RadControls for Winforms

Acknowledgements

Publisher
Telerik Inc.

Managing Editor
Noel Rice

Authors
Noel Rice
Rachel Hagerman

Cover Design
Matt Kurvin

Special Thanks to:
Stefan Stefanov
Vassil Petev
Iordan Pavlov
Boyko Markov
Todd Anglin
Lino Tadros
Gary Campbell

RadControls for Winforms

Table of Contents

1. Foreword 1
2. Introduction 2
   - 2.1. Who Should Read This Courseware 2
   - 2.2. What Do You Need To Have Before You Read This Courseware? 2
   - 2.3. What Do You Need To Know Before Reading This Courseware? 2
   - 2.4. How This Courseware Is Organized 2-5
   - 2.5. Introducing RadControls 5-11
   - 2.6. Before You Begin... 11-12
3. Buttons 13
   - 3.1. Objectives 13
   - 3.2. Introduction 13
   - 3.3. Getting Started 13-22
   - 3.4. Using the Design Time Interface 22-24
   - 3.5. Buttons Programming 24-37
   - 3.6. Summary 37
4. Editors 38
   - 4.1. Objectives 38
   - 4.2. Introduction 38-39
   - 4.3. Getting Started 39-45
   - 4.4. Using the Design Time Interface 45-46
   - 4.5. Working With the Editor Controls 46-79
   - 4.6. Summary 79
5. RichTextBox Beta 80
   - 5.1. Overview 80-81
   - 5.2. F.A.Q 81-83
   - 5.3. Getting Started 83-85
   - 5.4. Working with RichTextBox 85-86
   - 5.5. Layout Modes 86-87

5.6. Selection 87-88
5.7. Search 88-91
5.8. Positioning 91-92
5.9. History 92-93
5.10. ReadOnly 93
5.11. Formatting API 93
5.12. Import/Export 93-97
5.13. HTML Support 97-98
5.14. Spellcheck 98-101
5.15. RibbonBar UI 101-103
6. Menus 104
6.1. Objectives 104
6.2. Introduction 104-105
6.3. Getting Started 105-110
6.4. Using the Design Time Interface 110-112
6.5. Working with the Menu Controls 112-129
6.6. RadRibbonBar and RadApplicationMenu 129-139
6.7. RadRibbonBar Walk Through 139-143
6.8. Summary 143
7. Telerik Presentation Foundation 144
7.1. Objectives 144
7.2. Introduction 144-145
7.3. Getting Started 145-148
7.4. Class Hierarchy 148-149
7.5. Primitives 149-154
7.6. Elements 154
7.7. Programming the Telerik Presentation Foundation 154-164
7.8. Summary 164
8. PageView 165
8.1. Objectives 165

8.2. Introduction 165-172
8.3. Getting Started 172-180
8.4. Using the Design Time Interface 180-182
8.5. Working with RadPageView 182-190
8.6. Working with RadPageViewPages 190-194
8.7. Programming RadPageView 194-208
8.8. Summary 208-209
9. ListControl 210
9.1. Objectives 210
9.2. Introduction 210-211
9.3. Getting Started 211-215
9.4. Using the Design Time Interface 215-216
9.5. Programming ListControl and DropDownList 216-246
9.6. Summary 246
10. ListView 247
10.1. Objectives 247
10.2. Introduction 247-248
10.3. GettingStarted 248-256
10.4. Using the Design Time Interface 256-264
10.5. Programming ListView 264-277
10.6. Summary 277
11. PropertyGrid 279
11.1. Objectives 279
11.2. Introduction 279-280
11.3. Getting Started 280-284
11.4. Using the Design Time Interface 284-286
11.5. Programming 286-293
11.6. Summary 293
12. Data Binding 294
12.1. Objectives 294

12.2. Introduction 294-304
12.3. Getting Started 304-312
12.4. Binding to Simple Controls Using DataBindings 312-314
12.5. Using BindingSource 314-315
12.6. Binding to Business Objects 315-320
12.7. Binding to Services 320-322
12.8. Binding to LINQ Data Sources 322-325
12.9. Binding Image Data 325-328
12.10. Binding RadGridView 328-333
12.11. Summary 333
13. User Feedback 334
13.1. Objectives 334
13.2. Introduction 334-335
13.3. Getting Started 335-340
13.4. Using the Design Time Interface 340-344
13.5. Programming the User Feedback Controls 344-365
13.6. Summary 365-366
14. CommandBar 367
14.1. Objectives 367
14.2. Introduction 367-369
14.3. Getting Started 369-375
14.4. Using the Design Time Interface 375-379
14.5. Programming 379-384
14.6. Summary 384
15. Forms 385
15.1. Objectives 385
15.2. Introduction 385-386
15.3. Getting Started 386-390
15.4. Using the Design Time Interface 390
15.5. Programming Rad Form and Message Box 390-410

15.6. Summary 410
16. Screen "Real Estate" 411
16.1. Objectives 411
16.2. Introduction 411-414
16.3. Getting Started 414-418
16.4. Using the Design Time Interface 418-422
16.5. Programming the "Real Estate" Controls 422-444
16.6. Summary 444
17. Calendar 445
17.1. Objectives 445
17.2. Introduction 445-446
17.3. Getting Started 446-447
17.4. Using the Design Time Interface 447-451
17.5. Programming the Calendar 451-462
17.6. Summary 462
18. Shortcuts 463
18.1. Objectives 463
18.2. Introduction 463-464
18.3. Creating Shortcuts 464-466
18.4. Global Shortcuts 466-469
19. Scheduler 470
19.1. Objectives 470
19.2. Introduction 470-474
19.3. Getting Started 474-480
19.4. Using the Design Time Interface 480-481
19.5. Programming the Scheduler 481-501
19.6. Summary 501
20. Chart 502
20.1. Objectives 502
20.2. Introduction 502

20.3. Getting Started 502-520
20.4. Using the Design Time Interface 520-527
20.5. Programming the Chart 527-547
20.6. Unique RadChart Features 547-550
20.7. How To 550-556
20.8. Summary 556
21. Grid 557
21.1. Objectives 557
21.2. Introduction 557-558
21.3. Getting Started 558-560
21.4. Column Manipulation 560-563
21.5. Column Chooser 563-564
21.6. Conditional Formatting 564-569
21.7. Cell and Row Formatting 569-573
21.8. Custom Cells and Elements 573-581
21.9. Grouping, Sorting and Filtering 581-592
21.10. Hierarchy Support 592-625
21.11. Virtual Mode 625-630
21.12. Using RadGridView with Dynamic LINQ Query Library 630-635
21.13. Expression Editor 635-641
21.14. Exporting RadGridView Data 641-653
21.15. Summary 653-654
22. Multi-Column ComboBox 655
22.1. Objectives 655
22.2. Introduction 655
22.3. Getting Started 655-659
22.4. Programming the Multi Column Combo Box 659-662
22.5. Summary 662
23. Tree 663
23.1. Objectives 663

23.2. Introduction 663-665
23.3. Getting Started 665-669
23.4. Using the Design Time Interface 669-672
23.5. Programming the Tree View 672-731
23.6. Summary 731
24. Wizard 732
24.1. Objectives 732
24.2. Introduction 732-734
24.3. Using the Design Time Interface 734-735
24.4. Getting Started 735-745
24.5. Summary 745
25. Appearance and Styling 746
25.1. Objectives 746
25.2. Introduction 746
25.3. Using RadElements for Styling 746-749
25.4. HTML-Like Text Formatting 749-750
25.5. Themes 750-751
25.6. Visual Style Builder 751-752
25.7. Color Blending 752-755
25.8. Summary 755

1 Foreword

This is the fifth courseware Falafel has written for Telerik, the first two being on RadControls for ASP.NET and Telerik Reporting. This hands-on training material will be of great help to individuals, teams, and the community as a whole, and will assist them in quickly learning the power of RadControls and experience their true value.

It has been a fantastic experience to work with Falafel Software on producing the Telerik RadControls for WinForms courseware, and they have once again proved that they are skilled professionals, who are ready to go to great extends for their customers. On behalf of the whole Telerik team, I would like to personally thank Noel Rice for his kindliness, understanding, and most of all attention to detail – without him we would not have been able to produce this thorough and extensive courseware on RadControls for WinForms.

I hope you enjoy this reading and that it increases your proficiency in RadControls, bringing your valuable projects to a great success.

Nikolay Diyanov
Unit Manager, RadControls for WinForms
Telerik Inc.

2 Introduction

2.1 Who Should Read This Courseware

You should read this courseware if:

- You have used standard Microsoft WinForms controls and want to learn the Telerik RadControls for WinForms approach to application building.
- You have used RadControls for WinForms and want to make your knowledge more comprehensive.

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

Computer Setup

- Windows XP Professional or Windows Vista Service Pack 2.
- Microsoft .NET Framework 3.5.
- Microsoft Visual Studio 2008
- RadControls for WinForms. You can purchase RadControls for WinForms from:

  http://www.telerik.com/purchase.aspx

or download the trial from:

http://www.telerik.com/products/winforms.aspx

Learn more about **system requirements for RadControls for WinForms here (http://www.telerik.com/products/winforms/resources/system-requirements.aspx)**.

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

This courseware assumes that you are familiar with Windows Forms programming using either VB.NET or C# code. The courseware uses Visual Studio 2008 and assumes you now your way around this environment. You should be able to navigate the basic functional areas of the IDE (e.g. Solution Explorer, Properties, code/designer for windows forms, etc.) and be able to run and debug applications and class libraries.

2.4 How This Courseware Is Organized

Courseware Chapter Organization

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 programmatic management of the control.
- A brief review of the objectives that were accomplished.

Chapter Summary

Buttons

This chapter tours various types of super-charged Telerik buttons available in the RadControls for Winforms suite. You will learn the basic characteristics, behavior and important properties of each button type. You will use several design time interfaces to configure the buttons, work with the Items collections of both RadDropDownButton and RadSplitButton and respond to events both from the buttons and from individual Items collection members. Finally, you will learn how to manipulate button ToggleState and ToggleState related events.

Editors

This chapter shows how to retrieve information from the user with the RadSpinEditor, RadDateTimePicker, RadMaskedEditBox, RadTextBox and RadColorDialog controls. The chapter begins with a brief exploration of basic usability for each control and highlights special features. The chapter includes explanation how to implement tool tips, validate user entry and control specific information on edit masks, date formats and internationalization support.

Menus

You will learn how to add drop-down and context menus to your application using RadMenu, RadContextMenu and RadContextMenuManager controls. We begin with a tour of usability and feature highlights, then learn how to build menus at design-time using the Menu Designer. You will learn common menu item collection programmatic tasks such as adding and removing menu items, locating and modifying menu items. The unique aspects of RadContextMenu are discussed including programmatically popping up the context menu in a specific screen location. Finally, the chapter introduces the RadRibbonBar and RadApplicationMenu controls.

Telerik Presentation Foundation

In this chapter you will learn some of the capabilities, architecture and class hierarchy of the Telerik Presentation Framework. We will pay special attention to the classes used directly in the controls, i.e. RadElement and RadControl. We will take a brief look at the structure, WPF resemblance and how the elements interact. You will see how the elements can be accessed within RadControls and combined to create unique new controls. You will also learn how to add any Control descendant using RadHostItem as a wrapper.

Page View

This chapter explores tabbed interfaces and managing controls within associated content panels. You will perform common Items collection programmatic tasks such as adding and removing tabs, iterating, locating and selecting tabs. The chapter also explains how to implement handle layout and dimensions, overflow, tab images and text.

List Controls

This chapter explores creating tabbed and paneled interfaces and managing controls within associated content panels. The chapter starts with a tour of usability and feature highlights then demonstrates how to perform common Items collection programmatic tasks such as adding and removing tabs, iterating, locating and selecting tabs. The chapter also explains how to implement tab drag and drop, handle layout and dimensions, overflow, tab images and text.

Data Binding

In this chapter you will learn which controls have DataSource properties and the bindable types that can be assigned to them. The chapter starts with examples of binding to simple arrays and lists of objects, then moves on to use BindingSource to bind to database, business objects and web services. The chapter also shows how to bind simple controls using the DataBindings property and how to bind LINQ data sources.

User Feedback

Learn how to provide user feedback using RadStatusStrip, RadTrackBar, RadWaitingBar, RadProgressBar and RadLabel controls. You will also learn the appropriate circumstances to use a RadWaitingBar vs a RadProgressBar, how to use the RadStatusStrip at design time and programmatically and how to host hosted a

control inside a status strip. Finally, you will learn how to coordinate long running background processes with the user interface.

Command Bar

This chapter shows how to create tool strips that can be built up from multiple RadControls, dragged and docked within your user interface, resized, re-arranged, and customized by the user. The chapter explains the basic structure of the tool strip and how to include elements within the strip items.

Forms

This chapter shows how the special RadForm, ShapedForm and RadRibbonForm classes can be inherited from to create themeable, custom shaped forms to complete and polish the look-and-feel of an application. You will use the new project item templates to create instances of each form class. The chapter demonstrates how RadTitleBar is used as a replacement for the built-in Windows form title bar. Finally, the chapter introduces the RadMessage box as a themable replacement for the standard MessageBox.

Screen "Real Estate"

This chapter shows some stylish and unique ways of organizing form space using RadDoock to handle dockable windows and tabbed documents, RadCarousel to animate a series of clickable images and RadRotator to display constantly changing personalized content.

Calendar

This chapter focuses on the RadCalendar and includes how to use the extensive customization features to completely tailor the calendar header, calendar footer, number of displayed months, the number of rows and columns, date ranges, selected and focused dates, the first day of the week as well as the title and navigation areas of the calendar. The chapter also explains how to work with the “special days” collection to mark specific days or repeating events.

Shortcuts

This chapter explains how input bindings are created and initiated by RadControl actions. The chapter demonstrates creating and assigning bindings programmatically.

Scheduler

In this chapter you will learn how to work with RadScheduler, to work with appointments, reminders and views. Additionally, you will learn how to customize the appearance of the control components.

Chart

In this chapter you will build a simple chart with static items and also bind data to the chart. The chapter tours the basic RadChart elements as well as the types of charts that are available. You will learn how to use the tools in the designer to help navigate the many RadChart capabilities. You will create and configure many of the chart elements programmatically, including the chart series, items, legend and chart title.

Grid

This chapter explores the RadGridView control including design time support using the Smart Tag, Property Builder and Properties window to configure the grid and to bind the grid to data. You will learn how to add special purpose column types at design-time and in code, how to group/filter/sort data based on user input at runtime, by hand at design-time and programmatically in code. You will display hierarchical data from multiple related tables in the grid. You will use the RadGridView Virtual Mode feature to take low-level control over the grid's refresh process. You will learn how to bind RadGridView to LINQ data sources for lightening fast performance. Finally you will use multiple methods of exporting grid data to Excel and out through the Telerik Reporting engine.

Multi-Column ComboBox

This chapter explores the ways to work with and to customize the powerfull grid in the drop down and the drop down itself

Tree

This chapter shows how to display, navigate and manage hierarchical data using the RadTreeView control. RadBreadCrumb is introduced to help the user keep track of their location in deeply nested hierarchical structures. The chapter includes topics on tree view support for implementing drag-and-drop and context menus. RadTreeView specific data binding issues include binding to self referencing data, binding to related data, writing to and from XML and load-on-demand. The chapter covers techniques for adding and removing nodes programmatically, iterating and locating nodes, editing nodes and reordering nodes.

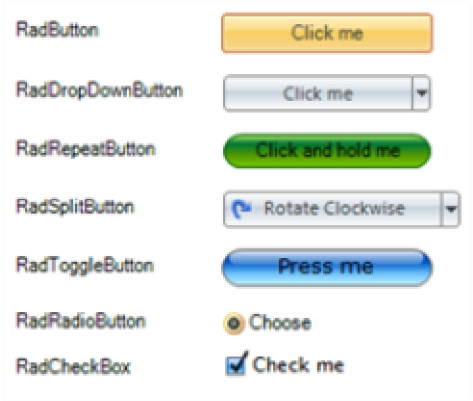
Wizard

This chapter reveils how with a few click you can create your very own step-by-step wizard to make your installations with less efforts and with better appearance and functionality.

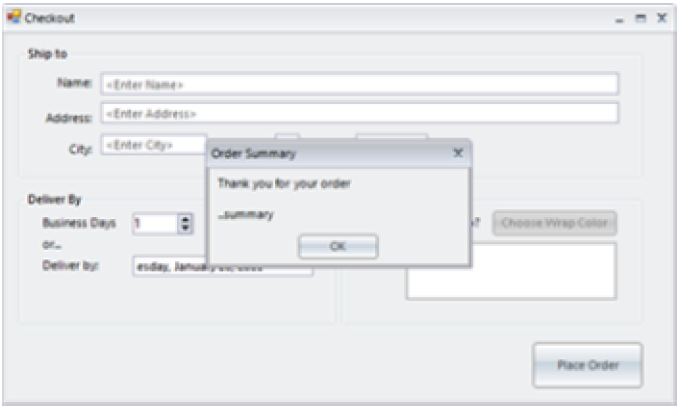
Appearance and Styling

This chapter briefly describes the concepts used to theme RadControls and provides a short description of the Visual Style Builder tool. The chapter also includes a section on “Styling text”, covering our HTML-like styling, text indentation and justification.

2.5 Introducing RadControls

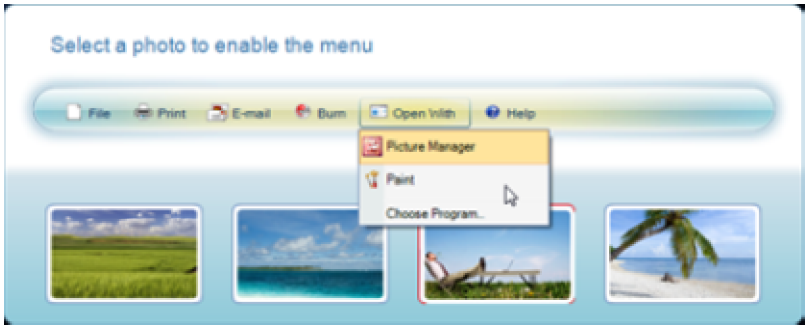


Telerik themable **buttons** replace their standard WinForms counterparts and add robust data binding, state management and design options. Developed to be very similar to the existing Windows Forms controls, they also allow you to take advantage of the features of RadControls, such as themes, rotation and transparency, images and text.

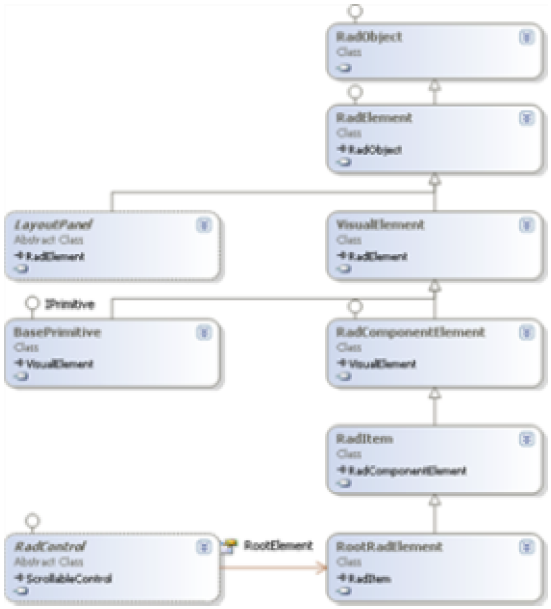


RadControls for Winforms **edit controls** are highly customizable, themable, intelligent components for controlled data input in Windows Forms applications that include text boxes, masked edit boxes, date time pickers, spin editor and a color dialog.

RadControls for Winforms



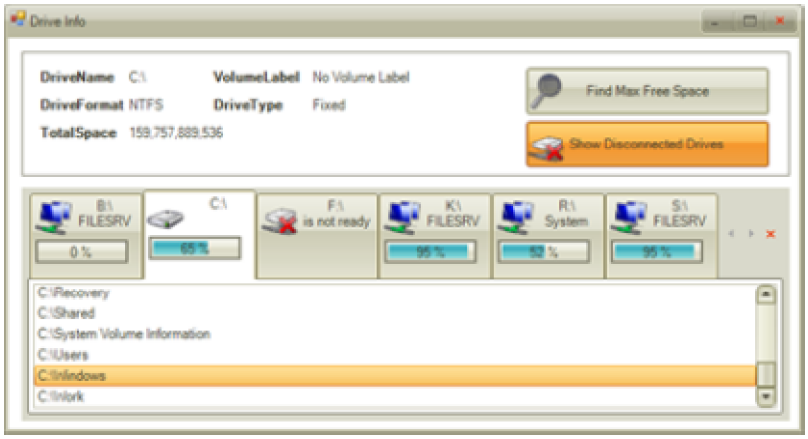
Telerik has an array of flexible **menuing** options that help structure user choices within your application into attractive navigation systems. Based on the Telerik Presentation Foundation, the components gives you full control over item appearance, orientation, and text-wrapping, while allowing you to easily nest checkboxes, images, or any other controls in design-time. The controls include RadMenu, RadContextMenu and RadRibbonBar.



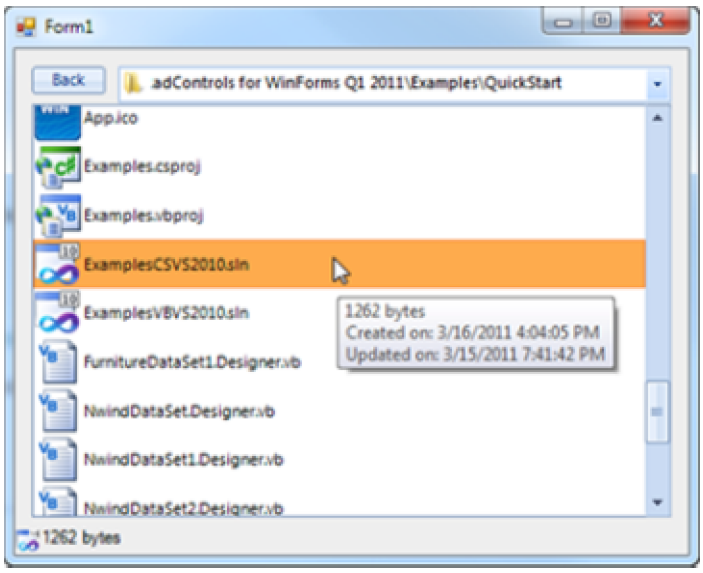
The **Telerik Presentation Foundation** (TPF) offers Windows Presentation Foundation (WPF) features within classic Windows Forms applications. TPF is the foundation for all RadControls for Winforms controls and objects.

All controls have access to common services provided by TPF such as theming, animation, layout and property binding. No matter how complex a control may be, a control is built up from a few simple types of elements. By working with these elements, you can customize any control on a very granular level.

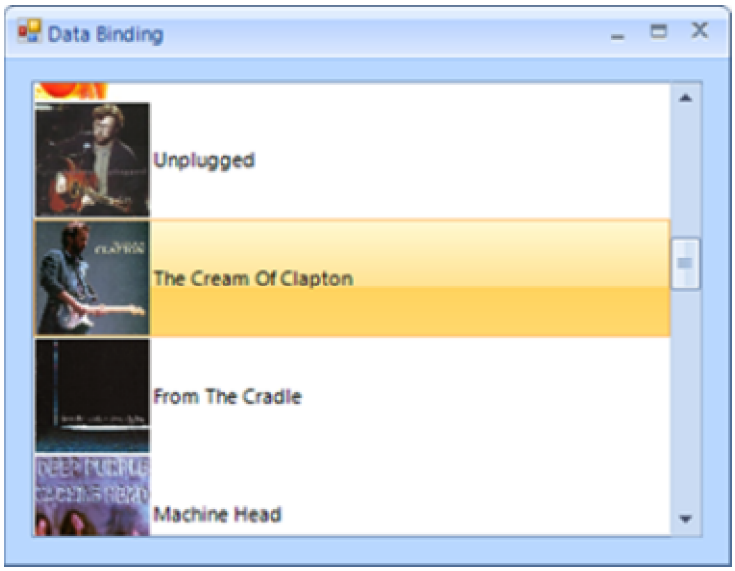
All elements in a control element tree inherit common property values from their parent elements unless these values are set locally. This feature helps each element use less memory while still providing fine-grained customization options.



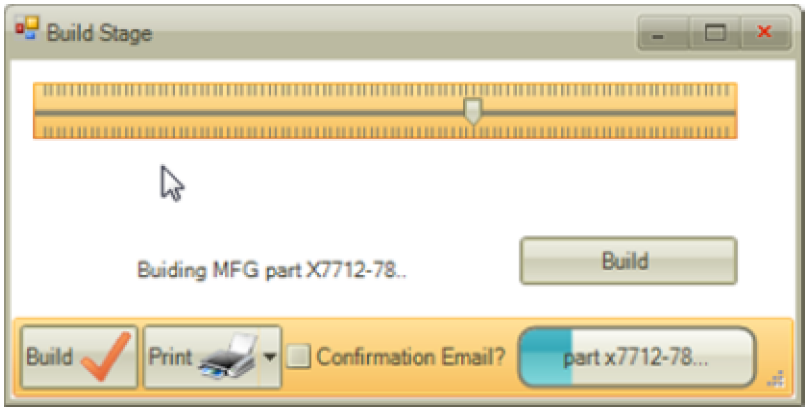
RadPageView supply intuitive navigation between application areas. **RadPageView** supplies the basis for building tabbed interfaces with rich formatting and behavior.



RadListControl and RadDropDownList controls are based on the Telerik Presentation Foundation and have a great deal of flexibility in what content can be included in theirs lists and how the content is arranged. Each item in the list can have specially formatted. The layout and formatting is highly configurable.

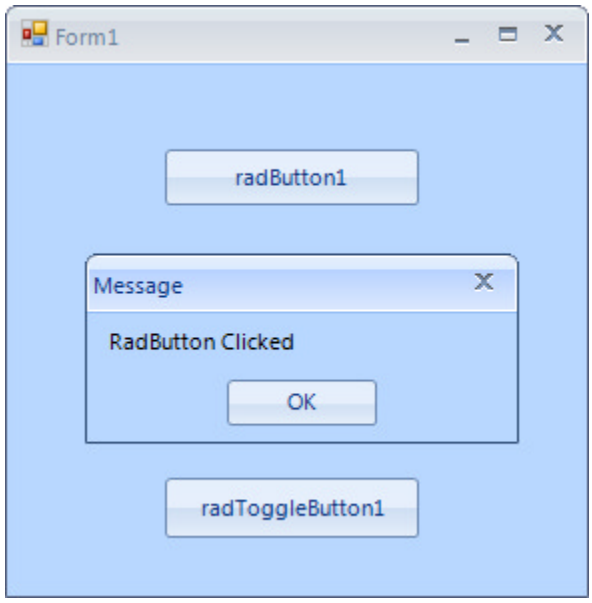


RadControls that have a DataSource property can **databind** to any object that implements IList or IListSource including array and generic lists, business objects, DataTable and DataSets, LINQ data sources, web services and WCF data. The binding mechanism is consistent between controls allowing you to best leverage your knowledge.

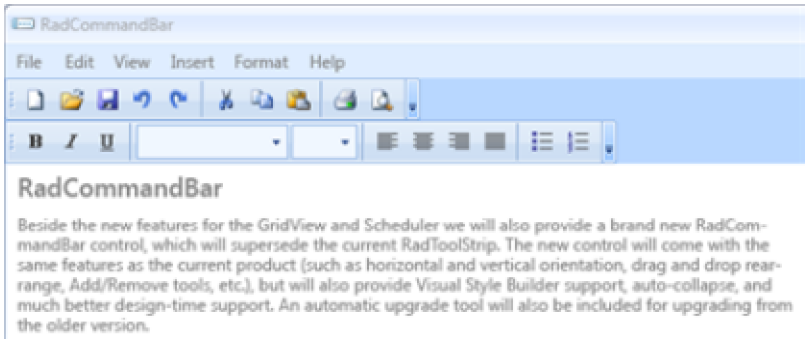


Telerik RadControls provide compelling **user feedback** so that your user knows what just happened, what's going on now and what's coming next. Enhanced themeable trackbars, status strips, progress bars, waiting bars and labels keep your user engaged.

RadControls for Winforms



RadShortcut provide a simple, comprehensive way to add keyboard shortcuts for all RadControls on a Windows Form.



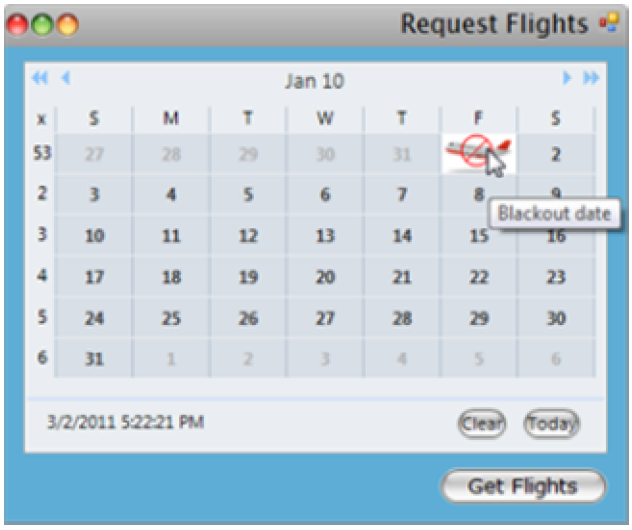
RadCommandBar fully themeable tool strip that provides unprecedented flexibility. More than just a collection of buttons, RadCommandBar hosts any RadControl, including dropdown lists, text boxes, split buttons, drop-down buttons, toggle buttons and more. Strips can be moved, rearranged and resized at run time for easy end-user customization.



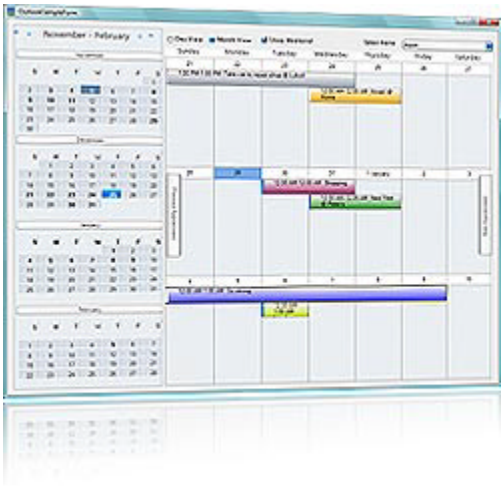
RadForm and **ShapedForm** let you design styled Windows forms of just about any shape and style for sharp, consistent user interfaces.



For interactive, unique navigation and form **"real estate" management**, RadDock, RadRotator and RadCarousel let you catch your users attention and make wise use of form space at the same time.

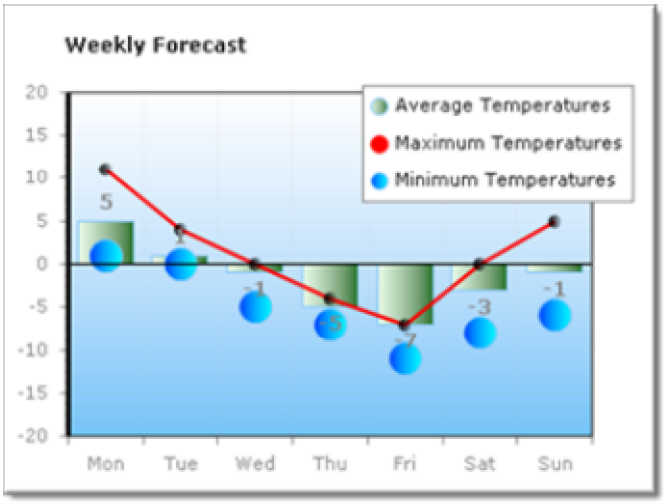


RadCalendar is a lightweight yet highly advanced date input control for Windows Forms. Zoom, select multiple dates, view several months at once, style special dates, use it as a date picker, or even add a Hebrew calendar to your forms.



RadScheduler is a highly-customizable appointment presentation component that offers rich Outlook®-style functionality. The product delivers swift performance, simplified deployment and limitless customization capabilities and is suitable for use in large-scale enterprise applications.

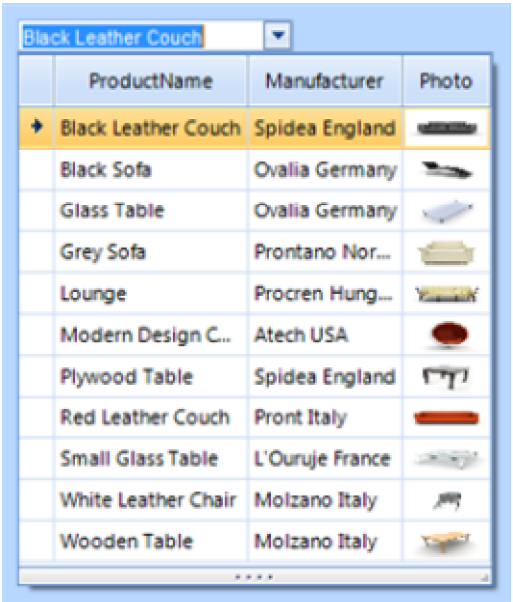
RadControls for Winforms



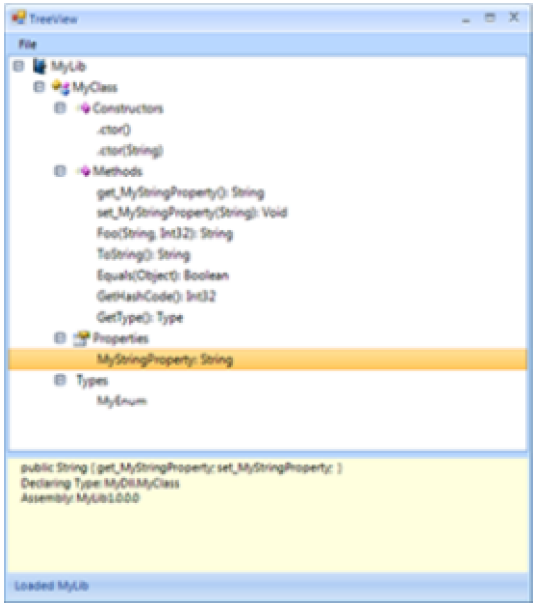
RadChart is a powerful business data presentation tool that can show your data off with striking impact. RadChart comes with many customizable chart types and skins to tailor the behavior and look of each chart.



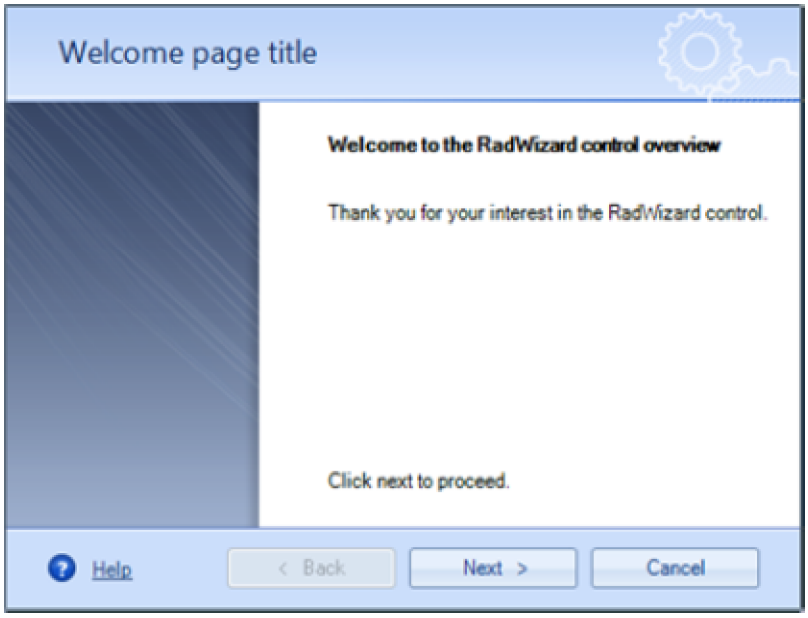
RadGridView is a powerful, highly performant grid component developed on top of the innovative Telerik Presentation Foundation, which allows for unprecedented performance, extensibility, customizability, and ease of use.



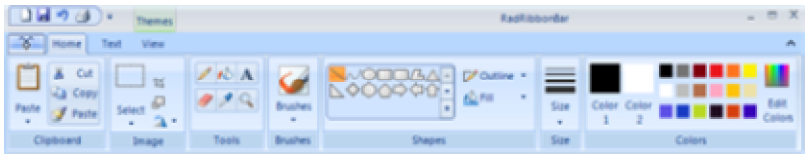
RadMultiColumnComboBox is a special case of combobox control with a RadGridView integrated in its drop-down. The drop down display has the rich capabilities of the grid view, but you can still use the control in a limited amount of space.



RadTreeView is a supercharged tree view component for Windows Forms that displays, manages, and navigates hierarchical data structures. The product offers advanced features like drag-and-drop, load-on-demand, context menus and data binding.



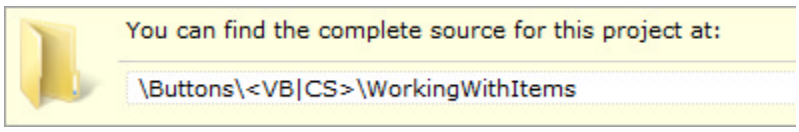
The **RadWizard** control helps you break a complex process into separate steps and provide your users with the opportunity to guide the process, just like the well-known installation wizards. RadWizard supports both Wizard97 and Wizard Aero specifications and provides developers with predesigned Welcome, Completion and Internal pages.



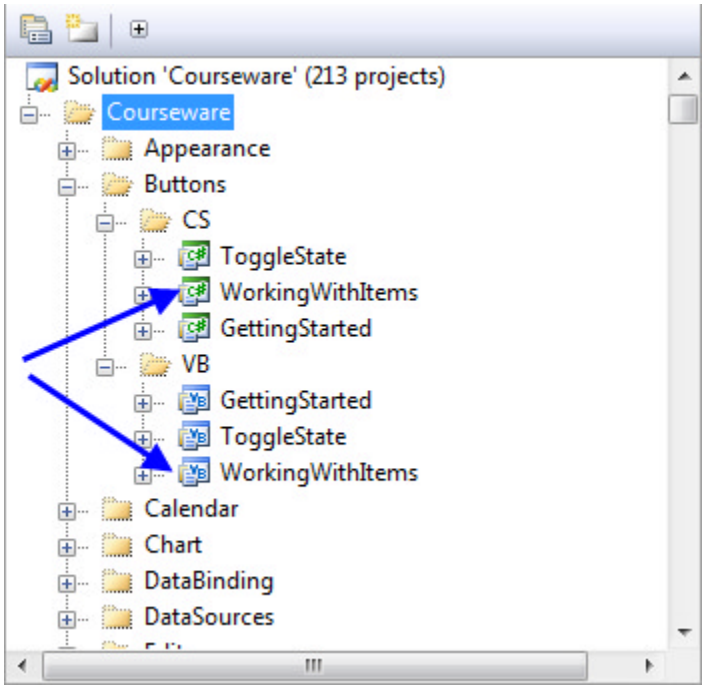
The most innovative user interfaces can be created in a flash with Telerik Windows Forms RibbonBar. In is an easy-to-use implementation of the Microsoft Office 2007 "ribbon' UI, which will allow you to codelessly organize all the functionality of you application into a single compact toolbar control. What is more, it can be coupled with RadRibbonForm or extended ease of use and easy imeplementation. And now, it also support BackstageView.

2.6 Before You Begin...

All sample projects are located in the Courseware solution (Courseware.sln). Each article that have an example project, specifies where you can find it in the solution. For example if article states that its example project is located at:



You can find it in the Courseware solution under:



The projects in this learning guide will assume the following:

1. That you will add the following assembly references and to the "Imports" (VB) or "uses" (C#) statements of your projects to reference the Telerik.WinControls and Telerik.WinControls.UI namespaces:

   [VB] Including the Telerik.Web.UI Namespace

   ```
   Imports Telerik.WinControls
   Imports Telerik.WinControls.UI
   ```

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

   ```
   using Telerik.WinControls.UI;
   using Telerik.WinControls;
   ```

2. You may need to change the path to assembly references within the supplied projects to local paths for your Telerik assemblies.

3 Buttons

3.1 Objectives

- Learn usage and basic characteristics for each type of RadControls buttons.
- Learn how to add members to the Items collection for drop down and split buttons.
- Respond to events for the buttons and Items collection members.
- Explore the design time interfaces to configure the buttons.
- Learn how to manipulate ToggleState and handle ToggleState related events.

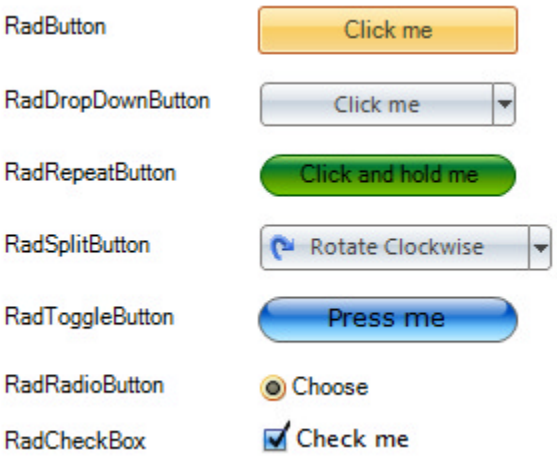
3.2 Introduction

These themable buttons replace their standard WinForms counterparts and add robust data binding, state management and design options. Developed to be very similar to the existing Windows Forms controls, they also allow you to take advantage of the features of RadControls, such as themes, rotation and transparency, images and text.

Along with the button, radio button and checkbox, you get:

- "Repeat" buttons that continuously fire as long as the button is held down
- A "Toggle" button that changes state like a checkbox. The state can be on, off and indeterminate.
- A "Drop Down" button displays a list of selections that can be clicked.
- A "Split" button is like the drop down but has a button and a drop down portion. If you click the arrow, the list drops down. One item in the list is considered the "default" and clicking the button acts as if that default item is being clicked.

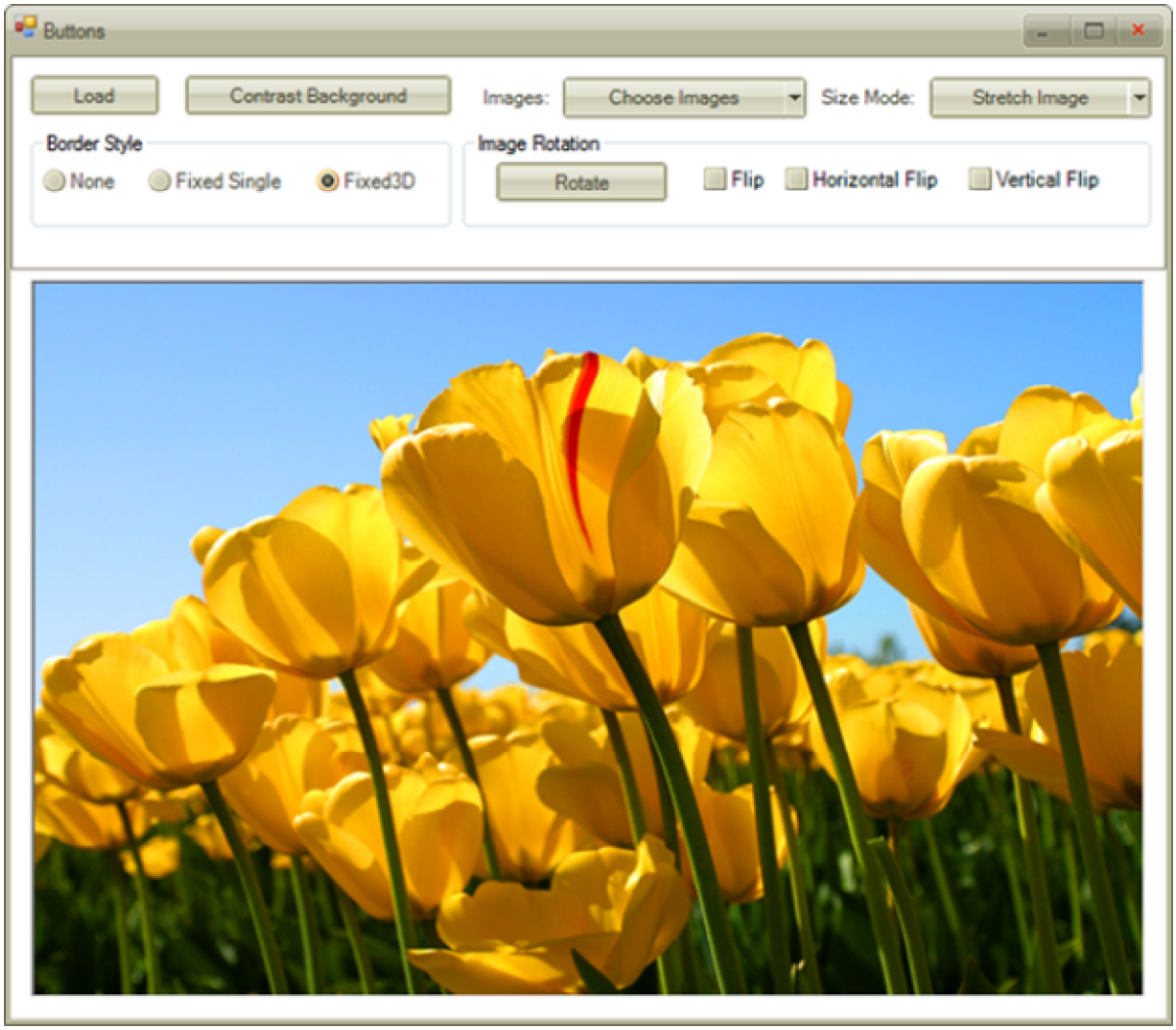
  The repeat and toggle buttons were inspired by analagous buttons found in the Windows Presentation Foundation (WPF).



3.3 Getting Started

In this project we will create a basic image viewer that will exercise all the available button types and where several themes will be used to style the UI. In later projects we will extend this example to handle state management capabilities and programmatic creation of list items.

Important Note: Be aware that in the beginning stages of this project we're going to audition several different themes so you can see the range of what's available. At the end of the project we'll bring it all together by applying a single theme to the entire application.

