreeViewTitle

```

<div id="divSkillTreeTitle" runat="server" class="skillTreeTitle">
    Exams
</div>
<div id="leftPane" runat="server" class="skillTree">
    <telerik:RadTreeView ID="RadTreeView1" runat="server"
        DataSourceID="dsExams" DataFieldID="ID" DataTextField="Title"
        DataValueField="ID" EnableDragAndDrop="True" OnClientNodeDropping="nodeDropping"
        OnNodeDrop="RadTreeView1_NodeDrop">
    </telerik:RadTreeView>
</div>

```

6. Add the markup for the scheduler. Notice the following about this markup:
 - AllowInsert and AllowEdit are set to false. We only want to allow adding appointments through dragging or deleting by clicking the delete button in the appointment.
 - ShowViewTabs is set to False. We just want to drag appointments to the default day view only.
 - EnableAdvancedForm and StartEditingAdvancedForm are set to false. We only want to see the inline form.
 - In the AppointmentTemplate there are two divs that will be toggled so that one will be visible for reserved appointments and the other invisible.
 - Also in the AppointmentTemplate is a LinkButton control with the CommandName set to "Reserve". We will re-visit the Reserve command later when we add the server-side code.

[ASP.NET] Adding the RadScheduler

```

<div id="schedulerPane" runat="server" class="schedulerPane">
  <telerik:RadScheduler ID="RadScheduler1" runat="server"
    OverflowBehavior="Scroll" DataSourceID="dsScheduler"
    DataEndField="EndTime" DataKeyField="ID" DataRecurrenceField="Recurrence"
    DataRecurrenceParentKeyField="RecurParentID" DataStartField="StartTime"
    DataSubjectField="Subject" ShowViewTabs="False" CustomAttributeNames="UserID,ExamI
    EnableAdvancedForm="False" StartEditingInAdvancedForm="False"
    OnAppointmentCommand="RadScheduler1_AppointmentCommand"
    OnAppointmentCreated="RadScheduler1_AppointmentCreated"
    AllowInsert="False" AllowEdit="false">
    <AppointmentTemplate>
      <%# Eval("Subject") %>
      <br />
      <div id="appointmentDiv" runat="server">
        Click to reserve:
        <asp:LinkButton ID="InsertButton" runat="server" CommandName="Reserve">
          <asp:Image runat="server" ID="insertImage"
            ImageUrl="..\skins\ActiveSkill\calendar\datePickerPopup.gif"
            AlternateText="Reserve" />
        </asp:LinkButton>
      </div>
      <div id="reservedDiv" runat="server" style="font-size: small;
        font-weight: bold; text-align: center">
        Reserved
      </div>
    </AppointmentTemplate>
  </telerik:RadScheduler>
</div>

```

44.3 Handling the Drag and Drop Client-Side

1. Add the script block below to the hidden field tag "TargetSlotHiddenField":

[JavaScript] Adding the Script Block

```

<telerik:RadScriptBlock ID="RadScriptBlock1" runat="server">
  <script type="text/javascript">

    </script>
  </telerik:RadScriptBlock>

```

2. Add a helper method to the script block "isPartOfSchedulerAppointmentArea". This function will determine if a given html element is part of the scheduler appointment area.

[JavaScript] Add Helper Method

```

function isPartOfSchedulerAppointmentArea(htmlElement) {
  // Determines if an html element is part of the scheduler appointment area
  // This can be either the rsContent or the rsAllDay div (in day and week view)
  var scheduler = $find('<%= RadScheduler1.ClientID %>');
  var allDayDiv = $telerik.getChildByClassName(scheduler.get_element(), "rsAllDay");
  var contentDiv = $telerik.getChildByClassName(scheduler.get_element(), "rsContent");
  return $telerik.isDescendant(contentDiv, htmlElement) || $telerik.isDescendant(allDayDiv,

```

```
htmlElement);  
}
```

3. Add the `nodeDropping()` event handler that responds to nodes being dropped from the treeview. The event args passed to this handler have a `get_htmlElement()` method that can be used in the scheduler's "active model" `getTimeSlotFromDomElement()` method which produces the `timeSlot` that was dropped onto. The time slot's index is stored into the hidden field for use within the server-side code.

[JavaScript] Add `nodeDropping` Client Event Handler

```
function nodeDropping(sender, eventArgs) {  
    // Fired when the user drops a TreeView node  
    var node = eventArgs.get_sourceNode();  
    var text = node.get_text();  
    var htmlElement = eventArgs.get_htmlElement();  
    var scheduler = $find('<%= RadScheduler1.ClientID %>');  
    if (isPartOfSchedulerAppointmentArea(htmlElement)) {  
        // The node was dropped over the scheduler appointment area  
        // Find the exact time slot and save its unique index in the hidden field  
        var timeSlot = scheduler.get_activeModel().getTimeSlotFromDomElement(htmlElement);  
        // Gotcha! Use the ClientID, not "TargetSlotHiddenField" directly or will  
        // come back null. Thats because this is in nested user controls, not directly  
        // on a web page.  
        var targetSlotHiddenField = $get("<%= TargetSlotHiddenField.ClientID %>");  
        targetSlotHiddenField.value = timeSlot.get_index();  
        // The HTML needs to be set in order for the postback to execute normally  
        eventArgs.set_htmlElement(targetSlotHiddenField);  
    }  
    else {  
        // The node was dropped elsewhere on the document  
        eventArgs.set_cancel(true);  
    }  
}
```

44.4 Handle Server-Side Events

1. You need to add the following assemblies to your "Imports" (VB) or "uses" (C#) portion of the code:
 - o System
 - o System.Drawing
 - o System.Web.Security
 - o System.Web.UI.HtmlControls
 - o Telerik.ActiveSkill.Common
 - o Telerik.Web.UI
 - o Telerik.Web.UI.Scheduler.Views
2. In the `Page_Load` event add this assignment by

[VB] Setting the Tree View Visibility

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)  
    Me.RadTreeView1.Visible = Roles.IsUserInRole("Admin")  
End Sub
```

[C#] Setting the Tree View Visibility