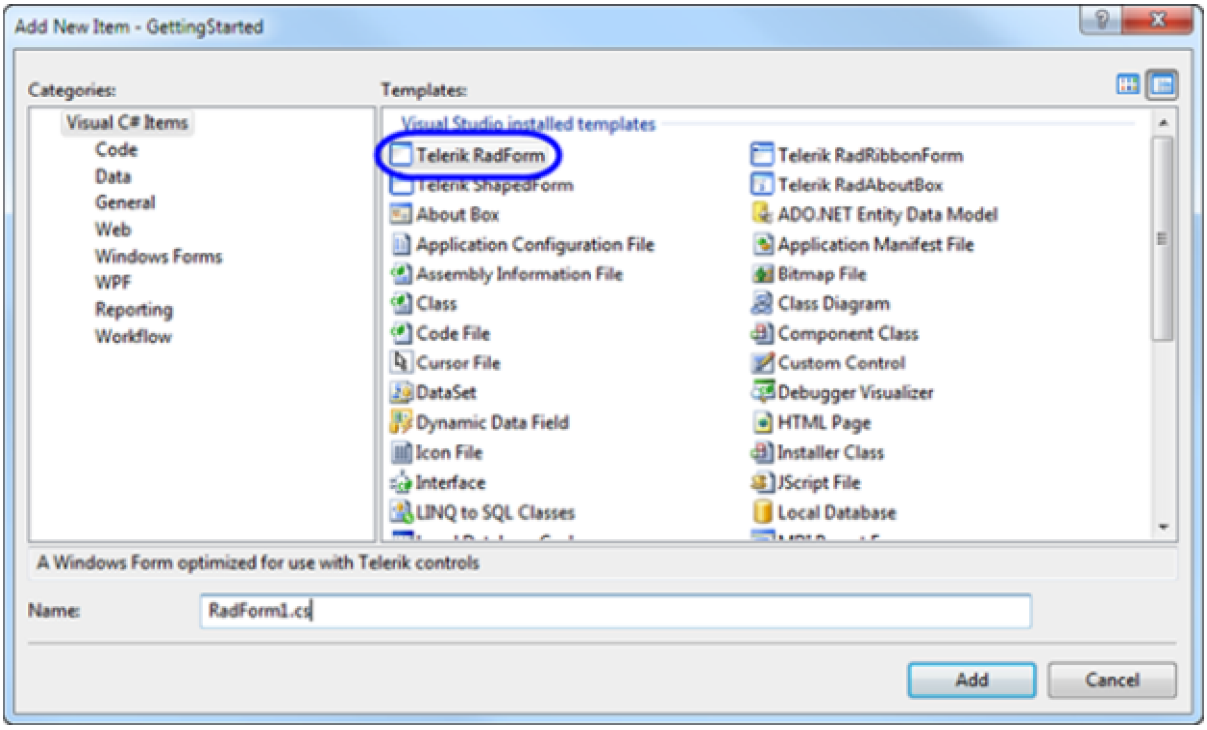


Preparing the Project

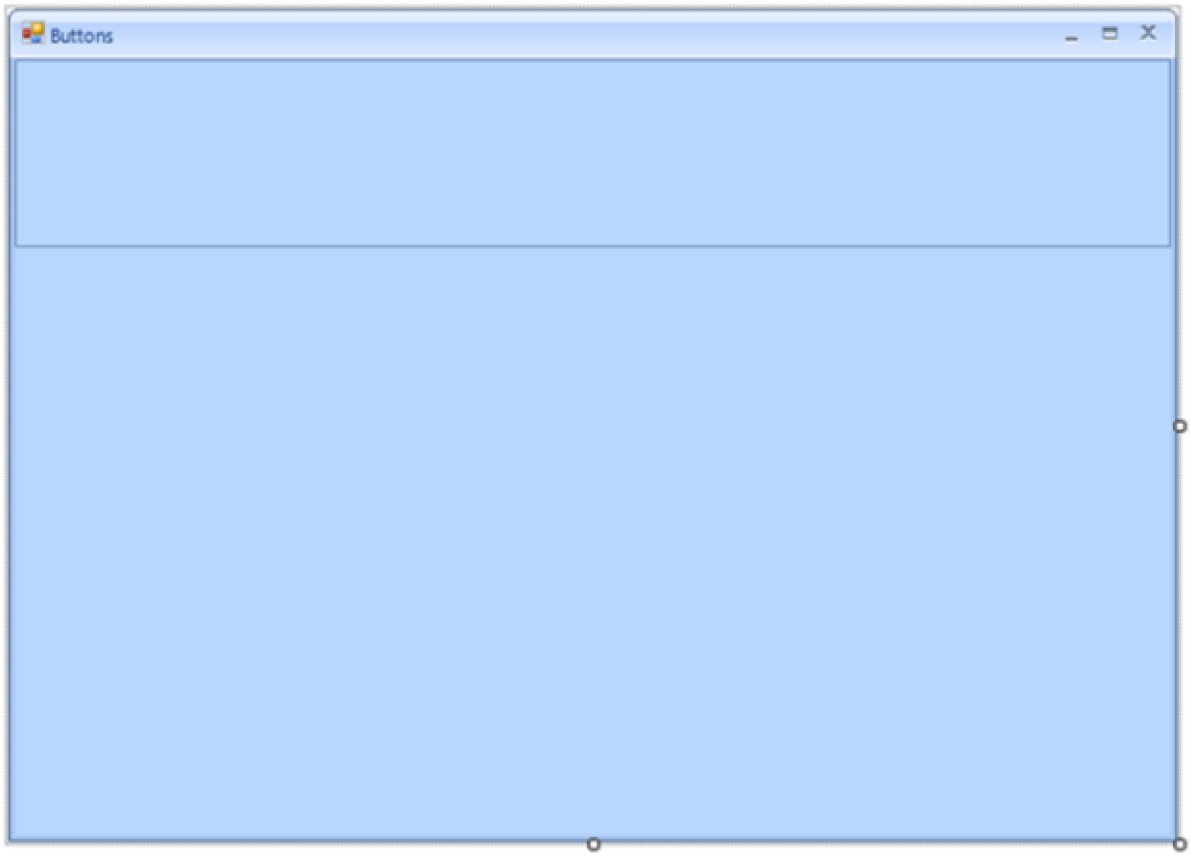
You can find the complete source for this project at:

\Buttons\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.

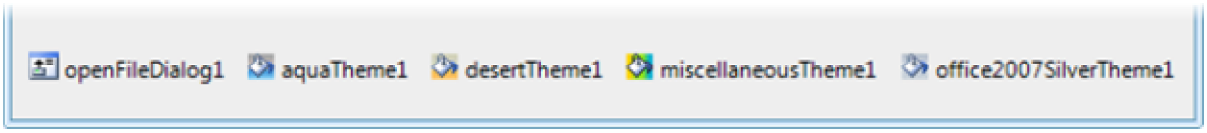


5. In the Properties window, set the form's **Size** property to "700, 500" and the **Text** property to "Buttons".
6. From the Toolbox, add a RadPanel to the form. Set the panel **Dock** property to "Top" and **Text** to "". Drag the bottom edge of the panel down so that it takes up about a third of the form space. The panel layout should now look something like this:



7. Add several themes from the Toolbox: **OpenFileDialog (from the standard MS controls)**, **AquaTheme**, **Office2007SilverTheme**, **MiscellaneousTheme** and **DesertTheme**.

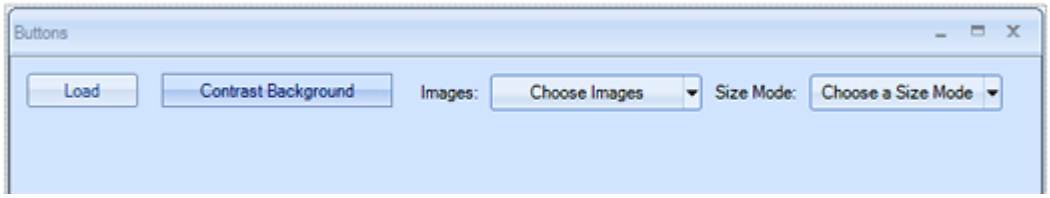
   The component tray should now have the components shown in the screenshot below:



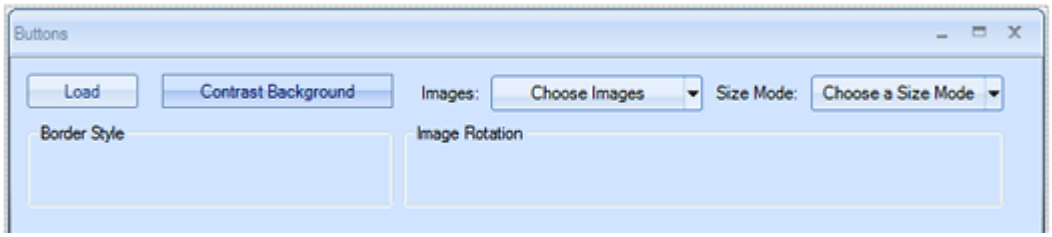
8. From the Dialogs tab of the Toolbox add a OpenFileDialog component. Set the **Filter** property to "jpg (*.jpg)|*.jpg|png (*.png)|*.png".
9. Set the RadForm **ThemeName** property to **ControlDefault**
10. Set the RadPanel **ThemeName** property to **ControlDefault**.
11. Add the following buttons to the top of the RadPanel (from left to right) and set properties:
    - RadButton: **Name** = "btnLoad", **Text** = "Load", **ThemeName** = "ControlDefault", **BackColor** = "Transparent".
    - RadToggleButton: **Name** = "tbBackgroundColor", **Text** = "Contrast Background", **ThemeName** = "ControlDefault", **BackColor** = "Transparent".
    - RadLabel: **Text** = "Images:", **ThemeName** = "ControlDefault", **BackColor** = "Transparent".
    - RadDropDownButton: **Name** = "ddbImages", **Text** = "Choose Images", **ThemeName** = "ControlDefault", **BackColor** = "Transparent", **DropDownDirection** = Right.
    - RadLabel: **Text** = "Size Mode:", **ThemeName** = "ControlDefault", **BackColor** = "Transparent".
    - RadSplitButton: **Name** = "spSizeMode", **Text** = "Choose a Size Mode", **ThemeName** = "ControlDefault",

BackColor = "Transparent".

The design surface should look something like the example below:



12. Below the row of buttons, add two standard GroupBox controls:
    - **BackColor** = "Transparent", **Text** = "Border Style".
    - **BackColor** = "Transparent", **Text** = "Image Rotation".



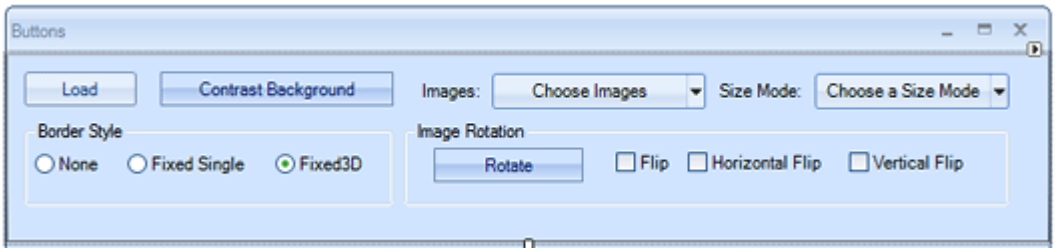
13. Add three RadRadioButtons to the "Border Style" group box:
    - **Name** = "rbNone", **Text** = "None", **ThemeName** = "ControlDefault", **Tag** = "0".
    - **Name** = "rbFixedSingle", **Text** = "Fixed Single", **ThemeName** = "ControlDefault", **Tag** = "1".
    - **Name** = "rbFixed3D", **Text** = "Fixed 3D", **ThemeName** = "ControlDefault", **Tag** = "2", **IsChecked** = "True".

    Using the Event () tab of the Properties Window add an event handler "BorderStyleClick". **Note**: We will add the code to this event handler later.

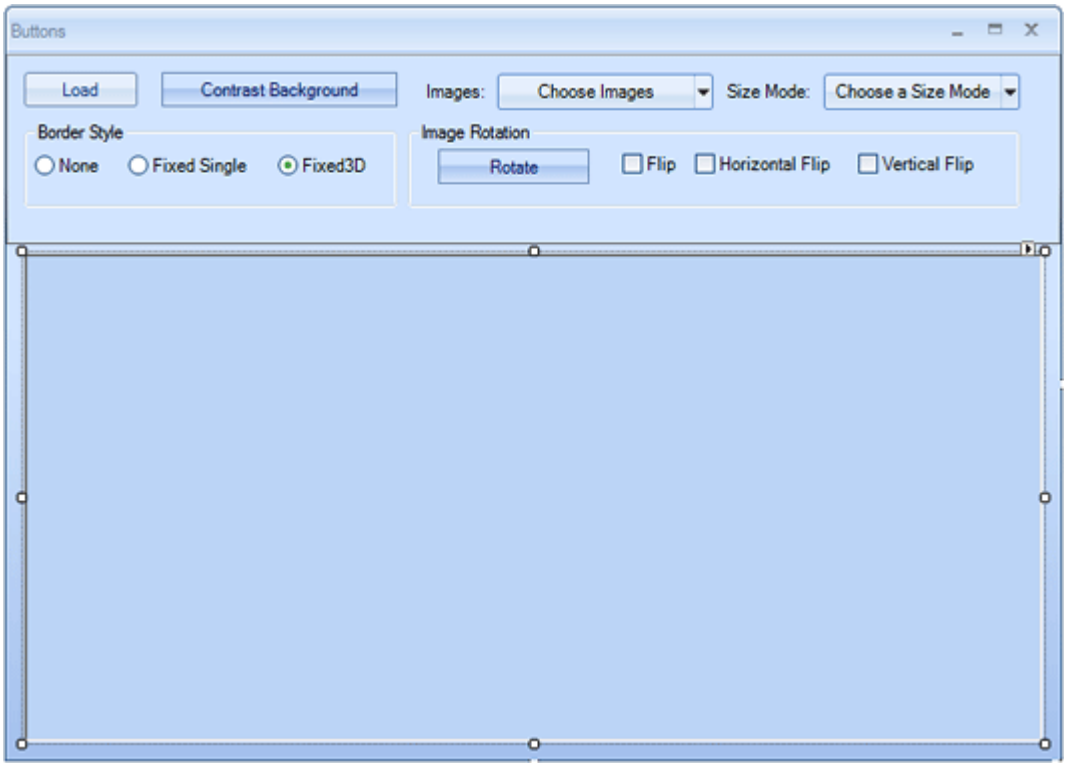
> Notice that the Tag properties above are set to 0, 1, 2. These correspond to the raw integer values of the BorderStyle enumeration. Later, when a button is clicked on, we will retrieve the tag value and cast it to be a BorderStyle.

14. In the "Image Rotation" group box add the following buttons:
    - RadRepeatButton: **Name** = "rbRotate", **Text** = "Rotate", **Interval** = "1000", **ThemeName** = "ControlDefault".
    - RadCheckBox: **Name** = "cbFlip", **Text** = "Flip", **ThemeName** = "ControlDefault".
    - RadCheckBox: **Name** = "cbHorizontalFlip", **Text** = "Horizontal Flip", **ThemeName** = "ControlDefault".
    - RadCheckBox: **Name** = "cbVerticalFlip", **Text** = "Vertical Flip", **ThemeName** = "ControlDefault".

    Now the panel should look something like the screenshot below:



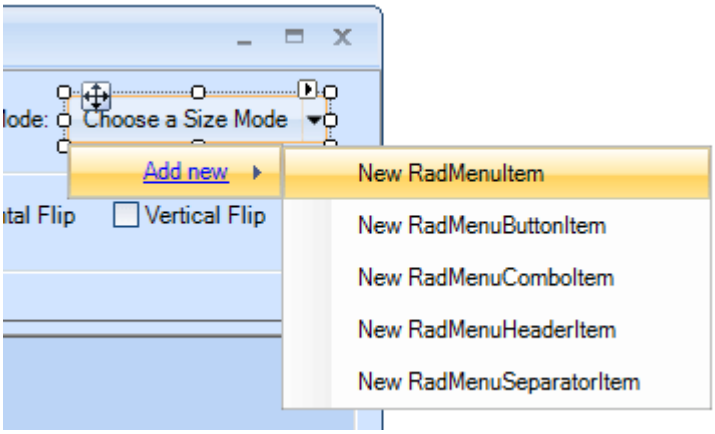
15. Below the RadPanel add a standard PictureBox control. Size it to take up most of the remaining space on the form. Set the **Anchor** property to "Top, Bottom, Left, Right".



Configure the RadSplitButton Items

In this example we are going to add items to the RadSplitButton at design time, rather than in code.

1. Click the RadSplitButton "Choose Size Mode" a single time, then click it again. An "Add new >" button will display. Click the button to see the list of possible RadItem types that can be added to the split button Items collection. Click "New RadMenuItem" to create a RadMenuItem in the collection.



2. Select the new RadMenuItem with the mouse and set the following properties in the Properties Window: **Name** = "btnSizeModeNormal", **Text** = "Normal" and **Tag** = "0". In the Events tab ( ) of the Properties Window, locate the **Click** event and add "SizeModeClick" as the event name and hit Enter to create the event handler.
3. Add four more RadMenuItems with the following Properties:
   - **Name** = "btnSizeModeStretch", **Text** = "Stretch Image", **Tag** = "1". Set the **Click** event to "SizeModeClick".
   - **Name** = "btnSizeModeAutoSize", **Text** = "AutoSize", **Tag** = "2". Set the **Click** event to "SizeModeClick".
   - **Name** = "btnSizeModeCenterImage", **Text** = "Center Image", **Tag** = "3". Set the **Click** event to "SizeModeClick".
   - **Name** = "btnSizeModeZoom", **Text** = "Zoom", **Tag** = "4". Set the **Click** event to "SizeModeClick".

Add Code

Here we will add some minimal functionality to the RadButton to load an image, the RadSplitButton to change the size mode and the RadRadioButtons to change the PictureBox BorderStyle.

1. Add references to the "Imports" (VB) or "uses" (C#) section of code:

   [VB] Adding References

```
Imports Telerik.WinControls
Imports Telerik.WinControls.UI
```

   [C#] Adding References

```
using Telerik.WinControls.UI;
using Telerik.WinControls;
```

2. In the design view, double-click the "Load" button to create a Click event handler. Add the code below to the handler.

   This code uses the OpenFileDialog to get the name of a "jpg" or "png" image file. A new Bitmap object is created and assigned to the PictureBox Image.

   [VB] Handling the Load Button Click Event

```
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnLoad.Click
If openFileDialog1.ShowDialog() = System.Windows.Forms.DialogResult.OK Then
Dim bitmap As Bitmap = New Bitmap(openFileDialog1.FileName)
```

```
pictureBox1.Image = bitmap
End If
End Sub
```

[C#] Handling the Load Button Click Event

```
private void btnLoad_Click(object sender, EventArgs e)
{
if (openFileDialog1.ShowDialog() == DialogResult.OK)
{
Bitmap bitmap = new Bitmap(openFileDialog1.FileName);
pictureBox1.Image = bitmap;
}
}
```

3. In the form's Load event, set the default item to execute when the button portion of the split button is clicked. Set the split button's text to the item's text. Finally, fire the sbSizeMode button Click event.

[VB] Handle the Load Event

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handle
MyBase.Load
sbSizeMode.DefaultItem = sbSizeMode.Items("btnSizeModeZoom")
sbSizeMode.Text = sbSizeMode.Items("btnSizeModeZoom").Text
sbSizeMode.DefaultItem.PerformClick()
End Sub
```

[C#] Handle the Load Event

```
private void RadForm1_Load(object sender, EventArgs e)
{
sbSizeMode.DefaultItem = sbSizeMode.Items["btnSizeModeZoom"];
sbSizeMode.Text = sbSizeMode.Items["btnSizeModeZoom"].Text;
sbSizeMode.DefaultItem.PerformClick();
}
```

4. Add the code below to handle the RadMenuMenuButton Click event.

Get the RadMenuItem from the "sender" parameter passed in. The Tag property will contain the enumeration value associated with the menu item. Cast the Tag value first from object to Int32, then to PictureBoxSizeMode and assign it to the SizeMode property of the PictureBox. Finally, set the split button Text property to the menu item's Text.

[VB] Handle the RadMenuButtonItem Click

```
Private Sub SizeModeClick(ByVal sender As Object, ByVal e As EventArgs)
Dim item As RadMenuButtonItem = TryCast(sender, RadMenuButtonItem)
pictureBox1.SizeMode = CType(Convert.ToInt32(item.Tag), PictureBoxSizeMode)
sbSizeMode.Text = item.Text
End Sub
```

[C#] Handle the RadMenuButtonItem Click

```
private void SizeModeClick(object sender, EventArgs e)
{
RadMenuButtonItem item = sender as RadMenuButtonItem;
pictureBox1.SizeMode = (PictureBoxSizeMode)Convert.ToInt32(item.Tag);
```

```
sbSizeMode.Text = item.Text;
}
```

5. Handle the "Border Style" RadRadioButton Click event. Again, get the enumeration value from the Tag, convert to Int32 and to the BorderStyle type then assign to the PictureBox BorderStyle property.

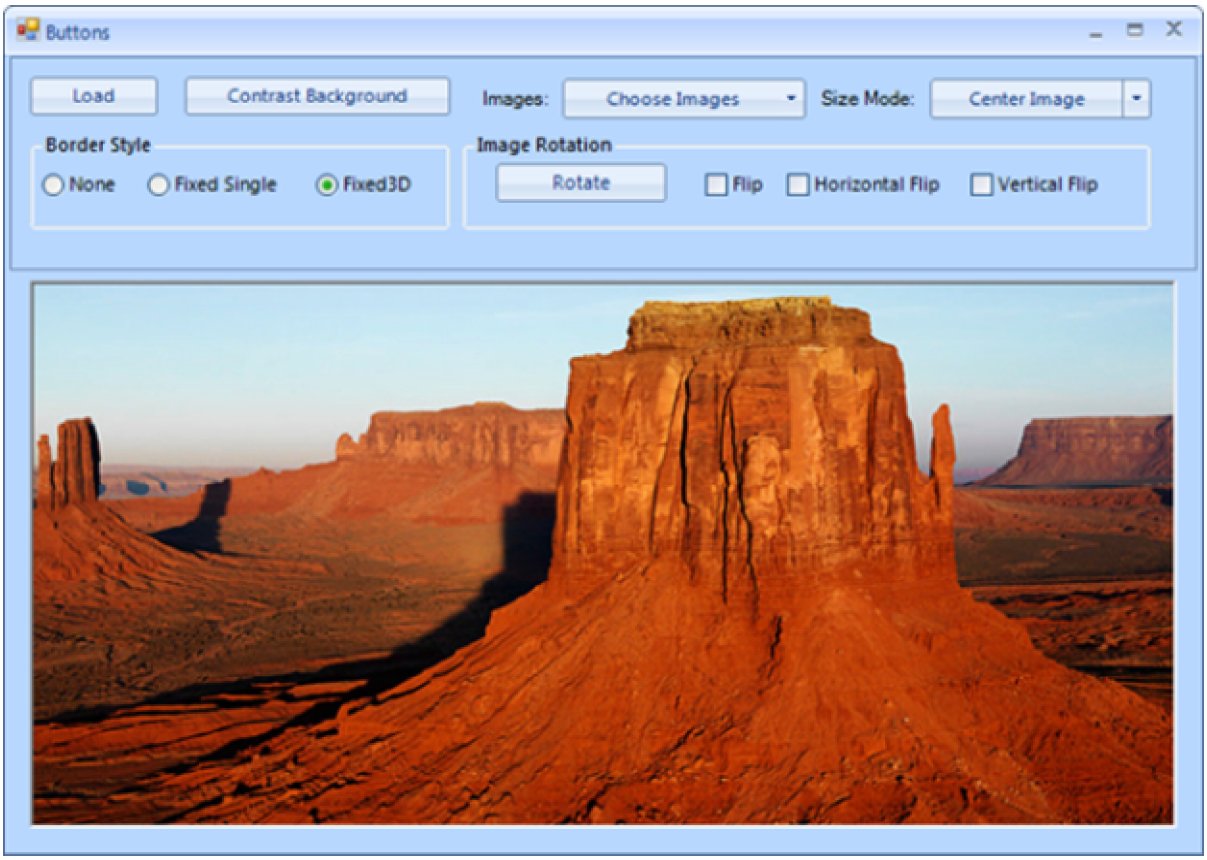
[VB] Handle the RadRadioButton Click

```
Private Sub BorderStyleClick(ByVal sender As Object, ByVal e As EventArgs)
pictureBox1.BorderStyle = CType(Convert.ToInt32((TryCast(sender, RadRadioButton)).Tag),
BorderStyle)
End Sub
```

[C#] Handle the RadRadioButton Click

```
private void BorderStyleClick(object sender, EventArgs e)
{
pictureBox1.BorderStyle = (BorderStyle)Convert.ToInt32((sender as RadRadioButton).Tag);
}
```

6. Run the application. Load a jpg or png image using the Load button. Try using the BorderStyle radio buttons. Then test the split button by selecting one of the items and also test clicking the button directly to get the default (zoom) action.



7. In the form's Load event, add the following lines of code to apply the "Desert" theme to the entire application. Note: you will need to add Telerik.WinControls to the "Imports" (VB) or "using" (C#) section of code.

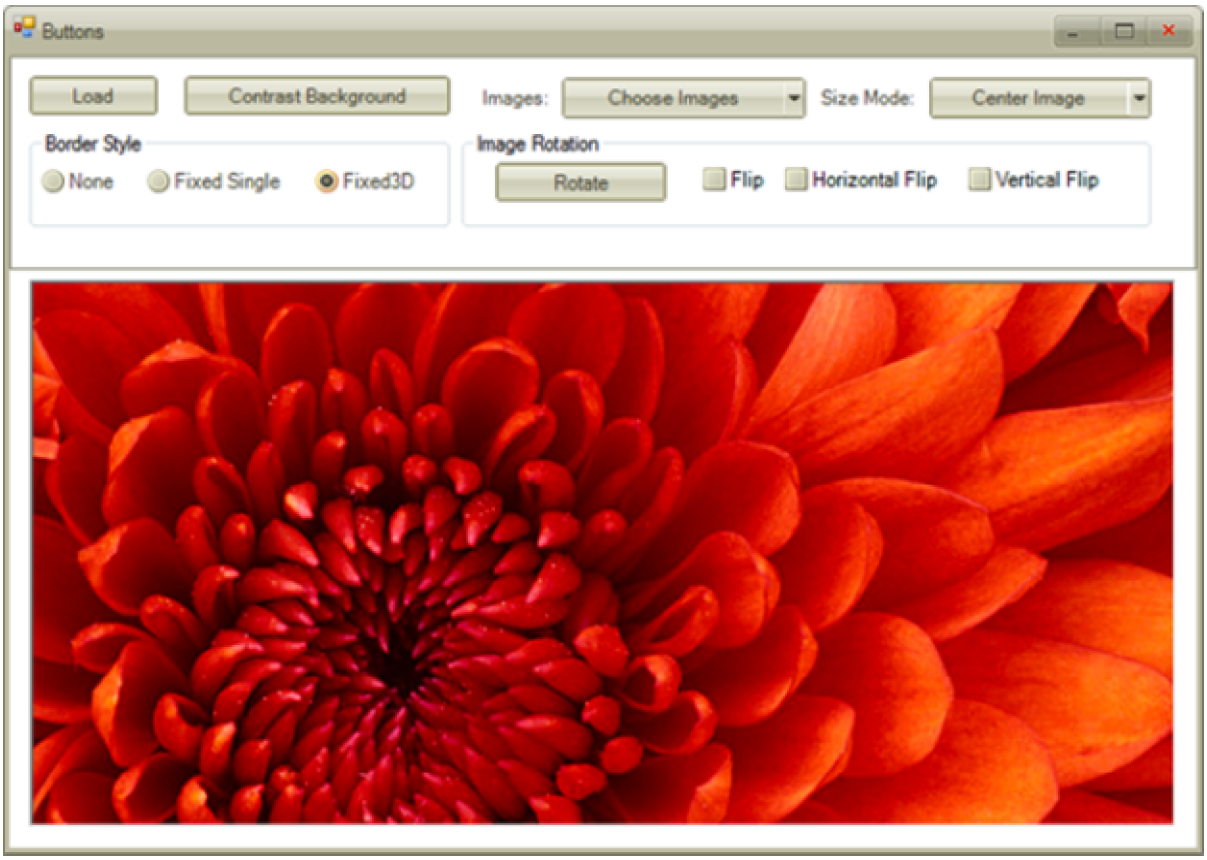
[VB] Setting the Application Theme

```
ThemeResolutionService.ApplicationThemeName = "Desert"
```

[C#] Setting the Application Theme

```
ThemeResolutionService.ApplicationThemeName = "Desert";
```

8. Re-run the application to see the theme applied to all elements of the form.



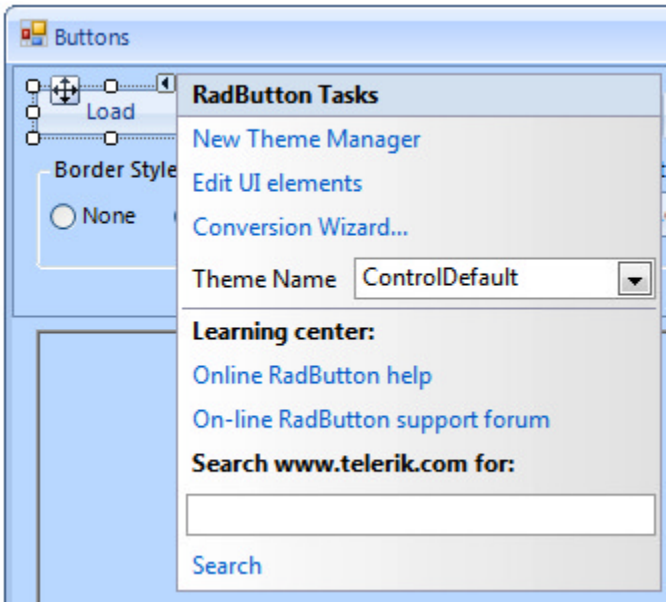
If you want to allow the user to tab through buttons and have each button visibly receive focus, set each button's **AllowShowFocusCues** property to true. The second button in the image below has the focus and AllowShowFocusCues is true.



3.4 Using the Design Time Interface

Smart Tag

The Smart Tag for all buttons except RadDropDownButton and RadSplitButton have the same basic layout as shown in the RadButton Smart Tag screenshot below. RadDropDownButton and RadSplitButton include an additional Tasks entry "Edit Items" that displays a RadItem Collection editor where you can add, change properties and delete items in the drop down list.



Tasks

From the Smart Tag Tasks menu you can Open Theme Builder to style all aspects of your control, select New Theme Manager to add a Theme Manager component to the component tray, Edit UI elements to browse all of the elements of the button and to change properties for any element. The Theme Name drop down lets you pick an existing theme to style your control.

Learning center

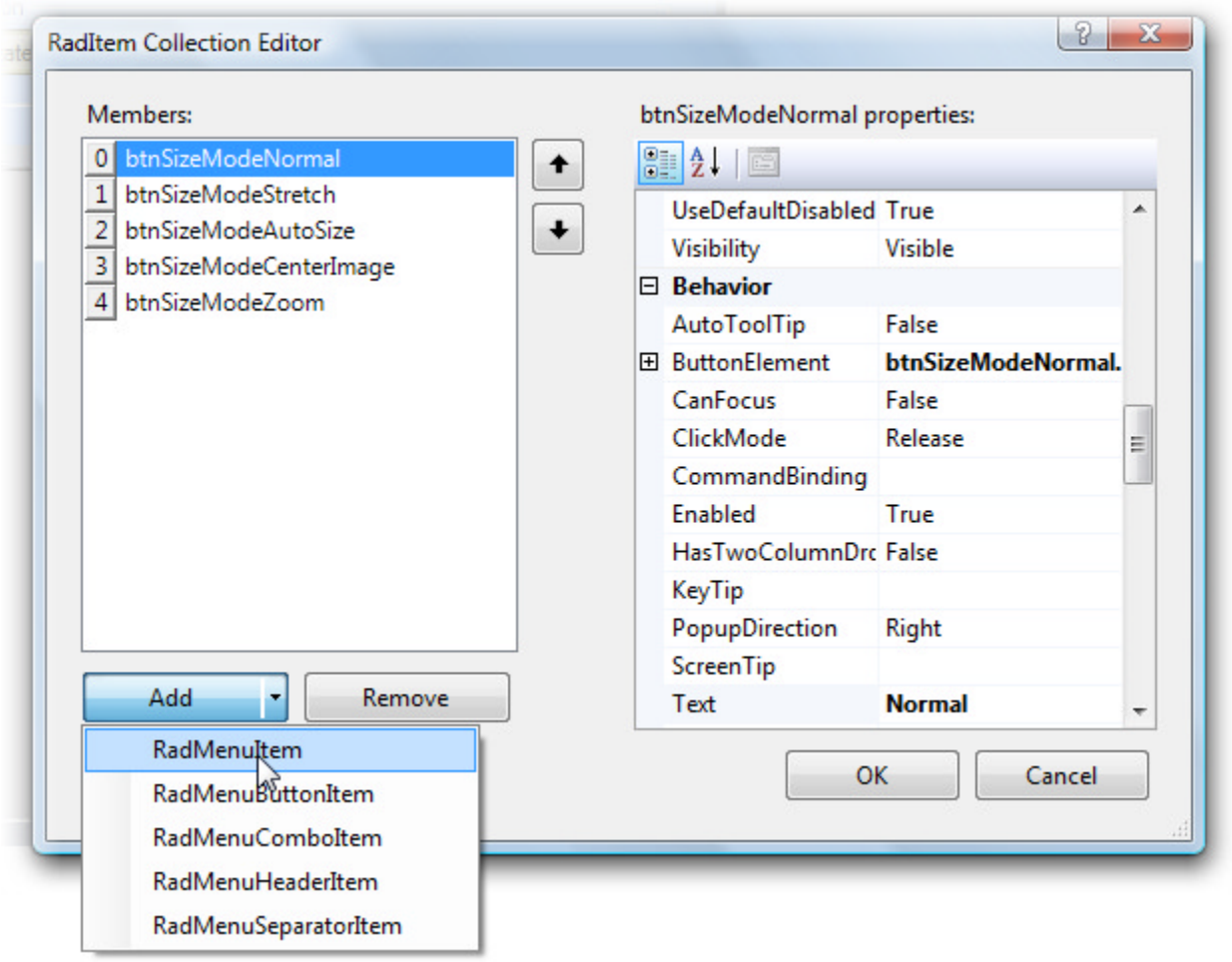
The Learning center lets you navigate to a web browser with online help for the currently selected button control or to the online support forums.

Search

Enter search criteria in the edit space provided and click the Search link to navigate directly to search on the Telerik web site.

Collection Editor

The RadItem Collection Editor is initiated by the RadDropDownButton and RadSplitButton Items property ellipses or from the Smart Tag "Edit Items" entry. The editor lets you add, edit properties and delete items that populate the drop down list of your button.



The Add button contains menu item types RadMenuItem, RadMenuButtonItem, RadMenuComboItem, RadMenuHeaderItem and RadMenuSeparatorItem.

3.5 Buttons Programming

Working with Items

To add an item to the Items collection of a RadDropDownButton or RadSplitButton, minimally you need to create one of the button types you find in the RadItem Collection Editor and add it. The example below creates a RadMenuButtonItem with text "Save".

[VB] Adding an Item, Minimal

```
Dim item As New RadMenuButtonItem("Save")
radDropDownButton1.Items.Add(item)
```

[C#] Adding an Item, Minimal

```
RadMenuButtonItem item = new RadMenuButtonItem("Save");
radDropDownButton1.Items.Add(item);
```

If you need to associate any extra data with a menu item, store it in the Tag property. Because Tag is an Object type, you can of course store simple values like integers, floats and strings. But you can also store any object in Tag, and that makes the storage possibilities unlimited. You may also want to associate an Image with the item and for that you can use the Image property directly or use a standard ImageList component along with the items ImageIndex or ImageKey property. To respond to the item being clicked by the user, assign the Click event. The Click event Sender parameter will be the clicked item.

The pattern for RadControls for Winforms items is to respond to individual item events rather than responding to events from the parent item.

Here's an example that creates a single RadMenuItem, sets the text, tag and image, also assigns a click event handler and adds the item to the Items collection.

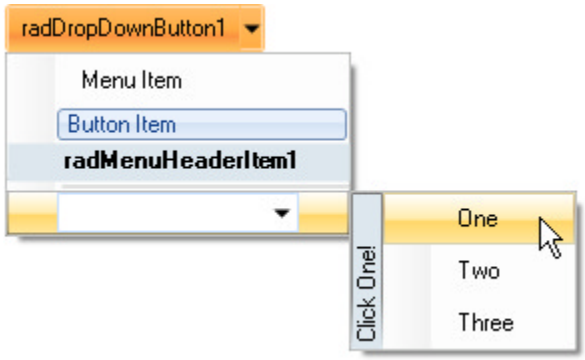
[VB] Adding an Item

```
Dim item As New RadMenuItem()
item.Text = name
item.Tag = myObject
item.Image = New Bitmap("SaveIcon.jpg")
AddHandler item.Click, AddressOf SaveClick
radDropDownButton1.Items.Add(item)
```

[C#] Adding an Item

```
RadMenuItem item = new RadMenuItem();
item.Text = name;
item.Tag = myObject;
item.Image = new Bitmap("SaveIcon.jpg");
item.Click += new EventHandler(SaveClick);
radDropDownButton1.Items.Add(item);
```

Depending on the item type there may be several constructor overloads that let you assign text, tag or images directly in the constructor. The example below creates multiple item types where the naming convention is "RadMenu<some type>Item". Notice the RadMenuComboItem that itself can hold any of the other menu item types. Also notice the combo item's DropDownHeaderText property. Note: you can also assign a DropDownHeaderImage. See the screenshot below for an example of drop down header text.



[VB] Adding Multiple Item Types

```
radDropDownButton1.Items.Add(New RadMenuItem("Menu Item"))
radDropDownButton1.Items.Add(New RadMenuButtonItem("Button Item"))
radDropDownButton1.Items.Add(New RadMenuSeparatorItem())
radDropDownButton1.Items.Add(New RadMenuHeaderItem("Header"))
Dim comboItem As New RadMenuComboItem()
comboItem.DropDownHeaderText = "Click One!"
comboItem.Items.Add(New RadMenuItem("One"))
comboItem.Items.Add(New RadMenuItem("Two"))
comboItem.Items.Add(New RadMenuItem("Three"))
radDropDownButton1.Items.Add(comboItem)
```

[C#] Adding Multiple Item Types

```
radDropDownButton1.Items.Add(new RadMenuItem("Menu Item"));
radDropDownButton1.Items.Add(new RadMenuButtonItem("Button Item"));
radDropDownButton1.Items.Add(new RadMenuSeparatorItem());
```

```
radDropDownButton1.Items.Add(new RadMenuHeaderItem("Header"));
RadMenuComboItem comboItem = new RadMenuComboItem();
comboItem.DropDownHeaderText = "Click One!";
comboItem.Items.Add(new RadMenuItem("One"));
comboItem.Items.Add(new RadMenuItem("Two"));
comboItem.Items.Add(new RadMenuItem("Three"));
radDropDownButton1.Items.Add(comboItem);
```

Working with Items Walkthrough

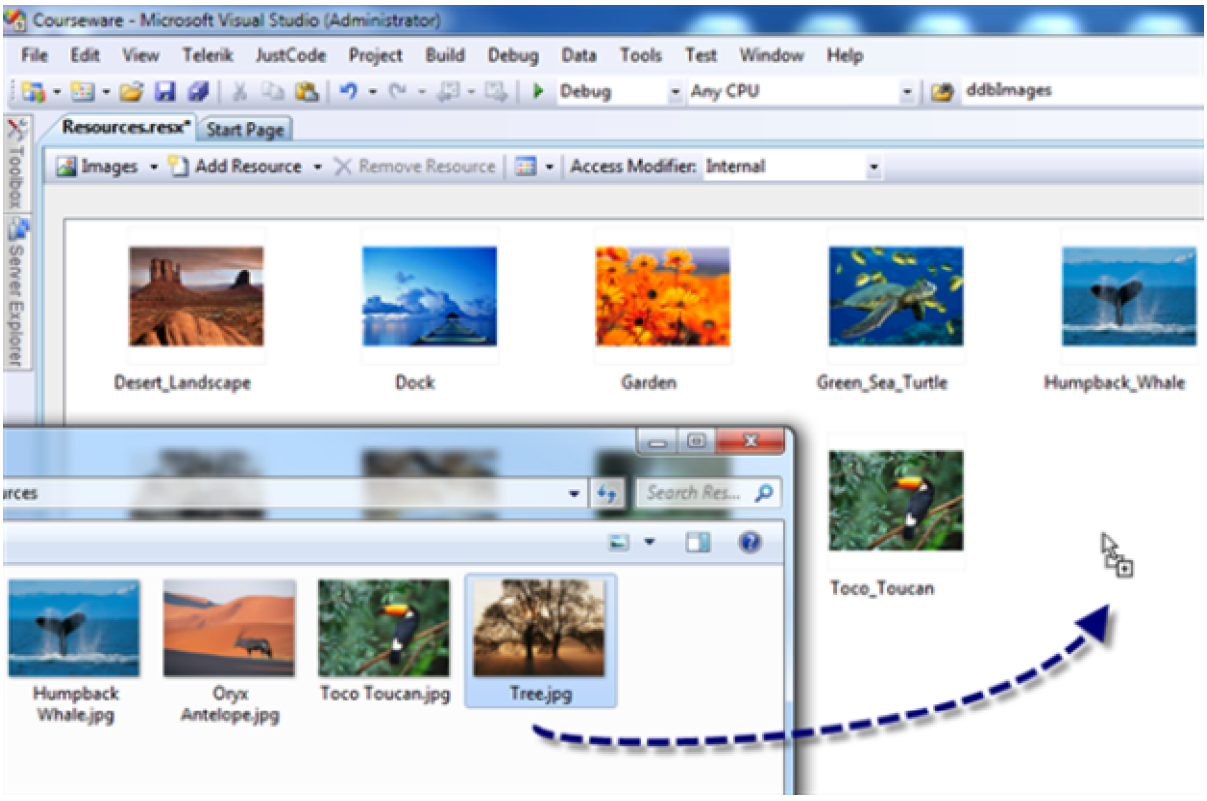
This project extends the Getting Started project so that a list of image resources are loaded into a RadDropDownButton items collection and also assigns the clicked-on item image to the picture box.



You can find the complete source for this project at:

\Buttons\<VB|CS>\WorkingWithItems

1. In the Solution Explorer, navigate to the Properties folder, Resources.resx item. Double-click Resources.resx to open the resources editor.
2. Drag an assortment of jpg or png images from the file explorer into the resources editor.



3. Make sure you have the following references in your "Imports" (VB) or "uses" (C#) section of your code:

[VB] Adding Namespace References

```
Imports System
' Supports IDictionaryEnumerator
Imports System.Collections
Imports System.Drawing
```

```
' Supports CultureInfo
Imports System.Globalization
' Supports GetFileNameWithoutExtension
Imports System.IO
' Supports ResourceSet, Resources
Imports System.Resources
Imports System.Windows.Forms
Imports Telerik.WinControls.UI
```

[C#] Adding Namespace References

```
using System;
// Supports IDictionaryEnumerator
using System.Collections;
using System.Drawing;
// Supports CultureInfo
using System.Globalization;
// Supports GetFileNameWithoutExtension
using System.IO;
// Supports ResourceSet, Resources
using System.Resources;
using System.Windows.Forms;
using Telerik.WinControls.UI;
```

4. Add the code below. This event handler will be attached later to the "Choose Images" drop down button items.

   The event handler retrieves the RadMenuItem from the sender parameter. The Tag property will contain an Image object. Get a thumbnail from the image and assign it to the drop down button image, and assign the item's text to the drop down button text. Set the TextImageRelation property to "ImageBeforeText".

   [VB] Handling the Click Event

```
Private Sub ChooseImagesClick(ByVal sender As Object, ByVal e As EventArgs)
 Dim item As RadMenuItem = (TryCast(sender, RadMenuItem))
 ddbImages.Image = (DirectCast(item.Tag, Image)).GetThumbnailImage(20, 20, Nothing, New
IntPtr())
 ddbImages.Text = item.Text
 ddbImages.TextImageRelation = TextImageRelation.ImageBeforeText
 pictureBox1.Image = DirectCast(item.Tag, Image)
End Sub
```

   [C#] Handling the Click Event

```
private void ChooseImagesClick(object sender, EventArgs e)
{
 RadMenuItem item = (sender as RadMenuItem);
 ddbImages.Image = ((Image)item.Tag).GetThumbnailImage(20, 20, null, new IntPtr());
 ddbImages.Text = item.Text;
 ddbImages.TextImageRelation = TextImageRelation.ImageBeforeText;
 pictureBox1.Image = (Image)item.Tag;
}
```

5. Add a helper method that creates a RadMenuItem, assigns the name of the image to the Text, assign the image to the Tag property, a thumbnail to the items Image and finally assign the ChooseImagesClick() event handler.

[VB] Getting a Image Menu Item

```
Private Function GetImageMenuItem(ByVal name As String, ByVal image As Image) As RadMenu
 Dim item As New RadMenuItem()
 item.Text = name
 ' store the full image in the tag property and display a thumbnail version
 item.Tag = image
 item.Image = image.GetThumbnailImage(20, 20, Nothing, New IntPtr())
 AddHandler item.Click, AddressOf ChooseImagesClick
 Return item
End Function
```

[C#] Getting a Image Menu Item

```
private RadMenuItem GetImageMenuItem(string name, Image image)
{
 RadMenuItem item = new RadMenuItem();
 item.Text = name;
 // store the full image in the tag property and display a thumbnail version
 item.Tag = image;
 item.Image = image.GetThumbnailImage(20, 20, null, new IntPtr());
 item.Click += new EventHandler(ChooseImagesClick);
 return item;
}
```

6. Add code to the start of the form's Load event to iterate the image resources in your project and add them to the drop down button's items. The new load event handler is shown below.

The code iterates all the resources in your Properties.Resources resource set. This example makes the assumption that all the members of the resource set are usable jpg or png files. Each resource has a Key (the image file name without extension) and the Value (the image itself). Both name and image are passed as parameters to the GetImageMenuItem() method we coded previously. The new menu item is added to the drop down button's Items collection. The PerformClick() method is called to sync up the first item in the list with the drop down button text and image.

[VB] Handling the Load Event

```
Private Sub RadForm1_Load(ByVal sender As Object, ByVal e As EventArgs)
 ' list the resources in Properties.Resources
 Dim resourceSet As ResourceSet = Properties.Resources.ResourceManager.GetResourceSet
(CultureInfo.CurrentCulture, True, True)
 Dim enumerator As IDictionaryEnumerator = resourceSet.GetEnumerator()
 ' for each image resources, retrieve the image name and object,
 ' create and populate the menu item, then add to the drop down
 ' button items collection
 While enumerator.MoveNext()
  ' the key will be the name of the image, minus extension,
  ' also remove underline
  Dim key As String = (DirectCast(enumerator.Key, String)).Replace("_"C, " "C)
  ' the value will be the image object
  Dim image As Image = DirectCast(enumerator.Value, Image)
  Dim item As RadMenuItem = GetImageMenuItem(key, image)
  ddbImages.Items.Add(item)
 End While
 ddbImages.Items(0).PerformClick()
 '. . .
End Sub
```

[C#] Handling the Load Event

```
private void RadForm1_Load(object sender, EventArgs e)
{
 // list the resources in Properties.Resources
 ResourceSet resourceSet = Properties.Resources.ResourceManager.GetResourceSet(
   CultureInfo.CurrentCulture, true, true);
 IDictionaryEnumerator enumerator = resourceSet.GetEnumerator();
 // for each image resources, retrieve the image name and object,
 // create and populate the menu item, then add to the drop down
 // button items collection
 while (enumerator.MoveNext())
 {
   // the key will be the name of the image, minus extension,
   // also remove underline
   string key = ((string)enumerator.Key).Replace('_', ' ');
   // the value will be the image object
   Image image = (Image)enumerator.Value;
   RadMenuItem item = GetImageMenuItem(key, image);
   ddbImages.Items.Add(item);
 }
 ddbImages.Items[0].PerformClick();
//. . .
}
```

7. Change the "Load" button's click event handler to create a new menu item based on the newly created bitmap, insert the item the head of the drop down button items and trigger the Click event for the item.

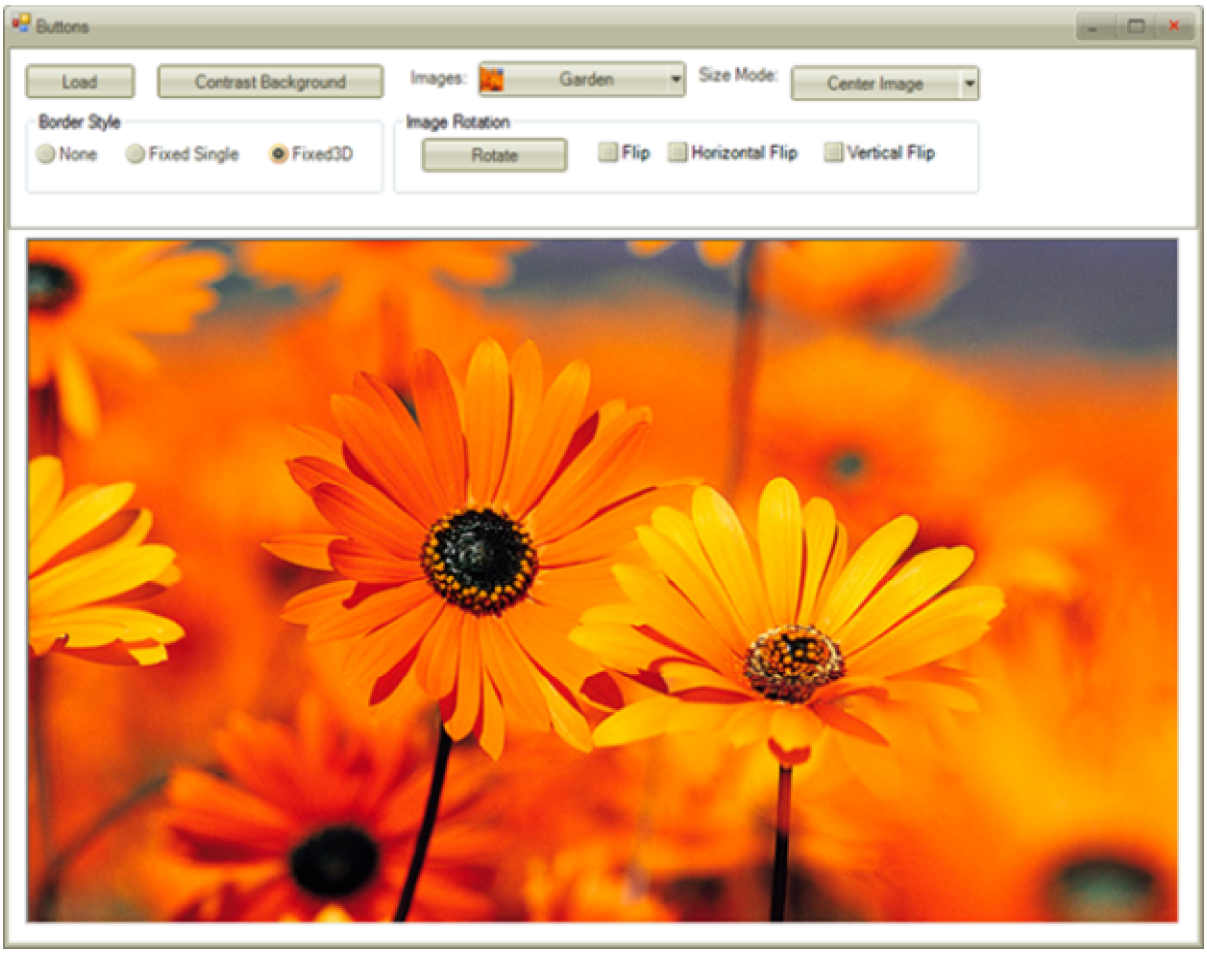
[VB] Modified "Load" Button Click Event Handler

```
Private Sub btnLoad_Click(ByVal sender As Object, ByVal e As EventArgs)
 If openFileDialog1.ShowDialog() = DialogResult.OK Then
  Dim bitmap As New Bitmap(openFileDialog1.FileName)
  Dim key As String = Path.GetFileNameWithoutExtension(openFileDialog1.FileName)
  Dim item As RadMenuItem = GetImageMenuItem(key, bitmap)
  ddbImages.Items.Insert(0, item)
  item.PerformClick()
  pictureBox1.Image = bitmap
 End If
End Sub
```

[C#] Modified "Load" Button Click Event Handler

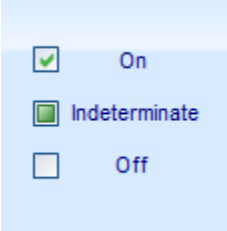
```
private void btnLoad_Click(object sender, EventArgs e)
{
 if (openFileDialog1.ShowDialog() == DialogResult.OK)
 {
   Bitmap bitmap = new Bitmap(openFileDialog1.FileName);
   string key = Path.GetFileNameWithoutExtension(openFileDialog1.FileName);
   RadMenuItem item = GetImageMenuItem(key, bitmap);
   ddbImages.Items.Insert(0, item);
   item.PerformClick();
   pictureBox1.Image = bitmap;
 }
}
```

8. Press **Ctl-F5** to run the application. Try selecting an image from the "Images" list and also try loading a new image to verify it is inserted to the head of the drop down button items.



Handling Toggle State

RadToggleButton, RadRadioButton and RadCheckBox are all capable of three states: On, Off and Indeterminate. These states usually, but not always, correspond to visual UI cues that let the user know which state the button is in. The visual cues are dependent on the theme. For example, RadCheckBox can use a "ControlDefault" theme which clearly shows the three states.



There are several properties and events related to toggle states:

- **ToggleState:** This property may be **Off**, **On** or **Indeterminate**.
- **Checked**: This is a simple boolean that is true when the button is checked (ToggleState = On). It's provided for compatibility only.
- **IsThreeState:** When this property is enabled, the user can cycle through the three states by clicking the button. If IsThreeState is off, you can still set the state of the button programmatically.
- **ToggleStateChanging:** This event occurs before the ToggleState property of the button is about to be changed. Using the StateChangingEventArgs passed in to the event, you can look at the **OldValue** (the

current ToggleState), the **NewValue** (the upcoming ToggleState) and you can set **Canceled** true to prevent the ToggleState being set.

- **ToggleStateChanged:** The StateChangedEventArgs passed to this event only contain a **ToggleState** property. The event is not cancelable.

ToggleState Walkthrough

In the "Getting Started" project we handled the "Border Style" radio buttons with a simple Click event and retrieved the Tag value. In this example we're going to exercise the button's ToggleState, both setting and retrieving the ToggleState On, Off and Indeterminate values in relatively complex relationships. In this case we're going to leave the IsThreeState property off and work with the ToggleState programmatically.

The "Contrast Background" will handle the ToggleStateChanged event to see if the ToggleState is On or Off, i.e. functionally equivalent to using IsChecked. If ToggleState is off, a color selected from a pixel in the center of the picture will be used as the background color. If ToggleState is on, a contrasting color will be calculated and used as the background color.

The checkboxes in the "Image Rotation" group box of controls will take all three toggle states into account. When the "Flip" check box toggle state changes, both the "Flip Horizontal" and "Flip Vertical" check boxes are set to the matching toggle state. When either the horizontal or vertical check boxes are toggled, the logic is:

- If both are On, the "Flip" check box is set On.
- If both are Off, the "Flip" check box is set Off.
- If only the horizontal or vertical check box is on, then the "Flip" check box toggle state is set to Indeterminate.

Based on the setting of these three checkboxes, a RadRepeater button will rotate and flip the image.

Working with Simple ToggleState - On/Off

This project extends the "WorkingWithItems" project.



You can find the complete source for this project at:

\Buttons\<VB|CS>\ToggleState

1. Add the namespace "Telerik.WinControls.Enumerations" to the "Imports" (VB) or "uses" (C#) section of code. This namespace will support the ToggleState enumeration.
2. Add a helper method to the form code to change the picture box's BackColor based on the RadToggleButton's ToggleState.

   The first color is obtained from a pixel grabbed from the center of the picture and the second is a contrasting color calculated off the first color.

   [VB] Setting Color Based on ToggleState

```
Private Sub SetBackgroundColor(ByVal toggleState As ToggleState)
 ' cast picture as bitmap so we can use the GetPixel method
 Dim bitmap As Bitmap = TryCast(pictureBox1.Image, Bitmap)
 ' grab a pixel color from the center of the image
 Dim color As Color = bitmap.GetPixel(bitmap.Width / 2, bitmap.Height / 2)
 ' calculate a contrasting color
 Dim contrastingColor As Color = Color.FromArgb(color.ToArgb() Xor 16777215)
 ' assign the back color based on the toggle state
 pictureBox1.BackColor = IIf(toggleState = ToggleState.Off,color,contrastingColor)
End Sub
```

[C#] Setting Color Based on ToggleState

```
private void SetBackgroundColor(ToggleState toggleState)
{
 // cast picture as bitmap so we can use the GetPixel method
 Bitmap bitmap = pictureBox1.Image as Bitmap;
 // grab a pixel color from the center of the image
 Color color = bitmap.GetPixel(bitmap.Width / 2, bitmap.Height / 2);
 // calculate a contrasting color
 Color contrastingColor = Color.FromArgb(color.ToArgb() ^ 0x00ffffff);
 // assign the back color based on the toggle state
 pictureBox1.BackColor = toggleState == ToggleState.Off ? color : contrastingColor;
}
```

3. Handle the ToggleStateChanged event for the tbBackgroundColor toggle button. Call SetBackgroundColor and pass the ToggleState.

   You can get the ToggleState from the StateChangedEventArgs passed to this event handler.

   [VB] Handle the Background Color Button ToggleStateChanged Event

```
Private Sub tbBackgroundColor_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs)
 SetBackgroundColor(args.ToggleState)
End Sub
```

   [C#] Handle the Background Color Button ToggleStateChanged Event

```
private void tbBackgroundColor_ToggleStateChanged(object sender, StateChangedEventArgs a
{
 SetBackgroundColor(args.ToggleState);
}
```

4. Add a call to SetBackgroundColor() in the "Choose Images" button click handler. This will set the new background color automatically as new images are loaded to the picture box.

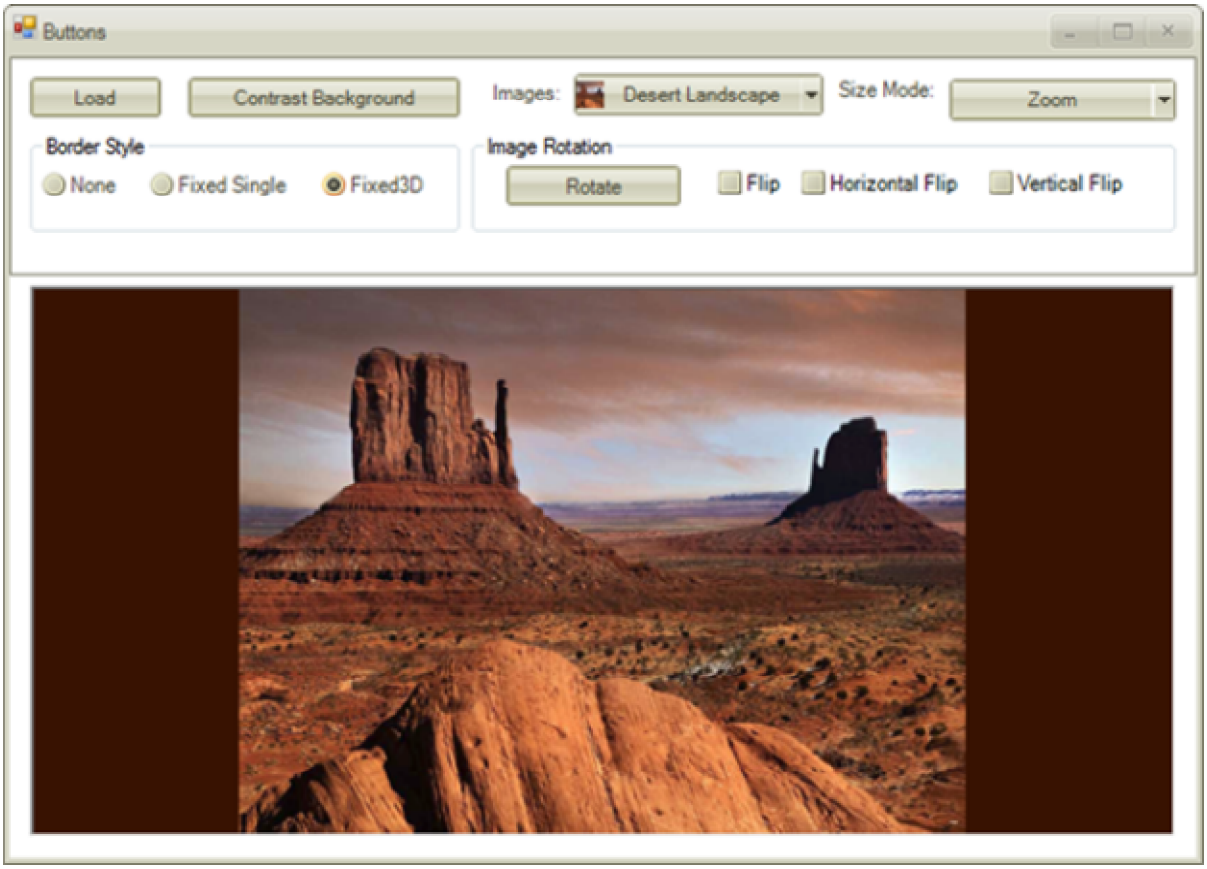
   [VB] Adding a Call to ChooseImagesClick

```
Private Sub ChooseImagesClick(ByVal sender As Object, ByVal e As EventArgs)
 ' . . .
 SetBackgroundColor(tbBackgroundColor.ToggleState)
End Sub
```

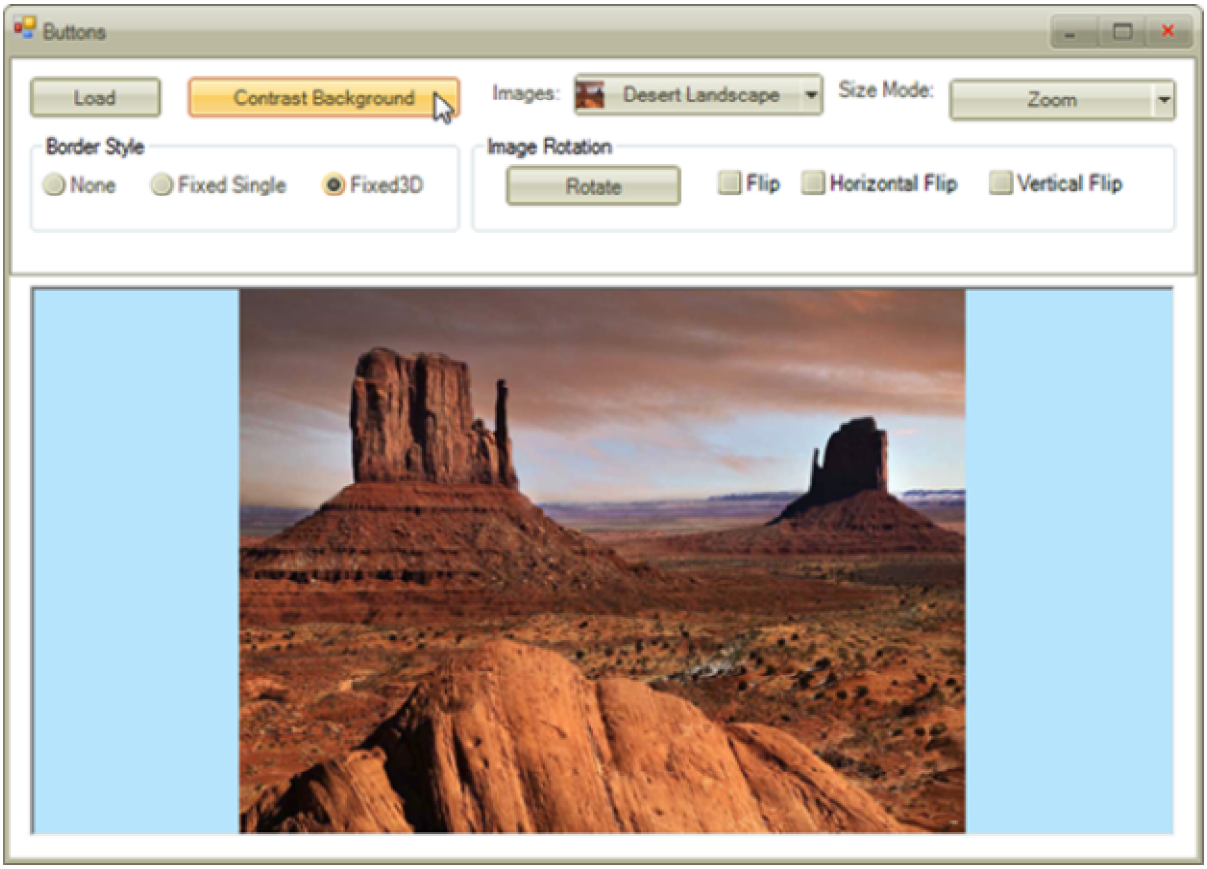
   [C#] Adding a Call to ChooseImagesClick

```
private void ChooseImagesClick(object sender, EventArgs e)
{
 // . . .
 SetBackgroundColor(tbBackgroundColor.ToggleState);
}
```

5. Press **Ctl-F5** to run the application to see how the application works so far. The initial view when loading any image using the "Load" button or "Images" split button list should display a background color taken from a pixel in the center of the image.



Clicking the "Contrast Background" color switches the background color to a contrasting color:



Working With ToggleState - On/Off/Indeterminate

1. Create a common event handler for both the "Horizontal Flip" and "Vertical Flip" checkboxes. If both checkboxes are on or off, the "Flip" check box toggle state is set to match. If one checkbox is On and another Off, the set the "Flip" check box toggle state to Indeterminate.

[VB] Handling the Horizontal and Vertical "Flip" Button ToggleStateChanged Events

```
Private Sub HorizVert_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs)
 ' true if both checkboxes are off
 Dim bothOff As Boolean = (cbHorizontalFlip.ToggleState = ToggleState.Off) AndAlso
(cbVerticalFlip.ToggleState = ToggleState.Off)
 ' true if both checkboxes are on
 Dim bothOn As Boolean = (cbHorizontalFlip.ToggleState = ToggleState.[On]) AndAlso
(cbVerticalFlip.ToggleState = ToggleState.[On])
 If bothOn Then
  cbFlip.ToggleState = ToggleState.[On]
ElseIf bothOff Then
  cbFlip.ToggleState = ToggleState.Off
 Else
  cbFlip.ToggleState = ToggleState.Indeterminate
 End If
End Sub
```

[C#] Handling the Horizontal and Vertical "Flip" Button ToggleStateChanged Events

```
private void HorizVert_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
 // true if both checkboxes are off
 bool bothOff = (cbHorizontalFlip.ToggleState == ToggleState.Off) &&
   (cbVerticalFlip.ToggleState == ToggleState.Off);
 // true if both checkboxes are on
 bool bothOn = (cbHorizontalFlip.ToggleState == ToggleState.On) &&
   (cbVerticalFlip.ToggleState == ToggleState.On);
 if (bothOn)
   cbFlip.ToggleState = ToggleState.On;
 else if (bothOff)
   cbFlip.ToggleState = ToggleState.Off;
 else
   cbFlip.ToggleState = ToggleState.Indeterminate;
}
```

2. Add a "Flip" checkbox ToggleStateChanged event handler to make the horizontal and vertical check box toggle states agree with the "Flip" checkbox.

   This handler simply assigns the toggle state of the "Flip" checkbox to both the horizontal and vertical check box toggle states. To avoid unexpected behaviors here, unhook the ToggleStateChanged event handlers for the horizontal and vertical check boxes, make the assignments, then rehook the event handlers.

   [VB] Handling the "Flip" Check Box ToggleStateChanged Event

```
Private Sub cbFlip_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs)
 If args.ToggleState <> ToggleState.Indeterminate Then
  ' unhook event handler temporarily
  cbHorizontalFlip.ToggleStateChanged -= HorizVert_ToggleStateChanged
  cbVerticalFlip.ToggleStateChanged -= HorizVert_ToggleStateChanged
  cbHorizontalFlip.ToggleState = args.ToggleState
  cbVerticalFlip.ToggleState = args.ToggleState
  ' re-hook event handler
  cbHorizontalFlip.ToggleStateChanged += HorizVert_ToggleStateChanged
  cbVerticalFlip.ToggleStateChanged += HorizVert_ToggleStateChanged
 End If
End Sub
```

   [C#] Handling the "Flip" Check Box ToggleStateChanged Event

```
private void cbFlip_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
 if (args.ToggleState != ToggleState.Indeterminate)
 {
   // unhook event handler temporarily
   cbHorizontalFlip.ToggleStateChanged -= HorizVert_ToggleStateChanged;
   cbVerticalFlip.ToggleStateChanged -= HorizVert_ToggleStateChanged;
   cbHorizontalFlip.ToggleState = args.ToggleState;
   cbVerticalFlip.ToggleState = args.ToggleState;
   // re-hook event handler
   cbHorizontalFlip.ToggleStateChanged += HorizVert_ToggleStateChanged;
   cbVerticalFlip.ToggleStateChanged += HorizVert_ToggleStateChanged;
 }
}
```

> The "ControlDefault" theme has a good representation of the Indeterminate state for the RadCheckBox control.

3. Handle the click event for the "Rotate" RadRepeatButton. As the button is held down by the user, this event fires continually, once a second. The image is rotated and fliped each time based on the check box states.

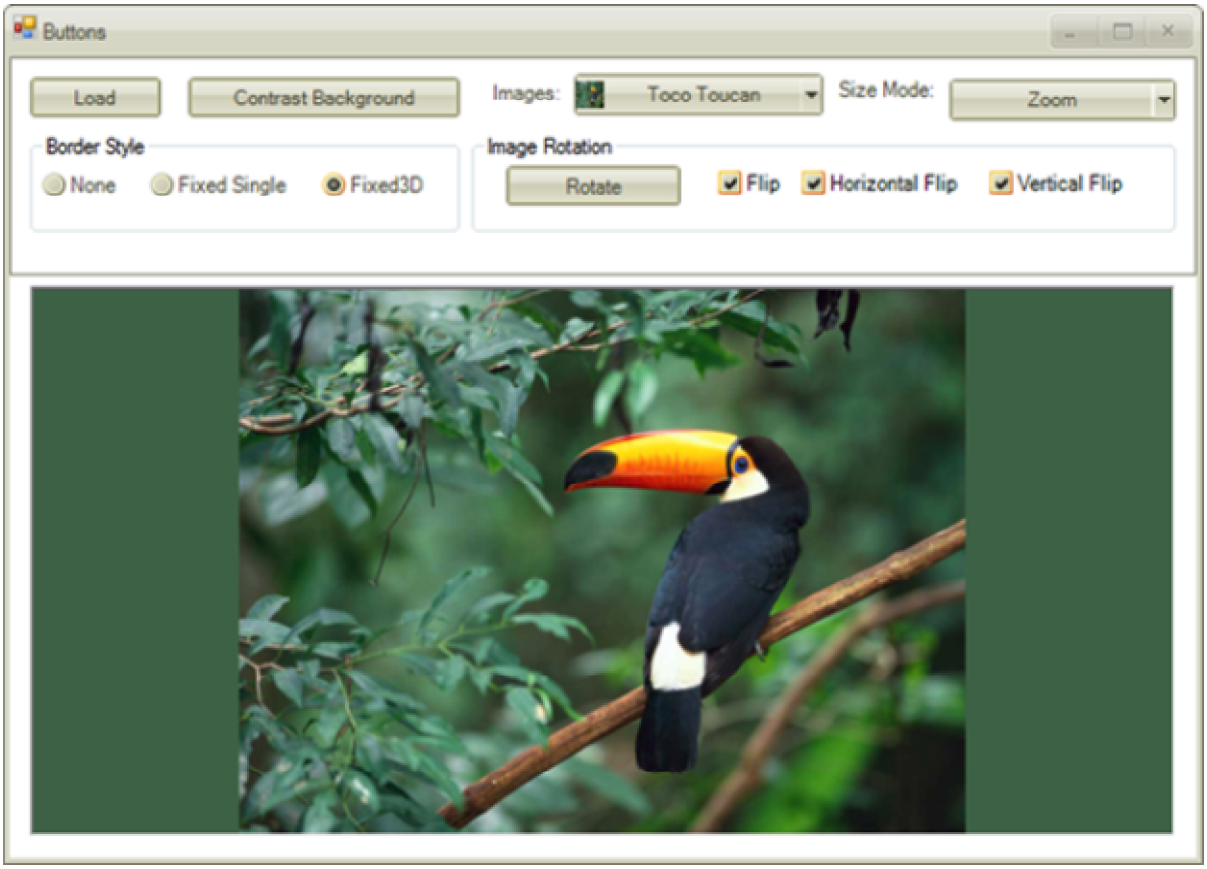
[VB] Handling the Click Event for the "Rotate" Button

```
Private Sub RotateClick(ByVal sender As Object, ByVal e As EventArgs)
 If cbFlip.ToggleState = ToggleState.[On] Then
  pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipXY)
ElseIf cbHorizontalFlip.ToggleState = ToggleState.[On] Then
  pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipX)
ElseIf cbVerticalFlip.ToggleState = ToggleState.[On] Then
  pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipY)
 Else
  pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipNone)
 End If
 pictureBox1.Refresh()
End Sub
```

[C#] Handling the Click Event for the "Rotate" Button

```
private void RotateClick(object sender, EventArgs e)
{
 if (cbFlip.ToggleState == ToggleState.On)
   pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipXY);
 else if (cbHorizontalFlip.ToggleState == ToggleState.On)
   pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipX);
 else if (cbVerticalFlip.ToggleState == ToggleState.On)
   pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipY);
 else
   pictureBox1.Image.RotateFlip(RotateFlipType.Rotate90FlipNone);
 pictureBox1.Refresh();
}
```

4. Press **Ctl-F5** to run the application. Try all the combinations of the check boxes to see how that changes the look and behavior.



3.6 Summary

In this chapter you took a tour of the various types of buttons available in the RadControls for Winforms suite. You learned the basic characteristics, behavior and important properties of each button type. You used several design time interfaces to configure the buttons. You worked with the Items collections of both RadDropDownButton and RadSplitButton. You responded to events both from the buttons and from individual Items collection members. Finally, you learned how to manipulate button ToggleState and how to handle ToggleState related events.

4 Editors

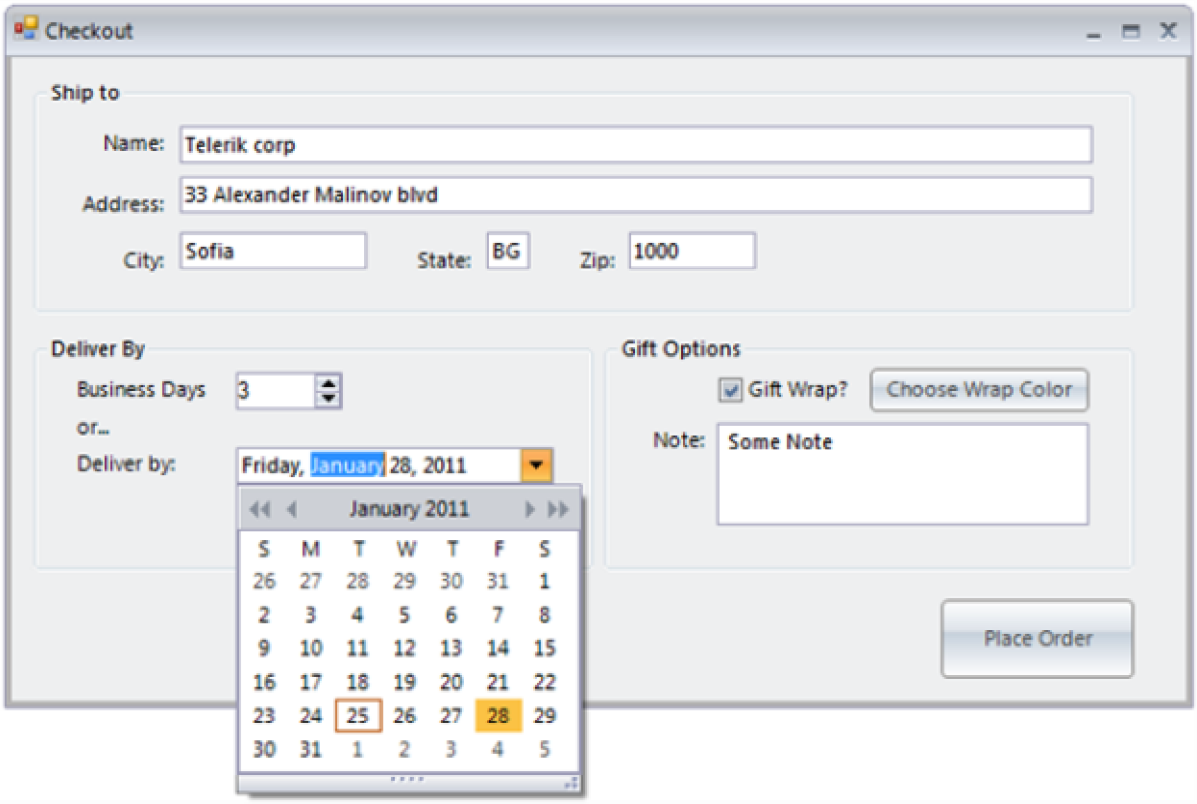
4.1 Objectives

- Learn usage and basic characteristics for each type of editor.
- Learn how to retrieve user information from RadTextBox, RadMaskedEditBox, RadSpinEditor, RadDateTimePicker, and RadColorDialog controls.
- Learn how to use edit masks and formats.
- Learn how to set tool tips.
- Learn how to validate input and different strategies for responding to incorrect data.

4.2 Introduction

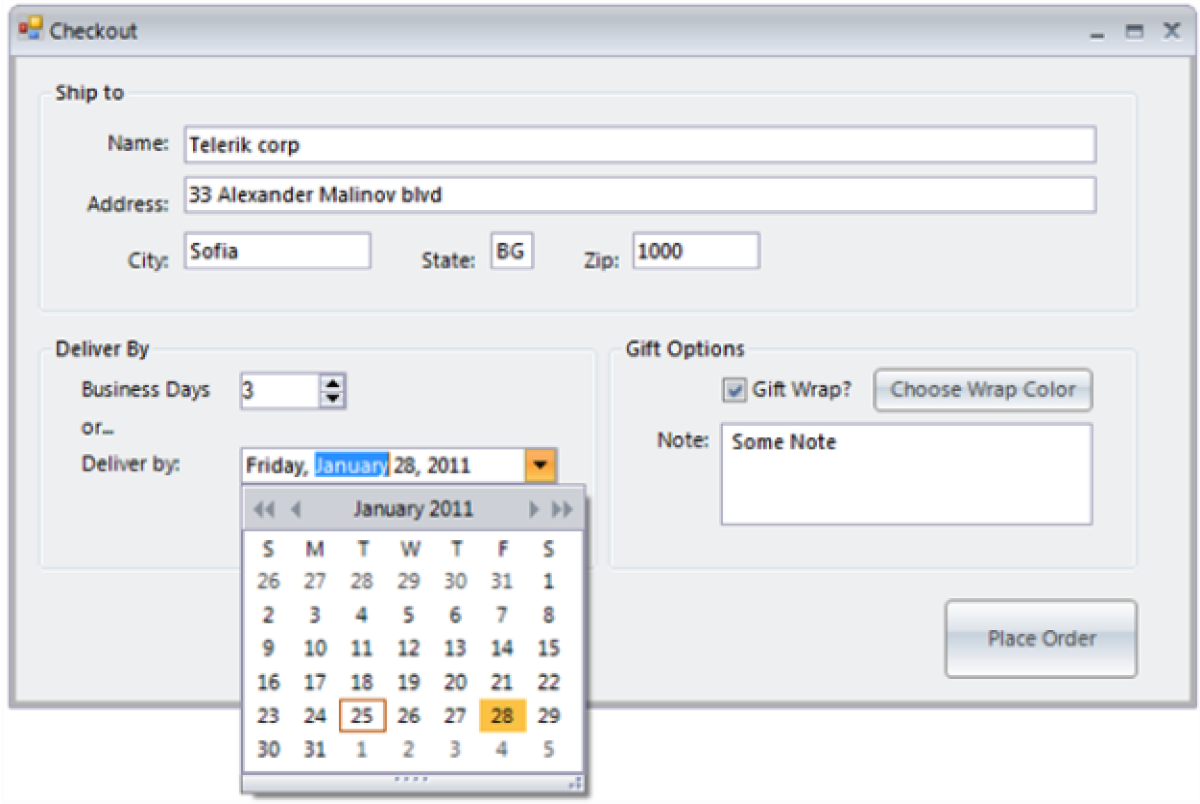
RadControls for Winforms edit controls are highly customizable, themable, intelligent components for controlled data input in Windows Forms applications. They include...

- **RadTextBox**: A themeable alternative to the standard TextBox control with enhanced properties such as NullText to hold prompt messages when there is no entry.
- **RadMaskedEditBox**: Adds input validation and masks, such as date, IP Address, SSN, phone number, digits, and decimals. This control is also themable and has a NullText property.
- **RadDateTimePicker:** Allows the user to enter dates directly or using a drop-down calendar. You can control the date format, set minimum and maximum dates, set the calendar to display for a specific culture and display a prompt when the control has a "null date".
- **RadSpinEditor**: A numeric entry control that allows direct input, keyboard control or the user may click on the up-down arrows. You can control the minimum and maximum value and the increment amount. You can also tailor formatting details such as the thousands separator and decimal places.
- **RadColorDialog**: A color dialog replacement with lots of bells and whistles including four different view tabs, Basic, Professional, Web and System, hex value entry, eye dropper control and custom colors.



4.3 Getting Started

In this project we create a no-frills checkout form using all editors. In later projects we will extend the example to set date time picker selections in code, assign tool tips and validate input.



Preparing the Project



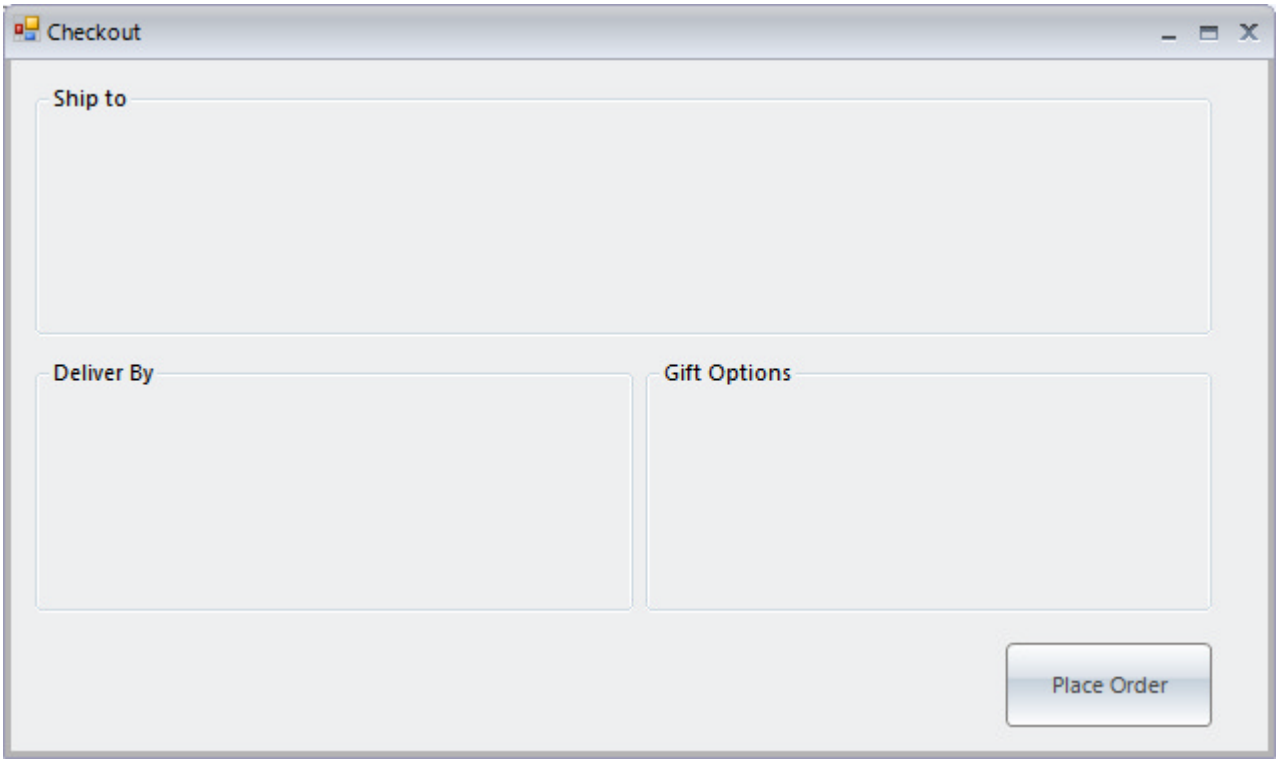
You can find the complete source for this project at:

\Editors\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.

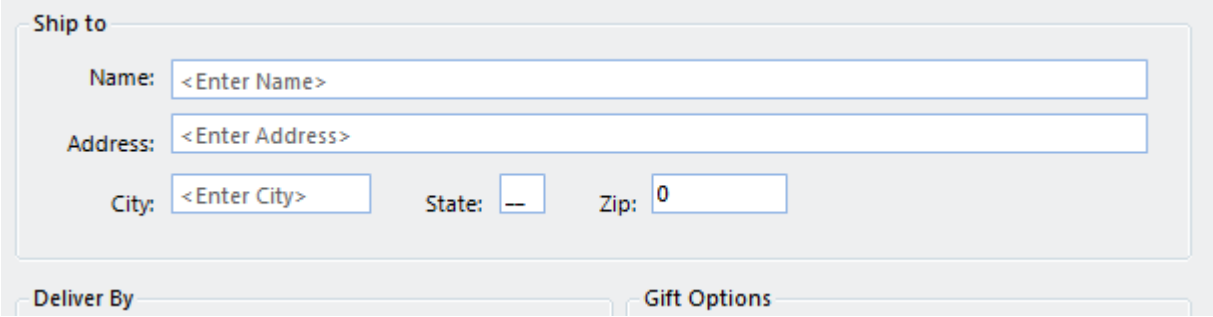
Adding Edit Controls to the RadForm

1. Drag an **Office2007SilverTheme** control to the RadForm. This step will add the theme control to the component tray below the form designer.
2. In the Properties window set the RadForm Properties:
   - **Size** = 630, 350
   - **MinimumSize** = 630, 350
   - **Text** = "Checkout"
   - **ThemeName** = "Office2007Silver"
3. Drop three standard GroupBox controls and a RadButton on the form. Arrange them to have roughly the same proportions as the example below:

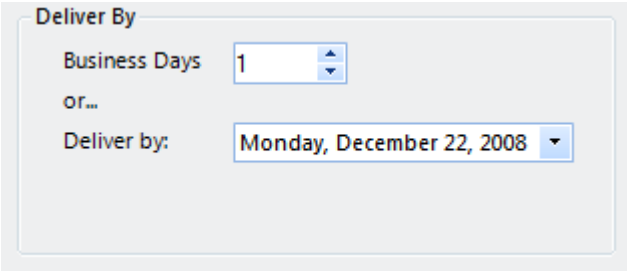


4. Set the control properties as follows:
   - The top group box: **Name** = "gbShipTo", **Anchor** = Top, Left, Right, **BackColor** = Transparent, **TabIndex** = 0, **Text** = "Ship to".
   - The bottom left group box: **Name** = "gbDelivery", **Anchor** = Top, Bottom, Left, **BackColor** = Transparent, **TabIndex** = 1, **Text** = "Deliver By".
   - The bottom right group box: **Name** = "gbGiftOptions", **Anchor** = Top, Bottom, Left, Right, **BackColor** = Transparent, **TabIndex** = 2, **Text** = "Gift Options".
   - The RadButton: **Name** = "btnPlaceOrder", **Anchor** = Bottom, Right, **TabIndex** = 3, **Text** = "Place Order", **Tag** = "Place Order".

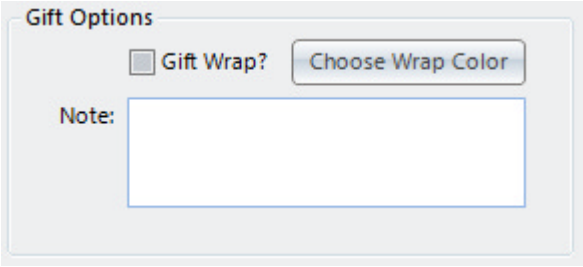
5. Add the following controls to the "gbShipTo" group box:
   1. RadLabel: **Text** = "Name:".
   2. RadTextBox: **Name** = tbName, **Anchor** = Left, Right", **NullText** = "<Enter Name>", **Tag** = "Name", **TabIndex** = 0.
   3. RadLabel: **Text** = "Address:".
   4. RadTextBox: **Name** = tbAddress, **Anchor** = Left, Right", **NullText** = "<Enter Address>", **Tag** = "Address", **TabIndex** = 1.
   5. RadLabel: **Text** = "City:".
   6. RadTextBox: **Name** = tbCity, **NullText** = "<Enter City>", **Tag** = "City", **TabIndex** = 2.
   7. RadLabel: **Text** = "State:".
   8. RadMaskedEditBox: **Name** = meState, **MaskType** = Standard, **Mask** = "LL", **Tag** = "State", **TabIndex** = 3.
   9. RadLabel: **Text** = "Zip:".
   10. RadMaskedEditBox: **Name** = meZip, **MaskType** = Standard, **Mask** = "99999-9999", **Tag** = "Zip", **TabIndex** = 4.



6. Add the following controls to the "gbDelivery" group box:
   1. RadLabel: **Text** = "Business Days".
   2. RadTextBox: **Name** = seDeliver, **CausesValidation** = False, **Tag** = "Business Days", **TabIndex** = 0.
   3. RadLabel: **Text** = "or...".
   4. RadLabel: **Text** = "Deliver by:".
   5. RadDateTimePicker: **Name** = "dtDeliver", **Tag** = "Deliver by", **TabIndex** = 1



7. Add the following controls to the "gbGiftOptions" group box:
   1. RadCheckBox: **Name** = "cbGiftWrap", **BackColor** = Transparent, **Text** = "Gift Wrap?", Tag = "Gift Wrap".
   2. RadButton: **Name** = "btnChooseColor", **Anchor** = Top, Left, Right, **Enabled** = False, **Text** = "Choose Wrap Color", **Tag** = "Choose Wrap Color".
   3. RadLabel: **Text** = "Note:".
   4. RadTextBox: **Name** = "tbNote", **Anchor** = "Top, Bottom, Left, Right", **Enabled** = False, **Tag** = "Gift Wrap Note".



8. Double-click the "Gift Wrap?" check box to create a ToggleStateChanged event handler. First add Telerik.Wincontrols.Enumerations to your "Imports" (VB) or "uses" (C#) section of code. Then add the following code to the ToggleStateChanged event handler:

[VB] Handling the ToggleStateChanged Event

```
Private Sub cbGiftWrap_ToggleStateChanged(sender As Object, args As
```

```
Telerik.WinControls.UI.StateChangedEventArgs)
 tbNote.Enabled = args.ToggleState = ToggleState.[On]
 btnChooseColor.Enabled = args.ToggleState = ToggleState.[On]
 If args.ToggleState <> ToggleState.[On] Then
  tbNote.Text = [String].Empty
 End If
End Sub
```

[C#] Handling the ToggleStateChanged Event

```
private void cbGiftWrap_ToggleStateChanged(object sender,
 Telerik.WinControls.UI.StateChangedEventArgs args)
{
 tbNote.Enabled = args.ToggleState == ToggleState.On;
 btnChooseColor.Enabled = args.ToggleState == ToggleState.On;
 if (args.ToggleState != ToggleState.On)
   tbNote.Text = String.Empty;
}
```

The behavior of the Gift Options group box will be to enable the "Choose Wrap Color" and "Note" text box when the "Gift Wrap?" check box is enabled.

9. Add two shell helper methods. These methods will be filled out later in this chapter.

[VB] Adding Helper Methods

```
Private Function ValidateControls(controls As Control.ControlCollection) As Boolean
 Return True
End Function
Private Function GetFormData(controls As Control.ControlCollection) As String
 Return Environment.NewLine + "...summary"
End Function
```

[C#] Adding Helper Methods

```
private bool ValidateControls(Control.ControlCollection controls)
{
 return true;
}
private string GetFormData(Control.ControlCollection controls)
{
 return Environment.NewLine + "...summary";
}
```

10. Double click the "Place Order" button to create a Click event handler. Add the code below to the event handler.

After ValidateControls() verifies the information on the form, the data is collected from the form and displayed using a RadMessageBox. Note: RadMessageBox is a themeable, flexible replacement for the standard MessageBox class. Also note that no validation or data collection is actually performed in this version of the project.

[VB] Creating a Click Event Handler

```
Private Sub btnPlaceOrder_Click(sender As Object, e As EventArgs)
 If ValidateControls(Me.Controls) Then
  Dim caption As String = "Order Summary"
  Dim message As String = "Thank you for your order" + Environment.NewLine + GetFormData
```

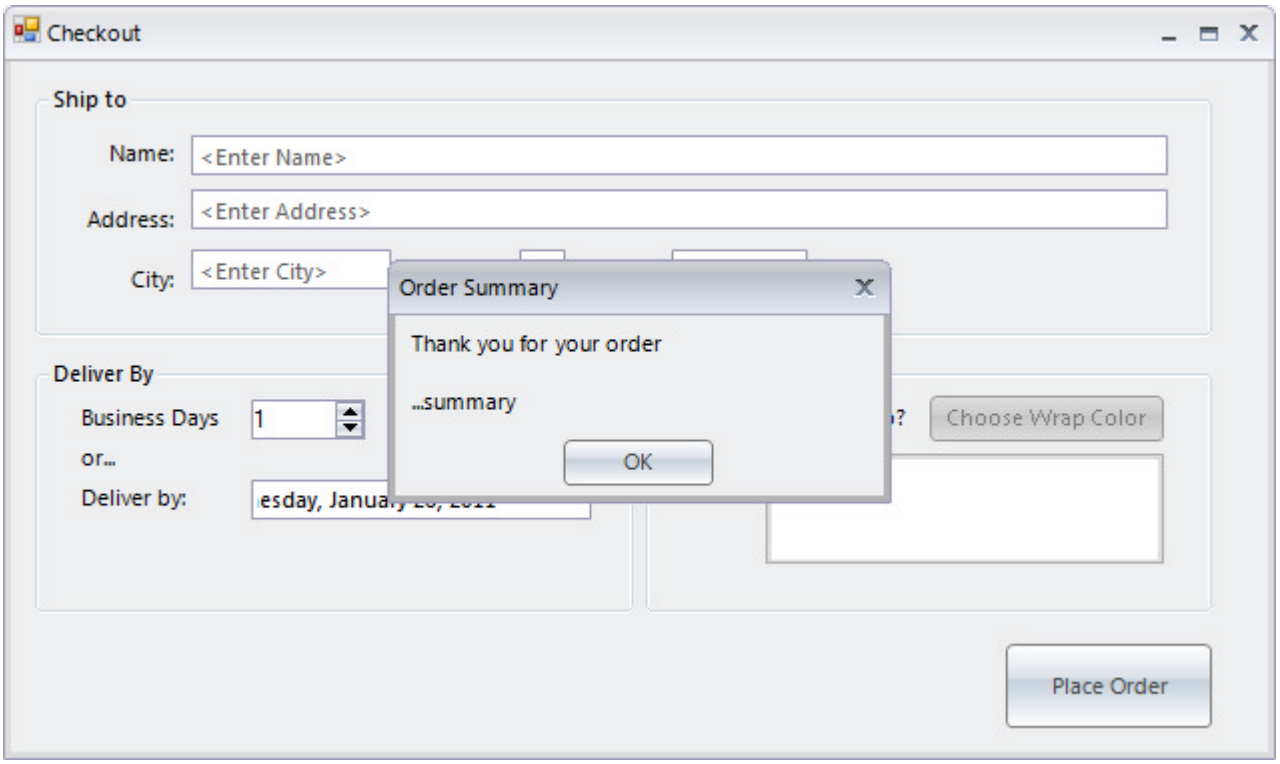
```
(Me.Controls)
RadMessageBox.SetThemeName("Office2007Silver")
  RadMessageBox.Show(message, caption, MessageBoxButtons.OK)
 End If
End Sub
```

[C#] Creating a Click Event Handler

```
private void btnPlaceOrder_Click(object sender, EventArgs e)
{
 if (ValidateControls(this.Controls))
 {
   string caption = "Order Summary";
   string message = "Thank you for your order" +
     Environment.NewLine +
     GetFormData(this.Controls);"
RadMessageBox.SetThemeName("Office2007Silver")