```
protected void Page_Load(object sender, EventArgs e)
{
    this.RadTreeView1.Visible = Roles.IsUserInRole("Admin");
}
```

3. Handle the NodeDrop Event. First retrieve the slot index from the hidden field. Also get the exam id and title. Setup a duration object based on the starting time of the slot. The duration will be set to either 1 hour or an entire day in length. Create a new appointment and assign the duration, exam id and exam title. The exam id gets assigned to the appointment as an attribute. Insert the new appointment and clear the hidden field for the next use.

[VB] Handling the NodeDrop Event

```
Protected Sub RadTreeView1_NodeDrop(ByVal sender As Object, ByVal e As
Telerik.Web.UI.RadTreeNodeDragDropEventArgs)
    ' retrieve the hidden field value
    Dim targetSlotIndex As String = TargetSlotHiddenField.Value
    If targetSlotIndex <> String.Empty Then
        ' get the dropped node id and subject
        Dim id As Integer = Int32.Parse(e.SourceDragNode.Value)
        Dim subject As String = e.SourceDragNode.Text
        RadScheduler1.Rebind()
        ' setup a duratin based on the starting time of the slot and either
        ' 1 hour or an entire day in length.
        Dim slot As ISchedulerTimeSlot = RadScheduler1.GetTimeSlotFromIndex(targetSlotIndex)
        Dim duration As TimeSpan = TimeSpan.FromHours(1)
        If slot.Duration = TimeSpan.FromDays(1) Then
            duration = slot.Duration
        End If
        ' create a new appointment, assign subject and duration.
        Dim appointment As New Appointment(0, slot.Start, slot.Start.Add(duration), subject)
        ' Add the exam id (derived from the dropped node) as an attribute
        appointment.Attributes.Add("ExamID", id.ToString())
        ' insert the appointment and clear the hidden field
        RadScheduler1.InsertAppointment(appointment)
        TargetSlotHiddenField.Value = String.Empty
    End If
End Sub
```

[C#] Handling the NodeDrop Event

```
protected void RadTreeView1_NodeDrop(object sender,
Telerik.Web.UI.RadTreeNodeDragDropEventArgs e)
{
    // retrieve the hidden field value
    string targetSlotIndex = TargetSlotHiddenField.Value;
    if (targetSlotIndex != string.Empty)
    {
        // get the dropped node id and subject
        int id = Int32.Parse(e.SourceDragNode.Value);
        string subject = e.SourceDragNode.Text;
        RadScheduler1.Rebind();
        // setup a duratin based on the starting time of the slot and either
        // 1 hour or an entire day in length.
        ISchedulerTimeSlot slot = RadScheduler1.GetTimeSlotFromIndex(targetSlotIndex);
        TimeSpan duration = TimeSpan.FromHours(1);
```

```
    if (slot.Duration == TimeSpan.FromDays(1))
    {
        duration = slot.Duration;
    }
    // create a new appointment, assign subject and duration.
    Appointment appointment = new Appointment(0, slot.Start, slot.Start.Add(duration),
subject);
    // Add the exam id (derived from the dropped node) as an attribute
    appointment.Attributes.Add("ExamID", id.ToString());
    // insert the appointment and clear the hidden field
    RadScheduler1.InsertAppointment(appointment);
    TargetSlotHiddenField.Value = string.Empty;
}
}
```

4. Handle the AppointmentCommand Event. When the user clicks the calendar button, the AppointmentCommand event fires, the event arguments CommandName is compared to the expected command name of "Reserve", the UserID GUID is stored in the appointment's attributes and the appointment is updated.

[VB] Handling the AppointmentCommand Event

```
Protected Sub RadScheduler1_AppointmentCommand(ByVal sender As Object, ByVal e As
AppointmentCommandEventArgs)
    If e.CommandName.Equals("Reserve") Then
        e.Container.Appointment.Attributes("UserID") = SessionManager.User.UserID.ToString()
        RadScheduler1.UpdateAppointment(e.Container.Appointment)
    End If
End Sub
```

[C#] Handling the AppointmentCommand Event

```
protected void RadScheduler1_AppointmentCommand(object sender, AppointmentCommandEventArg
e)
{
    if (e.CommandName.Equals("Reserve"))
    {
        e.Container.Appointment.Attributes["UserID"] = SessionManager.User.UserID.ToString();
        RadScheduler1.UpdateAppointment(e.Container.Appointment);
    }
}
```

5. Handle the AppointmentCreated event. When the appointment is first created we want to set the appearance and behavior for the appointment based on the user's role and if the appointment has been "reserved". The role can be derived by using the ASP.NET Membership Roles object and checking if the user is in the role "Admin". We can find out if the appointment is "reserved" or not by reading the event arguments Appointment attributes and seeing if the "UserID" attribute is populated.

Using this information we set the appointment back color to a slightly darker color if reserved. We also set the visibility of the div that contains "Click to reserve" and the calendar icon, as well as the "reserved" div. The first displays only when a user (not an admin) is logged in and the appointment is not reserved. The "reserved" div is visible when anyone is logged in and the appointment is reserved. The AllowDelete property for the appointment only (not the AllowDelete property for the entire scheduler) is allowed when the admin is logged in and the appointment is not reserved.

[VB] Handle the AppointmentCreated Event

```
Protected Sub RadScheduler1_AppointmentCreated(ByVal sender As Object, ByVal e As
AppointmentCreatedEventArgs)
    Dim isAdmin As Boolean = Roles.IsUserInRole("Admin")
    Dim isReserved As Boolean = Not e.Appointment.Attributes("UserID").Equals([String].Empty)
    e.Appointment.BackColor = IIf(isReserved, Color.FromArgb(40, 40, 40), Color.FromArgb(52, !
52))
    Dim appointmentDiv As HtmlGenericControl = TryCast(e.Container.FindControl
("appointmentDiv"), HtmlGenericControl)
    appointmentDiv.Visible = Not isAdmin And Not isReserved
    Dim reservedDiv As HtmlGenericControl = TryCast(e.Container.FindControl("reservedDiv"),
HtmlGenericControl)
    reservedDiv.Visible = isReserved
    ' Only the admin can delete appointments, but not reserved appointments
    e.Appointment.AllowDelete = isAdmin And Not isReserved
End Sub
```

[C#] Handle the AppointmentCreated Event