   RadMessageBox.Show(message, caption, MessageBoxButtons.OK);
 }
}
```

11. Press **Ctl-F5** to run the application. Notice the behavior of the **NullText** property in the RadTextBox controls and the **MaskType** and **Mask** properties for the two RadMaskedEditBox controls.



If you want to theme all controls to match the RadForm, you can create a method similar to this example that recursively sets the ThemeName for all RadControl types on the form. Here is the same form with the "Desert" theme set:



[VB] Setting the ThemeName for all Controls

```
' cycle through the controls collection recursively and set the
' theme for all rad controls
Private Sub SetTheme(controls As Control.ControlCollection)
 For Each control As Control In controls
  If TypeOf control Is RadControl Then
   (TryCast(control, RadControl)).ThemeName = Me.ThemeName
  End If
  If control.Controls.Count > 0 Then
   SetTheme(control.Controls)
  End If
 Next
End Sub
'...
' call it from the form's constructor:
Utils.SetTheme(Me.Controls)
```

[C#] Setting the ThemeName for all Controls

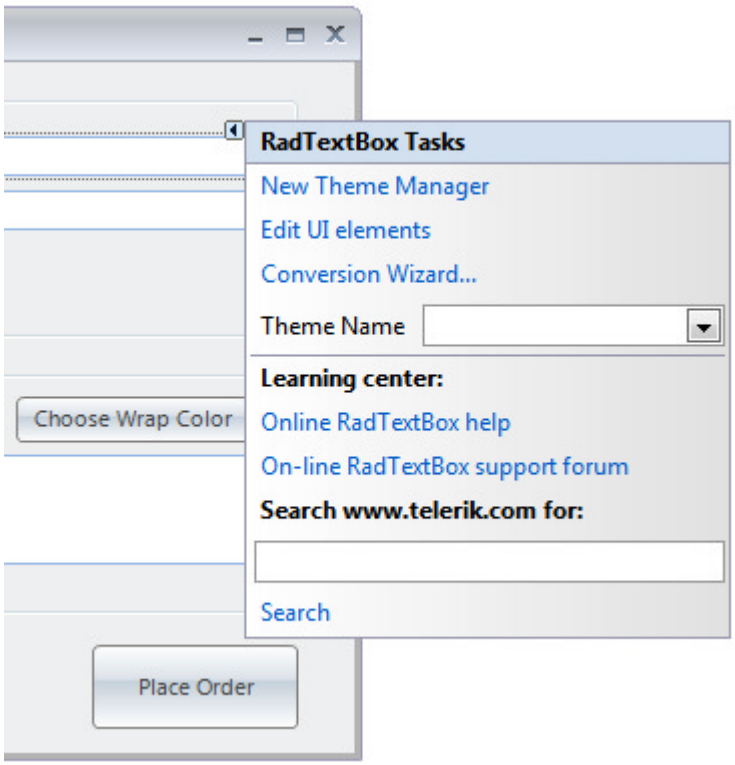
```
// cycle through the controls collection recursively and set the
// theme for all rad controls
private void SetTheme(Control.ControlCollection controls)
{
 foreach (Control control in controls)
 {
   if (control is RadControl)
     (control as RadControl).ThemeName = this.ThemeName;
   if (control.Controls.Count > 0)
     SetTheme(control.Controls);
 }
}
//...
// call it from the form's constructor:
Utils.SetTheme(this.Controls);
```

To set the theme for a RadMessageBox, call the static RadMessageBox.SetThemeName() method prior to calling Show().

4.4 Using the Design Time Interface

Smart Tag

The editor controls all have the same basic Smart Tag minimum layout that you see for all RadControls.



Tasks

From the Smart Tag Tasks menu you can Open Theme Builder to style all aspects of your control, select New Theme Manager to add a Theme Manager component to the component tray, Edit UI elements to browse all of the elements of the editor and to change properties for any element. The Theme Name drop down lets you pick an existing theme to style your control.

Learning center

The Learning center lets you navigate to a web browser with online help for the currently selected control or to the online support forums.

Search

Enter search criteria in the edit space provided and click the Search link to navigate directly to search on the Telerik web site.

4.5 Working With the Editor Controls

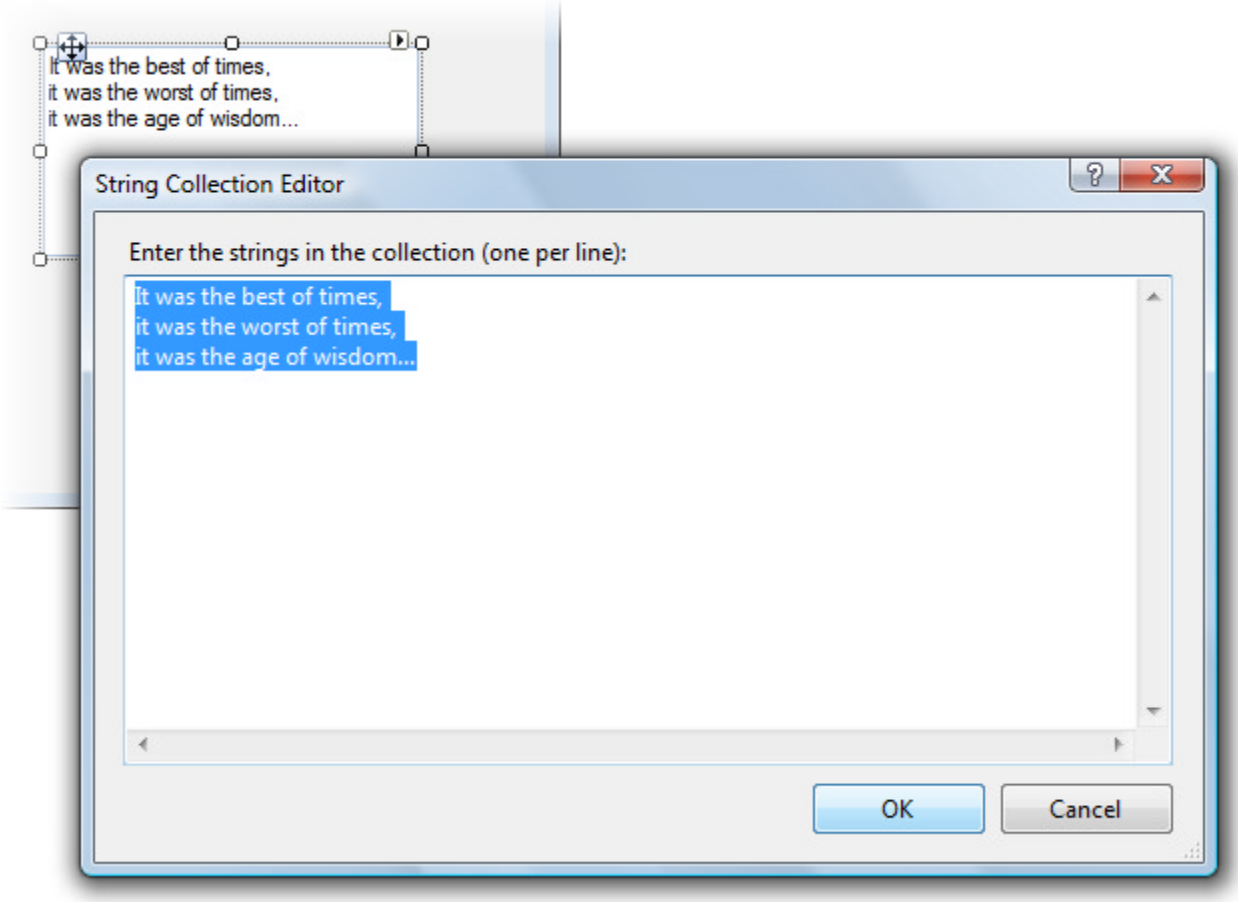
Text Box

Many of the properties of RadTextBox and RadMaskedEdit actually come from their common ancestor class, RadTextBoxBase. These properties match and surpass the same properties in the standard TextBox control. Here are some of the important functional groups of properties you should be looking for:

Multi-Line Behavior

When the **MultiLine** property is true, text can span more than one line and the text box can be resized. In

the Properties window, you can click the ellipses of the **Lines** property and use the Strings Collection Editor dialog:



You can programmatically set multiple lines of text by assigning an array of strings to the **Lines** property or a single string with embedded new line characters to the **Text** property. To indicate new lines you can add the System.Environment.NewLine string or use the language specific carriage return/line feed ("& vbCr & vbLf" in Visual Basic .NET and the "\r\n" escape characters in C#).

[VB] Assigning multiple lines of text

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 radTextBox1.Lines = New String(3) {"It was the best of times,", "it was the worst of
times,", "it was the age of wisdom..."}
 radTextBox1.Text = "It was the best of times," + Environment.NewLine + "it was the worst of
times," + Environment.NewLine + "it was the age of wisdom..."
 radTextBox1.Text = "It was the best of times," & vbCr & vbLf & "it was the worst of times,"
& vbCr & vbLf & "it was the age of wisdom..."
End Sub
```

[C#] Assigning multiple lines of text

```
private void Form1_Load(object sender, EventArgs e)
{
 radTextBox1.Lines = new string[3]
 { "It was the best of times,",
   "it was the worst of times,",
   "it was the age of wisdom..."};
 radTextBox1.Text =
   "It was the best of times," + Environment.NewLine +
```

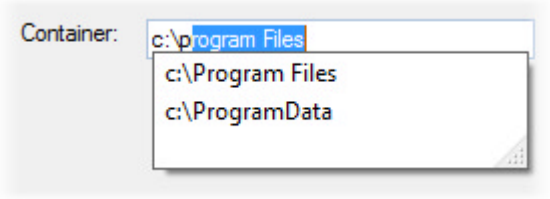
```
   "it was the worst of times," + Environment.NewLine +
   "it was the age of wisdom...";
 radTextBox1.Text =
 "It was the best of times,\r\nit was the worst of times,\r\nit was the age of wisdom...";
}
```

For edit controls accepting multiple lines of text, you may want the enter and tab keys to be interpreted literally instead of performing their usual functions. Set the **AcceptsReturn** property to true so that typing Enter creates a new line of text in the control instead of activating the default button for the form. Set **AcceptsTab** true so that pressing the Tab key types a tab character instead of moving the focus to the next control in the tab order.

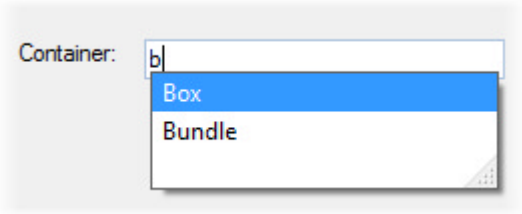
You can display scroll bars in your multi-line text box by setting the **ScrollBars** property away from the default "None" to "Horizontal", "Vertical" or "Both". Text will wrap automatically by default, but if you are using horizontal scroll bars, you could also set **WordWrap** to false.

Auto Completion

The "Auto Completion" feature provides "type ahead" and selection from a list of choices. You may have noticed this behavior in some web browsers as you type in a URL. To turn on this ability, set the **AutoCompleteMode** property from "None" to "Suggest", "Append" or "SuggestAppend". "Suggest" drops down a list of selections for the user to choose from and "Append" automatically completes the entry to the nearest match in the list. You can see the effect of how "SuggestAppend" combines both behaviors in the screenshot example below, where the entry starting with "c:\p" is automatically appended with "rogram Files" and a list of suggestions also appears.



Use the **AutoCompleteSource** property to designate where the suggestions come from, i.e. "FileSystem", "CustomSource", etc. To define your own auto completion suggestions, add to the **AutoCompleteCustomSource** array of strings and set the **AutoCompleteMode** to "CustomSource". The example below creates a custom list of choices:



[VB] Customizing Auto Complete Behavior

```
tbAutoComplete.AutoCompleteSource = AutoCompleteSource.CustomSource
Dim containers As New AutoCompleteStringCollection()
containers.Add("Box")
containers.Add("Bundle")
containers.Add("Pallet")
containers.Add("Carton")
tbAutoComplete.AutoCompleteCustomSource = containers
tbAutoComplete.AutoCompleteMode = AutoCompleteMode.SuggestAppend
```

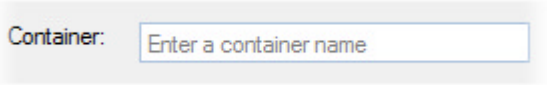
[C#] Customizing Auto Complete Behavior

```
tbAutoComplete.AutoCompleteSource = AutoCompleteSource.CustomSource;
```

```
AutoCompleteStringCollection containers = new AutoCompleteStringCollection();
containers.Add("Box");
containers.Add("Bundle");
containers.Add("Pallet");
containers.Add("Carton");
tbAutoComplete.AutoCompleteCustomSource = containers;
tbAutoComplete.AutoCompleteMode = AutoCompleteMode.SuggestAppend;
```

Null Text

This unique enhancement displays a prompt when there is no entry. The prompt appears as lightly colored text (SystemColors.GrayText) and disappears when the user tabs into the text box.



[VB] Setting Null Text

```
tbContainer.Text = [String].Empty
tbContainer.NullText = "Enter a container name"
```

[C#] Setting Null Text

```
tbContainer.Text = String.Empty;
tbContainer.NullText = "Enter a container name";
```

Events

To be notified when the contents of the text box changes, hook up the **TextChanging** and **TextChanged** events. Arguments supplied to TextChanging provide the NewValue and OldValue string properties; set Cancel true to prevent the text from changing.

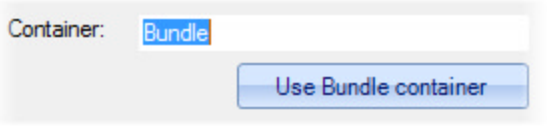
[VB] Handling the TextChanging Event

```
Private Sub tbContainer_TextChanging(sender As Object, e As
Telerik.WinControls.TextChangingEventArgs)
 btnPlaceOrder.Enabled = e.NewValue.Length > 0
End Sub
```

[C#] Handling the TextChanging Event

```
private void tbContainer_TextChanging(object sender,
Telerik.WinControls.TextChangingEventArgs e)
{
 btnPlaceOrder.Enabled = e.NewValue.Length > 0;
}
```

The TextChangedEvent fires as each character is typed. Use the Text or Lines property of the text box to access text box contents:



[VB] Handling the TextChanged Event

```
Private Sub tbContainer_TextChanged(sender As Object, e As EventArgs)
 btnUseContainer.Text = "Use " + (TryCast(sender, RadTextBox)).Text + " container"
End Sub
```

[C#] Handling the TextChanged Event

```
private void tbContainer_TextChanged(object sender, EventArgs e)
{
 btnUseContainer.Text = "Use " + (sender as RadTextBox).Text + " container";
}
```

Height

By default, both RadTextBox and RadMaskedEdit box cannot be resized vertically; set the MultiLine property true to size either control freely.

Borders

To remove the border of a text box so that it looks more like a label, you need to locate the TextBoxElement and set its visibility to "Collapsed". The background for this will be presented in the chapter "Telerik Presentation Foundation", but for now, either use the code below...

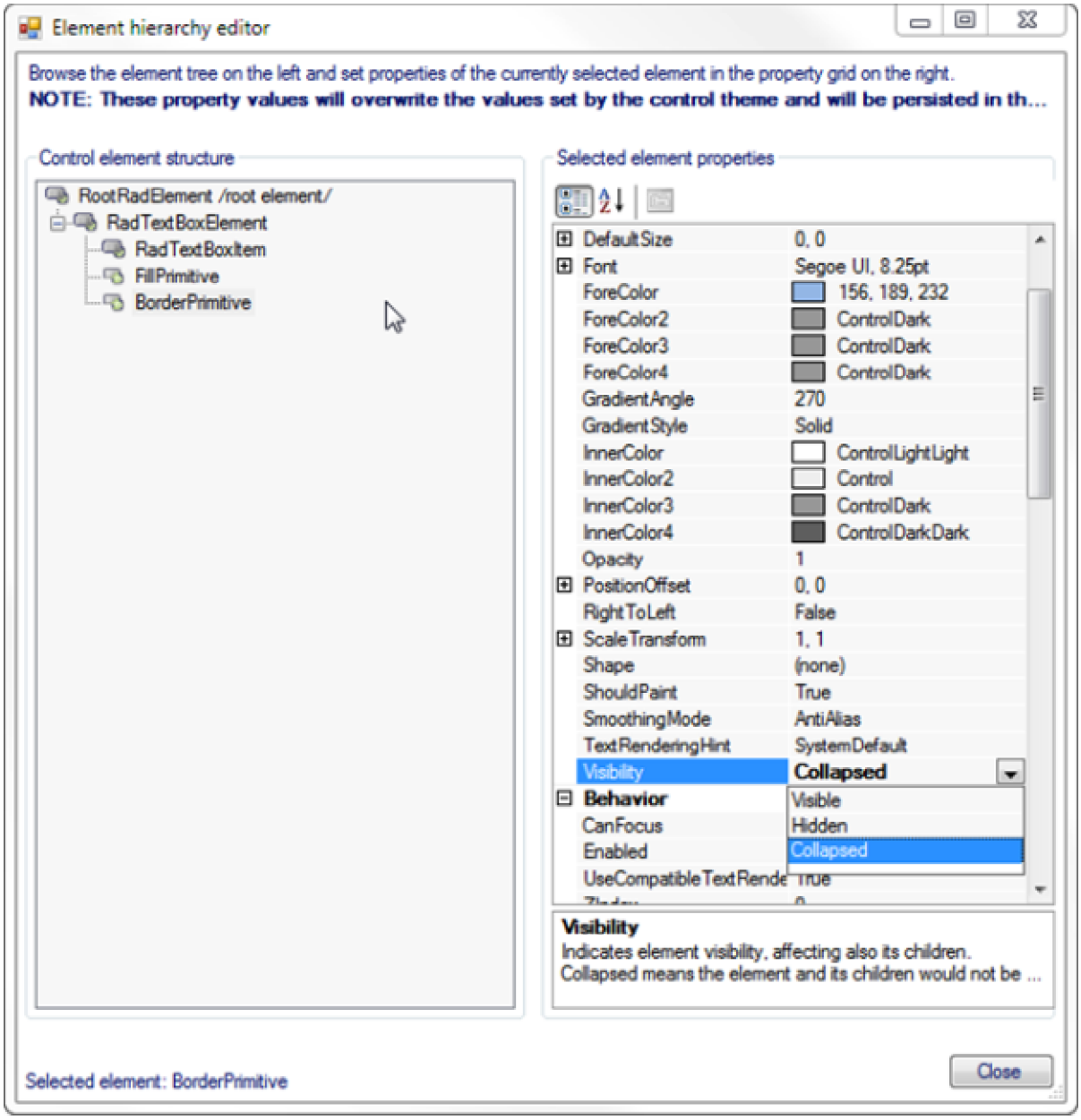
[VB] Hiding Text Box Borders

```
(TryCast(tbMultiLine.TextBoxElement.Children(2), BorderPrimitive)).Visibility =
ElementVisibility.Collapsed
```

[C#] Hiding Text Box Borders

```
(tbMultiLine.TextBoxElement.Children[2] as BorderPrimitive).Visibility =
ElementVisibility.Collapsed;
```

...or, click the Smart Tag "Edit UI Elements" link, locate the "BorderPrimitive" element in the tree view, navigate to the Appearance properties and set the Visibility property to "Collapsed".





You can find the complete source for this project at:

\Editors\<VB|CS>\EditorsOverview

Masked Edit Box

RadMaskedEditBox lets you set up a template or "Mask" that input must conform to. This is especially helpful for input with specific format requirements like social security numbers, phone numbers, dates and numerics.

Masks

The mask helps the user know what is expected and does not allow incorrect input. Instead of the Text property use the **Value** property. Value is an Object where the actual type varies depending on the type of mask you are using. The critical properties are **MaskType** and **Mask**. MaskType can be **None**, **Standard**,

DateTime, Numeric, Regex, IP, Email. The Mask string interpretation varies based on the MaskType setting. For example, "d" is a decimal for a Numeric mask type and a short date pattern for a DateTime mask type.

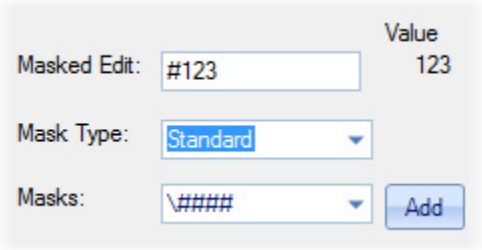
Standard Masks

The table below describe the mask characters that can be used when the **MaskType** property is set to **Standard**.

Mask Character	Description
0	Digit, required. This element will accept any single digit between 0 and 9.
9	Digit or space, optional.
#	Digit or space, optional. If this position is blank in the mask, it will be rendered as the character in the **PromptChar (http://www.telerik.com/help/winforms/editors-maskededitbox-working-with-radmaskededitbox.html)** property. For example, "$######.##" displays as a literal "$", accepts a numeric amount with six places and two places to the right of the decimal, i.e. $123456.56.
L	Accepts letters only. Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a back slash. For example, "\# ###", i.e. "# 123".
?	Letter, optional.
?	Character, required.
?	Character, optional.
A	Alphanumeric, required. Accepts any symbol.
a	Alphanumeric, optional.
.	Decimal placeholder.
,	Thousands placeholder.
:	Time separator.
/	Date separator.
$	Currency symbol.
<	Shift down. Converts all characters that follow to lowercase.
>	Shift up. Converts all characters that follow to uppercase.
|	Disable a previous shift up or shift down.
\	Escape. Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a backslash.

All other characters	All non-mask elements will appear as themselves within **RadMaskedEditBox**. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user

In the example below, the Mask is "\####", which produces a literal "#" pound sign and three numeric characters. If "123" is typed in to the masked edit, you will see "#123".



To get the input for a RadMaskedEditBox, use the Value property. In this case the Value property is "123" and the Text property is "#123".

Numeric Masks

If the mask type is "Numeric" another set of masks applies and the mask can be followed by a precision specifier. By default, these numbers will be formatted as defined by CultureInfo.CurrentCulture.NumberFormat. When the mask is "Numeric", the user can increment and decrement using the up and down arrow keys.

Standard Numeric Masks

Full list of the supported masks (except "E" and "X") can be found in the following MSDN article:**Standard Numeric masks (http://msdn.microsoft.com/en-us/library/dwhawy9k.aspx)**

Mask	Description
"C" or "c"	Currency
"D" or "d"	Decimal
"F" or "f"	Fixed-point
"G" or "g"	General
"N" or "n"	Number
"P" or "p"	Percent
"R" or "r"	Round-trip

Custom Numeric Masks

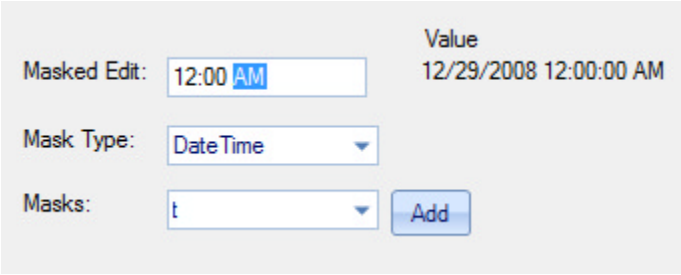
Full list of the supported masks (except "E" and "X") can be found in the following MSDN article:**Custom Numeric masks (http://msdn.microsoft.com/en-us/library/0c899ak8.aspx)**

Mask	Description

"0"	Zero placeholder
"#"	Digit placeholder
"."	Decimal point
","	Group separator and number scaling
"%"	Percentage placeholder
"‰"	Per mille placeholder
"\"	Escape character
'string' or "string"	Literal string delimeter
";"	Section separator

Date Time Masks

When the mask type is "DateTime", the masked edit allows the user to type the date in directly or use the arrow keys to increment and decrement all the elements of the date or time. The example below shows a short time mask. The RadMaskedEditBox has the AM/PM portion of the time highlighted. Using the arrow keys here toggles between AM and PM. The Value property is a DateTime type.



Standard Date Format codes

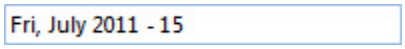
Format Character	Description
d	Short date pattern
D	Long date pattern
f	Full date and time (long date and short time)
F	Full date time pattern (long date and long time)
g	General (short date and short time)
G	General (short date and long time)

m, M	Month day pattern
r, R	RFC1123 pattern
s	Sortable date time pattern (based on ISO 8601) using local time
t	Short time pattern
T	Long time pattern

Custom Date Format codes

Where standard date codes are too restrictive, create your own custom formats using the format patterns listed below.

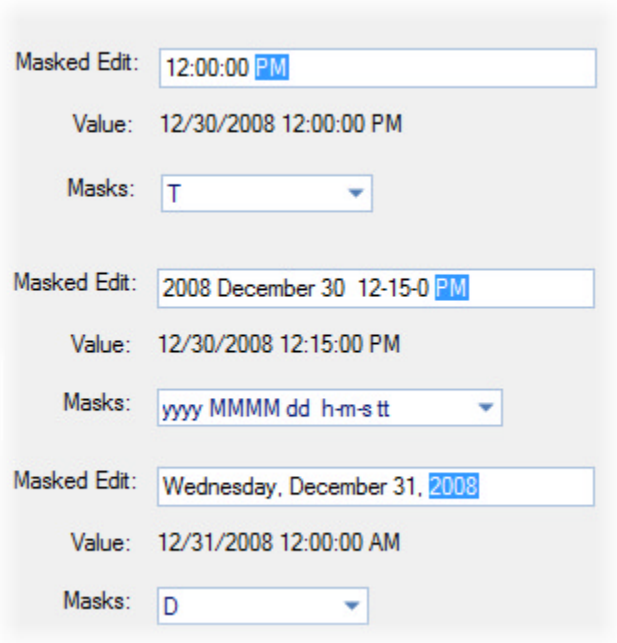
For example, a DateFormat of "ddd, MMMM yyyy - dd" might display:



Format Pattern	Description
dd	The numeric day of the month.
ddd	The abbreviated name of the day of the week.
M	The month name followed by the numeric day.
MM	The numeric month.
MMM	The abbreviated name of the month.
MMMM	The full name of the month.
y	The full month name and year numeric.
yy	The year without the century.
yyy	The year in four digits, including the century.
h	The hour in a 12-hour clock.
hh	The hour in a 12-hour clock.
H	The hour in a 24-hour clock.
HH	The hour in a 24-hour clock.
m	The minute.
mm	The minute.

s	The second.
ss	The second.
t	The first character in the AM/PM designator.
tt	The AM/PM designator.

Here are several examples that show the RadMaskedEditBox in action along with the Value property expressed using ToString() and the Mask:



Other Properties

- **Culture**: This property allows you to set the current language and culture from a drop down list at design-time or assign a new CultureInfo instance at run-time. For example, you can assign the culture code for Chinese:

  [VB] Setting the Culture

```
meDemo.Mask = "D"
meDemo.Culture = New CultureInfo("zh-CN")
```

  [C#] Setting the Culture

```
meDemo.Mask = "D";
meDemo.Culture = new CultureInfo("zh-CN");
```

  ...and this results in the RadEditor looking something like this example:

  2008年12月30日

- **PromptChar**: This property represents the character displayed in any blank space defined by a Mask character. By default the character is an underscore "_".

Events

Use the **ValueChanging** and **ValueChanged** events to retrieve new values. ValueChanging passes a CancelEventArgs property with a single Cancel property that can be set true to prevent the value from changing. To get at the Text and Value properties, you can reference the control directly or cast "sender" to be RadMaskedEditBox and access the properties that way.

[VB] Handling the ValueChanged Event

```
Private Sub meDemo_ValueChanged(sender As Object, e As EventArgs)
 lblValue.Text = Me.meDemo.Value.ToString()
End Sub
```

[C#] Handling the ValueChanged Event

```
private void meDemo_ValueChanged(object sender, EventArgs e)
{
 lblValue.Text = this.meDemo.Value.ToString();
}
```

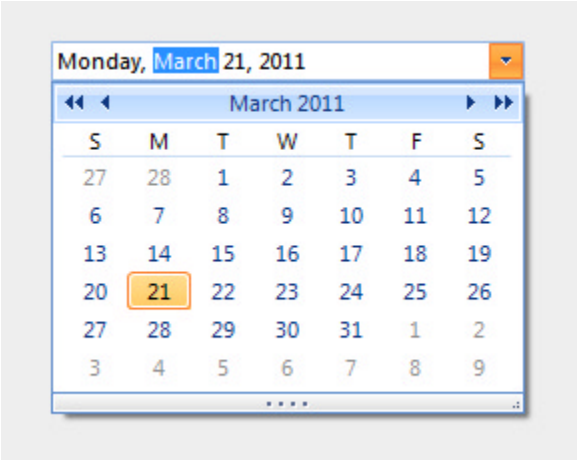


You can find the complete source for this project at:

\Editors\<VB|CS>\EditorsOverview

Date Time Picker

The RadDateTimePicker goes beyond the RadMaskedEditBox in DateTime mode by adding a drop down calendar and some additional specialized properties.



Date Time Picker Values

To set the current date, assign the **Value** property. You can also set **MinDate** and **MaxDate** to limit the dates that can be selected in the calendar. The calendar will not even allow the user to navigate to other months on the calendar that are not within the min and max dates. The example below shows the date picker Value set to "today", the earliest date that can be selected is "yesterday" and the latest date that can be selected is 7 days from now.

[VB] Setting the Value, MinDate and MaxDate

```
radDateTimePicker1.MinDate = DateTime.Today.AddDays(-1)
radDateTimePicker1.MaxDate = DateTime.Today.AddDays(7)
radDateTimePicker1.Value = DateTime.Today
```

[C#] Setting the Value, MinDate and MaxDate

```
radDateTimePicker1.MinDate = DateTime.Today.AddDays(-1);
radDateTimePicker1.MaxDate = DateTime.Today.AddDays(7);
radDateTimePicker1.Value = DateTime.Today;
```

Format

The **Format** property of the date time picker can be **Long**, **Short**, **Time** and **Custom**. These settings control date or time representation in the text portion of the control. If you set the Format to Custom, you can also set the **CustomFormat** property. The code example below displays something like: "2008 December 30 11:45:56 AM".

[VB] Assigning Format and CustomFormat Properties

```
radDateTimePicker1.Format = DateTimePickerFormat.[Custom]
radDateTimePicker1.CustomFormat = "yyyy MMMM dd h:m:s tt"
```

[C#] Assigning Format and CustomFormat Properties

```
radDateTimePicker1.Format = DateTimePickerFormat.Custom;
radDateTimePicker1.CustomFormat = "yyyy MMMM dd h:m:s tt";
```

See the Masked Edit Box "Date Time Formats" section for a complete list of characters that can be used in the CustomFormat property.

Internationalization

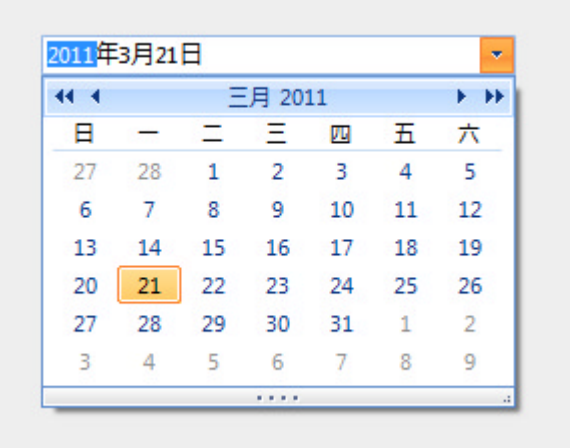
Like the RadMaskedEditBox in DateTime mode, RadDatePicker also has internationalization support.

[VB] Assigning the Culture

```
radDateTimePicker1.Culture = New CultureInfo("zh-CN")
```

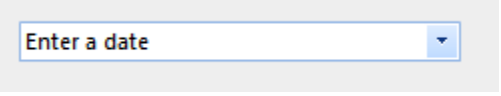
[C#] Assigning the Culture

```
radDateTimePicker1.Culture = new CultureInfo("zh-CN");
```



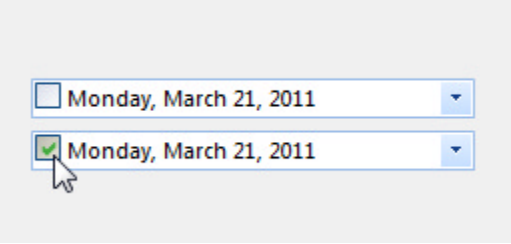
Null Text and Null Dates

You can set the **NullText** property for a RadDateTimePicker control like the other editors. The NullText appears when the **Value** property equals the **NullDate** property date. NullDate is 1/1/1900 by default. In the example below, the initial Value is set to "1/1/1900" and NullText to "Enter a date".



Adding CheckBoxes and Up Down Arrows

Enable **ShowCheckBox** to add a checkbox to the date time picker. When the checkbox is un-checked, the date is shown in a gray font, and the text area of the date time picker is not editable.



To get or set the check mark for the checkbox, use the **Checked** property. To access events for the checkbox, currently you need to drill down to the embedded RadCheckBoxElement and subscribe to its events. The example below navigates through the DateTimePickerElement to locate the RadCheckBoxElement.

[VB] Handling the ToggleStateChanged Event of the Date Time Picker Checkbox

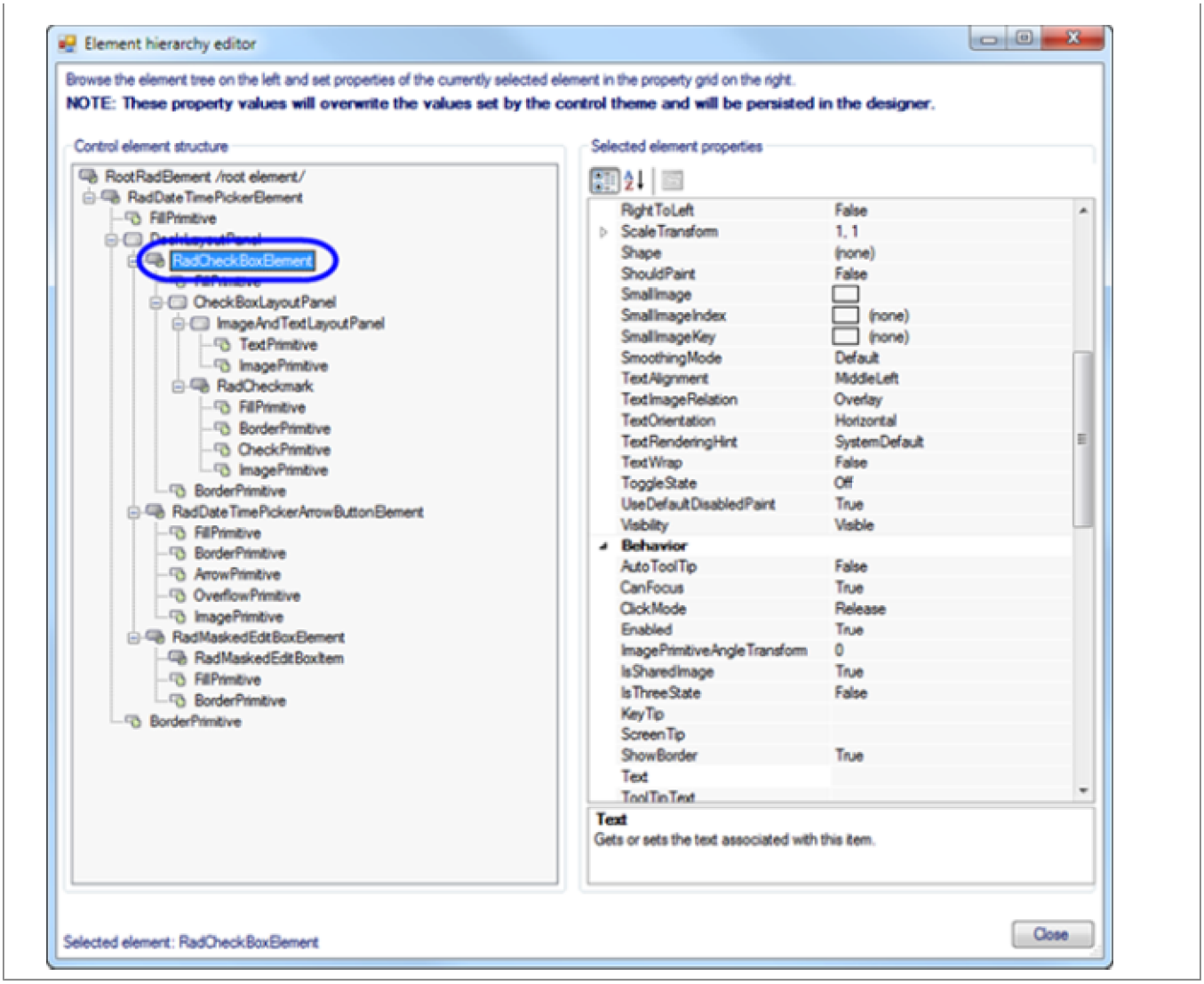
```
Dim checkBox As RadCheckBoxElement = TryCast
(Me.radDateTimePicker1.DateTimePickerElement.Children(1).Children(0), RadCheckBoxElement)
checkBox.ToggleStateChanged += New StateChangedEventHandler(checkBox_ToggleStateChanged)
'...
Sub checkBox_ToggleStateChanged(sender As Object, args As StateChangedEventArgs)
 RadMessageBox.Show(args.ToggleState.ToString())
End Sub
```

[C#] Handling the ToggleStateChanged Event of the Date Time Picker Checkbox

```
RadCheckBoxElement checkBox = radDateTimePicker1.DateTimePickerElement.Children[1].Children
[0] as RadCheckBoxElement;
checkBox.ToggleStateChanged += new StateChangedEventHandler(checkBox_ToggleStateChanged);
//...
void checkBox_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
 RadMessageBox.Show(args.ToggleState.ToString());
}
```

How could you have found the checkbox element on your own? You can find this element using the **Edit UI Elements** link from the Smart Tag. In the screen shot below, trace the hierarchy starting with the RootRadElement...

1. radDateTimePicker1.DateTimePickerElement
2. radDateTimePicker1.DateTimePickerElement.Children[1] (gets us to DockLayoutPanel)
3. radDateTimePicker1.DateTimePickerElement.Children[1].Children[0] (under DockLayoutPanel, the first child is RadCheckBoxElement).



You can also enable the **ShowUpDown** property to make the date picker more like a RadMaskedEditBox in DateTime mode. When ShowUpDown is true, the calendar does not appear, but you can page through the selected part of the date time string. For instance, if you have the AM/PM designator selected, the arrow will toggle between AM and PM.

You can find the complete source for this project at:

\Editors\<VB|CS>\EditorsOverview

Date Time Picker Walk Through

This walk through uses the "GettingStarted" project as a basis and sets the date time picker initial value to be the next business day.

You can find the complete source for this project at:

\Editors\<VB|CS>\DateTimePickerWalkthrough

1. Start with the "GettingStarted" project or a copy.
2. Add a class Utils.cs to the project.

   We will be adding various static helper methods to this class in later sections of this chapter.
3. Add the static method below to the Utils.cs class

   This method determines if a DateTime is a weekend day.

   [VB] Defining the IsWeekend() Method

```
Public Shared Function IsWeekend([date] As DateTime) As Boolean
 Return ([date].DayOfWeek = DayOfWeek.Saturday) OrElse ([date].DayOfWeek = DayOfWeek.Sun
End Function
```

   [C#] Defining the IsWeekend() Method

```
public static bool IsWeekend(DateTime date)
{
 return (date.DayOfWeek == DayOfWeek.Saturday) || (date.DayOfWeek == DayOfWeek.Sunday);
}
```

4. Add the static method below to the Utils.cs class.

   This method returns a business day based on a start date and number of days to add. This should always return a weekday date.

   [VB] Defining the AddBusinessDays Method

```
Public Shared Function AddBusinessDays(startDate As DateTime, daysToAdd As Integer) As
DateTime
 Dim result As DateTime = startDate
 Dim i As Integer = 0
 Dim temp As Double = 0
 ' cycle until the requested number of days to add is
 ' reached. Pass over any weekend days.
 While i < daysToAdd
  System.Math.Max(System.Threading.Interlocked.Increment(temp),temp - 1)
  result = startDate.AddDays(temp)
  If IsWeekend(result) Then
   Continue While
  End If
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
 Return result
End Function
```

   [C#] Defining the AddBusinessDays Method

```
public static DateTime AddBusinessDays(DateTime startDate, int daysToAdd)
{
 DateTime result = startDate;
 int i = 0;
 double temp = 0;
 // cycle until the requested number of days to add is
 // reached. Pass over any weekend days.
 while (i < daysToAdd)
 {
   temp++;
```

```
    result = startDate.AddDays(temp);
    if (IsWeekend(result))
      continue;
    i++;
  }
  return result;
}
```

5. Using the Utils AddBusinessDays() method, assign the date time picker Value property. Pass the DateTime "Today" and the current value of the RadSpinEditor (this should be "1").

> AddBusinessDays() should skip weekend days. If the current date at runtime is Friday, AddBusinessDays () should return the DateTime for the following Monday.

[VB] Assigning the Date Time Picker Value

```
Public Sub New()
 InitializeComponent()
 ' setup initial delivery date
 dtDeliver.Value = Utils.AddBusinessDays(DateTime.Today, DirectCast(seDeliver.Value,
Integer))
End Sub
```

[C#] Assigning the Date Time Picker Value

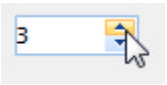
```
public RadForm1()
{
 InitializeComponent();
 // setup initial delivery date
 dtDeliver.Value = Utils.AddBusinessDays(DateTime.Today, (int)seDeliver.Value);
}
```

6. Press **Ctl-F5** to run the application and observe that the day in the date time picker is the next business day after "today".

> After we talk about the RadSpinEditor, we will hook this logic up to respond when the spin editor changes value.

Spin Editor

RadSpinEditor is a themeable alternative to the standard Windows Numeric Up Down control. It allows users to edit a number in a textbox by using up and down buttons.



To use the RadSpinEditor set or retrieve the **Value** property. Use the **Minimum** and **Maximum** properties to limit the upper and lower bounds that a value may be set to. The other unique properties of RadSpinEditor are...

- **InterceptArrowKeys**: This property when true (the default) allows the user to press the UP ARROW and DOWN ARROW keys to select values. The focus must be on the control for this functionality to work.
- **ReadOnly**: This property when true prevents direct text entry and allows the text values to be changed through the up and down arrows only. By default this property is turned off.
- **RightToLeft**: If this property is set to Yes the arrow buttons are displayed on the left, text on the right.
- **Increment**: The amount incremented or decremented when the user clicks the up or down arrow buttons. By default this value is "1".
- **ThousandsSeparator**: This property when true displays a thousands separator. By default this property is

set to false. The ThousandsSeparator display respects current culture settings.

- **DecimalPlaces**: Gets or sets the number of decimal places to display in the RadSpinEdit.

Events

The **ValueChanging** event fires before the value has changed and allows you to prevent a given value from being entered. The event passes a ValueChangingEventArgs parameter that includes the **OldValue**, **NewValue** and **Cancel** properties. Set Cancel true to prevent the change in NewValue. The example below stops the value from changing if the new value is greater than twice the size of the old value.

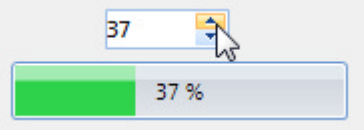
[VB] Handling the ValueChanging Event

```
Private Sub radSpinEditor1_ValueChanging(sender As Object, e As ValueChangingEventArgs)
 e.Cancel = Convert.ToDecimal(e.NewValue) > Convert.ToDecimal(e.OldValue) * 2
End Sub
```

[C#] Handling the ValueChanging Event

```
private void radSpinEditor1_ValueChanging(object sender, ValueChangingEventArgs e)
{
 e.Cancel = Convert.ToDecimal(e.NewValue) > Convert.ToDecimal(e.OldValue) * 2;
}
```

Once the value has changed you can handle the ValueChanged event. In the example below the Value is assigned to a label and to a progress bar.



[VB] Handling the ValueChanged Event

```
Private Sub radSpinEditor1_ValueChanged(sender As Object, e As EventArgs)
 lblSpin.Text = radSpinEditor1.Value.ToString()
 radProgressBar1.Value1 = DirectCast(radSpinEditor1.Value, Integer)
End Sub
```

[C#] Handling the ValueChanged Event

```
private void radSpinEditor1_ValueChanged(object sender, EventArgs e)
Handles radSpinEditor1.ValueChanged
{
 lblSpin.Text = radSpinEditor1.Value.ToString();
 radProgressBar1.Value1 = (int)radSpinEditor1.Value;
}
```



You can find the complete source for this project at:

\Editors\<VB|CS>\EditorsOverview

Spin Editor Walk Through

This walk through will use the "DateTimePickerWalkthrough" project as a basis and set the initial value to be the next business day.



You can find the complete source for this project at:

\Editors\<VB|CS>\DateTimePickerWalkthrough

1. Start with the "DateTimePickerWalkthrough" project or a copy.
2. Create a **ValueChanged** event handler for the RadSpinEditor "seDeliver" and add the following code:

[VB] Handling the ValueChanged Event

```
Private Sub seDeliver_ValueChanged(sender As Object, e As EventArgs)
 dtDeliver.Value = Utils.AddBusinessDays(DateTime.Today, DirectCast(seDeliver.Value,
Integer))
End Sub
```

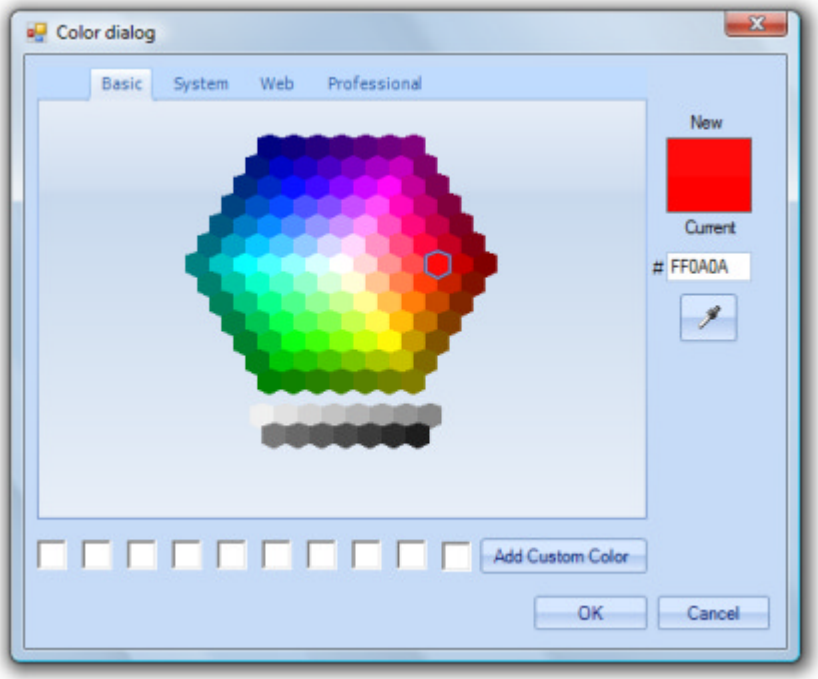
[C#] Handling the ValueChanged Event

```
private void seDeliver_ValueChanged(object sender, EventArgs e)
{
 dtDeliver.Value = Utils.AddBusinessDays(DateTime.Today, (int)seDeliver.Value);
}
```

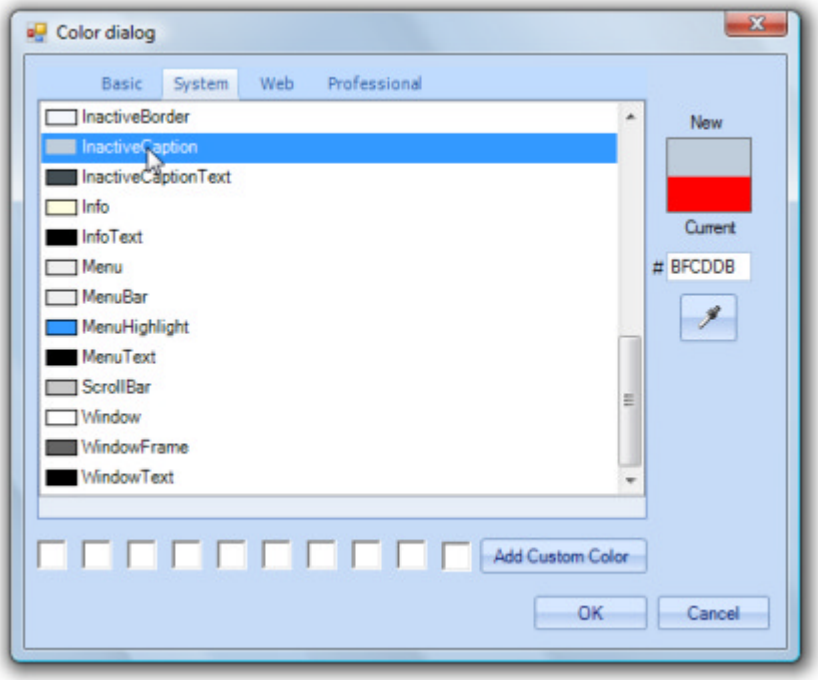
3. Press **Ctl-F5** to run the application. Try using the spin editor arrow buttons to change the date. Notice that the date time picker never displays a weekend date, while if the date is entered directly into the date picker or by using the calendar, any date can be selected.