```
protected void RadScheduler1_AppointmentCreated(object sender, AppointmentCreatedEventArg
e)
{
    bool isAdmin = Roles.IsUserInRole("Admin");
    bool isReserved = !e.Appointment.Attributes["UserID"].Equals(String.Empty);
    e.Appointment.BackColor = isReserved ? Color.FromArgb(40, 40, 40) : Color.FromArgb(52, !
52);
    HtmlGenericControl appointmentDiv = e.Container.FindControl("appointmentDiv") as
HtmlGenericControl;
    appointmentDiv.Visible = !isAdmin & !isReserved;
    HtmlGenericControl reservedDiv = e.Container.FindControl("reservedDiv") as
HtmlGenericControl;
    reservedDiv.Visible = isReserved;
    // Only the admin can delete appointments, but not reserved appointments
    e.Appointment.AllowDelete = isAdmin & !isReserved;
}
```

6. Override the OnPreRender event to set the images for the scheduler's datepicker calendar. If you miss this step, the images for the calendar will not show up.

[VB] Override the OnPreRender Event

```
Protected Overloads Overrides Sub OnPreRender(ByVal e As EventArgs)
    MyBase.OnPreRender(e)
    Dim datePicker As RadDatePicker = TryCast(RadScheduler1.FindControl("SelectedDatePicker"
RadDatePicker)
    If datePicker <> Nothing Then
        datePicker.Calendar.ImagesPath = "~/Skins/ActiveSkill/Calendar"
    End If
End Sub
```

[C#] Override the OnPreRender Event

```
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);
    RadDatePicker datePicker = RadScheduler1.FindControl("SelectedDatePicker") as
RadDatePicker;
    if (datePicker != null)
    {
        datePicker.Calendar.ImagesPath = "~/Skins/ActiveSkill/Calendar";
    }
}
```

```
}  
}
```

44.5 Integrate the Exam Scheduler

1. In the Login.aspx.cs code-behind, add the code below to the LoggedIn event handler. *This information is needed when we "reserve" an appointment by assigning user ID to the appointment UserID attribute.*

[VB] Persisting the ASP.NET Membership User

```
SessionManager.User = New ASUser(UserNameTextBox.Text)
```

[C#] Persisting the ASP.NET Membership User

```
SessionManager.User = new ASUser(UserNameTextBox.Text);
```

2. Add the markup below to the ScheduleExams.ascx in both the \Admin and \User folders. *This will include the ExamScheduler user control to both pages.*

[ASP.NET] Adding the ExamScheduler