Color Dialog

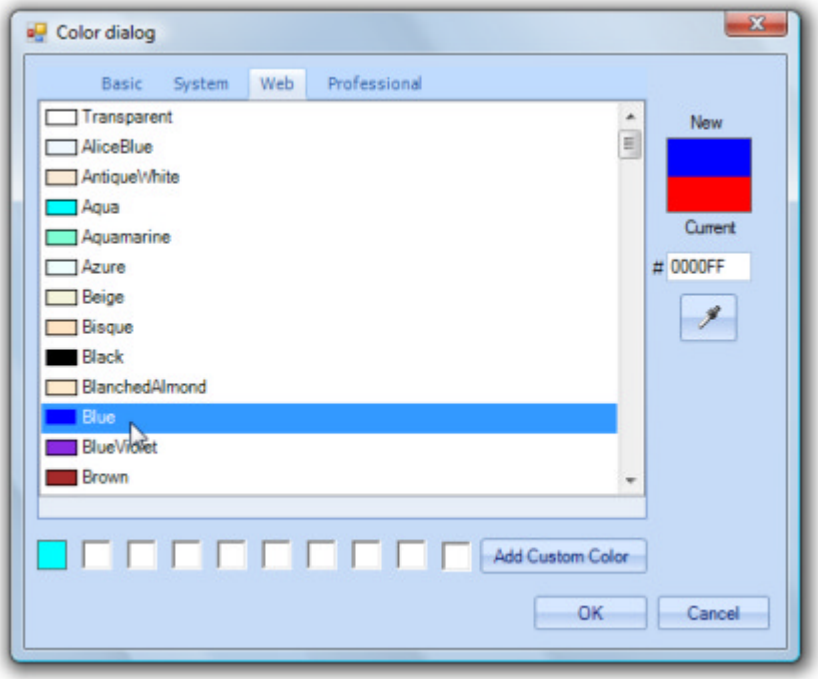
RadColorDialog is a CommonDialog descendant and replaces the standard ColorDialog. The RadColorDialog offers a high degree of configurability compared to its standard counterpart. There are tabs for "Basic", "System", "Web" and "Professional" color selection. See the screenshots below for examples.



Basic lets you choose a predefined color from a honeycomb interface where you click a colored cell to choose the color.

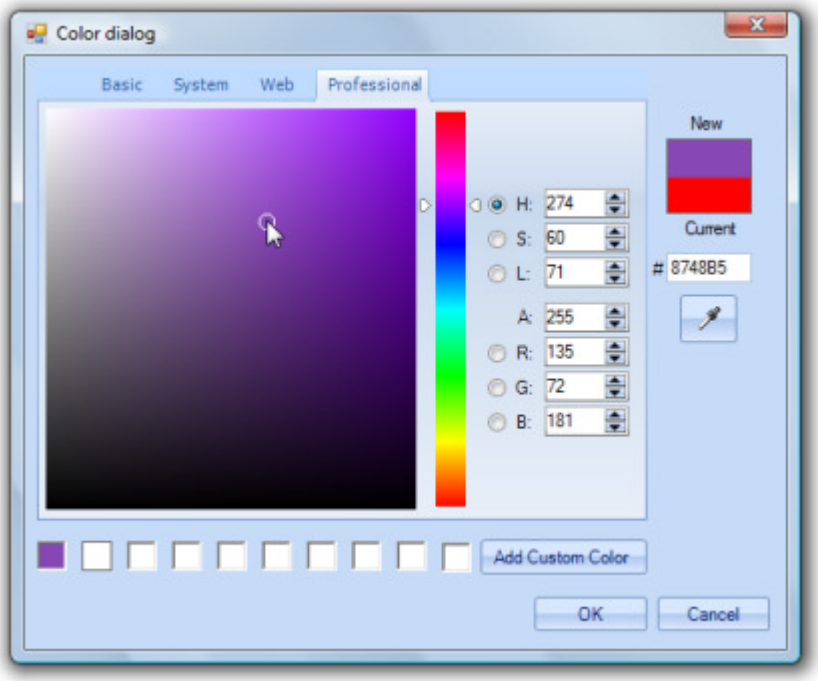


System lets you choose one of the System.Drawing.SystemColors enumeration from a list. Each is a color of a Windows display element (i.e. ButtonShadow, ActiveCaptionText, etc).



Web lets you choose one of the named web colors from a list.

Professional provides the greatest flexibility in selecting or tailoring the



exact color you require. Left click the mouse in the gradient area on the left to select a specific shade of color. Click on the rainbow bar in the middle to select a color.

Using RadColorDialog

To use the RadColorDialog, call its **ShowDialog()** method. If the returned DialogResult value is "OK", retrieve the **SelectedColor** property:

[VB] Using RadColorDialog ShowDialog()

```
Private Sub btnChangeColor_Click(sender As Object, e As EventArgs)
 radColorDialog1.SelectedColor = pnlColor.BackColor
 If radColorDialog1.ShowDialog() = DialogResult.OK Then
  pnlColor.BackColor = radColorDialog1.SelectedColor
 End If
End Sub
```

[C#] Using RadColorDialog ShowDialog()

```
private void btnChangeColor_Click(object sender, EventArgs e)
Handles btnChangeColor.Click
{
 radColorDialog1.SelectedColor = pnlColor.BackColor;
 if (radColorDialog1.ShowDialog() == DialogResult.OK)
 {
   pnlColor.BackColor = radColorDialog1.SelectedColor;
 }
}
```

RadColorDialog also has a **Reset()** method that resets the dialog properties to default values, including replacing the underlaying ColorDialogForm with a new instance.

RadColorDialog has properties for setting and retrieving color and for getting at properties used to configure the dialog:

- **SelectedColor:** A standard Color type.
- **SelectedHslColor:** A HslColor type where HSL stands for Hue, Saturation and Luminence.
- **CustomColors[]**: This is a read-only array of Color types. You can query the contents of the custom colors

displayed along the bottom of the selector.
RadColorDialogForm

- **ColorDialogForm**: A Form descendant that encapsulates dialog properties.

While RadColorDialog only has these few properties, the color dialog is configured through the ColorDialogForm.

RadColorDialogForm

The significant properties unique to RadColorDialogForm are:

- **ShowBasicColors**, **ShowSystemColors**, **ShowWebColors**, **ShowProfessionalColors**: If true (the default) the corresponding Basic, System, Web and Professional tabs are displayed.
- **ActiveMode**: Determines the currently selected tab. Valid enumeration values from Telerik.WinControls.ColorPickerActiveMode are Basic, System, Web, Professional.
- **AllowColorPickFromScreen**: When true (the default) the "eye dropper" tool is displayed.
- **AllowColorSaving**: Toggles the "Add a custom color" button visibility.
- **BackColor**: The background color of the color selector as a whole. You can also set this color to be Transparent to have it blend in with other UI elements.
- **ForeColor**: The color for labels and text entry.
- **SelectedColor**: The color swatch labeled "New".
- **OldColor**: The color swatch labeled "Current".
- **AllowEditHEXValue**: If true (the default) enables hexadecimal entry of color values.
- **ShowHEXColorValue**: If true (the default) displays the hexadecimal entry of color values.
- **ShowCustomColors**: If true displays custom colors boxes along the bottom of the dialog.

To localize or otherwise customize the text in the dialog, use properties **AddNewColorButtonText**, **BasicTabHeading**, **SystemTabHeading**, **WebTabHeading**, **ProfessionalTabHeading**, **SelectedColorLabelHeading** and **OldColorLabelHeading**.



You can find the complete source for this project at:

\Editors\<VB|CS>\EditorsOverview

Color Dialog Walk Through

This walk through will use the "SpinEditorWalkthrough" project as a basis and will set the gift wrap color using the RadColorDialog.



You can find the complete source for this project at:

\Editors\<VB|CS>\ColorDialogWalkthrough

1. Start with the "SpinEditorWalkthrough" project or a copy.
2. Drop a RadColorDialog component on the form.
3. In the constructor for the form's code-behind, add the code below to configure the RadColorDialog.

   This step hides all tabs except Web and System colors and customizes the Web tab heading to be "Named Colors". The dialog automatically displays the Web tab due to the ActiveMode property assignment.

   [VB] Configure the RadColorDialog

```
Dim dialog As RadColorDialogForm = radColorDialog1.ColorDialogForm
dialog.ShowBasicColors = False
dialog.ShowCustomColors = False
dialog.ShowProfessionalColors = False
dialog.ShowHEXColorValue = False
dialog.ShowCustomColors = False
dialog.AllowColorPickFromScreen = False
dialog.WebTabHeading = "Named Colors"
dialog.ActiveMode = ColorPickerActiveMode.Web
```

[C#] Configure the RadColorDialog

```
RadColorDialogForm dialog = (RadColorDialogForm)radColorDialog1.ColorDialogForm;
dialog.ShowBasicColors = false;
dialog.ShowCustomColors = false;
dialog.ShowProfessionalColors = false;
dialog.ShowHEXColorValue = false;
dialog.ShowCustomColors = false;
dialog.AllowColorPickFromScreen = false;
dialog.WebTabHeading = "Named Colors";
dialog.ActiveMode = ColorPickerActiveMode.Web;
```

4. Define a method to return a contrasting color.

   We will use this method to set the text of the "Choose Wrap Color" button so that it will stand out from the background color of the button.

   [VB] Defining the GetContrastingColor Method

```
Public Shared Function GetContrastingColor(color As Color) As Color
 Return Color.FromArgb(color.ToArgb() Xor &Hffffff)
End Function
```

   [C#] Defining the GetContrastingColor Method

```
public static Color GetContrastingColor(Color color)
{
 return Color.FromArgb(color.ToArgb() ^ 0x00ffffff);
}
```

5. In the design view for the form, double-click the "Choose Wrap Color" button to create a Click event handler. Add the code below to display the RadColorDialog, retrieve the color and set the background color for the button.

   [VB] Handling the Click Event

```
Private Sub btnChooseColor_Click(sender As Object, e As EventArgs)
 ' get the fill and text primitives for the button
 Dim fillPrimitive As FillPrimitive = TryCast(btnChooseColor.RootElement.Children
(0).Children(0), FillPrimitive)
 Dim textPrimitive As TextPrimitive = TryCast(btnChooseColor.RootElement.Children
(0).Children(1).Children(1), TextPrimitive)
 ' retrieve the current button color
 radColorDialog1.ColorDialogForm.OldColor = fillPrimitive.BackColor
 radColorDialog1.ColorDialogForm.SelectedColor = fillPrimitive.BackColor
 ' retrieve the user selected color and calculate a contrasting color,
 ' then set the BackColor properties to succesively more transparent
 ' shades of the primary color and set the text to the contrasting color.
 If radColorDialog1.ShowDialog() = DialogResult.OK Then
  Dim color As Color = radColorDialog1.SelectedColor
```

```
  textPrimitive.ForeColor = Utils.GetContrastingColor(color)
  fillPrimitive.BackColor = color
  fillPrimitive.BackColor2 = Color.FromArgb(200, color)
  fillPrimitive.BackColor3 = Color.FromArgb(100, color)
  fillPrimitive.BackColor4 = Color.FromArgb(50, color)
 End If
End Sub
```

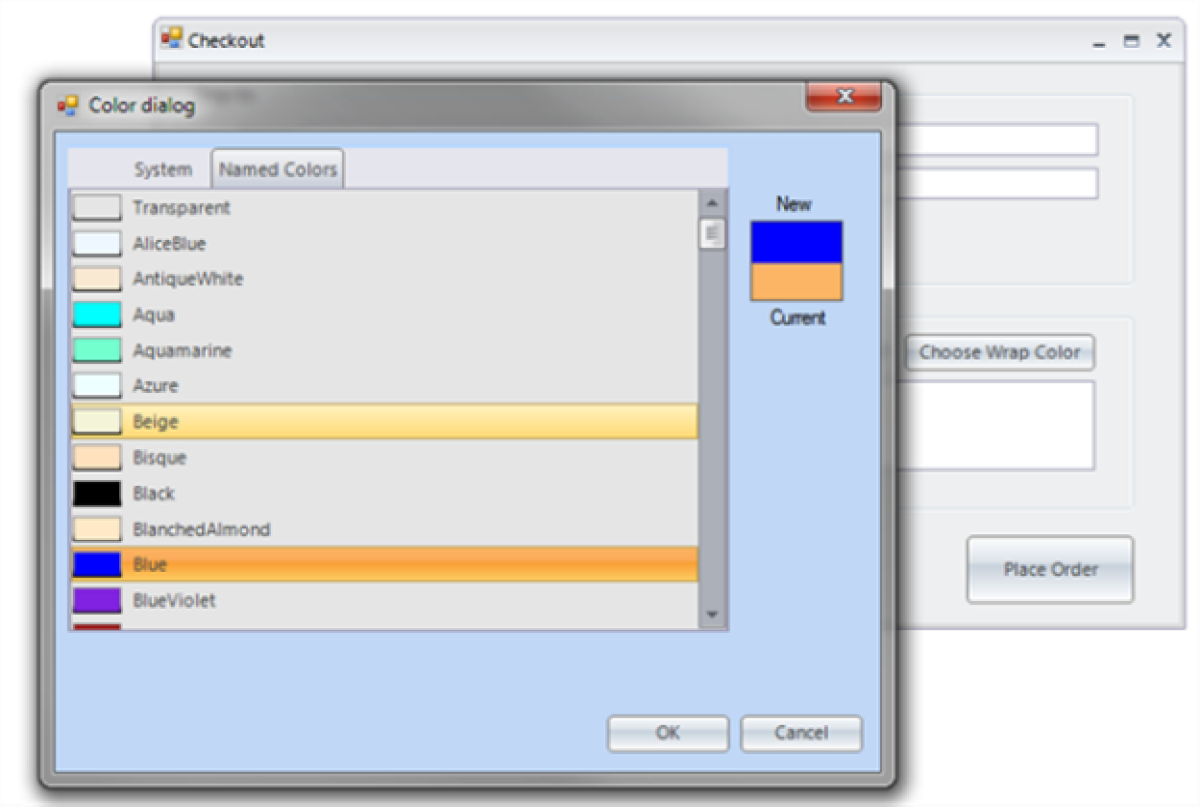
[C#] Handling the Click Event

```
private void btnChooseColor_Click(object sender, EventArgs e)
{
 // get the fill and text primitives for the button
 FillPrimitive fillPrimitive =
   btnChooseColor.RootElement.Children[0].Children[0] as FillPrimitive;
 TextPrimitive textPrimitive =
   btnChooseColor.RootElement.Children[0].Children[1].Children[1] as TextPrimitive;
 // retrieve the current button color
 radColorDialog1.ColorDialogForm.OldColor = fillPrimitive.BackColor;
 radColorDialog1.ColorDialogForm.SelectedColor = fillPrimitive.BackColor;
 // retrieve the user selected color and calculate a contrasting color,
 // then set the BackColor properties to succesively more transparent
 // shades of the primary color and set the text to the contrasting color.
 if (radColorDialog1.ShowDialog() == DialogResult.OK)
 {
   Color color = radColorDialog1.SelectedColor;
   textPrimitive.ForeColor = Utils.GetContrastingColor(color);
   fillPrimitive.BackColor = color;
   fillPrimitive.BackColor2 = Color.FromArgb(200, color);
   fillPrimitive.BackColor3 = Color.FromArgb(100, color);
   fillPrimitive.BackColor4 = Color.FromArgb(50, color);
 }
}
```

Notice the first step is to get the primitives for the button background and button text. We then set the color dialogs OldColor and SelectedColor to the button's current BackColor. If the user chooses a color we assign it back to the button background, and because the background can be made up of four different colors, we use the Color.FromArgb() method and make the color progressively transparent accross the four colors. We also use the Utils.GetContrastingColor() method to get an appropriate color for the Text, i.e. ForeColor.

6. Add a "Telerik.WinControls.Primitives" namespace reference to your "Imports" (VB) or "uses" (C#) section of code. This will support the FillPrimitive and TextPrimitive objects.
7. Press **Ctl-F5** to run the application. Check the "Gift Wrap" check box. Click the "Choose Wrap Color" button. Select a color from the RadColorDialog and click OK.

   Notice that the color is assigned to the button background and a contrasting color is assigned to the button text. Also, the color dialog only displays the System and Named Colors tabs. The Custom colors and Hex value entries are hidden.



Editors Input Walkthrough

This walk through will use the "ColorDialogWalkthrough" project as a basis and will gather input data from the form and display it in a message box.



You can find the complete source for this project at:

\Editors\<VB|CS>\EditorsInputWalkthrough

1. Start with the "ColorDialogWalkthrough" project or a copy.
2. Add a static helper function GetKnownColorName() to retrieve a readable name for a given color. This function should be added to the Utils.cs file.

   Remember that we only allowed system and web colors to be selected from the color dialog, so all the colors will have proper names, not just hex values strings. The method iterates the KnownColor enumeration and tries to match the rgb value against the color passed in.

   [VB] Define the GetKnownColorName() Method

```
Public Shared Function GetKnownColorName(color As Color) As String
 For Each knownColor As KnownColor In [Enum].GetValues(GetType(KnownColor))
  If color.ToArgb() = Color.FromKnownColor(knownColor).ToArgb() Then
   Return Color.FromKnownColor(knownColor).Name
  End If
 Next
 Return [String].Empty
End Function
```

   [C#] Define the GetKnownColorName() Method

```
public static string GetKnownColorName(Color color)
{
 foreach (KnownColor knownColor in Enum.GetValues(typeof(KnownColor)))
 {
   if (color.ToArgb() == Color.FromKnownColor(knownColor).ToArgb())
   {
     return Color.FromKnownColor(knownColor).Name;
   }
 }
 return String.Empty;
}
```

3. Add the following IsEditorControl() helper method to the form's class. This method returns true if a given control is:
   - Enabled
   - On a group box
   - A textbox base, date time picker or RadButton.

   [VB] Defining the IsEditorControl Method

```
Private Function IsEditorControl(control As Control) As Boolean
 If Not control.Enabled Then
  Return False
 End If
 ' only get input from controls within a groupbox
 If Not (TypeOf control.Parent Is GroupBox) Then
  Return False
 End If
 Return TypeOf control Is RadTextBoxBase OrElse TypeOf control Is RadDateTimePicker O
TypeOf control Is RadButton
End Function
```

   [C#] Defining the IsEditorControl Method

```
private bool IsEditorControl(Control control)
{
 if (!control.Enabled)
   return false;
 // only get input from controls within a groupbox
 if (!(control.Parent is GroupBox))
   return false;
 return
   control is RadTextBoxBase ||
   control is RadDateTimePicker ||
   control is RadButton;
}
```

4. Add another helper method GetControlValue() that takes a RadControl and returns the appropriate value object for that control.

   Notice that when we get to the "Choose Wrap Color" button we drill down to it's fill primitive and get the background color. The value in this case is the known color name of the button background. Also notice that there are only two RadButtons on the form, but the "Place Order" button is not parented by a GroupBox and so is ignored by IsEditorControl() method.

   [VB] Defining the GetControlValue Method

```
Private Function GetControlValue(control As RadControl) As Object
 Dim value As Object = Nothing
 If TypeOf control Is RadTextBox Then
  value = (TryCast(control, RadTextBox)).Text
 ElseIf TypeOf control Is RadMaskedEditBox Then
  value = (TryCast(control, RadMaskedEditBox)).Value.ToString()
 ElseIf TypeOf control Is RadDateTimePicker Then
  value = (TryCast(control, RadDateTimePicker)).Value
 ElseIf TypeOf control Is RadCheckBox Then
  value = (TryCast(control, RadCheckBox)).Checked.ToString()
 ElseIf TypeOf control Is RadButton Then
  Dim buttonPrimitive As FillPrimitive = TryCast((TryCast(control,
RadButton)).RootElement.Children(0).Children(0), FillPrimitive)
  value = Utils.GetKnownColorName(buttonPrimitive.BackColor)
 End If
 Return value
End Function
```

[C#] Defining the GetControlValue Method

```
private object GetControlValue(RadControl control)
{
 object value = null;
 if (control is RadTextBox)
   value = (control as RadTextBox).Text;
 else if (control is RadMaskedEditBox)
   value = (control as RadMaskedEditBox).Value.ToString();
 else if (control is RadDateTimePicker)
   value = (control as RadDateTimePicker).Value;
 else if (control is RadCheckBox)
   value = (control as RadCheckBox).Checked.ToString();
 else if (control is RadButton)
 {
   FillPrimitive buttonPrimitive =
     (control as RadButton).RootElement.Children[0].Children[0] as FillPrimitive;
   value = Utils.GetKnownColorName(buttonPrimitive.BackColor);
 }
 return value;
}
```

5. Replace the GetFormData() stub method implementation with the code below.

This function recurses a controls collection looking for objects we've defined as edit controls. When found, the control's Tag and value are appended to a string.

[VB] Replacing the GetFormData() Method

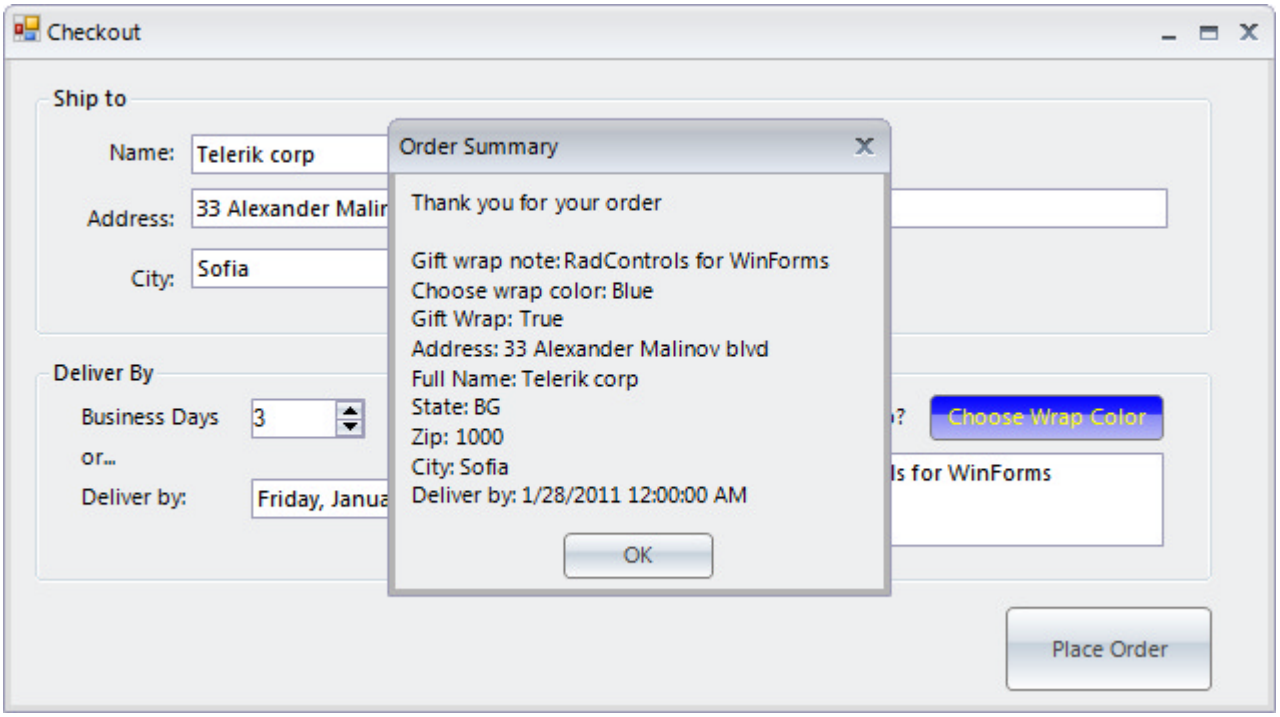
```
Private Function GetFormData(controls As Control.ControlCollection) As String
 Dim builder As New StringBuilder()
 For Each control As Control In controls
  If IsEditorControl(control) Then
   Dim value As Object = GetControlValue(TryCast(control, RadControl))
   If (value <> Nothing) AndAlso (Not value.ToString().Equals([String].Empty)) Then
    builder.Append(Environment.NewLine)
    builder.Append(control.Tag.ToString())
    builder.Append(": ")
    builder.Append(value)
```

```
   End If
  End If
  If control.Controls.Count > 0 Then
   builder.Append(GetFormData(control.Controls))
  End If
 Next
 Return builder.ToString()
End Function
```

[C#] Replacing the GetFormData() Method

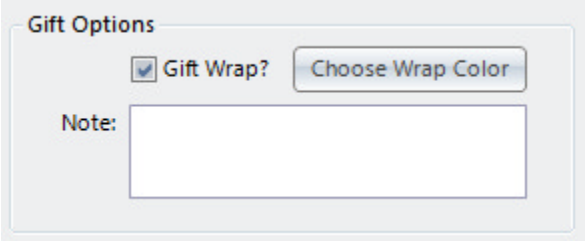
```
private string GetFormData(Control.ControlCollection controls)
{
 StringBuilder builder = new StringBuilder();
 foreach (Control control in controls)
 {
   if (IsEditorControl(control))
   {
     object value = GetControlValue(control as RadControl);
     if ((value != null) && (!value.ToString().Equals(String.Empty)))
     {
       builder.Append(Environment.NewLine);
       builder.Append(control.Tag.ToString());
       builder.Append(": ");
       builder.Append(value);
     }
   }
   if (control.Controls.Count > 0)
     builder.Append(GetFormData(control.Controls));
 }
 return builder.ToString();
}
```

6. Press **Ctl-F5** to run the application. Enter information to the editors and click the "Place Order" button. The information is collected from the form and displayed in the RadMessageBox.



Tool Tips

The most flexible way to display tool tips in your edit controls is to handle the ToolTipTextNeeded event. The ToolTipTextNeedeEventArgs have a single property ToolTipText that can be assigned to.





You can find the complete source for this project at:

\Editors\<VB|CS>\ToolTipsWalkthrough

1. Start with the "EditorsInputWalkthrough" project or a copy.
2. Select the "Name" RadTextBox. In the Properties window Event tab, locate the **ToolTipTextNeeded** event. Type in "HandleToolTipTextNeeded" and click Enter to create an event handler.
3. Add the code below to the event handler.

[VB] Handling the ToolTipTextNeeded Event

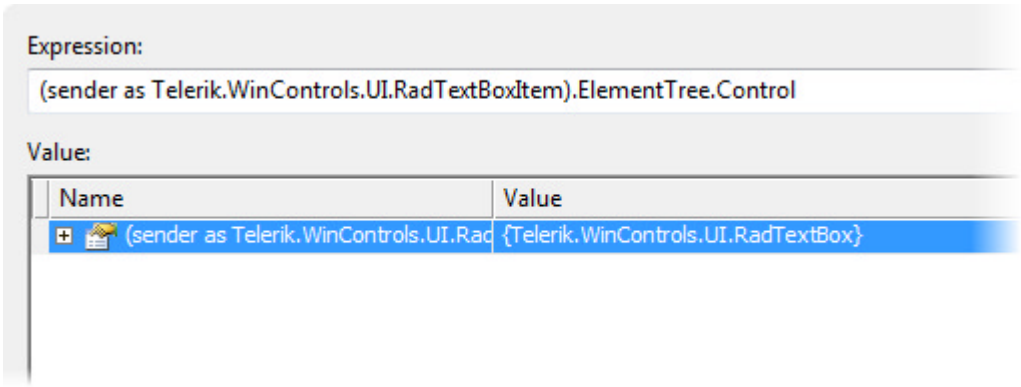
```
Private Sub HandleToolTipTextNeeded(sender As Object, e As ToolTipTextNeededEventArgs)
 e.ToolTipText = "Please enter " + (TryCast(sender,
RadItem)).ElementTree.Control.Tag.ToString()
End Sub
```

[C#] Handling the ToolTipTextNeeded Event

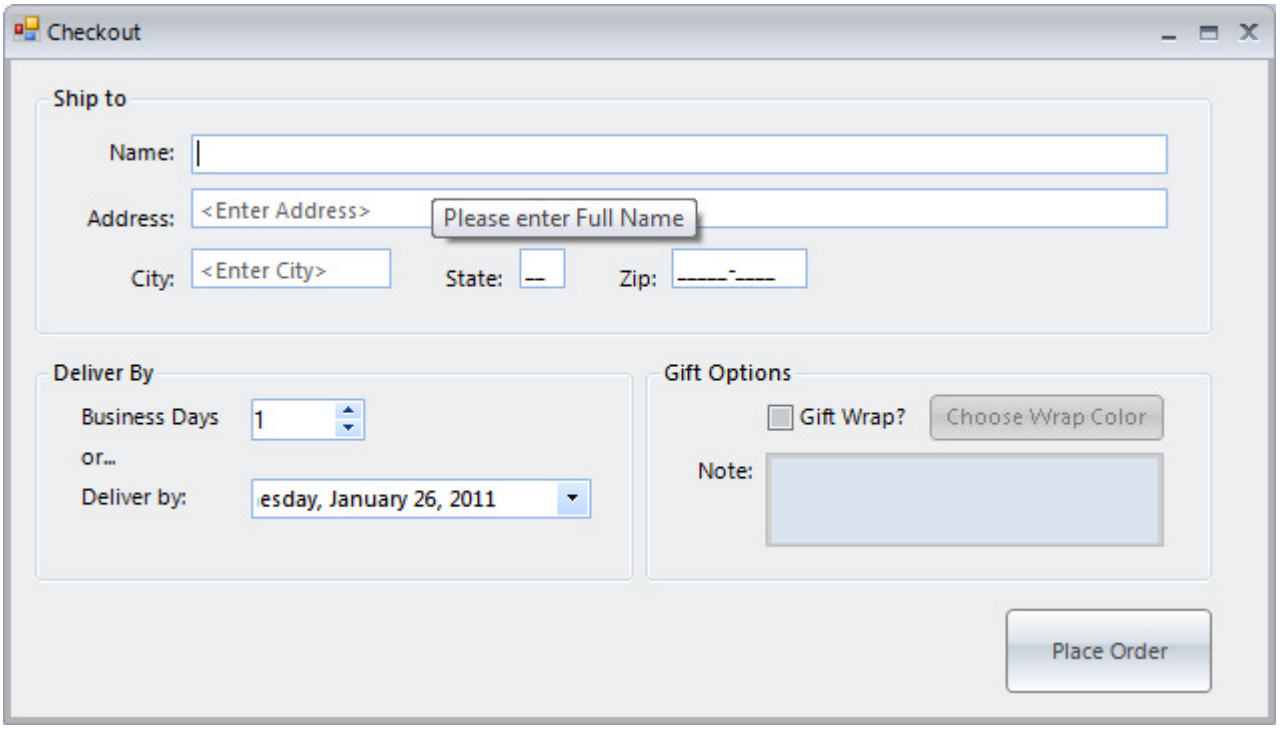
```
private void HandleToolTipTextNeeded(object sender, ToolTipTextNeededEventArgs e)
```

```
{
 e.ToolTipText = "Please enter " + (sender as RadItem).ElementTree.Control.Tag.ToString()
}
```

The code will re-use the Tag property text you entered in the "Getting Started" project. "sender" in this context will be a RadItem descendant, i.e. RadTextBoxItem. To get at the Tag property you need to use the RadItem descendent's ElementTree property and get the Control property. The control will be a RadControl descendant representing the actual control on the form. You can now access the Tag property of that control. The screenshot below shows what "sender" looks like when you debug into the ToolTipTextNeeded event handler.



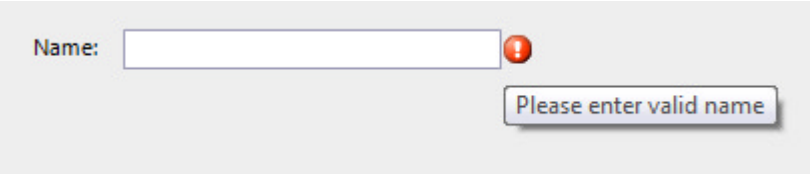
4. Assign this same event handler to all the editors on the form.
5. Click **Ctl-F5** to run the application. Move the mouse over each of the editors to see the tool tip.



Validation

All of the RadControls based editors work nicely with the built-in Microsoft validation mechanisms. One way we can handle validation is to add a standard Microsoft ErrorProvider component and subscribe to the edit control's Validating event. This fires when the user tabs off the control. If the input is invalid we can use the

ErrorProvider SetError() method that automatically displays the error message next to the control being validated. We can also set the Validating event Cancel argument property if we want to prevent any other input on the form.



[VB] Handling the Validating Event with Cancel

```
Private Sub tbName_Validating(sender As Object, e As CancelEventArgs)
 If (TryCast(sender, RadTextBox)).Text.Equals([String].Empty) Then
  errorProvider1.SetError(TryCast(sender, Control), "Please enter a valid name")
  e.Cancel = True
 End If
End Sub
```

[C#] Handling the Validating Event with Cancel

```
private void tbName_Validating(object sender, CancelEventArgs e)
{
 if ((sender as RadTextBox).Text.Equals(String.Empty))
 {
   errorProvider1.SetError(sender as Control, "Please enter a valid name");
   e.Cancel = true;
 }
}
```

This route is workable but a bit restrictive of the users ability to move around the form at will and enter data in any order. We can loosen this restriction a bit by not setting Cancel. This next example validates the delivery date time picker to verify the selected date is a weekday but does not set Cancel true. The error icon is still shown, but the user is not prevented from performing other actions on the form.

[VB] Handling the Validating Event handler

```
' validate the date time picker to allow only weekday dates
Private Sub dtDeliver_Validating(sender As Object, e As CancelEventArgs)
 Dim [error] As String = Nothing
 If Utils.IsWeekend((TryCast(sender, RadDateTimePicker)).Value) Then
  [error] = "Please enter a weekday date for " + (TryCast(sender, Control)).Tag.ToString()
 End If
 errorProvider1.SetError(DirectCast(sender, Control), [error])
End Sub
```

[C#] Handling the Validating Event handler

```
// validate the date time picker to allow only weekday dates
private void dtDeliver_Validating(object sender, CancelEventArgs e)
{
 string error = null;
 if (Utils.IsWeekend((sender as RadDateTimePicker).Value))
 {
   error = "Please enter a weekday date for " + (sender as Control).Tag.ToString();
 }
 errorProvider1.SetError((Control)sender, error);
}
```

When some completion action is about to take place (e.g. saving data), the data should be valid at that time. Extend the "EditorsInputWalkthrough" project to include a validity check when the "Place Order" button is

pressed.



You can find the complete source for this project at:

\Editors\<VB|CS>\ValidationWalkthrough

1. Start with the "ToolTipsWalkthrough" project or a copy.
2. Drop a standard **ErrorProvider** component on the form.
3. Replace the ValidateControls() stub method with the code below.

   This method traverses the form control's collection and for each control:

   - *First clears any existing errors using the ErrorProvider*
   - *Checks if a control is one we've defined as an "Editor control".*
   - *If it's an editor control then we retrieve the control value.*
   - *If the value is a DateTime type, validity checking consists of making sure the value is a weekend. If the value is a string, then the string must not be empty. If the value is not valid, the ErrorProvider SetError() method is called.*
   - *Even if the control is not an editor, the method recurses if there are child controls.*

   [VB] Defining the ValidateControls() Method

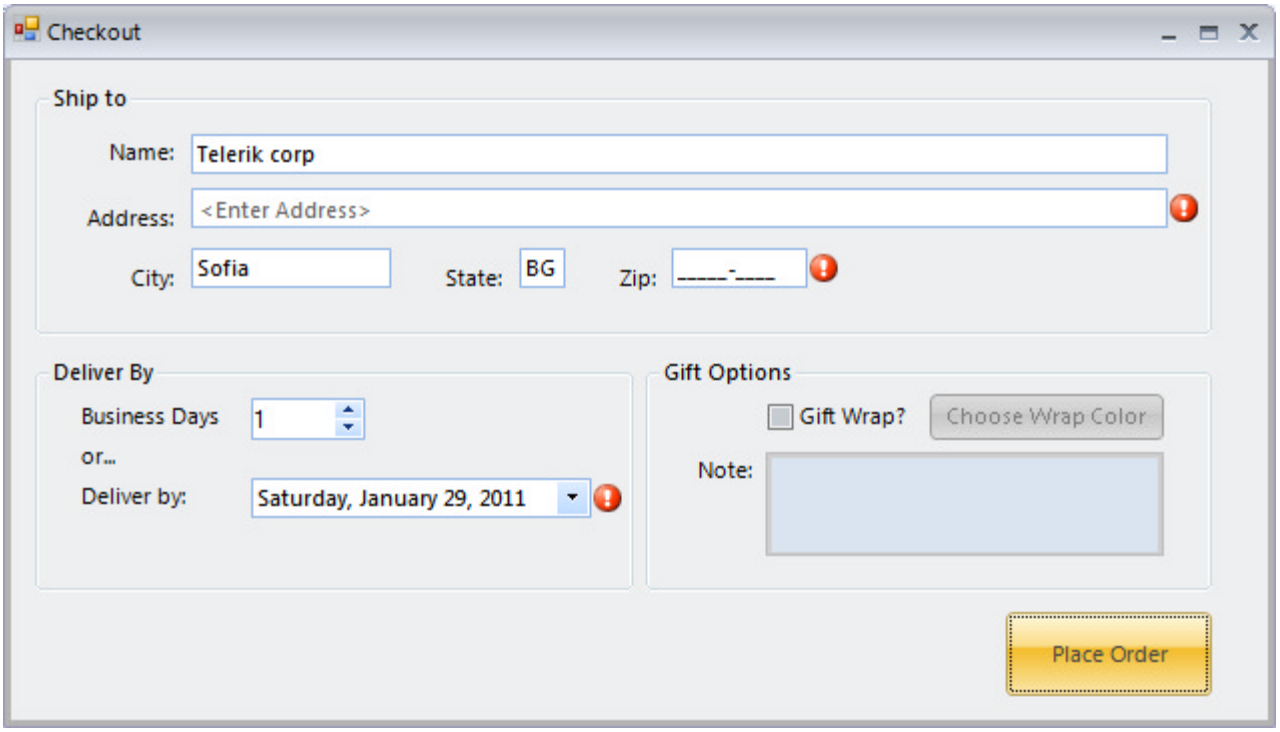
```
Private Function ValidateControls(controls As Control.ControlCollection) As Boolean
 Dim isValid As Boolean = True
 For Each control As Control In controls
  ' clear old errors
  errorProvider1.SetError(control, "")
  If IsEditorControl(control) Then
   Dim value As Object = GetControlValue(TryCast(control, RadControl))
   If (TypeOf value Is DateTime) AndAlso (Utils.IsWeekend(DirectCast(value, DateTime)
    errorProvider1.SetError(control, "Please enter a weekday date to " +
control.Tag.ToString())
    isValid = False
   ElseIf value.ToString().Equals([String].Empty) Then
    errorProvider1.SetError(control, "Please enter " + control.Tag.ToString())
    isValid = False
   End If
  End If
  ' recurse into child controls of this control
  ' so that we get items on the group panels.
  If control.Controls.Count > 0 Then
   If Not ValidateControls(control.Controls) Then
    isValid = False
   End If
  End If
 Next
 Return isValid
End Function
```

   [C#] Defining the ValidateControls() Method

```
private bool ValidateControls(Control.ControlCollection controls)
{
 bool isValid = true;
 foreach (Control control in controls)
```

```
{
  // clear old errors
  errorProvider1.SetError(control, "");
  if (IsEditorControl(control))
  {
    object value = GetControlValue(control as RadControl);
    if ((value is DateTime) && (Utils.IsWeekend((DateTime)value)))
    {
      errorProvider1.SetError(control, "Please enter a weekday date to " +
        control.Tag.ToString());
      isValid = false;
    }
    else if (value.ToString().Equals(String.Empty))
    {
      errorProvider1.SetError(control, "Please enter " + control.Tag.ToString());
      isValid = false;
    }
  }
  // recurse into child controls of this control
  // so that we get items on the group panels.
  if (control.Controls.Count > 0)
    if (!ValidateControls(control.Controls))
      isValid = false;
 }
 return isValid;
}
```

4. Press **Ctl-F5** to run the application. Try leaving out some of the data or select a date time picker weekend date.





If you want to prevent the user from closing the window, handle the FormClosing event. The FormClosingEventArgs include CloseReason to tell you why the window is being closed (e.g. system shut down, user is trying to close the window, etc) and Cancel. Setting Cancel to True prevents the form from closing.

[VB] Handling the FormClosing Event

```
Private Sub RadForm1_FormClosing(sender As Object, e As FormClosingEventArgs)
 If Not ValidateControls(Me.Controls) Then
  Dim result As DialogResult = RadMessageBox.Show("Please correct errors or press
Cancel to close the window.", "Input errors", MessageBoxButtons.OKCancel,
RadMessageIcon.[Error])
  ' Cancel == true prevents the form from closing
  e.Cancel = result = DialogResult.OK
 End If
End Sub
```

[C#] Handling the FormClosing Event

```
private void RadForm1_FormClosing(object sender, FormClosingEventArgs e)
{
 if (!ValidateControls(this.Controls))
 {
   DialogResult result = RadMessageBox.Show(
     "Please correct errors or press Cancel to close the window.", "Input errors",
     MessageBoxButtons.OKCancel, RadMessageIcon.Error);
   // Cancel == true prevents the form from closing
   e.Cancel = result == DialogResult.OK;
 }
}
```



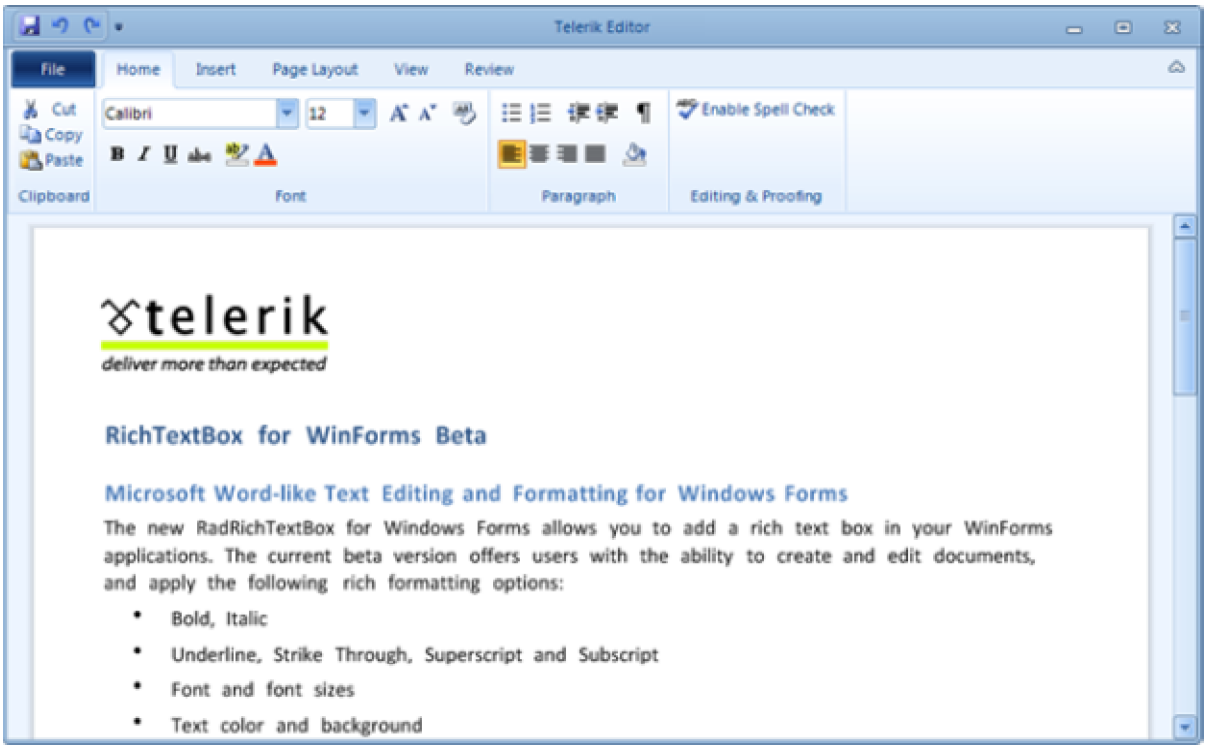
4.6 Summary

This chapter explained how to retrieve information from the user with the RadSpinEditor, RadDateTimePicker, RadMaskedEditBox, RadTextBox and RadColorDialog controls. The chapter began with a brief exploration of basic usability for each control and highlights special features. The chapter included explanation how to implement tool tips, validate user entry and control specific information on edit masks, date formats and internationalization support.

5 RichTextBox Beta

5.1 Overview

RadRichTextBox is a control that is able to display and edit rich-text content including formatted text arranged in pages, paragraphs, spans (runs), etc.



- **Rich Text Formatting** - Telerik RadRichTextBox control allows you to edit text and apply rich formatting options, like:
  - Bold, Italic
  - Underline, Strike Through
  - Text color and background
  - Bullet and numbered lists
  - Paragraph alignment and indentation
  - Show/Hide formatting symbols
  - Clear Formatting
- **Inserting pictures/symbols**
- **SpellChecker** - extensible spell checking for different languages, using built-in or custom dictionaries.
- **Multi-level Undo/Redo Support** - Telerik RadRichTextBox provides desktop-like usability and control thanks to its multilevel Undo/Redo feature. It allows past actions to be reviewed and reversed, just like in Microsoft Word.
- **Two types of layout** – Paged and Flow (as text in an HTML page viewed in a browser).
- **Multi-region selection** - the built-in multi-region support enables you to perform various operations for more than one selection simultaneously.
- **Import/export** - with Telerik RadRichTextBox control you can load XAML, HTML, RTF, DocX (rich-text) or

TXT (plain text) into the control, you can format and edit it and then export it back to any of the above formats or PDF.

5.2 F.A.Q

How to Get and Set the Text of RadRichTextBox

RadRichTextBox does not have a Text property because different formats for import and export of documents are supported – RTF, HTML, XAML, docX, plain text and PDF (export only). In order to set the contents of the document, it should be clear what format the data is in. For easier extensibility and separation of concerns, format providers that deal with the import and export of the documents are used. Here is a list of the currently available format providers and the assemblies they are included in:

- TxtFormatProvider (plain text) – Telerik.WinControls.RichTextBox.FormatProviders.Txt;
- DocxFormatProvider - Telerik.WinControls.RichTextBoc.FileFormats.OpenXml.Docx;
- HtmlFormatProvider - Telerik.WinControls.RichTextBox.FileFormats.Html;
- XamlFormatProvider - Telerik.WinControls.RichTextBox.FileFormats.Xaml;
- RtfFormatProvider – Telerik.WinControls.RichTextBox.FileFormats.Rtf;
- PdfFormatProvider - Telerik.WinControls.RichTextBox.FileFormats.Pdf.

You can read more about the use of format providers here. Overall, what you need to do to get the content of the document in a specific format is to create an instance of the corresponding provider and export the document.

An example is illustrated below:

[C#] Export XAML file

```
public string GetXAML(RadDocument document)
{
   XamlFormatProvider provider = new XamlFormatProvider();
   return provider.Export(document);
}
```

[VB.NET] Export XAML file

```
Public Function GetXAML(ByVal document As RadDocument) As String
    Dim provider As New XamlFormatProvider()
    Return provider.Export(document)
End Function
```

To get the text stripped of all formatting, you can use **TxtFormatProvider**.

Setting the content of RadRichTextBox can be done in the same manner, if you have the content in one of these formats. For example, importing an HTML string in the document of a RadRichTextBox can be done as follows:

[C#] Load HTML file

```
public RadDocument ImportHtml(string content)
{
   HtmlFormatProvider provider = new HtmlFormatProvider();
   return provider.Import(content);
}
```

[VB.NET] Load HTML file

```
Public Function ImportHtml(ByVal content As String) As RadDocument
    Dim provider As New HtmlFormatProvider()
    Return provider.Import(content)
End Function
```

If you wish to preserve the initial content of the document and insert text at different positions in the document, you can use the **Insert** methods of RadDocument and RadRichTextBox. The **Insert** method of RadRichTextBox uses the current span style of the document, i.e. the text is included in the document just as it would have been if you typed it at that position. With the method of RadDocument, you need to explicitly create a **StyleDefinition** and fill it up with the property values of the text that you are inserting – **FontSize**, **FontFamily**, **colors**, etc.

Here are some examples:

> The Insert method will work after the document object model is created. Please note that the DOM will be created only after RadRichTextBox is being measured. Simply said, The insert command will not work in the form constructor.

1. Inserting text in RadRichTextBox at caret position:

   [C#] Insert text in RadRichTextBox

```
this.radRichTextBox1.Insert(textToInsert);
```

   [VB.NET] Insert text in RadRichTextBox

```
Me.RadRichTextBox1.Insert(textToInsert)
```

   You can manipulate the caret position before invoking the insert method in order to change the position where you wish the text to appear. You can find more information on document positions and their usage here.

2. Inserting text in a custom style using RadDocument‘s method:

   [C#] Insert text in RadDocument

```
private void InsertTextInDocument(string textToInsert)
{
   StyleDefinition style = new StyleDefinition();
   style.SetPropertyValue(Span.FontFamilyProperty, "Comic Sans MS");
   style.SetPropertyValue(Span.FontSizeProperty, Unit.PointToDip(16));
   style.SetPropertyValue(Span.ForeColorProperty, Color.Blue);
   this.radRichTextBox1.Document.Insert(textToInsert, style);
}
```

   [VB.NET] Insert text in RadDocument

```
Private Sub InsertTextInDocument(ByVal textToInsert As String)
    Dim style As New StyleDefinition()
    style.SetPropertyValue(Span.FontFamilyProperty, New FontFamily("Comic Sans MS"))
    style.SetPropertyValue(Span.FontSizeProperty, Unit.PointToDip(16))
    style.SetPropertyValue(Span.ForeColorProperty, Color.Blue)
    Me.RadRichTextBox1.Document.Insert(textToInsert, style)
End Sub
```

   In case you wish to insert the text in the current editing style of the rich text box, you can do that in the following manner:

   [C#] Insert text in RadDocument

```
private void InsertTextInDocumentInCurrentSpanStyle(string textToInsert)
{
   StyleDefinition style = new StyleDefinition();
   style.CopyPropertiesFrom(this.radRichTextBox1.CurrentEditingStyle);
   this.radRichTextBox1.Document.Insert(textToInsert, style);
}
```

[VB.NET] Insert text in RadDocument

```
Private Sub InsertTextInDocumentInCurrentSpanStyle(ByVal textToInsert As String)
    Dim style As New StyleDefinition()
    style.CopyPropertiesFrom(Me.radRichTextBox1.CurrentEditingStyle)
    Me.radRichTextBox1.Document.Insert(textToInsert, style)
End Sub
```

Using methods from RadDocument's API to apply changes to the document clears the history stack and the undo function will not be able to restore the document before this change.

Setting the Margins of the Document

With paged layout mode, you can set the margin of the document by using the following code:

[C#] Change page margin

```
radRichTextBox1.Document.LayoutMode = DocumentLayoutMode.Paged;
radRichTextBox1.Document.SectionDefaultPageMargin = new System.Windows.Forms.Padding(100);
```

[VB.NET] Change page margin

```
RadRichTextBox1.Document.LayoutMode = DocumentLayoutMode.Paged
RadRichTextBox1.Document.SectionDefaultPageMargin = New System.Windows.Forms.Padding(100)
```

5.3 Getting Started

RadRichTextBox is a control that allows you to display rich text content including sections, paragraphs, spans, italic text, bold text, in-line images etc. This topic will help you to quickly get started using the control.

Formatting at design time

There are cases when you want to display a non-editable content that has been defined and formatted as far back as design time. In most of the cases the **TextBlock**control is used, but it doesn't provide you with a way to format your text in the same control instance. Although the RadRichTextBox is a rich text input control, it can also be used as a rich text viewer. In order to format the text inside it you can use the following classes:

- Section
- Paragraph
- Span
- InlineImage

As you can see the text is still editable. To make it read only you have to set the **IsReadOnly** property of the RadRichTextBox to *True*.

[C#] Read only mode

```
radRichTextBox1.IsReadOnly = true;
```

[VB.NET] Read only mode

```
RadRichTextBox1.IsReadOnly = True
```

Formatting via a sample UI

If you want to allow the user to edit and format the content of the RadRichTextBox, you have to create a UI

and use the API exposed by the RadRichTextBox. The API exposes methods (like ToggleBold(), ToggleItalic() etc.) that modify the text in the control when called. Here is an example of creating a UI for making the text bold, italic and underlined.

[C#] Bold, Italic, Underline

```
private void boldButton_Click(object sender, EventArgs e)
{
   this.radRichTextBox1.ToggleBold();
}

private void italicButton_Click(object sender, EventArgs e)
{
   this.radRichTextBox1.ToggleItalic();
}

private void underlineButton_Click(object sender, EventArgs e)
{
   this.radRichTextBox1.ToggleUnderline();
}
```

[VB.NET] Bold, Italic, Underline

```
Private Sub boldButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles boldButton.Click
    Me.RadRichTextBox1.ToggleBold()
End Sub

Private Sub italicButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles italicButton.Click
    Me.RadRichTextBox1.ToggleItalic()
End Sub

Private Sub unserlineButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles underlineButton.Click
    Me.RadRichTextBox1.ToggleUnderline()
End Sub
```

The UI should also response, when the caret is on a position where the text is modified. For example, the **boldButton** should be toggled if the caret is on a bold text. This can be done by handling the **CurrentSpanStyleChanged** event. It is raised every time when the span, over which the caret is, gets changed. In the event handler you can use the **CurrentEditingStyle** property in order to get the desired information for the style of the current Span. This property is of type**StyleDefinition** and by using its **GetPropertyValue()** method and passing the appropriate DependancyProperty, you can get the desired information. This makes the**StylesDefinition** class universal for **Span** and **Paragraph** elements. Here is an example:


The same approach can be used for the **CurrentParagraphStyleChanged** event

[C#] CurrentSpanStyleChanged and CurrentParagraphStyleChanged event handling

```
void radRichTextBox1_CurrentSpanStyleChanged(object sender, EventArgs e)
{
```

```
   StyleDefinition style = this.radRichTextBox1.CurrentEditingStyle;
   TextStyle fontStyle = (TextStyle)style.GetPropertyValue(Span.FontStyleProperty);
   UnderlineType underlineDecoration = (UnderlineType)style.GetPropertyValue
(Span.UnderlineTypeProperty);
   this.boldButton.IsChecked = fontStyle.HasFlag(TextStyle.Bold);
   this.italicButton.IsChecked = fontStyle.HasFlag(TextStyle.Italic);
   this.underlineButton.IsChecked = underlineDecoration != UnderlineType.None;
}
```

[VB.NET] CurrentSpanStyleChanged and CurrentParagraphStyleChanged event handling

```
Private Sub radRichTextBox1_CurrentSpanStyleChanged(ByVal sender As Object, ByVal e As
EventArgs)
    Dim style As StyleDefinition = Me.RadRichTextBox1.CurrentEditingStyle
    Dim fontStyle As TextStyle = DirectCast(style.GetPropertyValue(Span.FontStyleProperty),
TextStyle)
    Dim underlineDecoration As UnderlineType = DirectCast(style.GetPropertyValue
(Span.UnderlineTypeProperty), UnderlineType)
    Me.boldButton.IsChecked = fontStyle.HasFlag(TextStyle.Bold)
    Me.italicButton.IsChecked = fontStyle.HasFlag(TextStyle.Italic)
    Me.underlineButton.IsChecked = underlineDecoration <> UnderlineType.None
End Sub
```

5.4 Working with RichTextBox

The RadRichTextBox is a control that allows you to visualize rich text content and allows the user to format it.

Design-Time Documents

You are able to predefine the content of the RadRichTextBox at design time. This means that the RadRichTextBox can be used to display static rich content. Defining such a content can be done via the following UI Elements:

- Section
- Paragraph
- Span
- InlineImage

RadRichTextBox API

Instead of visualizing a static rich content, you may want to use the RadRichTextBox as an input control. In this case, in order to provide the user with the ability to format the inputted content, you have to provide a UI that communicates with the RadRichTextBox. For that purpose the RadRichTextBox exposes an API, which contains various methods that can apply different formatting to the inputted content. To learn more about the API methods read this topic. To see an example of a RadRichTextBox that allows to apply bold, italic and underline formatting, take a look at this topic.

Scrolling

When the available size for the control becomes less than the size of the content, the RadRichTextBox will automatically display horizontal or vertical scrollbars respectively.

Current Span

The **CurrentEditingStyle** property returns an instance of the **StyleDefinition** class, which allows you to get information abou the current element. You can combine the usage of this property with the usage of

the **CurrentSpanStyleChanged** event in order to update the UI (if any) when the text style changes.

> The same approach can be used for the **CurrentParagraphStyleChanged** event.

For example, you have a button that makes the text bold:

[C#] Bold text

```
private void boldButton_Click(object sender, EventArgs e)
{
   radRichTextBox1.ToggleBold();
}
```

[VB.NET] Bold text

```
Private Sub boldButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles boldButton.Click
    RadRichTextBox1.ToggleBold()
End Sub
```

in the event handler for the **CurrentSpanStyleChanged** event, you can do the following:

[C#] Bold text

```
void radRichTextBox1_CurrentSpanStyleChanged(object sender, EventArgs e)
{
   StyleDefinition style = this.radRichTextBox1.CurrentEditingStyle;
   TextStyle fontWeight = (TextStyle)style.GetPropertyValue(Span.FontStyleProperty);
   this.boldButton.IsChecked = fontWeight == TextStyle.Bold;
}
```

[VB.NET] Bold text

```
Private Sub radRichTextBox1_CurrentSpanStyleChanged(ByVal sender As Object, ByVal e As
EventArgs)
    Dim style As StyleDefinition = Me.RadRichTextBox1.CurrentEditingStyle
    Dim fontWeight As TextStyle = DirectCast(style.GetPropertyValue(Span.FontStyleProperty),
TextStyle)
    Me.boldButton.IsChecked = fontWeight = TextStyle.Bold
End Sub
```

This will keep the button synchronized with the current position of the caret.

5.5 Layout Modes

The RadRichTextBox allows you to choose between several layout modes.

Paged

When using the paged mode, the content of the edited document is divided into pages. The size and layout of each page are defined by the**DefaultPageLayoutSettings** property of the **RadDocument** and more specifically - the **Width** and **Height** properties of the **PageLayoutSettings** object.

Next, the margins of the control in a page are specified by the **PageMargin** property of each **Section**. Since several sections can appear on a single page, the margin properties of the first **Section** that appears on a certain page would be valid. In **Page** mode, resizing a RadRichTextBox will not affect the document layout but scrollbars will appear if the document does not fit in the view.

Flow

In **Flow** layout mode, document content is not divided into pages. Instead there is a single page which displays the whole content. This layout option resembles **MS Word's Web-Layout** mode. Since the document is not constrained in width, changing the control's width will also resize the content of the document and this will affect its height.

FlowNoWrap

The **FlowNoWrap** layout mode is similar to the **Flow** layout mode, but it doesn't allow the text in the separate paragraphs to get wrapped when the free space gets exceeded. Instead a horizontal scroll bar will appear.

5.6 Selection

The RadRichTextBox supports not only selection via the UI, but also programmatic selection. This topic will explain you how to:

UI Selection

The user is able to select the content of the RadRichTextBox in the same way as in MS Word. This is done by clicking on the desired position and dragging to the desired end of the selection. A multiple ranges selection is also allowed. This one is done by holding the **Ctrl** key while selecting the different ranges.

Programmatic Selection

The developer is allowed to work with the selection programmatically. This can be used when having a **Find** functionality in your RadRichtextBox and you want to select the found string. Examples that involve the programmatic selection can be found in the How To section.

The programmatic selection gets implemented via the **DocumentSelection** class. The instance of the class gets associated with the **RadDocument** of the RadRichTextBox and allows you to specify selection starts and ends, selection ranges and other. You can manage the selection by either using the **Selection**property of the **RadDocument** or by creating an instance of the **DocumentSelection** class.

Here is an example of how to select the current word.


To learn more about the **DocumentPosition** read the Positioning topic.

[C#] Select the current word

```
DocumentPosition startPosition = this.radRichTextBox1.Document.CaretPosition; //new
DocumentPosition( this.radRichTextBox.Document );
DocumentPosition endPosition = new DocumentPosition(startPosition);
startPosition.MoveToCurrentWordStart();
endPosition.MoveToCurrentWordEnd();
this.radRichTextBox1.Document.Selection.AddSelectionStart(startPosition);
this.radRichTextBox1.Document.Selection.AddSelectionEnd(endPosition);
```

[VB.NET] Select the current word

```
Dim startPosition As DocumentPosition = Me.RadRichTextBox1.Document.CaretPosition
'new DocumentPosition( this.radRichTextBox.Document );
Dim endPosition As New DocumentPosition(startPosition)
startPosition.MoveToCurrentWordStart()
endPosition.MoveToCurrentWordEnd()
Me.RadRichTextBox1.Document.Selection.AddSelectionStart(startPosition)
Me.RadRichTextBox1.Document.Selection.AddSelectionEnd(endPosition)
```

Multi-Range Selection

You can implement Multi-Range Selection by either calling multiple times the **AddSelectionStart()** and **AddSelectionEnd()** methods or by working with the **Ranges** collection.

Here is an example of selecting each "RadRichTextBox" word in the text. This example uses the first approach.

[C#] Multi selection

```
DocumentPosition position = new DocumentPosition( this.radRichTextBox1.Document );
do
{
  //GetCurrentSpan().Text returns the word at the position
  string word = position.GetCurrentSpanBox().Text;
  if ( word.Contains( "RadRichTextBox" ))
  {
      DocumentPosition wordEndPosition = new DocumentPosition( position );
      wordEndPosition.MoveToCurrentWordEnd();
      this.radRichTextBox1.Document.Selection.AddSelectionStart( position );
      this.radRichTextBox1.Document.Selection.AddSelectionEnd( wordEndPosition );
  }
}
while (position.MoveToNextWordStart());
```

[VB.NET] Multi selection

```
Dim position As New DocumentPosition(Me.RadRichTextBox1.Document)
Do
    'GetCurrentSpan().Text returns the word at the position
    Dim word As String = position.GetCurrentSpanBox().Text
    If word.Contains("RadRichTextBox") Then
        Dim wordEndPosition As New DocumentPosition(position)
        wordEndPosition.MoveToCurrentWordEnd()
        Me.RadRichTextBox1.Document.Selection.AddSelectionStart(position)

        Me.RadRichTextBox1.Document.Selection.AddSelectionEnd(wordEndPosition)
    End If
Loop While position.MoveToNextWordStart()
```

5.7 Search

RadRichTextBox supports searching the contents of the document along with providing some methods for manipulating the selection. Used in combination, they become quite a powerful tool enabling scenarios like highlighting of specific parts of the document or replacing words and collocations.



To learn more about the selection API read the **Selection (Section 5.6)** topic.

The simplest scenario – finding a string in the content of the document can be implemented with the following code:

[C#] Select all matches

```
private void SelectAllMatches(string toSearch)
{
   this.radRichTextBox1.Document.Selection.Clear(); // this clears the selection before
```

```
processing
   DocumentTextSearch search = new DocumentTextSearch(this.radRichTextBox1.Document);
   foreach (var textRange in search.FindAll(toSearch))
   {
       this.radRichTextBox1.Document.Selection.AddSelectionStart(textRange.StartPosition);
       this.radRichTextBox1.Document.Selection.AddSelectionEnd(textRange.EndPosition);
   }
}
```

[VB.NET] Select all matches

```
Private Sub SelectAllMatches(ByVal toSearch As String)
    Me.RadRichTextBox1.Document.Selection.Clear() ' this clears the selection before
processing
    Dim search As New DocumentTextSearch(Me.RadRichTextBox1.Document)
    For Each textRange In search.FindAll(toSearch)
        Me.RadRichTextBox1.Document.Selection.AddSelectionStart(textRange.StartPosition)
        Me.RadRichTextBox1.Document.Selection.AddSelectionEnd(textRange.EndPosition)
    Next textRange
End Sub
```



Note that the **FindAll()** method of the **DocumentTextSearch** class takes a string as an argument, but regards it as a regular expression. For example, searching **"[a-z]*control[a-z]*"**will find all words that contain the word **"control"** as a substring. Thus, if you have the word **"RadControls"** in the document, executing the **SelectAllMatches("control")** will result in only these six letters being selected, as opposed to the whole word **"RadControls"** being selected when**SelectAllMatches("[a-z]*control[a-z]*")**.

You can use all kind of regular expressions, such as **"(asp|silverlight)\s*control(s)?"** (which would match both **"ASP Controls"** and **"SilverlightControl"**), **[a-z]*(.)\1[a-z]*** (which finds all words that have a doubled letter), and basically everything else you can come up with. Make sure to use correct expressions if you want to utilize this functionality, or escape the string in case you want a simple search for a word that can contain special symbols like **"?"**, **"+"**, **"{"** etc. This can be automatically done by invoking the following code before proceeding with the search.

[C#] Escape string

```
toSearch = Regex.Escape(toSearch);
```

[VB.NET] Escape string

```
toSearch = Regex.Escape(toSearch)
```

Selecting the results of the search is particularly useful, as most formatting commands that are executed on RadRichTextBox (or its document) are selection-based, i.e. they operate on the currently selected parts of the document. You can use that fact by invoking one of these commands after having selected the words in the document. Here is an example.

[C#] Customize selection

```
this.radRichTextBox1.ChangeTextHighlightColor(Color.LightGray);  // will highlight all
selected words in LightGray
this.radRichTextBox1.ChangeFontSize(Unit.PointToDip(32));   // will increase the font size
of the words to 30 DIP
this.radRichTextBox1.ChangeFontFamily("Comic Sans MS"); // will change the font family of
the spans, containing these words.
```

[VB.NET] Customize selection

```
Me.RadRichTextBox1.ChangeTextHighlightColor(Color.LightGray) ' will highlight all selected
words in LightGray
Me.RadRichTextBox1.ChangeFontSize(Unit.PointToDip(32)) ' will increase the font size of the
words to 30 DIP
Me.RadRichTextBox1.ChangeFontFamily("Comic Sans MS") ' will change the font family of the
spans, containing these words.
```

Most of the time, you would like to remove the selection at the end of the operations, which is done by calling the **Clear()** method of the **DocumentSelection** object.

[C#] Clear selection

```
this.radRichTextBox1.Document.Selection.Clear();
```

[VB.NET] Clear selection

```
Me.RadRichTextBox1.Document.Selection.Clear()
```

> You should have in mind that performing these actions in **DocumentContentChanged** event handler will not only raise performance issues, but will also produce an infinite loop, as the changes of the formatting are also considered document changes.

On a side note, the find and select functionality can be extended in order to implement a replace of all occurrences matching the searched string with another string. Here is a sample code:

[C#] Replace all matches

```
private void ReplaceAllMatches(string toSearch, string toReplaceWith)
{
   this.radRichTextBox1.Document.Selection.Clear(); // this clears the selection before
processing
   DocumentTextSearch search = new DocumentTextSearch(this.radRichTextBox1.Document);
   List<TextRange> rangesTrackingDocumentChanges = new List<TextRange>();
   foreach (var textRange in search.FindAll(toSearch))
   {
       TextRange newRange = new TextRange(new DocumentPosition(textRange.StartPosition,
true), new DocumentPosition(textRange.EndPosition, true));
       rangesTrackingDocumentChanges.Add(newRange);
   }
   foreach (var textRange in rangesTrackingDocumentChanges)
   {
       this.radRichTextBox1.Document.Selection.AddSelectionStart(textRange.StartPosition);
       this.radRichTextBox1.Document.Selection.AddSelectionEnd(textRange.EndPosition);
       this.radRichTextBox1.Insert(toReplaceWith);
       textRange.StartPosition.Dispose();
       textRange.EndPosition.Dispose();
   }
}
```

[VB.NET] Replace all matches

```
Private Sub ReplaceAllMatches(ByVal toSearch As String, ByVal toReplaceWith As String)
    Me.RadRichTextBox1.Document.Selection.Clear() ' this clears the selection before
processing
    Dim search As New DocumentTextSearch(Me.RadRichTextBox1.Document)
    Dim rangesTrackingDocumentChanges As New List(Of TextRange)()
    For Each textRange In search.FindAll(toSearch)
        Dim newRange As New TextRange(New DocumentPosition(textRange.StartPosition, True),
New DocumentPosition(textRange.EndPosition, True))
```

```
            rangesTrackingDocumentChanges.Add(newRange)
        Next textRange
        For Each textRange In rangesTrackingDocumentChanges
            Me.RadRichTextBox1.Document.Selection.AddSelectionStart(textRange.StartPosition)
            Me.RadRichTextBox1.Document.Selection.AddSelectionEnd(textRange.EndPosition)
            Me.RadRichTextBox1.Insert(toReplaceWith)
            textRange.StartPosition.Dispose()
            textRange.EndPosition.Dispose()
        Next textRange
End Sub
```

There is a need to create new **TextRanges** with document positions that track the changes to the document, because the deletion and insertion affects the document positions. The document positions created using the 2-parameter constructor with the second parameter set to "*true*" keep their relative position as the document changes. They, however, have to be manually disposed of, as they subscribe to events in order to track the changes to the document and must detach their handlers at the end of their use. This is done using the **Dispose ()** method. If you skip that step, there will be memory leaks in your application.

5.8 Positioning

The positioning feature in the RadRichTextBox is used to navigate through document's content and to get information about the document's elements at a specific position. The **RadDocument** uses the positioning to track the movement of the caret and to control the selection.

The positioning is implemented via the **DocumentPosition** class. This class can be used by the developer to programmatically control the positioning or the selection. **DocumentPosition** offers methods, such as **MoveToNextWord()**, **MoveToPreviousWord()**,**MoveToCurrentLineStart/End()** and so on, which will navigate to the given document element. In order to get information about the element at a given position you can use several methods such as**GetCurrentSpanBox()**, **GetCurrentParagraphBox()**, **GetCurrentSectionBox()** and so on.

DocumentPosition also redefines equality and comparison operators for more convenience, when you should find whether the **DocumentPosition** is before or after another position in the natural flow of the document. By default RadRichTextBox moves **Document.CaretPosition** using arrow keys or on mouse click. **DocumentPosition** can also be obtained by giving the coordinates of a point in the document using the method **DocumentPosition.SetPosition**.

You can manage the caret position for a specific **RadDocument** by either accessing its**CaretPosition** property, which is of type **DocumentPosition**, or by creating a new instance of the**DocumentPosition** class and associating it with the desired **RadDocument**.

> When using the **CaretPosition** property you are directly managing the caret position in the**RadDocument**. By using the **DocumentPosition** class you can create instances of several positions inside the document without changing the current caret position.

CaretPosition property

Here is an example of how to use the **CaretPosition** property to get the current word.

[C#]

```
string currentSpanText = this.radRichTextBox1.Document.CaretPosition.GetCurrentSpanBox
().Text;
```

[VB.NET]

```
Dim currentSpanText As String = Me.RadRichTextBox1.Document.CaretPosition.GetCurrentSpanBox
().Text
```

DocumentPosition class

An alternative of using the **CaretPosition** property is to create an instance of the**DocumentPosition** class. Here is the same example from the previous chapter done with an instance of the **DocumentPosition** class.

[C#]

```
DocumentPosition position = new DocumentPosition(this.radRichTextBox1.Document);
string currentSpanText1 = position.GetCurrentSpanBox().Text;
```

[VB.NET]

```
Dim position As DocumentPosition = New DocumentPosition(Me.RadRichTextBox1.Document)
Dim currentSpanText1 As String = position.GetCurrentSpanBox().Text
```

5.9 History

The RadRichTextBox supports not only selection via the UI, but also programmatic selection. This topic will explain you how to:

Enable/Disable History

You can enable or disable the history for the **RadDocument** via the **Enabled** property of the**DocumentHistory**.

[C#] Enable history

```
this.radRichTextBox1.Document.History.Enabled = true;
```

[VB.NET] Enable history

```
Me.RadRichTextBox1.Document.History.Enabled = False
```

Clear History

To clear the history you just have to call the **Clear()** method of the **DocumentHistory** class.

[C#] Clear history

```
this.radRichTextBox1.Document.History.Clear();
```

[VB.NET] Clear history

```
Me.RadRichTextBox1.Document.History.Clear()
```

Undo/Redo

To undo and redo some actions, you can call the **Undo()** and **Redo()** methods of the RadRichTextBox.

[C#] Undo/Redo history

```
this.radRichTextBox1.Undo();
this.radRichTextBox1.Redo();
```

[VB.NET] Undo/Redo history

```
Me.RadRichTextBox1.Undo()
Me.RadRichTextBox1.Redo()
```

Change History Depth

To change the history capacity you have to set the desired value of the **Depth** property of the**DocumentHistory**. The default one is **1000**.

[C#] History depth

```
this.radRichTextBox1.Document.History.Depth = 500;
```

[VB.NET] History depth

```
Me.RadRichTextBox1.Document.History.Depth = 500
```

5.10 ReadOnly

To make the RadRichTextBox read only, you have to set its **IsReadOnly** property to *True*.

[C#] Read only mode

```
radRichTextBox1.IsReadOnly = true;
```

[VB.NET] Read only mode

```
RadRichTextBox1.IsReadOnly = True
```

Except the **ReadOnly** property, there are several other properties that can be used to control the response of the RadRichTextBox towards the user actions against it:

- **Enabled** - setting this property to *False* will disable the entire control. The user won't be able to enter any input in it and to scroll the contents.
- **IsSelectionEnabled** - setting this property to *False* will disable the user to perform any selection inside the RadRichTextBox control.

5.11 Formatting API

If you want to format the content of the RadRichTextBox at run time, you have to use the API exposed by the RadRichTextBox. This is essential, as the main purpose of RadRichTextBox is to allow the users to format their input via UI. The UI should call the respective API methods of the RadRichTextBox.

The RadRichTextBox exposes methods that change the style of the text or the paragraph. When a method is called, the respective style is applied to the selected text. If there is no selection available, the style is applied to the word, in which the caret is located.

Here is an example of a toggle button that upon checking should make the selection or the current word bold. In the handler for the **Click** event of the **RadToggleButton**, the **ToggleBold()** method of the RadRichTextBox gets called.

[C#] Toggle bold

```
private void boldButton_Click(object sender, EventArgs e)
{
   radRichTextBox1.ToggleBold();
}
```

[VB.NET] Toggle bold

```
Private Sub boldButton_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles boldButton.Click
    RadRichTextBox1.ToggleBold()
End Sub
```

5.12 Import/Export

Import/Export

The RadRichTextBox allows you to export and import its content. This is useful in case you want to save the user's input into a data base and then load it from there, or if you want to save/load the content of the RadRichTextBox to/from a file. To import and export you have to use a specific class that implements the **IDocumentFormatProvider**. You can find built-in classes, that implement this interface, for each of the supported formats. Currently the RadRichTextBox can export and import the following formats:

- **XAML** - to import/export **XAML** documents you have to use the **XamlFormatProvider** class.
- **DOCX** - to import/export **DOCX** documents you have to use the **DocxFormatProvider** class.
- **HTML** - to import/export **HTML** documents you have to use the **HtmlFormatProvider** class.
- **RTF** - to import/export **RTF** documents you have to use the **RtfFormatProvider** class.
- **Plain text** - to import/export plain text documents you have to use the **TxtFormatProvider**class.
- **PDF** - to export documents to **PDF** you have to use the **PdfFormatProvider** class.

Here are some examples on how to export and import.

> The "Export to String" and "Import from String" examples are only valid for the text-based format providers (Html, Xaml, Rtf and TxtFormatProvider). The "Export to File" and "Import from File" are applicable to each of the format providers (save for PDF import). To use them with the desired format just replace the format provider and change the settings of the **SaveFileDialog** or the**OpenFileDialog**.

Export to String

[C#] Export to String

```
public string ExportToXAML(RadDocument document)
{
    XamlFormatProvider provider = new XamlFormatProvider();
    return provider.Export(document);
}
```

[VB.NET] Export to String

```
Public Function ExportToXAML(ByVal document As RadDocument) As String
    Dim provider As New XamlFormatProvider()
    Return provider.Export(document)
End Function
```

Export to File

[C#] Export to File

```
public void ExportToDocx(RadDocument document)
{
    DocxFormatProvider provider = new DocxFormatProvider();
    SaveFileDialog saveDialog = new SaveFileDialog();
    saveDialog.DefaultExt = ".docx";
    saveDialog.Filter = "Documents|*.docx";
    DialogResult dialogResult = saveDialog.ShowDialog();
    if (dialogResult == System.Windows.Forms.DialogResult.OK)
    {
        using (Stream output = saveDialog.OpenFile())
```

```
        {
            provider.Export(document, output);
            MessageBox.Show("Saved Successfuly!");
        }
    }
}
```

[VB.NET] Export to File

```
Public Sub ExportToDocx(ByVal document As RadDocument)
   Dim provider As New DocxFormatProvider()
   Dim saveDialog As New SaveFileDialog()
   saveDialog.DefaultExt = ".docx"
   saveDialog.Filter = "Documents|*.docx"
   Dim dialogResult As DialogResult = saveDialog.ShowDialog()
   If dialogResult = System.Windows.Forms.DialogResult.OK Then
       Using output As Stream = saveDialog.OpenFile()
           provider.Export(document, output)
           MessageBox.Show("Saved Successfuly!")
       End Using
   End If
End Sub
```

Import from String

[C#] Import from String

```
public RadDocument ImportXaml(string content)
{
   XamlFormatProvider provider = new XamlFormatProvider();
   return provider.Import(content);
}
```

[VB.NET] Import from String

```
Public Function ImportXaml(ByVal content As String) As RadDocument
    Dim provider As New XamlFormatProvider()
    Return provider.Import(content)
End Function
```

Import from File

[C#] Import from File

```
public RadDocument ImportDocx()
{
   RadDocument document = null;
   IDocumentFormatProvider provider = new DocxFormatProvider();
   OpenFileDialog openDialog = new OpenFileDialog();
   openDialog.Filter = "Documents|*.docx";
   openDialog.Multiselect = false;
   DialogResult dialogResult = openDialog.ShowDialog();
   if (dialogResult == System.Windows.Forms.DialogResult.OK)
   {
       using (FileStream stream = new FileStream(openDialog.FileName, FileMode.Open))
       {
           document = provider.Import(stream);
       }
   }
```

```
    return document;
}
```

[VB.NET] Import from File

```
Public Function ImportDocx() As RadDocument
    Dim document As RadDocument = Nothing
    Dim provider As IDocumentFormatProvider = New DocxFormatProvider()
    Dim openDialog As New OpenFileDialog()
    openDialog.Filter = "Documents|*.docx"
    openDialog.Multiselect = False
    Dim dialogResult As DialogResult = openDialog.ShowDialog()
    If dialogResult = System.Windows.Forms.DialogResult.OK Then
        Using stream As New FileStream(openDialog.FileName, FileMode.Open)
            document = provider.Import(stream)
        End Using
    End If
    Return document
End Function
```

Import/Export settings

As sometimes the formats may support several ways for presenting one and the same content, some customization options have been provided in order to specify the result which the user expects. This is achieved with the help of import and export settings, which some format providers expose (**HtmlFormatProvider** and **PdfFormatProvider**). You need to create an instance of the settings and assign it to the property of the provider you will be using. Then, you can set (or retrieve) the properties you need. **HtmlFormatProvider** makes use of **HtmlExportSettings** and**HtmlImportSettings** to control import/export.

1. **HtmlExportSettings** provides the following options:
    - **DocumentExportLevel** – you can choose between **Document** and **Fragment**.**Document** is the default value, which includes the HTML declaration, the **<HTML>**,**<TITLE>**, **<HEAD>** and **<BODY>** tags, whereas setting the document export level to**Fragment** results in exporting the content of the **<BODY>** tag only.
    - **StylesExportMode** – the options here are **Inline** and **Classes**, the default one being**Classes**. Predefined classes is the preferred way for setting styles, yet inline styles may be useful with regard to the consumer of the HTML. For instance, in order to use the exported HTML in **Telerik Reporting**, the **StylesExportMode** needs to be set to **Inline**.
    - **ImageExportMode** – the user can choose between several options or even provide his own implementation by choosing the **ImageExportingEvent** option for the**ImageExportMode** property and handling the **ImageExportingEvent**. The **UriSource**option can be used if you want to export an image by setting its **src** property to the **URL**rather than having the raw data in the exported document.
2. **HtmlImportSettings** provides the following options:
    - **UseDefaultStylesheetForFontProperties** – a **Boolean** property indicating whether the default font properties of RadRichTextBox or the defaults in the HTML specification should be used for the elements that do not set their **FontSize**, **FontFamily**, **FontWeight** and**FontStyle** explicitly.
    - **LoadImageFromUrl** event – this event was introduced at a time when**HtmlFormatProvider** did not automatically load images from URLs. The feature is currently supported out of the box, but this event can be useful if using virtual directories and files on the server.

PdfFormatProvider exposes an **ExportSettings** property of type **PdfExportSettings**, which can be used in the same way as the export settings of **HtmlFormatProvider**.

PdfExportSettings include the following options:

- **ContentsCompressionMode** – this property allows you to choose if you wish to make use of compression (by setting it to **Deflate** or **Automatic**) or not (**PdfContentsCompressionMode.None**) of the text content of the document.
- **ContentsDeflaterCompressionLevel** – an integer between -1 and 9, used to get or set the compression level to be used when deflating the content of the document. Default Compression is **-1**, No Compression is **0** and Best Compression is **9**;
- **ImagesCompressionMode** – the user can choose between **None**, **Jpeg** (supported only for images, imported as JPEG), Deflate (the deflate algorithm will be applied to compress the images) or **Automatic** (the best algorithm will be automatically decided upon for you).
- **ImagesDeflaterCompressionLevel** – same as **ContentsDeflaterCompressionLevel**, but applied to the images in the document. This property is respected when an image is compressed with **Deflate**.

PDF import is currently **not** supported, so there are not any import settings.

There are not any **Import** or **ExportSettings** for **XAML** either, as the **XAML** serialization is lossless and all elements are imported and exported as they would appear if declared in a **XAML**page in the application.

DocxFormatProvider does not currently provide any settings as the document content is matched as closely as possible to the Word document.

5.13 HTML Support

A major requirement for providing rich text editing capabilities is support for HTML. The RadRichTextBox control is designed to offer **import/export (Section 5.12)** functionality for **HTML** out of the box – through the **HtmlFormatProvider**. However, as **HTML** is a diverse and complicated standard, supporting it comes with certain limitations. **HtmlFormatProvider** is designed to handle slightly malformed **HTML** (missing closing tags, missing quotes around attributes), however in this case correct parsing cannot be guaranteed. For best results we recommend importing valid **XHTML**content.

The purpose of this article is to outline which parts of the HTML are supported.

Importing

When the **HtmlFormatProvider** imports an **HTML** document, it parses it and converts the HTML element tree to **RadDocument**. Importing is basically made up of two parts – content and styling.

Content

HTML content is mapped to several types of **RadDocument** elements – paragraphs, spans and tables.

Tags mapped to paragraphs are:

- **<p>**
- **<h1>** through **<h6>**
- **<ul>, <ol>, **
- **<tr>**

Tags mapped to spans are:

- **formatting tags, , <font>**

Additionally, line breaks are inserted between consequent block elements (** **).

Styling

Styling is supported through CSS styles (classes or inline) and a number of formatting tags (including**<font>**). Although the **<font>** tag has been deprecated, legacy software and markup requires support for it.

Supported formatting tags are:

- **<b>, ** – turn on bold
- **<i>, ** – turn on italic
- **<u>** – turn on underline
- **<h1>** through **<h6>** – apply heading

Supported CSS properties are:

- **text-align**
- **margin-left**
- **margin-right**
- **color**
- **background-color**
- **font-family**
- **font-size**
- **font-style**
- **font-weight**
- **text-decoration (underline, line-through)**
- **vertical-align (sub, super)**

Supported **<font>** tag attributes are:

- **face** – equivalent to CSS property **font-family**
- **size** – equivalent to CSS property **font-size**
- **color** – equivalent to CSS property **color**

Exporting

When exporting to **HTML**, the **HtmlFormatProvider** iterates through the **RadDocument** tree and generates **HTML** nodes. It is designed to generate valid **XHTML** document in order to preserve the formatting as much as possible. Styles are exported as CSS classes in the head of the document and used through the body. The structure of the **HTML** document closely resembles that of**RadDocument** – paragraphs and spans.

5.14 Spellcheck

The RadRichTextBox control is designed to support "spell checking as you type" by setting a single property and specifying a proper dictionary to it. This topic will explain you the following:

Enabling SpellCheck

To enable or disable the spell checking functionality (present as red wavy underlines below unrecognized

words), you can use the **IsSpellCheckingEnabled** property on the RadRichTextBox. When the property is *False*, no dictionaries are loaded and no overhead is incurred for spell checking.

You can customize the spell checker by using the **SpellChecker** property of RadRichTextBox. It's of type **ISpellChecker**. By default an object of type **DocumentSpellChecker** that implements the interface, is used for this property. You can either use it or provide your custom class that implements the **ISpellChecker** interface.

Dictionaries

The dictionaries in RadRichTextBox implement the **IWordDictionary** interface. Easy interoperability with dictionaries from **RadSpell** for ASP.NET is achieved through the **WordDictionary** class, which supports the loading of a dictionary directly from the ***.tdf** files, used with **RadSpell**.

Here is an example of a **WordDictionary** loaded from a **TDF** file.

When adding a **WordDictionary** or similar object use the **AddDictionary(IWordDictionary dictionary, CultureInfo culture)** method of the **DocumentSpellChecker**. You can also associate a dictionary with a specific culture. The method to remove this dictionary is**RemoveDictionary(CultureInfo culture)**.

The given example doesn't contain the logic used to read the **TDF** file as a **Stream**.

[C#] Load dictionary

```
private void LoadDictionary(Stream tdfFileStream)
{
   WordDictionary dictionary = new WordDictionary();
   dictionary.Load(tdfFileStream);
   ((DocumentSpellChecker)this.radRichTextBox1.SpellChecker).AddDictionary(dictionary,
CultureInfo.InvariantCulture);
}
```

[VB.NET] Load dictionary

```
Private Sub LoadDictionary(ByVal tdfFileStream As Stream)
    Dim dictionary As New WordDictionary()
    dictionary.Load(tdfFileStream)
    CType(Me.RadRichTextBox1.SpellChecker, DocumentSpellChecker).AddDictionary(dictionary,
CultureInfo.InvariantCulture)
End Sub
```

Adding a Word

To add a word to a dictionary you can either use the **AddWord()** method of the**DocumentSpellChecker** or of the dictionary itself. Using the first one you can add a word to multiple dictionaries associated to the same culture. This done done by passing the desired culture as parameter to the method.

Using the overload of the **AddWord()** method that takes only the word as argument is equal to using the second overload and passing **CultureInfo.InvariantCulture** as argument.

Using the **AddWord()** method of the dictionary itself will add the word only to the respective dictionary.

Here is an example:

[C#] Add word to dictionary

```
this.radRichTextBox1.SpellChecker.AddWord("RadRichTextBox", CultureInfo.InvariantCulture);
```

[VB.NET] Add word to dictionary

```
Me.RadRichTextBox1.SpellChecker.AddWord("RadRichTextBox", CultureInfo.InvariantCulture)
```

Internationalization

The spell checking component is designed to suit scenarios where different cultures take place in the same application. Internationalization is achieved through associating each dictionary and custom dictionary with a specific culture (or the **InvariantCulture** as the default one).

> The given example doesn't contain the logic used to read the **TDF** file as a **Stream**.

[C#] Load international dictionary

```
private void LoadDictionaryDE(Stream tdfFileStream)
{
    WordDictionary dictionary = new WordDictionary();
   dictionary.Load( tdfFileStream );
   (( DocumentSpellChecker )this.radRichTextBox1.SpellChecker ).AddDictionary( dictionary,
new CultureInfo( "de-DE" );
}
```

[VB.NET] Load international dictionary

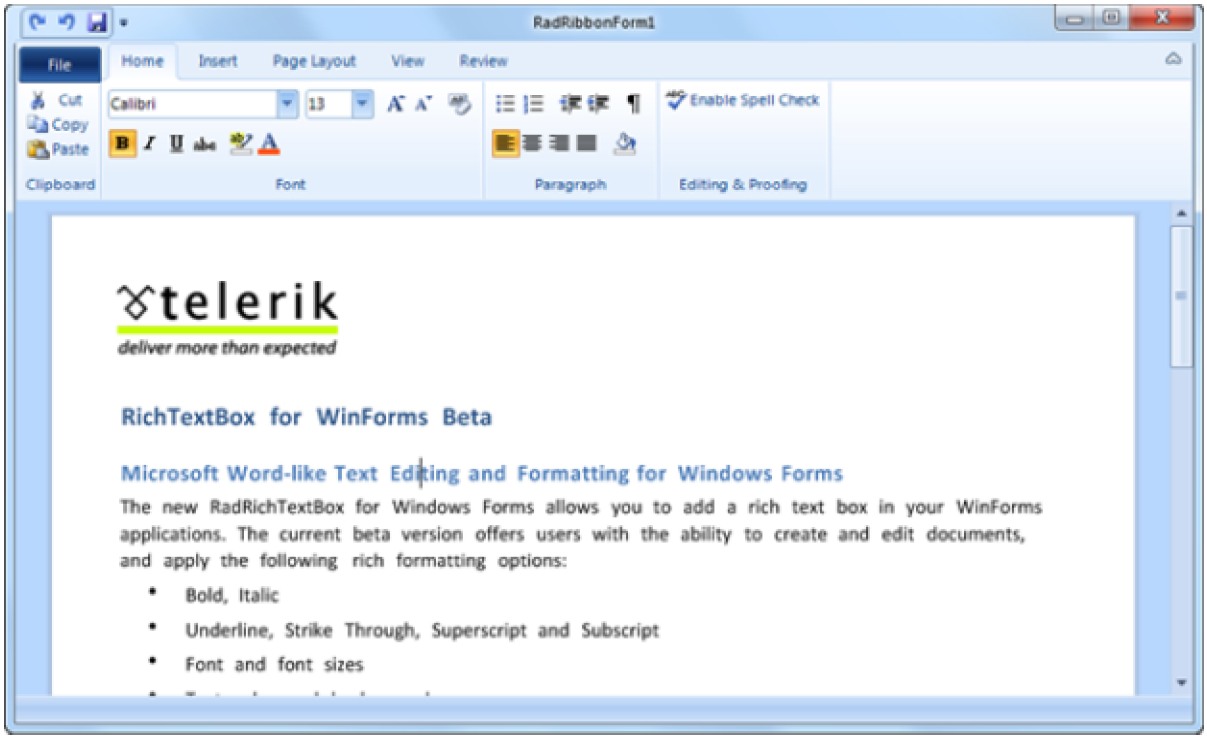
```
Private Sub LoadDictionaryDE(ByVal tdfFileStream As Stream)
    Dim dictionary As New WordDictionary()
    dictionary.Load(tdfFileStream)
    CType(Me.RadRichTextBox1.SpellChecker, DocumentSpellChecker).AddDictionary(dictionary,
New CultureInfo("de-DE"))
End Sub
```

RadRichTextBox has an integration with the keyboard and exposes some hot key combinations. Here is a list of them.

Action	Hotkey
Copy	Ctrl+C
Paste	Ctrl+V
Cut	Ctrl+X
Undo	Ctrl+Z
Redo	Ctrl+Y
Toggle Bold	Ctrl+B, Ctrl+Shift+B
Toggle Italic	Ctrl+I, Ctrl+Shift+I
Toggle Underline	Ctrl+U, Ctrl+Shift+U
Align Justify	Ctrl+J, Ctrl+Shift+J
Align Right	Ctrl+R, Ctrl+Shift+R
Align Left	Ctrl+L, Ctrl+Shift+L

Align Center	Ctrl+E, Ctrl+Shift+E

5.15 RibbonBar UI



RadRichTextBox control comes with a predefined number of shortcuts which trigger commands that allow you to manipulate the selected text the way you want. However, in many cases it is more convenient for the user to click buttons which execute these commands instead of pressing a shortcut key. The purpose of this article is to demonstrate how you can build user interface appropriate for the purposes of RadRichTextBox with the help of RadRibbonBar for WinForms.

What is more, at the end of the article you will find a link to a project which implements a RadRibbonBar built according to the RadRichTextBox capabilities. You will also find a description of how you can bind this RadRibbonBar to an existing RadRichTextBox.

Building RadRibbonBar UI

Building the user interface of the RadRichTextBox-enabled is pretty simple. You can use the **design-time capabilities (http://www.telerik.com/help/winforms/ribbonbar-getting-started.html)** of RadRibbonBar to insert and arrange elements in it. The other option is to **insert elements through code (http://www.telerik.com/help/winforms/ribbonbar-programming-radribbonbar-adding-and-removing-tabs-and-ribbonbar-groups.html)**. The following code snippet demonstrates how to insert a 'Home' tab, a 'Font' group and a button which toggles the bold mode of the text:

[C#]

```
public RichTextBoxRibbonUI()
{
   InitializeComponent();

   RibbonTab tabHome = new RibbonTab();
   tabHome.Text = "Home";
   this.radRibbonBar1.CommandTabs.Add(tabHome);
```

```
    RadRibbonBarGroup grpFont = new RadRibbonBarGroup();
    grpFont.Text = "Font";
    tabHome.Items.Add(grpFont);

    RadToggleButtonElement btnBold = new RadToggleButtonElement();
    btnBold.Text = "Bold";
    grpFont.Items.Add(btnBold);

    btnBold.ToggleStateChanged += new StateChangedEventHandler(btnBold_ToggleStateChanged);
}

void btnBold_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
    this.radRichTextBox1.ToggleBold();
    this.radRichTextBox1.Focus();
}
```

[VB.NET]

```
Public Sub New()
    InitializeComponent()

    Dim tabHome As New RibbonTab()
    tabHome.Text = "Home"
    Me.radRibbonBar1.CommandTabs.Add(tabHome)

    Dim grpFont As New RadRibbonBarGroup()
    grpFont.Text = "Font"
    tabHome.Items.Add(grpFont)

    Dim btnBold As New RadToggleButtonElement()
    btnBold.Text = "Bold"
    grpFont.Items.Add(btnBold)

    AddHandler btnBold.ToggleStateChanged, AddressOf btnBold_ToggleStateChanged
End Sub

Private Sub btnBold_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs)
    Me.radRichTextBox1.ToggleBold()
    Me.radRichTextBox1.Focus()
End Sub
```

Of course, RadRichTextBox and RadRibbonBar UI should always be in sync, so you should handle the CurrentEditingStyleChanged event as it is described for CurrentParagraphStyleChanged event in the following documentation article: **Working with RadRichTextBox (Section 5.4)**.