```
<%@ Register Src="../Controls/ExamScheduler.ascx" TagName="ExamScheduler"  
    TagPrefix="uc1" %>  
<div id="schedulerPane" runat="server">  
    <uc1:ExamScheduler ID="ExamScheduler1" runat="server" />  
</div>
```

44.6 Summary

In this chapter you implemented the ExamScheduler.ascx control. You configured a RadTreeView and RadScheduler for drag and drop. You learned how to handle scheduler events to create new appointments and modify the attributes of existing appointments based on commands set within the appointment template. You also learned how to format appointments as they are created based on the logged in user role and appointment attribute data. Finally, you integrated the ExamScheduler for use within ScheduleExams.ascx controls located within the \user and \admin directories.

Index

Important Properties, 712
Introduction, 611
\$find, 80-82, 82-83
\$find(), 360-361
\$get, 80-82
a, 136-141
AccessDataSource, 297-298
ActiveRegion, 867-882
Add PageView, 915-918
Add ScriptManager, 915-918
Add Utility Classes, 216-224
add_itemClicked, 86-91
AddAjaxSetting(), 145-148
AJAX, 246-256, 390-402
AJAX callback, 110-113
AjaxRequest, 805-812
ajaxRequest(), 148-162
AjaxRequestEventArgs, 148-162
AjaxSettings, 141-145, 145-148
AjaxUpdate, 105-110, 110-113
Alert(), 29-37
AllowCustomText, 386-390, 390-402, 414-427
AllowDelete, 987-991
AllowDrag, 432-437, 437-447
AllowDrop, 432-437, 437-447
AllowEdit, 432-437, 983-986
AllowInsert, 983-986
AllowNodeEditing, 432-437, 447-458
AllowPaging, 812-816
AllowRowSelect, 812-816, 812-816, 812-816
Animation, 14-17, 100-105
Appearance and Styling, 736-739
AppendDataBoundItems, 22-29
Appointment, 966-972
AppointmentClick, 966-972
AppointmentCommand, 966-972, 987-991
AppointmentCreated, 987-991

- AppointmentDataBound, 966-972
- AppointmentTemplate, 978-981, 983-986
- args, 86-91
- arguments, 265-280
- ASP.NET Membership, 181-188, 987-991
- attachDropDown, 390-402
- Attributes, 14-17, 148-162
- Auto Save RadEditor's content and notify the user, 649-650
- AutoCompleteType, 49-57
- AutoGenerateColumns, 812-816
- AutoPostBack, 57-60, 165-174, 919-928
- AxisMode, 886-890
- Barcode types, 588-589
- Before You Begin..., 13
- Bigger Icons and Buttons , 602
- Bind(), 346-350, 745-747, 978-981
- Binding Expressions, 340-346
- Binding Hierarchical Data, 309-315
- Binding to Business Objects, 322-330
- Binding to Linq, 330-337
- Block, 141-145
- Build the Admin Page, 362-368
- Build the Choose Exam Control, 812-816
- Build the Exam Question Control, 816-838
- Build the User Home Page, 805-812
- Building the Categories Tree Control, 747-758
- Building the Exam Finish Page, 903-913
- Button Types And Button Collections, 711-712
- Calendar, 918-919
- CalendarDayTemplates, 919-928
- CalendarPopupButton, 928-933
- CalendarTableStyle, 919-928
- CalendarView, 928-933
- callback functions, 256-265
- Callback Support, 647-648
- callbacks, 265-280
- Carousel mode , 132-134
- CausesValidation, 64-69, 936-938
- CellDayFormat, 919-928

- Chart Types, 896-900
- ChartClickEventArgs, 867-882
- Checkable, 430-432, 432-437
- CheckAllNodes, 437-447
- CheckBoxes, 432-437, 437-447
- CheckChildNodes, 437-447
- CheckListBox, 174-176
- ChildrenCreated, 928-933
- Choose Data Source, 22-29, 309-315
- clear(), 928-933
- ClickedCssClass, 22-29
- Client Events Walk Through, 91-95
- Client Side Code, 525-537
- Client Side Programming, 113-117
- ClientChanges, 408-414
- ClientDataKeyNames, 812-816
- ClientEvents, 812-816
- ClientID, 80-82
- ClientSettings, 812-816
- client-side API, 14-17
- Client-Side Data Binding, 679-687
- client-side events, 256-265, 458-463
- Client-Side Items Management, 127-128
- Client-Side Programming, 60-64, 126-127, 148-162, 256-265, 360-361, 381-382, 408-414, 458-463, 478-479, 573-578, 667-671, 691-692, 697-698, 882-886
- ClientValidationFunction, 936-938
- CollapseAnimation, 384-386, 386-390
- Collection Editors, 236-242
- CollectionBase, 297-298
- Columns, 497-503, 812-816
- Command event, 246-256
- CommandName, 966-972, 983-986
- commitChanges, 408-414, 437-447
- Common Features, 44-46, 49-57
- Configure Ajax Manager, 136-141, 141-145, 148-162, 165-174
- Configure Data Source..., 322-330
- Configure RadCaptcha audio, 658-660
- Configure Special Days, 919-928
- Configure the Data Source, 22-29

- Configure the Profile, 224-225
- Configuring the Toolbar, 613-615
- Configuring the ToolsFile, 566-567
- Configuring the XmlHttpPanel, 662-667
- Confirm postback with RadButton, 602-604
- ConnectionString, 812-816
- Container, 346-350, 966-972
- Container.DataItem, 124-126
- Content, 551-562
- ContentAreaCssFile, 551-562
- ContentContainer, 246-256, 265-280
- ContentFilters, 551-562
- ContentProviderTypeName, 551-562
- ContentTemplate, 231-236, 246-256
- ContentUrl, 231-236
- ContextMenuID, 437-447
- ContextMenuItemClick, 437-447
- ContextMenus, 432-437, 437-447
- Control Specifics, 37-42, 128-129, 242-246, 390-402, 437-447, 702-703, 865-867
- Controlling the URL and the Title, 717-718
- Controls collection, 463-465
- ControlToCompare, 936-938
- ControlToValidate, 936-938
- Coverflow mode, 129-132
- Create ActiveSkill Skin, 372-374
- Create Registration Page, 192-201
- Create the ActiveSkill Login Page, 188-192
- Create the Billing Control Code-Behind, 210-214
- Create the BillingControl User Control, 214-216
- Create User Controls, 368-372
- Creating a Custom Skin, 292-296
- Creating a Custom Tool, 616-617
- Creating a single click button, 601-602
- CRUD, 297-298
- CssClass, 340-346, 812-816
- Custom Attributes, 390-402
- custom commands, 246-256, 256-265
- custom skin, 14-17
- custom sort, 414-427

- CustomAttributeNames, 959
- Customizing Content Area, 563-566
- CustomValidator, 936-938
- Data Editing, 735-736
- data entry, 113-117
- DataBind, 390-402
- DataBind(), 346-350
- DataBinder, 346-350
- DataBinder.Eval, 124-126
- DataBinder.Eval(), 346-350
- Databinding, 675-678, 900-902
- DataBindings, 309-315, 432-437
- data-bound items, 384-386, 386-390
- data-bound tree view, 430-432
- DataField, 812-816
- DataFieldID, 22-29, 309-315, 430-432, 432-437
- DataFieldParentID, 22-29, 309-315, 430-432, 432-437
- DataItem, 346-350
- DataKeyNames, 812-816
- DataListItem, 928-933
- DataMember, 22-29
- DataNavigateUrlField, 22-29, 309-315
- DataRowView, 346-350
- DataSet, 297-298
- DataSource, 22-29, 330-337, 346-350
- DataSourceControl, 297-298
- DataSourceID, 22-29, 298-309, 330-337, 432-437
- DataTable, 297-298
- DataTextField, 22-29, 309-315, 330-337, 340-346, 430-432, 432-437
- DataTextFormatString, 22-29
- DataValueField, 22-29, 309-315, 330-337, 340-346
- DataView, 297-298
- Date Format Dialog, 46-49, 49-57
- Date-Time and Calendar Controls Client-Side Programming, 938-944
- Date-Time and Calendar Controls Designer Interface, 919-928
- Date-Time and Calendar Controls Getting Started, 915-918
- Date-Time and Calendar Controls Server-Side Programming, 928-933
- Date-Time and Calendar Controls Server-Side Walk-through, 933-936
- Date-Time Picker Validation, 936-938

- DateTimeFormatInfo, 938-944
- DayEndTime, 959-963
- DayRender, 928-933
- DayStartTime, 959-963
- DayTemplate, 928-933
- DecoratedControls, 105-110
- DecorationZoneID, 105-110
- DefaultCellPadding, 919-928
- DefaultCellSpacing, 919-928
- DefaultLoadingPanelID, 141-145
- DefaultViewChanged, 928-933
- DefaultViewChangedEventArgs, 928-933, 928-933
- Defining the Markup, 983-986
- deleteAppointment(), 972-978
- DeleteParameters, 322-330
- DeletePaths, 551-562
- Designer Interface, 22-29, 49-57, 105-110, 124-126, 141-145, 236-242, 350-353, 386-390, 432-437, 486-491, 551-562, 857-865
- Different Ways to Show A Notification, 643-645
- DisabledCssClass, 22-29
- DisabledImageUrl, 22-29
- disableEvents, 86-91
- disabling, 60-64
- display mode, 44-46
- DisplayToUtc, 963-966
- DockCommand Collection Editor, 236-242
- DockMode, 231-236, 246-256, 265-280
- DockState, 246-256, 265-280, 265-280, 265-280, 265-280
- Dynamic User Controls for Ajax-Enabling Entire Page, 165-174
- Edit Bindings..., 812-816
- Edit Databindings...., 340-346
- edit mode, 44-46
- Edit Templates, 350-353
- editAppointment(), 972-978
- EditorTool, 551-562
- EditorToolGroup, 551-562
- Embedded Icons, 643
- Empty Message, 49-57
- Empty Values, 865-867

- Enable AJAX, 141-145
- Enable AJAX history, 141-145
- Enable RadEditor Dialogs, 548-551
- Enable Spell Check for RadEditor, 548-551
- Enable update of Page <head> element, 141-145
- EnableAdvancedForm, 983-986
- EnableContextMenu, 432-437, 437-447
- EnableCustomAttributeEditing, 959
- EnableDragAndDrop, 432-437, 437-447, 983-986
- EnableDragAndDropBetweenNodes, 437-447
- EnableEmbeddedSkins, 372-374
- enableEvents, 86-91
- EnableItemCaching, 390-402
- EnableLoadOnDemand, 390-402
- EnableMultiSelect, 919-928
- EnableNavigationAnimation, 919-928
- EnableResize, 551-562
- EnableRowHoverStyle, 812-816
- EnableViewState, 265-280
- EnableVirtualScrolling, 390-402
- enabling, 60-64
- End Template Editing, 350-353
- EndOfItems, 390-402
- ErrorMessage, 936-938
- Eval(), 745-747
- event bubbling, 414-427
- EventDuration, 963-966
- Events, 610
- expand, 458-463, 959-963
- ExpandAnimation, 384-386, 386-390
- ExpandedCssClass, 22-29
- ExpandedImageUrl, 22-29
- ExpandMode, 465-469
- ExportToCalendar(), 966-972
- external content, 236-242, 265-280
- Failed to load viewstate, 165-174
- FastNavigationStyle, 919-928
- Field binding, 340-346
- FileBrowser, 551-562

FileDelete, 570-573
FileSystemContentProvider, 551-562
FileUpload, 570-573
Filter, 386-390
Filtering and Sorting of the TagCloud Items, 678-679
FindControl(), 353-360
findItemByText, 82-83, 402-408
FindItemByValue, 353-360, 402-408
FindItemByValue(), 353-360
findItemsByText, 408-414
findItemsByValue, 408-414
findNodeByAttribute, 458-463
findNodeByText, 458-463
FireFox, 340-346
First Steps, 714-717
FirstDayOfWeek, 919-928, 919-928, 959-963
FitDocks, 231-236, 236-242
Floating, 551-562
focus, 408-414
FocusedCssClass, 22-29
FocusedDate, 919-928
FocusedDateColumn, 919-928
FocusedDateRow, 919-928
FooterTemplate, 390-402
ForeignKeyField, 955-959
FormatString, 309-315
FormCreated, 966-972
FormCreatedEventArgs, 966-972
FormCreating, 966-972
FormCreatingEventArgs, 966-972, 966-972
FormView, 745-747, 758-776
FrameDuration, 128-129
frameset, 340-346
Generating TagCloud from External Sources, 679
get_ajaxSettings(), 148-162
get_allNodes, 458-463
get_count, 82-83
get_domEvent(), 938-944
get_element, 458-463

- get_element(), 938-944
- get_enableAJAX(), 148-162
- get_htmlElement(), 986-987
- get_isSelecting(), 938-944
- get_item, 86-91
- get_items, 82-83, 408-414
- get_level, 86-91
- get_newDate(), 938-944
- get_newValue(), 938-944
- get_nodes, 458-463
- get_oldDate(), 938-944
- get_oldValue(), 938-944
- get_parent, 458-463
- get_popupButton(), 938-944