Using the RadRichTextBox-enabled RadRibbonBar

You can download the customized RadRibbonBar component from the following links (C#/VB.NET):

TelerikWinFormsEditorCS (http://www.telerik.com/download/TelerikWinFormsEditorCS.zip) - **http://www.telerik.com/download/TelerikWinFormsEditorCS.zip**

TelerikWinFormsEditorVB (http://www.telerik.com/download/TelerikWinFormsEditorVB.zip) - **http://www.telerik.com/download/TelerikWinFormsEditorVB.zip**

After you download the project, please follow the steps below:

1. Add the project to your solution.
2. Add a project reference from the downloaded project to the project in which you plan to use the customized RadRibbonBar.
3. Set the target framework of the project in which you are going to use the customized RadRibbonBar instance to .NET Framework 4.0 or .NET 4.0 Client Profile.
4. Add the following Telerik references to the main project:
   - Telerik.WinControls.dll
   - Telerik.WinControls.RichTextBox.dll
   - Telerik.WinControls.SpellChecker.dll
   - Telerik.WinControls.UI.dll
   - TelerikCommon.dll

   If there is a RichTextBox on one of the forms in the main project, these references should should already exist.
5. Build the downloaded project. The customized RadRibbonBar component should appear in your Toolbox.
6. Drag it from your Toolbox to your Form.
7. Set the **AssociatedRichTextBox** property to the RadRichTextBox instance that will be controlled by the RadRibbonBar user interface.
8. Run your solution.
9. The commands that you trigger by using the RadRibbonBar interface will be applied to the text of the associtated RadRichTextBox instance.
10. The following properties will allow you to control the visibility of the tabs of the customized RadRibbonBar implementation:

[C#]

```
richRibbonBar1.ShowHomeTab = false;
richRibbonBar1.ShowInsertTab = false;
richRibbonBar1.ShowPageLayoutTab = false;
richRibbonBar1.ShowViewTab = false;
richRibbonBar1.ShowReviewTab = false;
```

[VB.NET]

```
richRibbonBar1.ShowHomeTab = False
richRibbonBar1.ShowInsertTab = False
richRibbonBar1.ShowPageLayoutTab = False
richRibbonBar1.ShowViewTab = False
richRibbonBar1.ShowReviewTab = False
```



In case you are using RadRibbonForm, you can substitute the Telerik.WinControls.UI.RadRibbonBar type with TelerikEditor.RichRibbonBar type in the Designer code file of the RadRibbonForm.

Feel free to add more properties that will be suitable for your scenario.

6 Menus

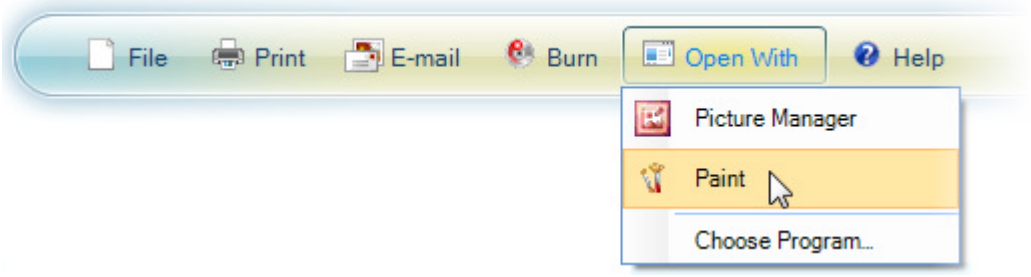
6.1 Objectives

- Learn how to add drop-down and context menus to your application using RadMenu, RadContextMenu and RadContextMenuManager controls.
- Learn how to build menus at design-time using the Menu Designer.
- Learn common programmatic tasks such as adding and removing menu items, locating and modifying menu items.
- Learn how to programmatically pop up up a context menu in a specific screen location.
- Become familiar with the RadRibbonBar and RadApplicationMenu controls.

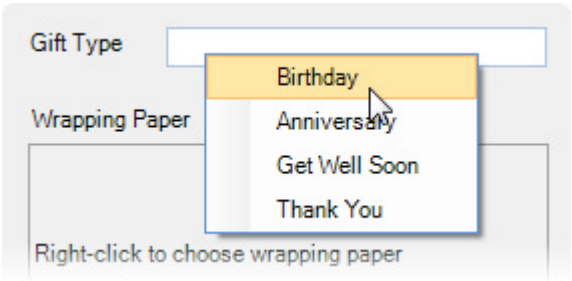
6.2 Introduction

Telerik has an array of flexible menuing options that help structure user choices within your application:

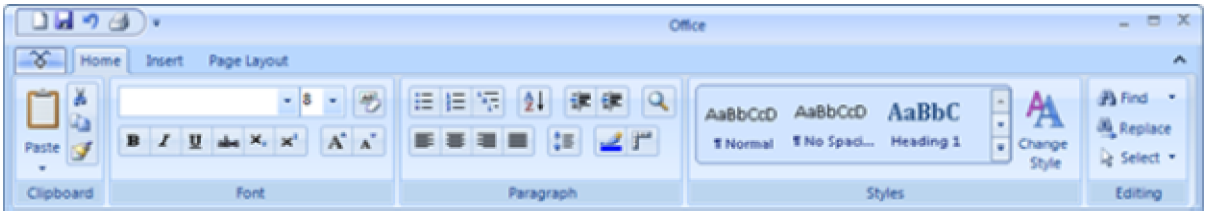
- **RadMenu** enables you to implement attractive navigation systems, comprised of classic or Office 2007-style menus. Based on the Telerik Presentation Foundation, the component gives you full control over item appearance, orientation, and text-wrapping, while allowing you to easily nest checkboxes, images, or any other controls in design-time.



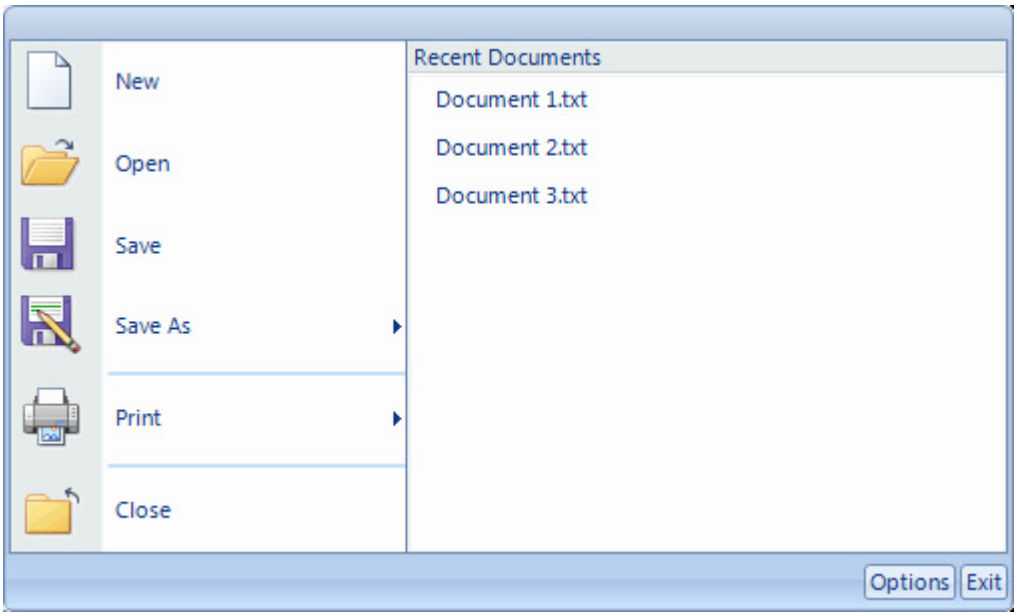
- The same RadMenu functionality is available anywhere in the form using **RadContextMenu**. Context menus can be associated with specific controls or popped up programmatically at any X/Y location.



- **RadRibbonBar** combines the functions of menus, tab strips and toolbars to make an interface similar to those introduced in Microsoft Office 2007. With RadRibbonBar you can organize all of the functionality of your application into a single compact ribbon.



- **RadApplicationMenu** is the Telerik counterpart of the "application menu" that displays controls used to perform actions on entire documents and forms, such as Save and Print.



6.3 Getting Started

In this Getting Started demonstration you will create a simple rich text edit and use the RadMenu to save and load the file, cut/copy/paste, format the text in bold or italic and display an "About" box.

Preparing the Project



You can find the complete source for this project at:

\Menus\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.

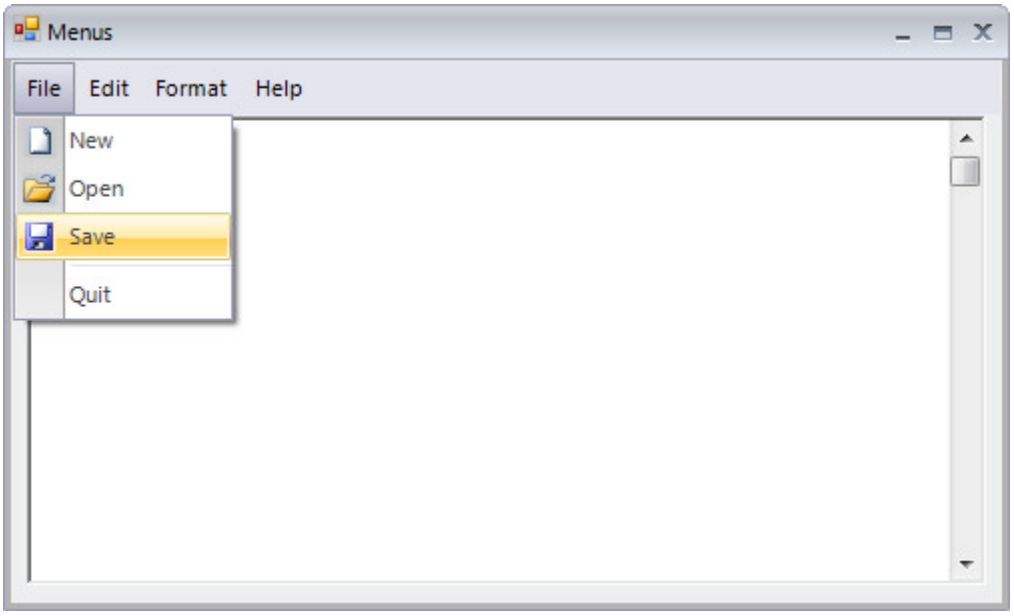
Adding Controls to the RadForm

1. Drop a **Office2007SilverTheme, OpenFileDialog, SaveFileDialog** and **ImageList** components on the form. These will all show up in the component tray below the form design surface. Set the ThemeName property of the RadForm to "Office2007Silver".
2. Click the ImageList Smart Tag "Choose Images" option and add images to represent the actions below.

> You can typically find images in the C:\Program Files\Microsoft Visual Studio 9.0\Common7\VS2008ImageLibrary\1033 directory in "VS2008ImageLibrary.zip".

- New
- Open
- Save
- Cut
- Copy
- Paste
- Help

3. Add a **RadMenu** control to the form. Set the **ThemeName** property of the menu to "Office2007Silver". Set the **ImageList** property to point to the ImageList you added to the form.
4. Add a standard WinForms **RichTextBox** control to the form. Set the Name property to "tbContent". Size the RichTextBox within the available area on the form and set the Anchor property to "Top, Bottom, Left, Right". Also set the **HideSelection** property to False and **ShowSelectionMargin** property to True.
5. In the RadMenu designer, click the "Type Here" prompt and enter "File". Repeat this step to create more top level menu items "Edit", "Format" and "Help".
6. Under the top level "File" menu item, click the "Add New" prompt and add a RadMenuItem. Set the name to "miNew", Text to "New" and the ImageKey to the image list image key representing "new". Add the "File" menu items:
   - **Name** = "miNew", **Text** = "New", **ImageKey** = <image for New>, **TextImageRelation** = ImageBeforeText
   - **Name** = "miOpen", **Text** = "Open", **ImageKey** = <image for Open>, **TextImageRelation** = ImageBeforeText
   - **Name** = "miSave", **Text** = "Save", **ImageKey** = <image for Save>, **TextImageRelation** = ImageBeforeText
   - Add a **RadMenuSeparatorItem.**
   - **Name** = "miQuit", Text = "Quit", **TextImageRelation** = ImageBeforeText
7. Add the "Edit" menu items:
   - **Name** = "miCut", **Text** = "Cut", **ImageKey** = <image for Cut>, **TextImageRelation** = ImageBeforeText
   - **Name** = "miCopy", **Text** = "Copy", **ImageKey** = <image for Copy>, **TextImageRelation** = ImageBeforeText
   - **Name** = "miPaste", **Text** = "Paste", **ImageKey** = <image for Paste>, **TextImageRelation** = ImageBeforeText
8. Add the "Format" menu items:
   - **Name** = "miBold", **Text** = "Bold", **ImageKey** = <image for Bold>, **CheckOnClick** = True, **TextImageRelation** = ImageBeforeText
   - **Name** = "miItalic", **Text** = "Italic", **ImageKey** = <image for Italic>, **CheckOnClick** = True, **TextImageRelation** = ImageBeforeText
9. Add the "Help" menu item:
   - **Name** = "miAbout", **Text** = "About", **ImageKey** = <image for About>, **TextImageRelation** = ImageBeforeText
10. Press **Ctl-F5** to run the application and see what the menu looks like so far.



11. Double-click the "New" menu item and add the following code to the Click event handler:

[VB] Handling the Menu Item Click Event

```
Private Sub miNew_Click(sender As Object, e As EventArgs)
 tbContent.Clear()
End Sub
```

[C#] Handling the Menu Item Click Event

```
private void miNew_Click(object sender, EventArgs e)
{
 tbContent.Clear();
}
```

12. Double click each menu item except for the "Bold" and "Italic" items and add the corresponding event handlers below:

[VB] Handling the Menu Item Click Events

```
Private Sub miOpen_Click(sender As Object, e As EventArgs)
 If openFileDialog1.ShowDialog() = DialogResult.OK Then
  tbContent.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.RichText)
 End If
End Sub
Private Sub miSave_Click(sender As Object, e As EventArgs)
 If saveFileDialog1.ShowDialog() = DialogResult.OK Then
  tbContent.SaveFile(saveFileDialog1.FileName, RichTextBoxStreamType.RichText)
 End If
End Sub
Private Sub miAbout_Click(sender As Object, e As EventArgs)
 RadMessageBox.SetThemeName(Me.ThemeName)
 RadMessageBox.Show(" By " + Environment.UserName + ", " + DateTime.Today.ToLongDateStri
(), "About RadMenu Demo")
End Sub
Private Sub miCut_Click(sender As Object, e As EventArgs)
 tbContent.Cut()
End Sub
```

```
Private Sub miCopy_Click(sender As Object, e As EventArgs)
 tbContent.Copy()
End Sub
Private Sub miPaste_Click(sender As Object, e As EventArgs)
 tbContent.Paste()
End Sub
Private Sub miQuit_Click(sender As Object, e As EventArgs)
 Me.Close()
End Sub
```

[C#] Handling the Menu Item Click Events

```
private void miOpen_Click(object sender, EventArgs e)
{
 if (openFileDialog1.ShowDialog() == DialogResult.OK)
 {
   tbContent.LoadFile(openFileDialog1.FileName, RichTextBoxStreamType.RichText);
 }
}
private void miSave_Click(object sender, EventArgs e)
{
 if (saveFileDialog1.ShowDialog() == DialogResult.OK)
 {
   tbContent.SaveFile(saveFileDialog1.FileName, RichTextBoxStreamType.RichText);
 }
}
private void miAbout_Click(object sender, EventArgs e)
{
 RadMessageBox.SetThemeName(this.ThemeName);
 RadMessageBox.Show(" By " + Environment.UserName + ", "
   + DateTime.Today.ToLongDateString(), "About RadMenu Demo");
}
private void miCut_Click(object sender, EventArgs e)
{
 tbContent.Cut();
}
private void miCopy_Click(object sender, EventArgs e)
{
 tbContent.Copy();
}
private void miPaste_Click(object sender, EventArgs e)
{
 tbContent.Paste();
}
private void miQuit_Click(object sender, EventArgs e)
{
 this.Close();
}
```

13. Create ToggleStateChanged event handlers for the "miBold" and "miItalic" menu items. Add the following code to implement the font style changes:

[VB] Handling the ToggleStateChanged Events

```
Private Sub miBold_ToggleStateChanged(sender As Object, args As StateChangedEventArgs)
 If args.ToggleState = ToggleState.[On] Then
  tbContent.SelectionFont = New Font(tbContent.SelectionFont, tbContent.SelectionFont.Sty
Or FontStyle.Bold)
```

```
 Else
  tbContent.SelectionFont = New Font(tbContent.SelectionFont, tbContent.SelectionFont.Sty
And Not FontStyle.Bold)
 End If
End Sub
Private Sub miItalic_ToggleStateChanged(sender As Object, args As StateChangedEventArgs)
 If args.ToggleState = ToggleState.[On] Then
  tbContent.SelectionFont = New Font(tbContent.SelectionFont, tbContent.SelectionFont.Sty
Or FontStyle.Italic)
 Else
  tbContent.SelectionFont = New Font(tbContent.SelectionFont, tbContent.SelectionFont.Sty
And Not FontStyle.Italic)
 End If
End Sub
```

[C#] Handling the ToggleStateChanged Events

```
private void miBold_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
 if (args.ToggleState == ToggleState.On)
 {
   tbContent.SelectionFont =
     new Font(tbContent.SelectionFont,
      tbContent.SelectionFont.Style | FontStyle.Bold);
 }
 else
 {
   tbContent.SelectionFont =
     new Font(tbContent.SelectionFont,
       tbContent.SelectionFont.Style & ~FontStyle.Bold);
 }
}
private void miItalic_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
 if (args.ToggleState == ToggleState.On)
 {
   tbContent.SelectionFont =
     new Font(tbContent.SelectionFont,
      tbContent.SelectionFont.Style | FontStyle.Italic);
 }
 else
 {
   tbContent.SelectionFont =
     new Font(tbContent.SelectionFont,
       tbContent.SelectionFont.Style & ~FontStyle.Italic);
 }
}
```

14. Add one more event handler for the RichTextBox SelectionChanged event so that the current font attributes of the selection will be reflected in the Bold and Italic menu items:

[VB] Handling the SelectionChanged Event

```
Private Sub tbContent_SelectionChanged(sender As Object, e As EventArgs)
 miBold.IsChecked = tbContent.SelectionFont.Bold
 miItalic.IsChecked = tbContent.SelectionFont.Italic
End Sub
```

[C#] Handling the SelectionChanged Event

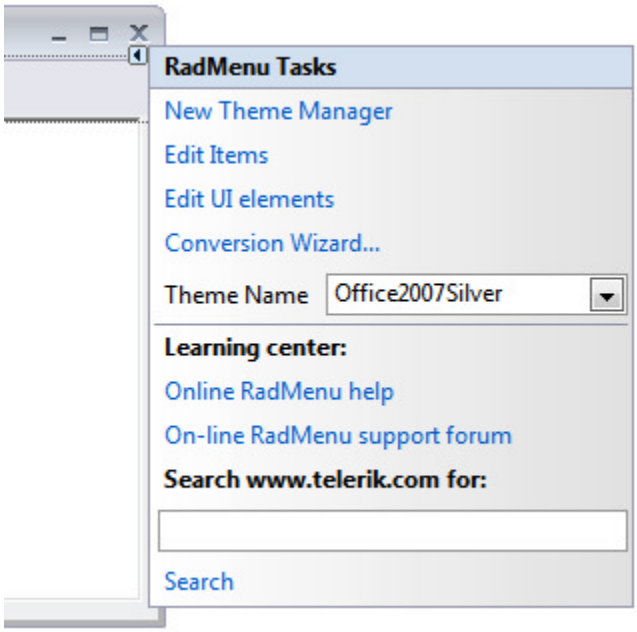
```
private void tbContent_SelectionChanged(object sender, EventArgs e)
{
 miBold.IsChecked = tbContent.SelectionFont.Bold;
 miItalic.IsChecked = tbContent.SelectionFont.Italic;
}
```

15. Press **Ctl-F5** to run the application. Try entering text, saving, loading and test each of the menu items.

6.4 Using the Design Time Interface

RadMenu Smart Tag

The RadMenu has the same basic Smart Tag minimum layout that you see for all RadControls. In addition you have an Edit Items item that displays the RadItem Collection Editor.



Tasks

From the Smart Tag Tasks menu you can Open Theme Builder to style all aspects of your control, select New Theme Manager to add a RadThemeManager component to the component tray, Edit Items to display the RadItem Collection Editor, Edit UI elements to browse all of the elements of the menu and to change properties for any element. The Theme Name drop down lets you pick an existing theme to style your control.

Learning center

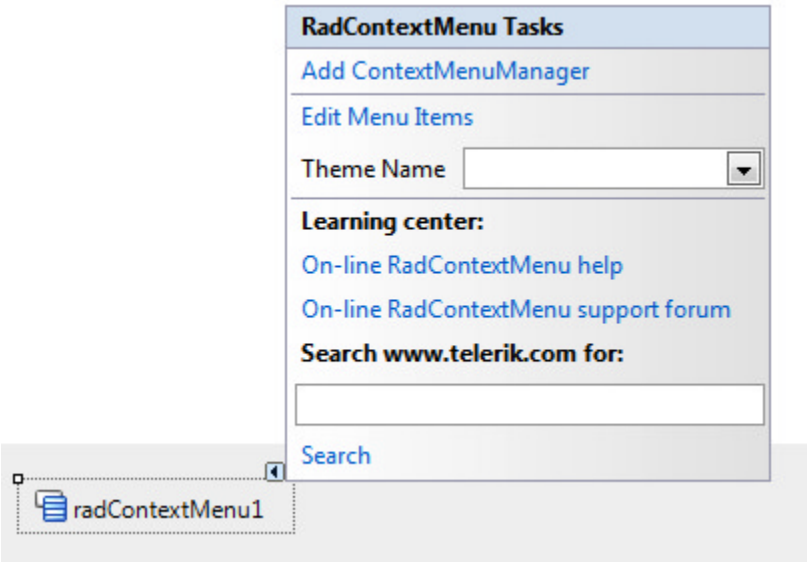
The Learning center lets you navigate to a web browser with online help for the currently selected control or to the online support forums.

Search

Enter search criteria in the edit space provided and click the Search link to navigate directly to search on the Telerik web site.

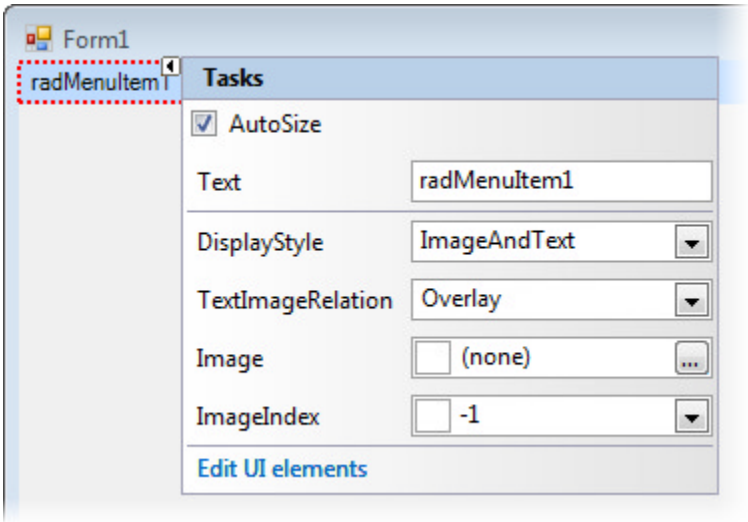
RadContextMenu Smart Tag

The RadContextMenu Smart Tag lets you add a RadContextMenuManager control to the form and to Edit Menu Items by way of the RadItem Collection Editor.



Menu Item Smart Tags

Each individual RadMenuItem within the RadMenu also has a Smart Tag. Use this Smart Tag to quickly toggle the AutoSize property, set the Text property, set image properties and to display the Element Hierarchy Editor for the selected RadMenuItem.



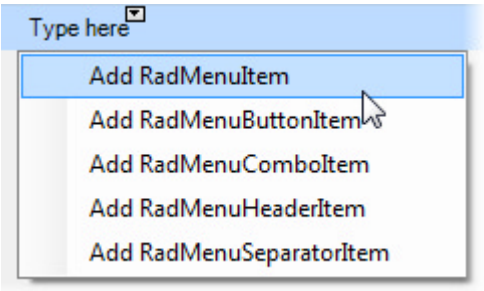
Menu Designer

Adding Main Menu Items

There are multiple methods for building menus using the property collection editor or the RadMenu designer. To add a new main menu item, click the the RadMenu area labeled "Type here", and type your top level menu item directly into the entry space provided. When you're finished click ESC to abandon your edits or Enter to accept the edits and create a new RadMenuItem.



...Or click the drop-down arrow to the right of the existing main menu items and select one of the "Add" choices" to create an item of the corresponding type. Once the menu item is created you can use the Smart Tag to configure the Text, Image properties and edit the Items collection for the menu item.

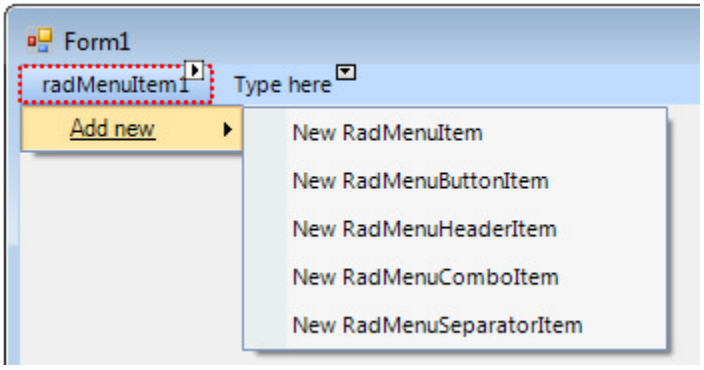


...Or, click the RadMenu control, open its Smart Tag menu and select Edit Items. Add new RadMenuItems using the RadElement Collection Editor.

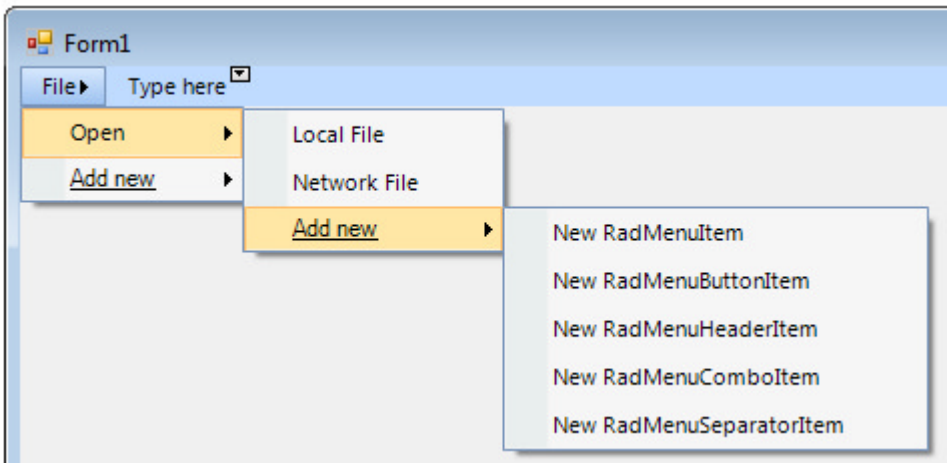
Adding Sub Menu Items

To add a new sub-menu item to a main menu item, select the main menu item, click in its Items property in the Properties window, click the ellipsis button, and then use the RadElement Collection Editor.

...Or click a main menu item in the designer to invoke the "Add new" item. Select one of the menu item types to create it and add it below the selected menu item.



Each RadMenuItem can have its own items to allow menu designs that require multiple levels of hierarchy.



Removing Menu Items

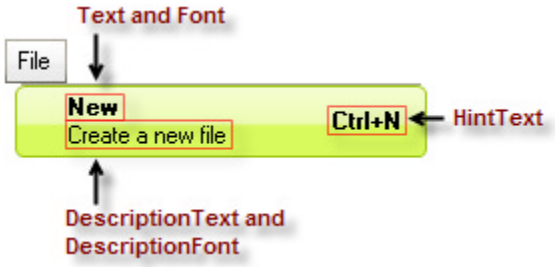
To remove a main menu or sub-menu item, select the item and press Delete or right click the menu item and select Delete from the context menu.

6.5 Working with the Menu Controls

Working with Menu Items

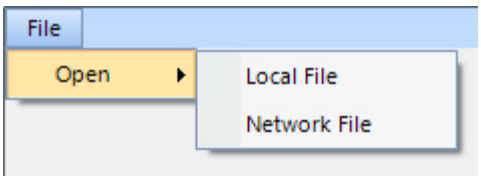
RadMenuItem

The RadMenuItem represents a standard menu item.

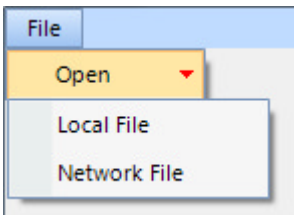


Here are some of the significant RadMenuItem properties:

- **AngleTransform** sets the angle offset from horizontal with which the item will be displayed.
- **CommandBinding** allows you to configure keyboard shortcuts to fire a given menu item. See the Getting Started topic for an example.
- **DisplayStyle** lets you choose whether to show an image, text, or both on the item.
- **DescriptionText** contains the text for the description that appears just below the Text. DescriptionFont controls the font characteristics of the DescriptionText display.
- **Enabled** when set to false shows the menu item as grayed out and does not react to mouse clicks. To allow non-standard painting when the menu item is not enabled set the UseDefaultDisabledPaint property to false and then use the Visual Style Builder to customize the appearance of this state.
- **FlipText** flips the text of the item vertically if set to True.
- **BackColor** and **ForeColor** set the colors of the background and text.
- **Font** sets the font and size of the item text.
- **HintText** is the text that can appear on the right side of the menu item, such as "Ctrl+N" for "New".
- **Image**, **ImageIndex**, and **ImageKey** let you assign an image to display on the item. To use ImageIndex and ImageKey you will need to assign the RadMenu ImageList property to an ImageList in the component tray.
- **CheckOnClick** indicates whether selecting the item should set a check mark.
- **IsChecked** controls whether a check mark is displayed on the item.
- **TextImageRelation** controls the relationship between the text and image displayed on the item.
- **Text** specifies the text to show on the item.
- **TextOrientation** allows you to choose vertical or horizontal text display.
- **PopupDirection** indicates whether the direction in which the drop-down menu will open will be to the left, right, up, or down.
- **ShowArrow** displays an arrow button on the drop-down menu when there are sub-menus as shown on the "Open" menu item below.



Use the menu items Layout.ArrowPrimitive to alter the appearance of the arrow graphic. You can use this together with PopupDirection to make sub menus popup below.



[VB] Altering the Arrow and Popup Direction

```
miOpen.Layout.ArrowPrimitive.Direction = Telerik.WinControls.ArrowDirection.Down
miOpen.Layout.ArrowPrimitive.ForeColor = Color.Red
miOpen.PopupDirection = Telerik.WinControls.UI.RadDirection.Down
```

[C#] Altering the Arrow and Popup Direction

```
miOpen.Layout.ArrowPrimitive.Direction = Telerik.WinControls.ArrowDirection.Down;
miOpen.Layout.ArrowPrimitive.ForeColor = Color.Red;
miOpen.PopupDirection = Telerik.WinControls.UI.RadDirection.Down;
```

- **StringAlignment** formats the alignment of the text string so it is positioned near, center, or far from the left border of the menu item.

RadMenuComboItem

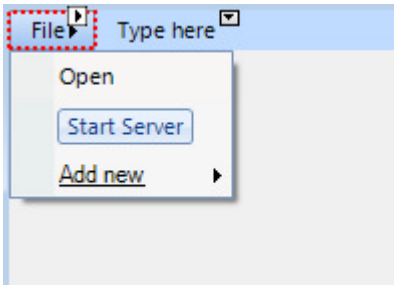
The RadMenuComboItem allows you to put a combo box on a menu. To add items to the combo box work with the Items collection of the RadMenuComboItem.ComboBoxElement property. Because the ComboBoxElement returns a RadComboBox, you can also use data binding to put items in the combo box from any data source.

RadMenuSeparatorItem

The RadMenuSeparatorItem is used to add a horizontal line for visual separation between successive items on a menu.

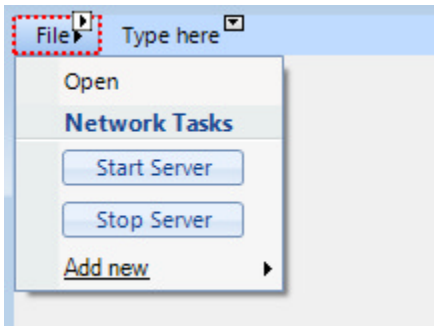
RadMenuButtonItem

This menu item has button appearance and behavior. It also does not have HintText, DescriptionText and does not have an IsChecked property. The screenshot below shows a RadMenuButtonItem with text "Start Server".



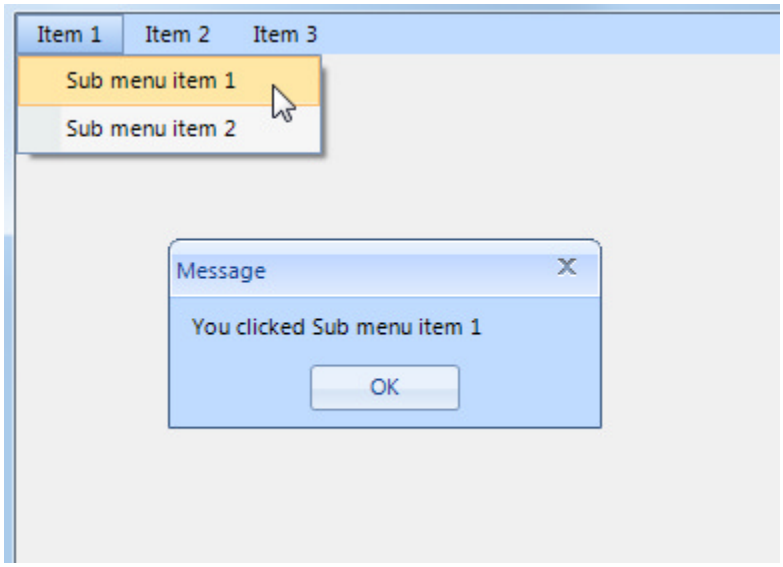
RadMenuHeaderItem

This menu item displays header text as shown in the screenshot below with text "Network Tasks".



Adding Menu Items in Code

To add a series of menu items in code use one of the several RadMenuItem constructor overloads to create the item then add the item to the RadMenu Items collection.



[VB] Adding Menu Items

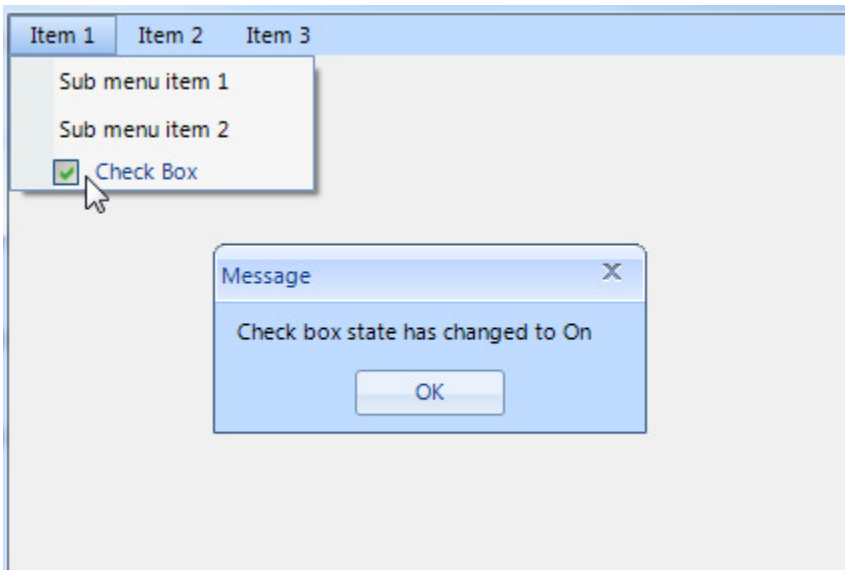
```
Public Sub New()
 InitializeComponent()
 radMenu1.Items.Add(New RadMenuItem("Item 1"))
 radMenu1.Items.Add(New RadMenuItem("Item 2"))
 radMenu1.Items.Add(New RadMenuItem("Item 3"))
 Dim item1 As RadMenuItem = DirectCast(radMenu1.Items(0), RadMenuItem)
 item1.Items.Add(New RadMenuItem("Sub menu item 1"))
 item1.Items.Add(New RadMenuItem("Sub menu item 2"))
 (DirectCast(item1.Items(0), RadMenuItem)).Click += New EventHandler(Form1_Click)
End Sub
Sub Form1_Click(sender As Object, e As EventArgs)
 RadMessageBox.Show("You clicked " + (TryCast(sender, RadMenuItem)).Text)
End Sub
```

[C#] Adding Menu Items

```
public Form1()
{
 InitializeComponent();
 radMenu1.Items.Add(new RadMenuItem("Item 1"));
 radMenu1.Items.Add(new RadMenuItem("Item 2"));
```

```
 radMenu1.Items.Add(new RadMenuItem("Item 3"));
 RadMenuItem item1 = (RadMenuItem)radMenu1.Items[0];
 item1.Items.Add(new RadMenuItem("Sub menu item 1"));
 item1.Items.Add(new RadMenuItem("Sub menu item 2"));
 ((RadMenuItem)item1.Items[0]).Click += new EventHandler(Form1_Click);
}
void Form1_Click(object sender, EventArgs e)
{
 RadMessageBox.Show("You clicked " + (sender as RadMenuItem).Text);
}
```

You can also add other item types like check box menu item:



Only certain element types can be nested at design-time, i.e. RadMenuItem, RadMenuButtonItem, RadMenuHeaderItem, RadMenuSeparatorItem and RadComboItem. All other elements can be nested by wrapping the in RadMenuItemBase. Follows an example of creating check box menu item:

[VB] Creating check box menu item

```
Public Class CustomCheckMenuItem
Inherits RadMenuItemBase
Private checkBoxElement As RadCheckBoxElement
Protected Overrides Sub CreateChildElements()
MyBase.CreateChildElements()
checkBoxElement = New RadCheckBoxElement()
checkBoxElement.Padding = New Padding(20, 0, 0, 0)
checkBoxElement.CheckMarkPrimitive.Margin = New Padding(0, 0, 5, 0)
Me.Children.Add(checkBoxElement)
End Sub
Public Overrides Property Text() As String
Get
Return checkBoxElement.Text
End Get
Set(ByVal value As String)
checkBoxElement.Text = value
End Set
End Property
Public Custom Event ToggleStateChanged As StateChangedEventHandler
```

```
AddHandler(ByVal value As StateChangedEventHandler)
AddHandler checkBoxElement.ToggleStateChanged, value
End AddHandler
RemoveHandler(ByVal value As StateChangedEventHandler)
RemoveHandler checkBoxElement.ToggleStateChanged, value
End RemoveHandler
RaiseEvent(ByVal sender As Object, ByVal e As StateChangedEventArgs)
End RaiseEvent
End Event
End Class
```

[C#] Creating check box menu item

```
public class CustomCheckMenuItem : RadMenuItemBase
{
private RadCheckBoxElement checkBoxElement;
protected override void CreateChildElements()
{
base.CreateChildElements();
checkBoxElement = new RadCheckBoxElement();
checkBoxElement.Padding = new Padding(20, 0, 0, 0);
checkBoxElement.CheckMarkPrimitive.Margin = new Padding(0, 0, 5, 0);
this.Children.Add(checkBoxElement);
}
public event StateChangedEventHandler ToggleStateChanged
{
add
{
checkBoxElement.ToggleStateChanged += value;
}
remove
{
checkBoxElement.ToggleStateChanged -= value;
}
}
public override string Text
{
get
{
return checkBoxElement.Text;
}
set
{
checkBoxElement.Text = value;
}
}
}
```

Now all you have to do is to create an instance of the custom item and add it to the Items collection of RadMenu or any other item inheriting from RadMenuItemBase

[VB] Adding the custom item to the Items collection

```
Public Sub New()
InitializeComponent()
Dim checkBoxItem As New CustomCheckMenuItem()
```

```
checkBoxItem.Text = "Check Box"
AddHandler checkBoxItem.ToggleStateChanged, AddressOf checkBoxItem_ToggleStateChanged
RadMenuItem1.Items.Add(checkBoxItem)
End Sub
Private Sub checkBoxItem_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs)
RadMessageBox.Show("Check box state has changed to " + args.ToggleState.ToString())
End Sub
```

[C#] Adding the custom item to the Items collection

```
public Form1()
{
InitializeComponent();
CustomCheckMenuItem checkBoxItem = new CustomCheckMenuItem();
checkBoxItem.Text = "Check Box";
checkBoxItem.ToggleStateChanged += new StateChangedEventHandler
(checkBoxItem_ToggleStateChanged);
radMenuItem1.Items.Add(checkBoxItem);
}
void checkBoxItem_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
RadMessageBox.Show("Check box state has changed to " + args.ToggleState);
}
```



You can find the complete source for this project at:

\Menu\<VB or CS>\MenuAddCheckBoxItem

Finding Items

RadMenu Items can be located by name through accessing the item through the indexer. The example below gets the top level "File" menu, then indexes the next level of Items to get the "Open" menu item, then removes the "Open" menu item.

[VB] Locating Items Using Indexer

```
Dim fileItem As RadMenuItem = DirectCast(radMenu1.Items("miFile"), RadMenuItem)
Dim openItem As RadMenuItem = DirectCast(fileItem.Items("miOpen"), RadMenuItem)
If openItem <> Nothing Then
 fileItem.Items.Remove(openItem)
End If
```

[C#] Locating Items Using Indexer

```
RadMenuItem fileItem = (RadMenuItem)radMenu1.Items["miFile"];
RadMenuItem openItem = (RadMenuItem)fileItem.Items["miOpen"];
if (openItem != null)
{
 fileItem.Items.Remove(openItem);
}
```

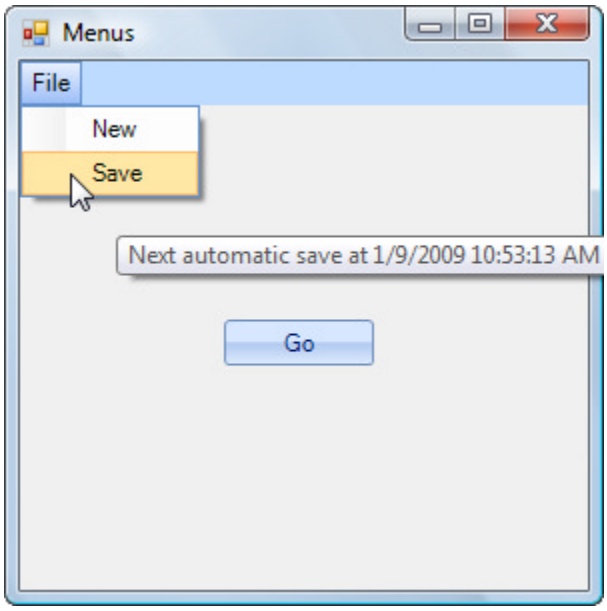
RadMenu Items collection also enables using LINQ clauses for finding menu items based on RadItem properties, such as the Text property. See the articles in MSDN (Microsoft Developer's Network) **Language Integrated Query (http://msdn.microsoft.com/en-us/library/bb397926.aspx)** page for more information about LINQ. Here is an example that shows finding menu items based on the Text property of the item, changing the tool tip

for an item and removing one of the menu items. The menu has a single top level item "File", with "New", "Open" and "Save" child items.



You can find the complete source for this project at:

\Menu\<VB or CS>\MenuFindItem



[VB] Finding and Removing Menu Items

```
Private Sub btnGo_Click(sender As Object, e As EventArgs)
 Dim fileItem As RadMenuItem = DirectCast(radMenu1.Items("miFile"), RadMenuItem)
 Dim openItem As RadMenuItem = DirectCast(fileItem.Items("miOpen"), RadMenuItem)
 If openItem <> Nothing Then
  fileItem.Items.Remove(openItem)
 End If

 ' finds the "Save" item from the "File" Items collection
 Dim saveItem As RadMenuItem = DirectCast(fileItem.Items.FirstOrDefault(Function(item As )
item.Name.Equals("miSave")), RadMenuItem)
 If saveItem <> Nothing Then
  saveItem.ToolTipText = "Next automatic save at " + DateTime.Now.AddHours(0.5)
 End If
End Sub
```

[C#] Finding and Removing Menu Items

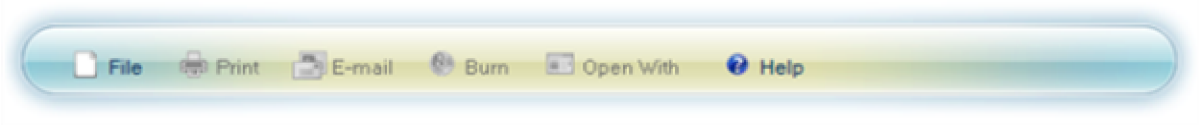
```
private void btnGo_Click(object sender, EventArgs e)
{
 RadMenuItem fileItem = (RadMenuItem)radMenu1.Items["miFile"];
 RadMenuItem openItem = (RadMenuItem)fileItem.Items["miOpen"];
 if (openItem != null)
 {
   fileItem.Items.Remove(openItem);
 }

 // finds the "Save" item from the "File" Items collection
 RadMenuItem saveItem =
```

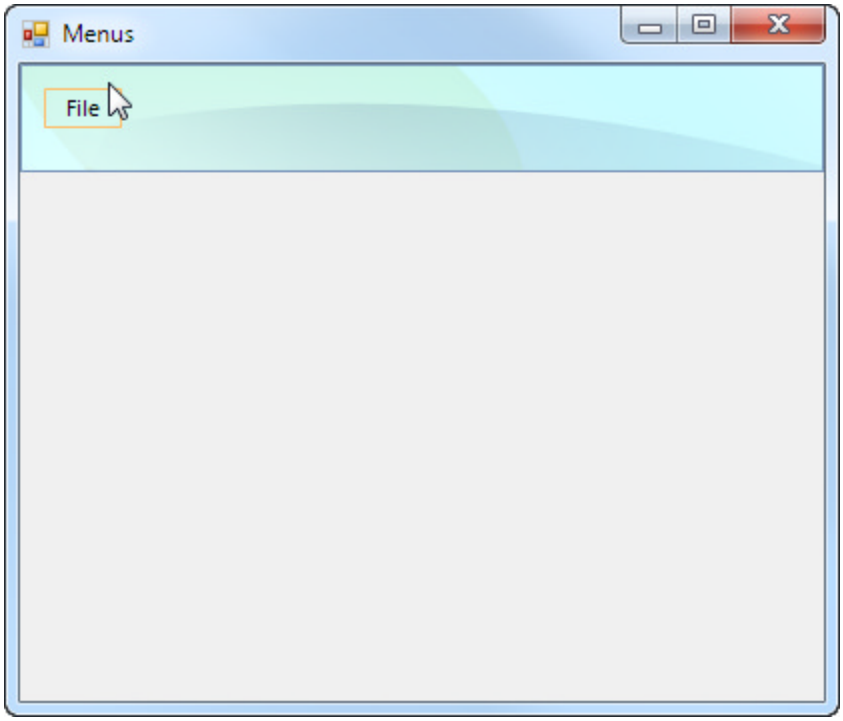
```
    (RadMenuItem)fileItem.Items.FirstOrDefault(item => item.Name.Equals("miSave"));
  if (saveItem != null)
  {
    saveItem.ToolTipText = "Next automatic save at " + DateTime.Now.AddHours(0.5);
  }
}
```

Background Images

You can place a background image underneath your menu. It can even be an irregular image as shown in the in example demos that ship with the product.