- get_popupContainer(), 938-944
- get_renderDay(), 938-944
- get_selectedDates(), 938-944
- get_targetSlot(), 972-978
- get_text, 82-83, 86-91
- get_timePopupButton(), 938-944
- get_timePopupContainer(), 938-944
- get_value, 86-91
- getDate(), 938-944, 938-944
- getFullYear(), 938-944
- getItem, 82-83
- getMonth(), 938-944
- GetRadWindow, 265-280
- GetRegisteredDocksState, 246-256, 265-280
- getTimeSlotFromDomElement(), 986-987
- Getting Started, 17-22, 46-49, 100-105, 120-124, 136-141, 231-236, 282-283, 298-309, 340-346, 376-379, 384-386, 430-432, 471-474, 484-486, 497-503, 548-551, 591-593, 607-608, 609-610, 612-613, 621-625, 640-642, 655-657, 672-674, 688-689, 695-696, 701-702, 705-706, 839-857, 891-896
- Getting Started with RadScheduler, 944-955
- Getting-Started, 720-729
- global RadTreeNode template, 432-437
- GridBoundColumn, 496-497, 497-503
- GridButtonColumn, 496-497, 497-503
- GridCheckBoxColumn, 496-497, 497-503
- GridClientSelectColumn, 496-497
- GridColumnCollection, 497-503

GridDateTimeColumn, 496-497
GridDropDownColumn, 496-497, 497-503
GridEditCommandColumn, 496-497, 497-503
GridHTMLEditorColumn, 496-497
GridHyperLinkColumn, 496-497, 497-503
GridLines, 812-816
GridMaskedColumn, 496-497
GridNumericColumn, 496-497
GridTemplateColumn, 497-503
GridTemplateColumns, 496-497
GroupBy, 959-963
GroupingDirection, 959-963
GUID, 983-986
Handle Server-Side Events, 987-991
Handling the Drag and Drop Client-Side, 986-987
HeaderCellRender, 928-933
HeaderTemplate, 390-402
HeaderText, 497-503, 812-816
hierarchical relationships, 432-437
Horizontal, 959-963
HoverImageUrl, 22-29
How This Courseware Is Organized, 2-6
How To, 64-69, 117-119, 265-280, 414-427, 463-465, 479-482, 492-495, 578-587, 886-890
How -to, 698-699
How To Combine Properties, 648-649
How-to, 703
IBindingList, 297-298
ICollection, 297-298
IComparer, 414-427
ID, 80-82
IDynamicControl, 165-174
IEnumerable, 297-298, 322-330, 322-330, 330-337
IList, 297-298
IListSource, 297-298, 297-298
Image, 340-346
ImageManager, 551-562
ImageMap, 117-119
ImageUrl, 22-29, 309-315, 384-386, 430-432
Implement Categories Control, 758-776

Implement CreateExams Control, 789-803
Implement Questions Control, 776-789
Implement the Registration Page, 201-202
Important Information, 1
Important Properties, 379, 599-601, 657-658, 674-675
individual item templates, 432-437
Initial delay time, 141-145
InitialItemIndex, 124-126
Inline, 141-145
InlineInsertTemplate, 978-981
Input Mask Dialog, 46-49, 49-57
InsertAppointment, 963-966
InsertSnippet, 570-573
Integrate the Exam Scheduler, 991
IntelliSense, 29-37, 80-82, 86-91
internationalize, 14-17
Introducing RadControls, 6-13
Introduction, 14-17, 44-46, 80, 99-100, 120, 135-136, 177, 226-231, 281-282, 297-298, 338-340, 362, 375-376, 383-384, 429-430, 471, 483-484, 496-497, 547-548, 588, 590-591, 606-607, 609, 621, 640, 651, 654-655, 672, 688, 693-695, 700-701, 705, 711, 720, 745-747, 839, 891
Introduction and Overview, 661
Is Sticky, 141-145
IsAjaxRequest, 145-148
IsCallback, 390-402
IsLogarithmic, 865-867, 865-867
IsPostBack, 390-402
IsSeparator, 22-29, 384-386
Item Builder, 22-29
ItemClick, 322-330
ItemCreated, 928-933
ItemDataBound, 309-315, 928-933
ItemHeight, 124-126
Items, 22-29
Items Drag and Drop, 740-741
Items property, 128-129
ItemsRequested, 390-402, 408-414
ItemTemplate, 128-129, 390-402, 414-427, 812-816
ItemWidth, 124-126
JavaScript, 29-37, 360-361
JavaScript Intellisense, 83-85

- JSON, 903, 913
- JSON: Fat-Free Data Interchange, 95-98
- Known Issues, 671
- Language Integrated Query, 330-337
- LastDayOfWeek, 959-963
- Level, 447-458
- like, 496
- LinkButton, 983-986
- LINQ, 330-337
- LINQ to SQL Classes, 330-337
- LinqDataSource, 297-298, 330-337
- Load On Demand, 741-742
- LoadDockLayout, 246-256, 265-280
- LoadingMessage, 390-402
- LoadingPanelID, 136-141
- Load-on-demand, 390-402, 408-414, 414-427, 465-469
- LoadUserControl(), 165-174
- LoadViewState(), 163-165
- Localization, 615-616, 959-963
- LogarithmBase, 865-867
- Logarithmic Y-Axis, 865-867, 865-867
- look-and-feel, 49-57, 105-110
- MarkFirstMatch, 384-386, 386-390
- Mask, 49-57
- MaskPart Collection Editor, 49-57
- MasterPage, 162-163
- MasterPage/Content Page, 174-176
- MasterTableView, 350-353, 497-503, 812-816
- MaxBindDepth, 309-315
- MaxDataBindDepth, 22-29
- MaxDate, 919-928
- MaxOccurrences, 963-966
- MaxUploadFileSize, 551-562
- medium, 141-145
- Min display time, 141-145
- MinDate, 919-928
- minimize zones, 242-246
- Modal, 113-117
- modal dialogs, 265-280

ModifiedAppointment, 966-972
Modules, 551-562
MonthView, 928-933, 944-955
MS AJAX Library, 98
multi-column combo box, 390-402
Multi-Line Labels, 865-867
MultiPageID, 915-918, 933-936
MultiSelect, 919-928
MultiViewColumns, 919-928
MultiViewRows, 919-928
Naming Conventions, 85-86
navigateToDate(), 938-944
NavigateUrl, 22-29, 29-37, 86-91, 236-242, 256-265
NavigationCommand, 966-972
NavigationComplete, 966-972
NewDate, 928-933
NewLineBr, 551-562
NodeCheck event, 437-447
NodeClick event, 447-458
NodeDrop, 987-991
NodeDrop event, 437-447
NodeEdit event, 447-458
NodeExpand event, 465-469
Nodes, 432-437, 447-458
Northwind.mdf, 309-315, 309-315, 309-315, 309-315
Notification Menu, 642-643
NumberOfItems, 390-402
Numeric Type, 49-57
ObjectDataSource, 297-298, 322-330, 330-337
Objectives,
14, 44, 80, 99, 120, 135, 177, 226, 281, 297, 338, 362, 375, 383, 429, 471, 483, 496, 547, 588, 590, 606,
Office, 14-17
OldDate, 928-933
OnAjaxRequest, 148-162
OnClick, 867-882
OnClientAppointmentContextMenu, 972-978
OnClientCommand, 256-265
OnClientContextMenuItemClicking, 437-447
OnClientDockPositionChanged, 256-265

OnClientDropDownClosed, 408-414
OnClientDropDownOpened, 408-414
OnClientDropDownOpening, 414-427
OnClientFocus, 408-414
OnClientItemClicked, 86-91
OnClientItemClicking, 86-91
OnClientItemsRequested, 408-414
OnClientItemsRequestFailed, 408-414
OnClientItemsRequesting, 408-414
OnClientMouseOut, 91-95
OnClientMouseOver, 91-95
OnClientNodeChecked, 458-463
OnClientNodeDropping, 437-447, 983-986
OnClientSelectedIndexChanged, 390-402, 408-414
OnClientShowing, 86-91
OnClientTimeSlotClick, 972-978
OnColumnHeaderClick, 938-944
OnDateClick, 938-944
OnDateSelected, 938-944
OnDateSelecting, 938-944
OnError, 60-64
onfocus, 458-463
OnNodeDrop, 983-986
OnPreRender, 987-991
OnResponseEnd, 148-162
OnRowHeaderClick, 938-944
OnRowSelected, 812-816
OnValueChanged, 60-64
OnValueChanging, 60-64
OnViewSelectorClick, 938-944
Optimize for Maximum Security, 658
Orientation, 236-242, 919-928
OverflowBehavior, 959-963
Page Lifecycle, 163-165
Page vs MasterPage vs UserControl, 162-163
Page_Init, 265-280
Page_Load, 165-174
PageSize, 812-816
PageTop, 551-562

- panes, 226-231
- Parameter Collection Editor, 322-330
- ParentNode, 447-458
- pause(), 126-127
- Performance, 465-469
- Placeholder, 165-174
- PlotArea, 865-867
- Populating Plain Text And Rich Content, 645-647
- pop-up window, 226-231
- pop-up windows, 242-246, 256-265
- PopupClosing, 938-944
- PopupOpening, 938-944
- portal page, 265-280
- portal sites, 226-231
- PresentationType, 915-918, 919-928
- printing, 256-265
- Property Builder, 22-29
- ProviderName, 959-963
- RadAjaxLoadingPanel, 135-136
- RadAjaxManager, 22-29, 135-136, 136-141, 141-145, 145-148, 148-162, 174-176
- RadAjaxManagerProxy, 135-136, 141-145, 174-176
- RadAjaxPanel, 135-136, 136-141, 141-145
- radalert(), 256-265
- RadButton as a Toggle Button, 596-599
- RadButton as an Image Button, 594-596
- RadCalendar, 136-141, 174-176, 915-918, 918-919, 933-936
- RadCalendarDay, 928-933
- RadChart, 297-298, 903, 903, 903, 913
- RadComboBox, 297-298, 330-337, 350-353, 383-384, 384-386, 745-747
- RadComboBox Item Builder, 386-390
- RadComboBoxContext, 414-427
- RadComboBoxData, 414-427
- RadComboBoxItemData, 414-427
- radconfirm(), 256-265
- RadContextMenu, 429-430
- RadDataPager, 637-639
- RadDate, 928-933
- RadDateInput, 44-46, 46-49, 928-933
- RadDatePicker, 915-918, 918-919, 919-928, 933-936

RadDateTimePicker, 915-918, 918-919, 919-928
RadDock, 14-17, 226-231, 231-236, 236-242, 236-242, 246-256, 256-265, 265-280
RadDock Collection Editor, 236-242
RadDockLayout, 236-242, 246-256, 265-280
RadDockZone, 226-231, 231-236, 236-242, 246-256
RadFormDecorator, 99-100, 100-105, 105-110, 174-176, 915-918
RadGrid, 297-298, 322-330, 338-340, 350-353, 745-747, 812-816
RadInputManager, 69-79
radio button pattern, 458-463
RadMaskedTextBox, 44-46, 46-49
RadMenu, 82-83
RadMenuItem, 29-37
RadMultiPage, 915-918
RadNumericTextBox, 44-46, 46-49, 353-360
RadPageView, 350-353
RadPane, 231-236, 236-242, 256-265
RadPanelBar, 322-330, 322-330, 338-340, 340-346, 340-346, 346-350
RadPivotGrid Fields, 706-710
radprompt(), 256-265
RadRotator, 120-124, 124-126, 350-353
RadScheduler, 297-298, 959-963
RadScriptManager, 22-29, 80-82
RadSlidingPane, 242-246
RadSlidingZone, 226-231, 242-246
RadSpell, 64-69
RadSplitBar, 231-236, 236-242
RadSplitter, 226-231, 231-236, 236-242, 265-280
RadStyleSheetManager, 22-29
RadTabStrip, 91-95, 165-174, 330-337, 915-918
RadTextBox, 44-46, 46-49, 340-346, 745-747, 758-776
RadTimePicker, 918-919, 919-928
RadTimeView, 928-933
RadToolBar, 22-29, 297-298, 350-353, 758-776
RadToolBar Item Builder, 22-29
RadToolBarButton, 22-29
RadToolBarDropDown, 22-29
RadToolBarSplitButton, 22-29
RadToolTip, 99-100, 100-105, 105-110, 110-113, 113-117, 117-119
RadToolTipManager, 99-100, 100-105, 105-110, 110-113

RadTreeNode, 458-463
RadTreeView, 309-315, 350-353, 429-430, 430-432, 432-437, 437-447, 447-458, 463-465, 465-469, 745-747
RadTreeView Item Builder, 432-437
RadTreeViewContextMenu Collection Editor, 432-437
RadWindow, 226-231, 231-236, 236-242, 242-246, 256-265, 265-280
RadWindow Collection Editor, 236-242
RadWindowManager, 226-231, 231-236, 236-242, 256-265
RangeMaxDate, 919-928
RangeMinDate, 919-928
RecurrenceRule, 963-966
Referencing RadControl Client Objects, 80-82
Refresh Schema, 22-29
Registering and Assigning Skins, 283-288
related tables, 447-458
Remove(), 29-37
RemoveAt(), 29-37
RenderInColumns, 919-928
RenderInRows, 919-928
requestItems, 414-427
RequestQueueSize, 141-145
RequiredFieldValidator, 936-938
Resource Types, 959-963
ResourceType, 963-966
Response.Write(), 360-361
ResponseScripts, 145-148
RestoreOriginalRenderDelegate, 141-145
Ribbon Bar, 14-17
RibbonBar and Editor, 567-570
Right-to-left, 14-17
RotatorType, 124-126, 126-127
RoundToNearestHour(), 966-972
Rows, 545-546
Save a Thumbnail, 617-620
SaveDockLayout, 246-256, 265-280
Scale Breaks, 865-867
ScaleBreaks, 865-867
Scheduler Client-Side Programming, 972-978
Scheduler Designer Interface, 959-963

- Scheduler Resources, 955-959
- Scheduler Server-Side Events, 966-972
- Scheduler Server-Side Programming, 963-966
- SchedulerFormMode, 966-972
- ScriptManager, 22-29, 80-82, 91-95, 165-174, 346-350
- Scroll, 959-963
- Scrolling, 739-740
- scrollIntoView, 458-463
- SearchPatterns, 551-562
- select, 458-463
- SelectCommand, 812-816
- SelectCommandType, 812-816
- selectDate(), 938-944
- selectDates(), 938-944
- Selected property, 402-408
- SelectedDate, 919-928, 933-936, 944-955, 959-963
- SelectedDateChanged, 928-933
- SelectedDateChangedEventArgs, 928-933
- SelectedDates, 928-933
- SelectedDatesEventArgs, 928-933
- SelectedDay, 966-972
- SelectedDayStyle, 915-918
- SelectedImageUrl, 22-29
- SelectedIndex, 402-408
- SelectedIndexChanged, 402-408
- SelectedView, 944-955, 959-963
- Selecting, 812-816
- SelectionChanged, 928-933
- semantic rendering, 14-17
- sender, 86-91
- Server Side Code, 514-525, 628-637
- Server Side Programming, 447-458, 491-492
- Server Tags, 29-37
- server-side events, 447-458
- Server-Side Programming, 29-37, 57-60, 110-113, 145-148, 246-256, 315-322, 353-360, 380-381, 402-408, 475-478, 570-573, 671, 689-691, 696-697, 703, 867-882
- set_allowDelete(), 972-978
- set_allowEdit(), 972-978
- set_allowInsert(), 972-978

- set_cancel, 86-91
- set_cancelCalendarSynchronization(), 938-944
- set_displayDeleteConfirmation(), 972-978
- set_enableAJAX(), 148-162
- Setting Up the Database, 178-181
- Setup ActiveSkill Project Structure, 177-178
- Show More Results box, 390-402
- show(), 256-265
- ShowColumnHeaders, 174-176
- ShowEvent, 105-110
- showInsertFormAt(targetSlot), 972-978
- ShowMoreResultsBox, 390-402
- ShowOnFocus, 551-562
- ShowOtherMonthDays, 174-176, 919-928
- showPopup(), 938-944
- ShowResourceHeaders, 959-963
- ShowRowHeaders, 174-176
- ShowStatusBar, 497-503
- ShowViewTabs, 983-986
- SingleExpandPath, 430-432, 432-437
- SingleViewColumns, 919-928
- SingleViewRows, 919-928
- Skin, 174-176, 236-242, 256-265, 372-374
- skins, 14-17
- sliding panes, 226-231, 242-246
- sliding zone, 226-231
- sliding zones, 242-246
- Smart Tag, 22-29, 49-57, 236-242, 386-390, 432-437
- Sort method, 414-427
- sorting items, 402-408
- SortItems, 402-408, 414-427
- SpecialDays, 919-928
- Specifying RadButton Icons, 593-594
- Specifying the content of a RadButton , 604-605
- Spell checking, 64-69
- splitter, 226-231, 265-280
- SqlDataSource, 297-298, 346-350, 812-816, 983-986
- startAutoPlay(), 126-127
- StartEditingAdvancedForm, 983-986

- StateBags, 163-165
- statically declared items, 384-386, 386-390, 390-402
- statically declared nodes, 430-432
- Sticky, 105-110
- StoredProcedure, 812-816
- Strict Mode, 865-867
- StripFormattingOptions, 551-562
- style sheets, 256-265
- substring, 86-91
- Summary, 42-43, 79, 98, 119, 134, 176, 225, 280, 296, 337, 361, 374, 382, 427-428, 469-470, 482, 495, 537-538, 587, 610, 639, 652-653, 692, 699, 703-704, 710, 739, 803-804, 838, 890, 913, 981-982, 991
- Supported Scenarios, 661-662
- System.Random, 148-162
- System.Web.UI.HtmlControls, 148-162
- TabClick, 165-174
- TableCell, 928-933
- Tabs, 22-29, 933-936
- Tag, 265-280
- Target, 22-29, 29-37
- target attribute, 256-265
- target element, 105-110
- TargetControls, 110-113
- Telerik.mdf, 309-315
- Telerik.Web.UI.Scheduler.Views, 987-991
- template, 120-124
- template design surface, 124-126, 432-437
- templates, 14-17, 390-402
- TextBlock, 865-867
- TextChanged, 57-60, 165-174, 402-408, 570-573
- The CreateUserWizardWrapper Code-Behind, 202-204
- The CreateUserWizardWrapperUI, 204-210
- Thumbnails Mode, 474-475
- TimePickerEventArgs, 928-933
- TimePopupButton, 928-933
- TimeSlotCreated, 966-972
- tool window, 265-280
- ToolBarMode, 551-562
- ToolProviderID, 551-562
- Tools, 551-562

ToolsFile, 551-562
ToolTip, 309-315, 346-350, 384-386, 430-432
ToolTipTargetControl Collection Editor, 105-110
ToolTipZoneID, 100-105
Tour of Date-Time and Calendar Controls, 918-919
trackChanges, 408-414
trackChanges(), 437-447
TrackViewState(), 163-165
Transparency, 141-145
traversing the node hierarchy, 447-458, 458-463
TriStateCheckBoxes, 437-447
Understanding the Skin CSS File, 288-292
UNIQUEIDENTIFIER, 983-986
UniqueName, 265-280, 812-816
unselectDate(), 938-944
unselectDates(), 938-944
UpdatePanel, 110-113
UpdatePanelsRenderMode, 141-145
Upload Modules, 379-380
UploadPaths, 551-562
UseColumnHeadersAsSelectors, 919-928
UserControls, 162-163
UseRowHeadersAsSelectors, 919-928
UserSelectable, 959-963
Using Client Events, 86-91
Using RadAjaxManagerProxy, 174-176
Using RadCompression, 651-652
Using RadControl Client Properties and Methods, 82-83
Using Scheduler Templates, 978-981
Using The Configurator, 712-714
Using the Design Time Interface, 503-514, 625-628
Using the design-time interface, 729-735
Using the NewLineMode Property, 562-563
Using Third Party Buttons, 718-719
UtcToDisplay, 963-966
validation, 64-69
ValidationGroup, 64-69, 936-938
ValidationSummary, 936-938
validators, 64-69

- Value, 49-57
- ValueChanging, 938-944
- Vertical, 959-963
- ViewPaths, 551-562
- ViewState, 163-165
- Virtual Scrolling, 390-402, 414-427
- Visible, 812-816
- Visual Studio 2008, 86-91
- Web service, 414-427, 465-469
- Web User Control, 165-174
- WebServiceSettings, 414-427
- WebUserControl, 174-176
- What Do You Need To Have Before You Read This Courseware?, 1
- What Do You Need To Know Before Reading This Courseware?, 1-2
- Who Should Read This Courseware, 1
- WorkDayEndTime, 959-963
- WorkDayStartTime, 959-963
- WrapFrames, 124-126
- wrapping text, 463-465
- XmlDataSource, 297-298
- XPath, 124-126
- YAxis, 865-867
- YAxis2, 865-867