In the example below a RadPanel has the BackGroundImage property assigned and the RadMenu sits within the middle of the panel space. To make the image show through, hide the fill primitive of the menu and the "File" menu item.



[VB] Setting a Transparent Background

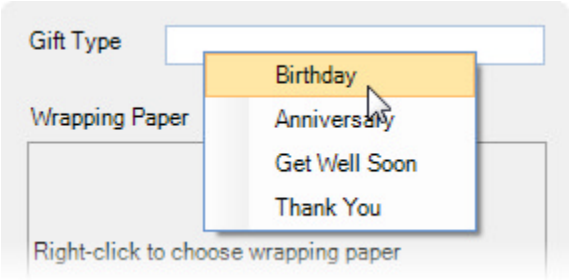
```
radMenu1.MenuElement.Children(1).Visibility = ElementVisibility.Hidden
radMenu1.Items(0).Children(0).Visibility = ElementVisibility.Hidden
```

[C#] Setting a Transparent Background

```
radMenu1.MenuElement.Children[1].Visibility = ElementVisibility.Hidden;
radMenu1.Items[0].Children[0].Visibility = ElementVisibility.Hidden;
```

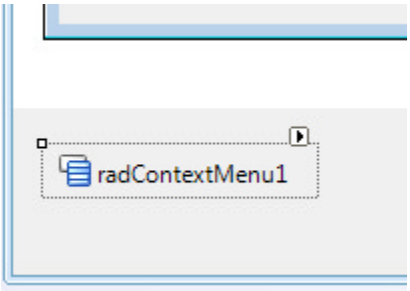
Context Menus

With context menus you can pop up a context-specific set of choices anywhere in the user interface, either attaching the menu to a control or programmatically in any location on the form that suits you. Context menus can be attached to RadControls and any other System.Windows.Forms.Control descendant for that matter.

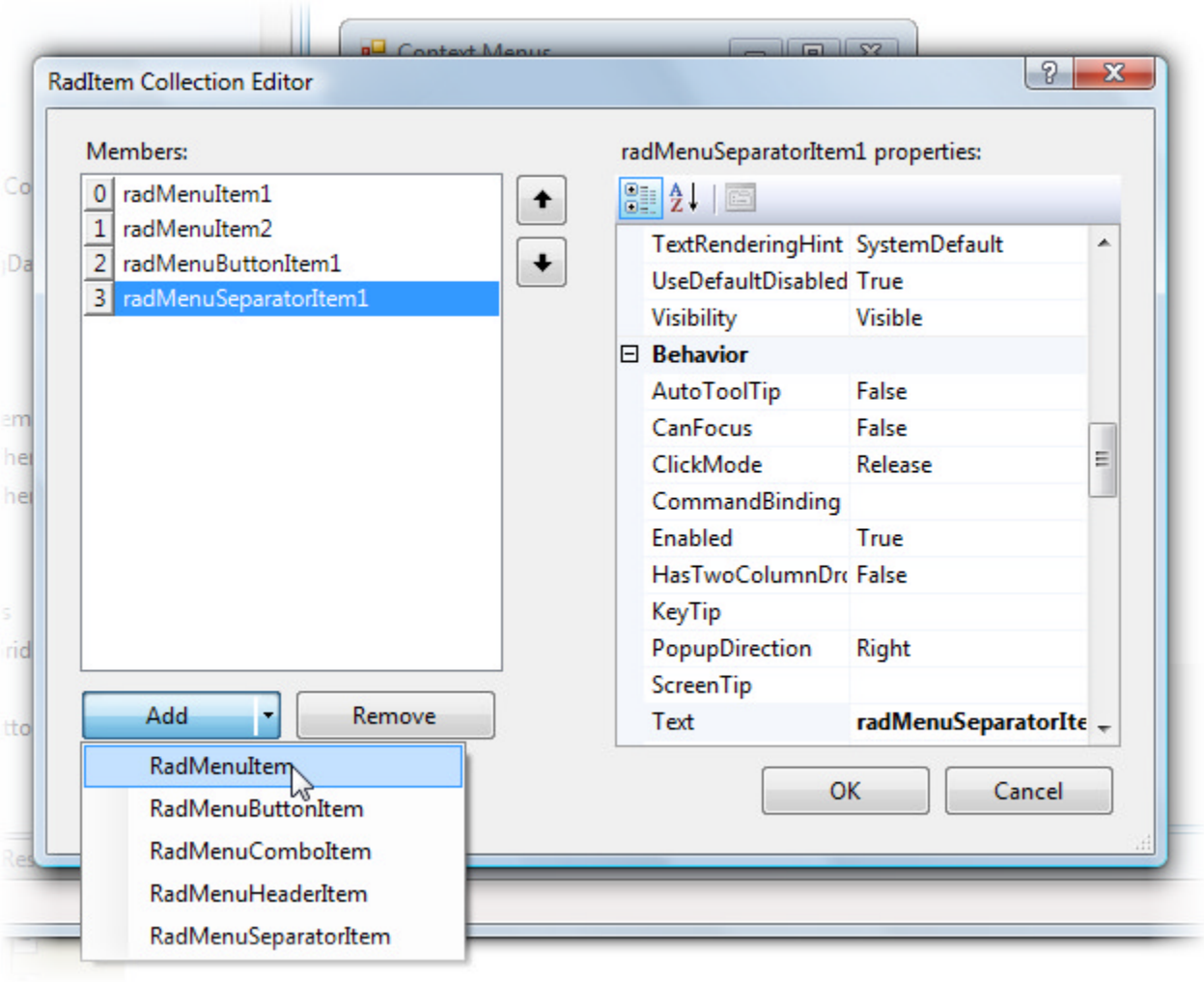


Using RadContextMenu

To get started with context menus, drop a **RadContextMenu** on the form. This is a non-visual component that will be placed below the form in the component tray.

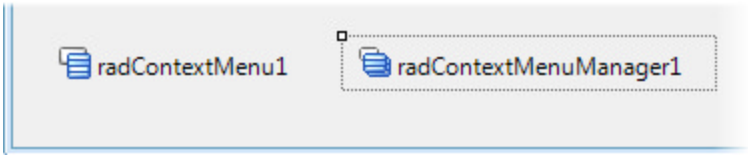


Add menu entries by adding to the Items collection at design time or programmatically. Although there is no Menu Designer, the Smart Tag "Edit Menu Items" will still bring up the RadItem Collection Editor where you can add/edit/delete items just as you can with RadMenu.



Using RadContextMenuManager

Standard Winforms visual controls all have a ContextMenuStrip property that is *not* used by RadContextMenu. Instead, each control that you want to associate a context menu with needs to have a **RadContextMenu** property. To get this property to appear, drop a **RadContextMenuManager** component onto the component tray to automatically associate a RadContextMenu property with every control on the form. RadContextMenuManager is a non-visual component and has no significant properties. The sole purpose of RadContextMenuManager is to populate controls on the form with the RadContextMenu property.



Showing a Context Menu at a Specific Location

Another way you can make the context menu appear is to call the RadContextMenu **Show()** method to have the menu appear in a specific location. A common use for this would be to have the menu appear in response to a right-click. For example, you can handle the form's MouseDown event looking for right button clicks and feed the mouse coordinates to the menu's Show() method.

[VB] Showing the Context Menu

```
' respond to right mouse clicks in the text box
Sub TextBoxElement_MouseDown(sender As Object, e As MouseEventArgs)
 If e.Button = MouseButtons.Right Then
  cmGiftType.Show(TryCast(sender, RadTextBoxElement), e.X, e.Y)
 End If
End Sub
```

[C#] Showing the Context Menu

```
// respond to right mouse clicks in the text box
void TextBoxElement_MouseDown(object sender, MouseEventArgs e)
{
 if (e.Button == MouseButtons.Right)
 {
   cmGiftType.Show(sender as RadTextBoxElement, e.X, e.Y);
 }
}
```



A PointToScreen() method to convert coordinates is not needed when using the specialized Show() method of RadContextMenu: Show(Control control, int x, int y). The Show is automatically displayed in relation to the control.

Context Menus and TextBoxes

Can you attach a context menu to a RadTextBox or control that hosts a Microsoft TextBox? Yes and no... Remember that the MS TextBox has its own standard context menu strip that wants to kick in and show the usual Cut, Copy and Paste options. How do you erradicate the standard menu and put in your own? The context menu is a feature of the hosted Microsoft TextBox and it can be stopped from showing using a work around: assign an empty ContextMenuStrip to the text box:

[VB] Assigning an Empty ContextMenuStrip

```
radTextBox1.TextBoxElement.TextBoxItem.HostedControl.ContextMenuStrip = New ContextMenuStrip
()
```

[C#] Assigning an Empty ContextMenuStrip

```
radTextBox1.TextBoxElement.TextBoxItem.HostedControl.ContextMenuStrip = new ContextMenuStrip
();
```

While that kills the built-in Microsoft context menu, when you right-click the RadTextBox, the context menu only appears when you click the very edges. The TextBoxElement portion of the control doesn't respond. To fix this, add a MouseDown event handler that traverses to the TextBoxElement portion of the control. Now the code will look something like the example below:

[VB] Hooking up a MouseDown Event Handler

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 tbGiftType.TextBoxElement.TextBoxItem.HostedControl.ContextMenuStrip = New ContextMenuStrip
()
 tbGiftType.TextBoxElement.MouseDown += New MouseEventHandler(TextBoxElement_MouseDown)
 LoadWrappingContextMenu()
End Sub
Sub TextBoxElement_MouseDown(sender As Object, e As MouseEventArgs)
 If e.Button = MouseButtons.Right Then
  Dim p As Point = (TryCast(sender, RadTextBoxElement)).PointToScreen(e.Location)
  cmGiftType.Show(p.X, p.Y)
 End If
End Sub
```

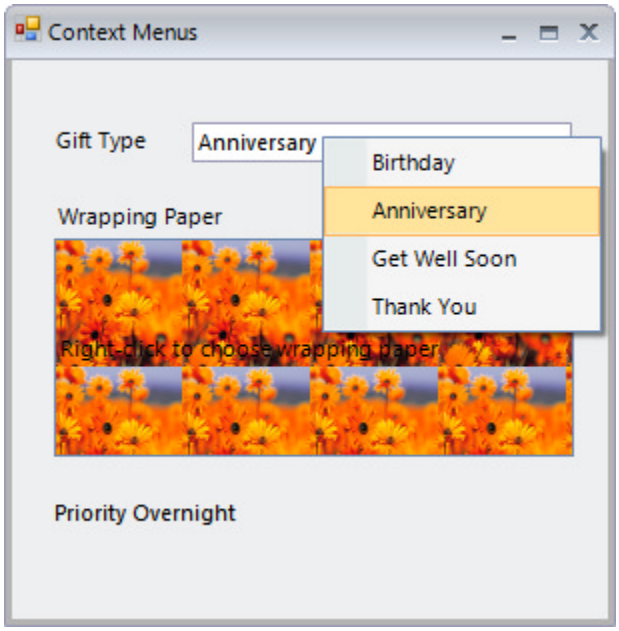
[C#] Hooking up a MouseDown Event Handler

```
private void Form1_Load(object sender, EventArgs e)
{
 tbGiftType.TextBoxElement.TextBoxItem.HostedControl.ContextMenuStrip =
  new ContextMenuStrip();
 tbGiftType.TextBoxElement.MouseDown += new MouseEventHandler(TextBoxElement_MouseDown);

 LoadWrappingContextMenu();
}
void TextBoxElement_MouseDown(object sender, MouseEventArgs e)
{
 if (e.Button == MouseButtons.Right)
 {
   Point p = (sender as RadTextBoxElement).PointToScreen(e.Location);
   cmGiftType.Show(p.X, p.Y);
 }
}
```

Context Menus Walk Through

This walk through demonstrates multiple methods of displaying a RadContextMenu: by associating with a RadControl, associating with a standard control and displaying programmatically at a specific location. This demonstration also shows how to make the context menu appear over a text box.



Preparing the Project



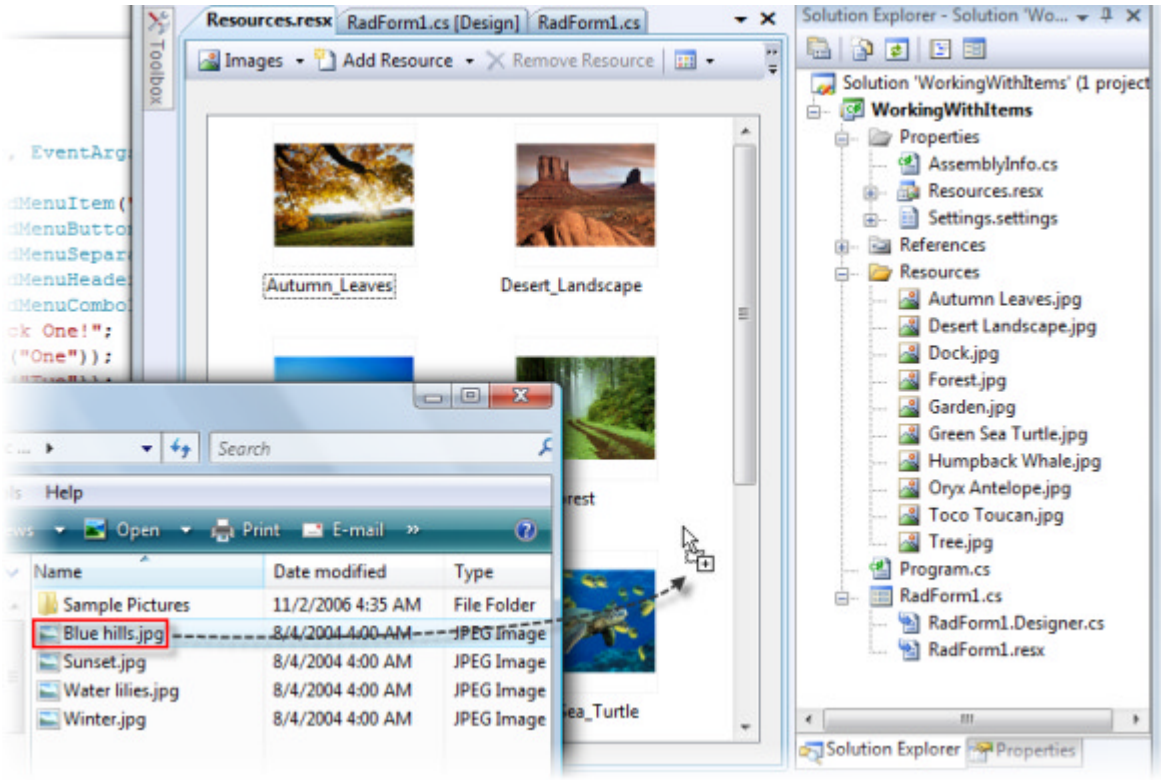
You can find the complete source for this project at:

\Menus\<VB|CS>\ContextMenu

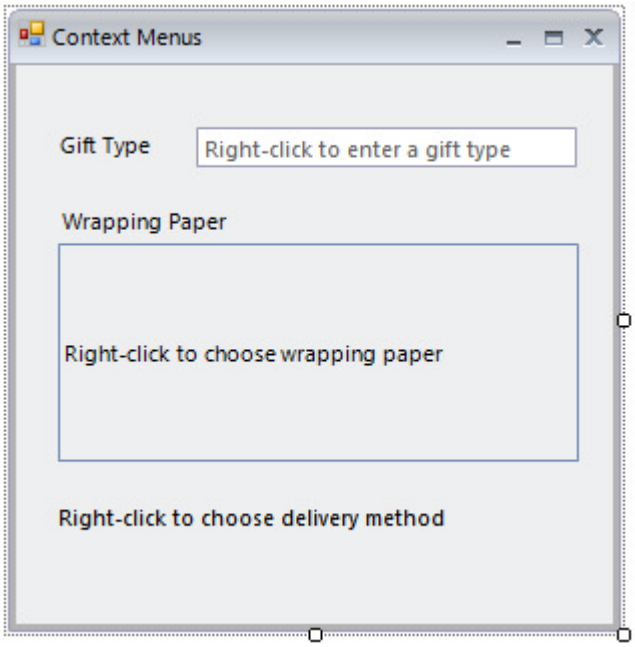
1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.

Adding Controls to the RadForm

1. Drop a **Office2007SilverTheme, RadContextMenuManager** and three **RadContextMenu** components on the form. Name the three RadContextMenu components "cmGiftType", "cmWrapping" and "cmDeliveryMethod".
2. Select **Edit Menu Items** from the Smart Tag of "cmGiftType". Add four **RadMenuItems** with the following properties:
    - **Name** = "miGiftType1", **Text** = "Birthday".
    - **Name** = "miGiftType2", **Text** = "Anniversary".
    - **Name** = "miGiftType3", **Text** = "Thank You".
    - **Name** = "miGiftType4, **Text** = "Get Well Soon".
3. Select **Edit Menu Items** from the Smart Tag of "cmDeliveryMethod". Add four **RadMenuButtonItems** with the following properties:
    - **Name** = "mbiDeliveryMethod1", **Text** = "Priority Overnight".
    - **Name** = "mbiDeliveryMethod2", **Text** = "Standard Overnight".
    - **Name** = "mbiDeliveryMethod3", **Text** = "2 Day".
    - **Name** = "mbiDeliveryMethod4, **Text** = "3 Day".
4. In the Solution Explorer, navigate to the Properties folder, Resources.resx item. Double-click Resources.resx to open the resources editor.
5. Drag an assortment of jpg or png images from the file explorer into the resources editor.



6. On the form's surface drop the following controls and set properties. The layout should look something like the screenshot below:



- **RadLabel**: Text = "Gift Type", ThemeName = Office2007Silver.
- **RadTextBox**: Name = "tbGiftType", NullText = "Right-click to enter a gift type", ThemeName = Office2007Silver.
- **RadLabel**: Text = "Wrapping", ThemeName = Office2007Silver.
- **RadPanel**: Name = "pnlWrappingPaper", Text = "Right-click to choose wrapping paper", ThemeName = Office2007Silver.
- **Label**: Name = "lblDeliveryMethod", Text = "Right-click to chooses delivery method"

7. Double click the form to create a form load event handler. Add the code below to disable the "gift type" Microsoft context menu, hook up a new mouse down event handler and call helper method LoadWrappingContextMenu(). The mouse down event handler and LoadWrappingContextMenu() will be written later.

[VB] Handling the Form Load Event

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 tbGiftType.TextBoxElement.TextBoxItem.HostedControl.ContextMenuStrip = New ContextMenuSt
()
 tbGiftType.TextBoxElement.MouseDown += New MouseEventHandler(TextBoxElement_MouseDown)
 LoadWrappingContextMenu()
End Sub
```

[C#] Handling the Form Load Event

```
private void RadForm1_Load(object sender, EventArgs e)
{
 tbGiftType.TextBoxElement.TextBoxItem.HostedControl.ContextMenuStrip =
   new ContextMenuStrip();
 tbGiftType.TextBoxElement.MouseDown += new MouseEventHandler(TextBoxElement_MouseDown);
 LoadWrappingContextMenu();
}
```

8. Make sure you have the following references in your "Imports" (VB) or "uses" (C#) section of your code:

[VB] Adding Namespace References

```
Imports System
' Supports IDictionaryEnumerator
Imports System.Collections
Imports System.Drawing
' Supports CultureInfo
Imports System.Globalization
' Supports GetFileNameWithoutExtension
Imports System.IO
' Supports ResourceSet, Resources
Imports System.Resources
Imports System.Windows.Forms
Imports Telerik.WinControls.UI
```

[C#] Adding Namespace References

```
using System;
// Supports IDictionaryEnumerator
using System.Collections;
using System.Drawing;
// Supports CultureInfo
using System.Globalization;
// Supports GetFileNameWithoutExtension
using System.IO;
// Supports ResourceSet, Resources
using System.Resources;
using System.Windows.Forms;
using Telerik.WinControls.UI;
```

9. Add a helper method to get the resourced images and create menu items:

[VB] Adding the LoadWrappingContextMenu() Method

```
' enumerate the image resources, creating menu items
Private Sub LoadWrappingContextMenu()
 Dim resourceSet As ResourceSet = Properties.Resources.ResourceManager.GetResourceSet
(CultureInfo.CurrentCulture, True, True)
 Dim enumerator As IDictionaryEnumerator = resourceSet.GetEnumerator()
 While enumerator.MoveNext()
  Dim key As String = (DirectCast(enumerator.Key, String)).Replace("_"C, " "C)
  Dim image As Image = DirectCast(enumerator.Value, Image)
  Dim item As New RadMenuItem()
  item.Text = Path.GetFileNameWithoutExtension(key)
  item.Image = image.GetThumbnailImage(64, 64, Nothing, New IntPtr())
  item.Click += New EventHandler(item_Click)
  cmWrapping.Items.Add(item)
 End While
End Sub
```

[C#] Adding the LoadWrappingContextMenu() Method

```
// enumerate the image resources, creating menu items
private void LoadWrappingContextMenu()
{
 ResourceSet resourceSet = Properties.Resources.ResourceManager.GetResourceSet(
   CultureInfo.CurrentCulture, true, true);
 IDictionaryEnumerator enumerator = resourceSet.GetEnumerator();
 while (enumerator.MoveNext())
 {
```

```
    string key = ((string)enumerator.Key).Replace('_', ' ');
    Image image = (Image)enumerator.Value;
    RadMenuItem item = new RadMenuItem();
    item.Text = Path.GetFileNameWithoutExtension(key);
    item.Image = image.GetThumbnailImage(64, 64, null, new IntPtr());
    item.Click += new EventHandler(item_Click);
    cmWrapping.Items.Add(item);
  }
}
```

10. Add a mouse down event handler to display the context menu for the "Gift Type" text box.

[VB] Handling the Mouse Down Event

```
Sub TextBoxElement_MouseDown(sender As Object, e As MouseEventArgs)
 If e.Button = MouseButtons.Right Then
  Dim p As Point = (TryCast(sender, RadTextBoxElement)).PointToScreen(e.Location)
  cmGiftType.Show(p.X, p.Y)
 End If
End Sub
```

[C#] Handling the Mouse Down Event

```
void TextBoxElement_MouseDown(object sender, MouseEventArgs e)
{
 if (e.Button == MouseButtons.Right)
 {
   Point p = (sender as RadTextBoxElement).PointToScreen(e.Location);
   cmGiftType.Show(p.X, p.Y);
 }
}
```

11. Add event handlers that will be fire for each menu item's Click event.

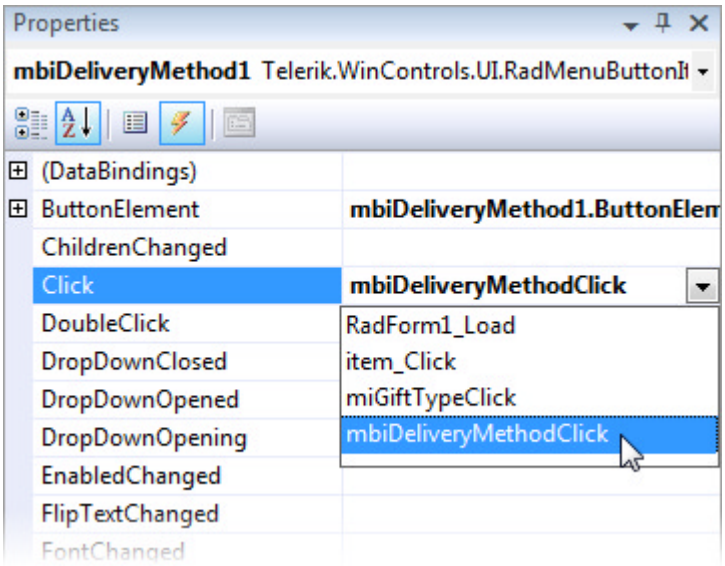
[VB] Handling the Menu Item Click Events

```
Sub item_Click(sender As Object, e As EventArgs)
 pnlWrappingPaper.BackgroundImage = (TryCast(sender, RadMenuItem)).Image
End Sub
Private Sub miGiftTypeClick(sender As Object, e As EventArgs)
 tbGiftType.Text = (TryCast(sender, RadMenuItem)).Text
End Sub
Private Sub mbiDeliveryMethodClick(sender As Object, e As EventArgs)
 lblDeliveryMethod.Text = (TryCast(sender, RadMenuButtonItem)).Text
End Sub
```

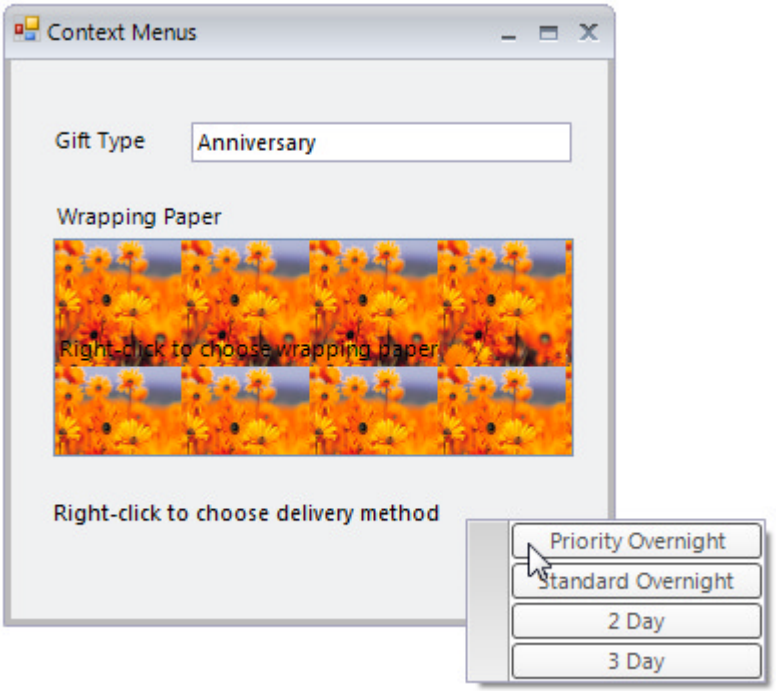
[C#] Handling the Menu Item Click Events

```
void item_Click(object sender, EventArgs e)
{
 pnlWrappingPaper.BackgroundImage = (sender as RadMenuItem).Image;
}
private void miGiftTypeClick(object sender, EventArgs e)
{
 tbGiftType.Text = (sender as RadMenuItem).Text;
}
private void mbiDeliveryMethodClick(object sender, EventArgs e)
{
 lblDeliveryMethod.Text = (sender as RadMenuButtonItem).Text;
}
```

12. In the Properties window, events tab, locate "miDeliveryMethod1" and set the Click event to "miDeliveryMethodClick". Repeat this step for "miDeliveryMethod2", "miDeliveryMethod3" and "miDeliveryMethod4".



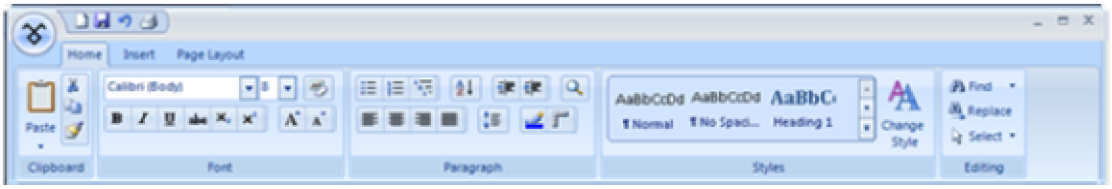
13. In the Properties window, events tab, locate "miGiftType1" and set the Click event to "miGiftTypeClick". Repeat this step for "miGiftType2", "miGiftType3" and "miGiftType4".
14. Press **Ctl-F5** to run the application. Right-click each of the prompts to see the context menus.



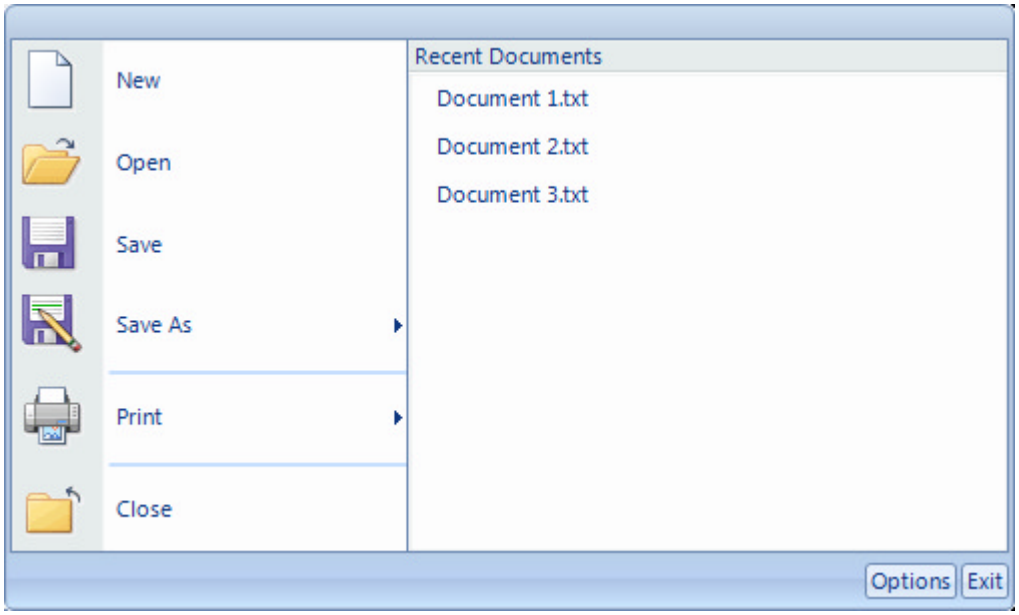
6.6 RadRibbonBar and RadApplicationMenu

RadRibbonBar combines the functions of menus, tab strips and toolbars to make an interface similar to those introduced in Microsoft Office 2007. With RadRibbonBar you can organize all of the functionality of your

application into a single compact ribbon. When resized, the ribbon bar efficiently shows more or less detail as space becomes available. If you use a Multiple Document Interface (MDI) layout, the ribbon bar will automatically manage the child forms.

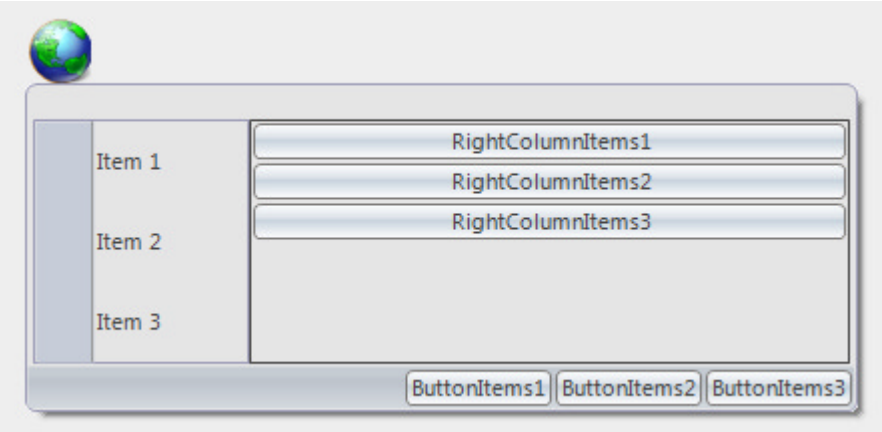


RadApplicationMenu is the Telerik counterpart of the "application menu" that displays controls used to perform actions on entire documents and forms, such as Save and Print. It also provides a list of recent documents, access to form options, and the ability to exit the form or application.



RadApplicationMenu

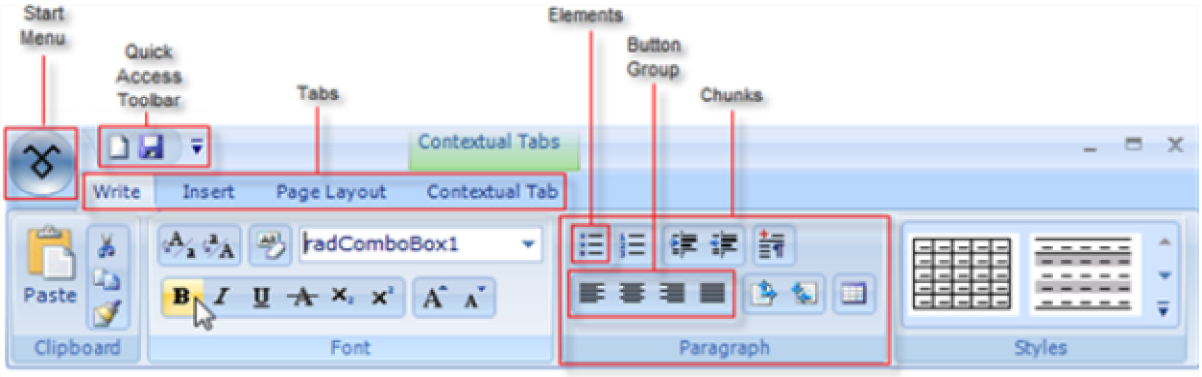
RadApplicationMenu is essentially a RadDropDownButton and has many of the same properties and design time interface as a RadMenu, i.e. same Smart Tag items, RadItem Collection Editor and the same RadItem types that may be added to the collection. In the case of RadApplicationMenu, there are three RadItem collections: **Items**, **RightColumnItems**, and **ButtonItems.** The **RightColumnWidth** property allows you to reserve extra screen real estate. RadApplicationMenu also has a **DisplayStyle** property that may be Image (default), None, Text and ImageAndText. Set the **Image** property to include your logo or whatever icon you want to use. The screenshot below shows all three collections populated and the Image set to a "world globe" icon.



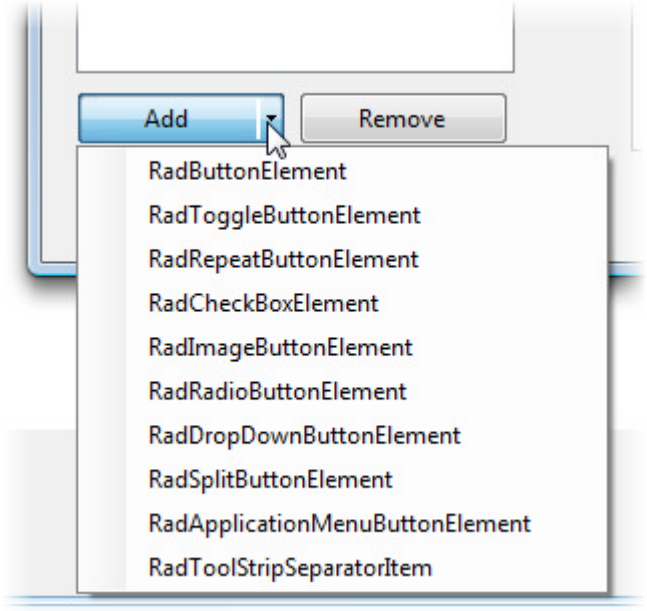
RadRibbonBar

RadRibbonBar Structure

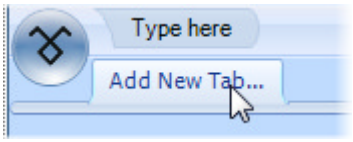
The ribbon bar has a complex structure, but can be broken down into these main pieces. The screenshot below should help you visually place where the pieces go.



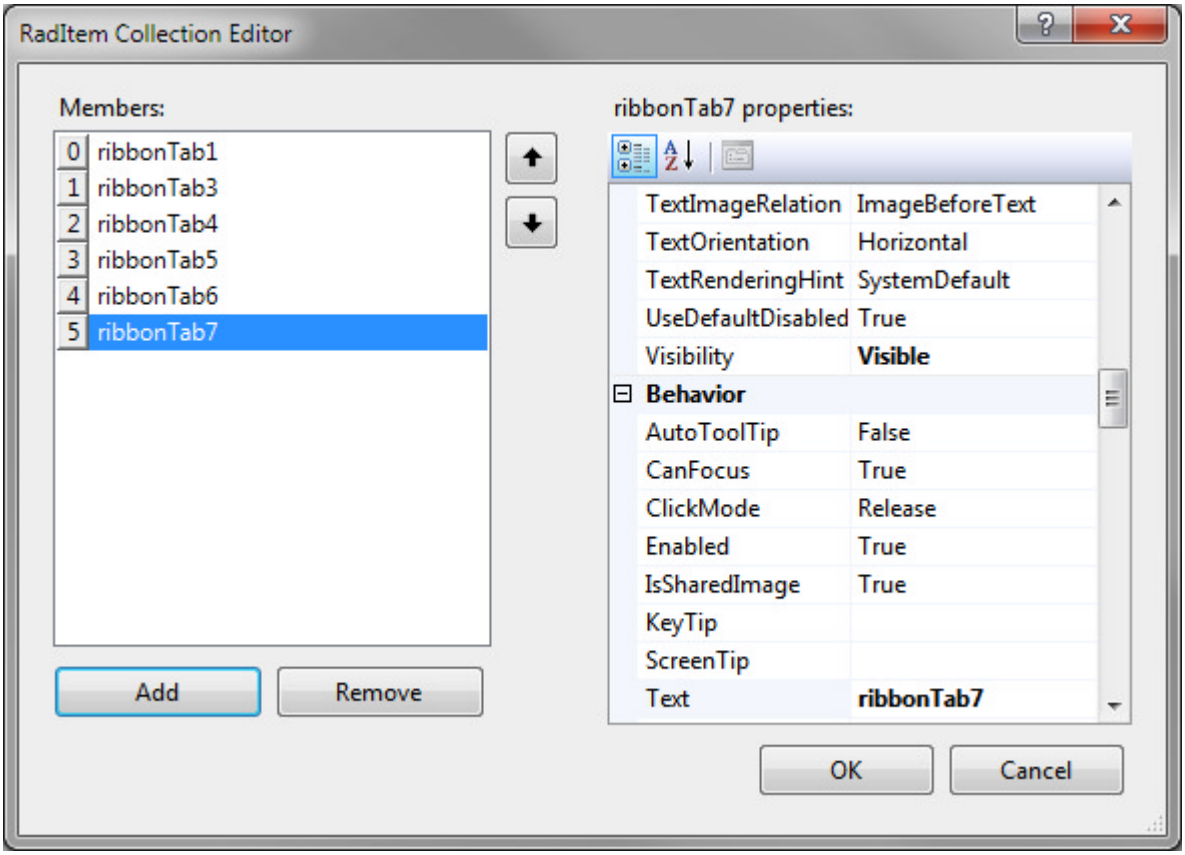
- **Start menu**: The Start Menu is a RadApplicationMenu described earlier in this chapter.
- **Quick Access Tool Bar**: This contains a few often used items that can be any of the button types shown in the screenshot below or a separator item. Use the **QuickAccessToolBarItems** collection to populate the tool bar and toggle **QuickAccessToolbarBelowRibbon** on if the tool bar should appear below the tabbed area.



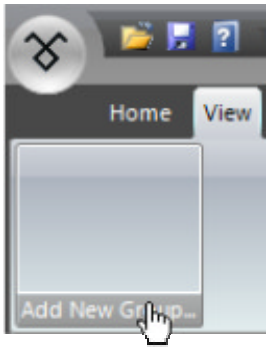
- **Tabs**: The tabs are the first level of organization in the main part of RadRibbonBar. Tabs are typically used to split the functionality of an application into major areas. Each tab is a member of the ribbon bar's **CommandTabs** collection. You can create tabs by clicking the **Add New Tab...** prompt in the designer:



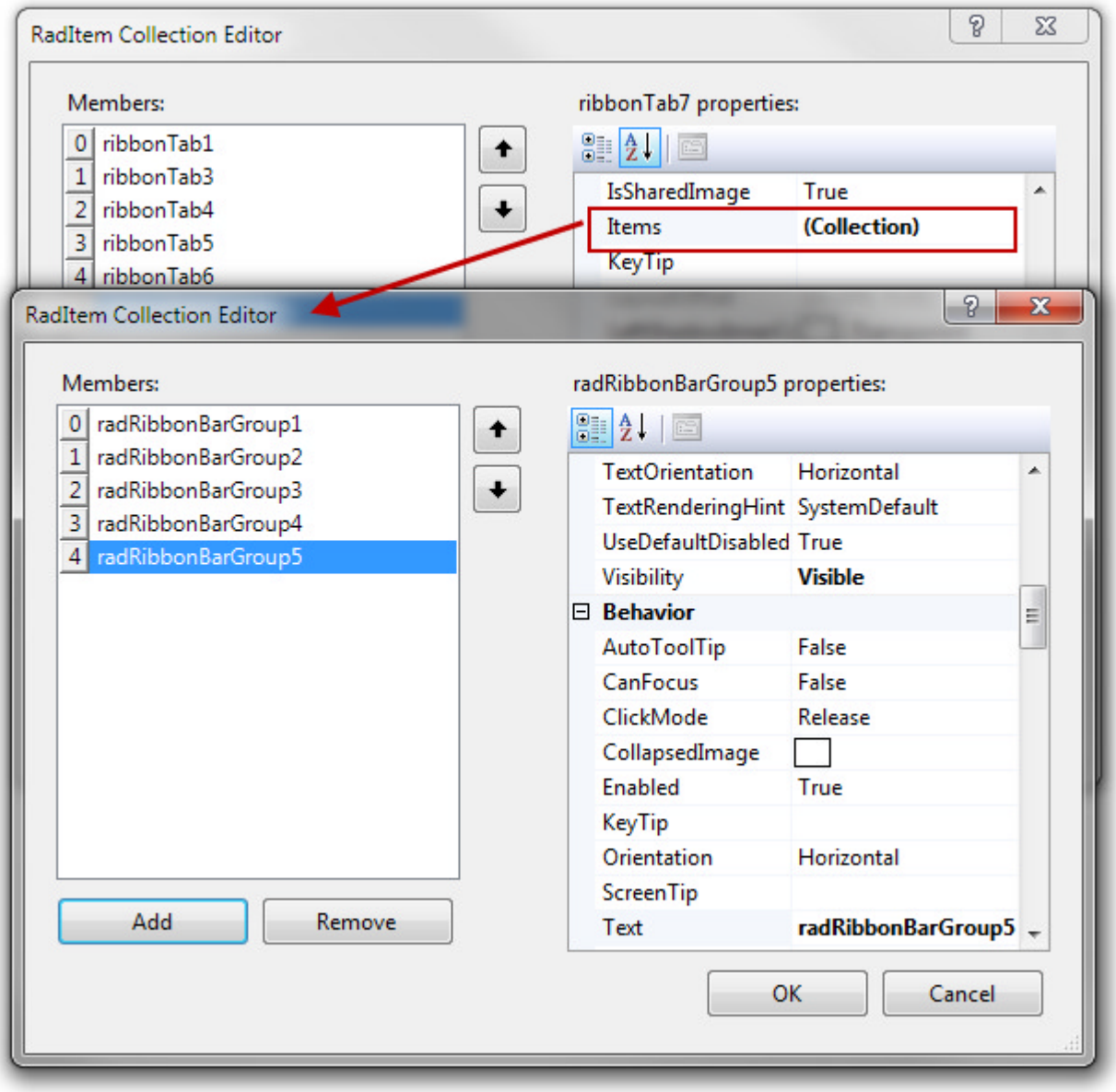
...or clicking the Add Tab Smart Tag command or editing the ribbon bar's **CommandTabs** collection:



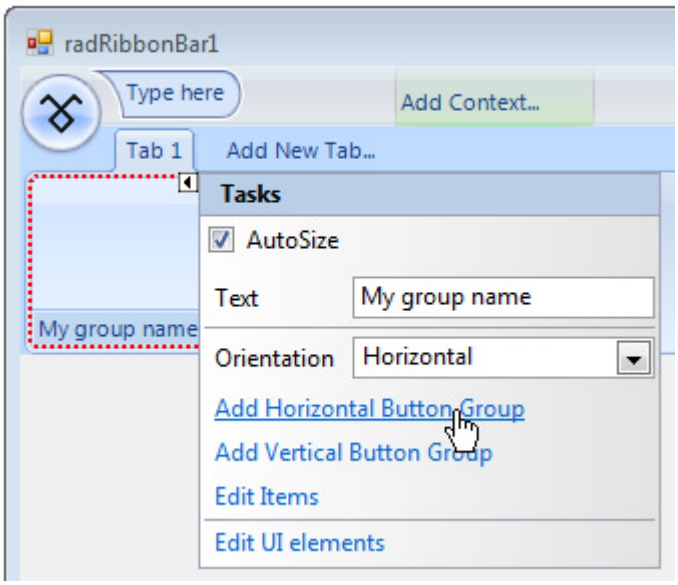
- **Groups**: Each tab hosts one or more groups. A groups is a container for other containers and individual functional elements. Groups can individually collapse if there is not enough room to display them all. You can add groups by clicking the **Add New Group...** prompt in the designer:



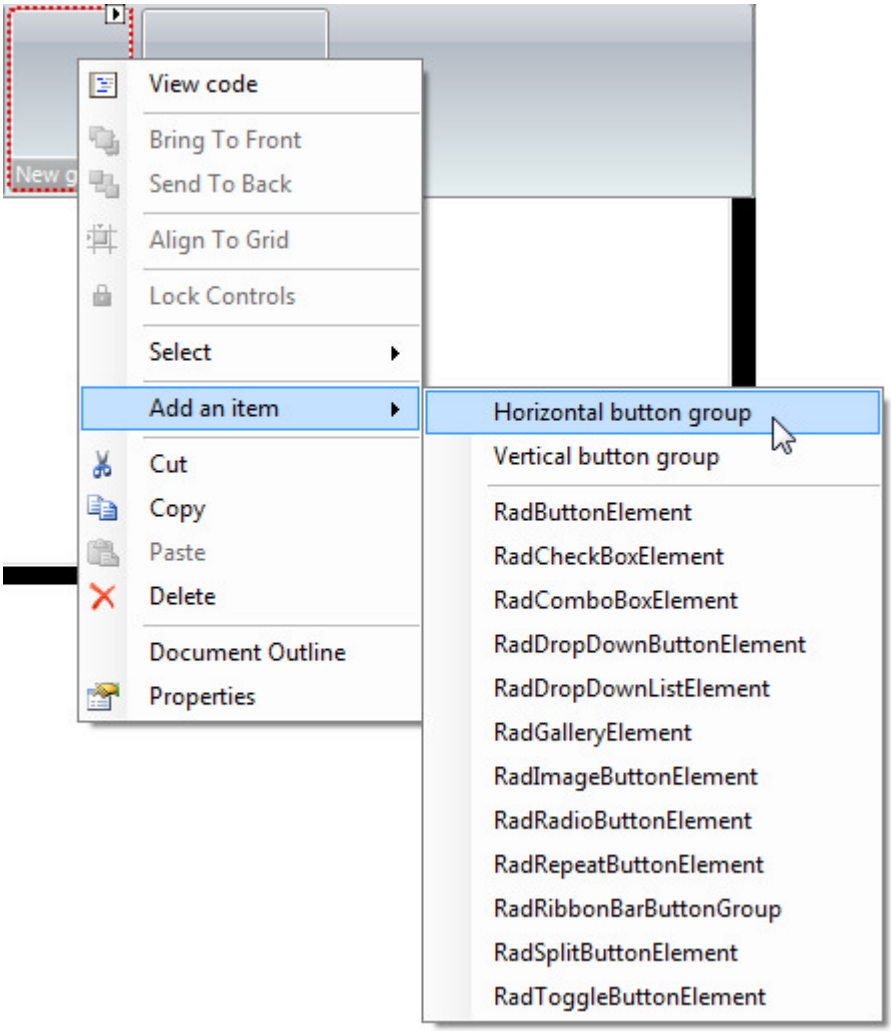
Each CommandTab contains an object representing a tab, but also contains an Items collection that contain RadRibbonBarGroup objects. So you can also add groups by using the tab's collection editor:



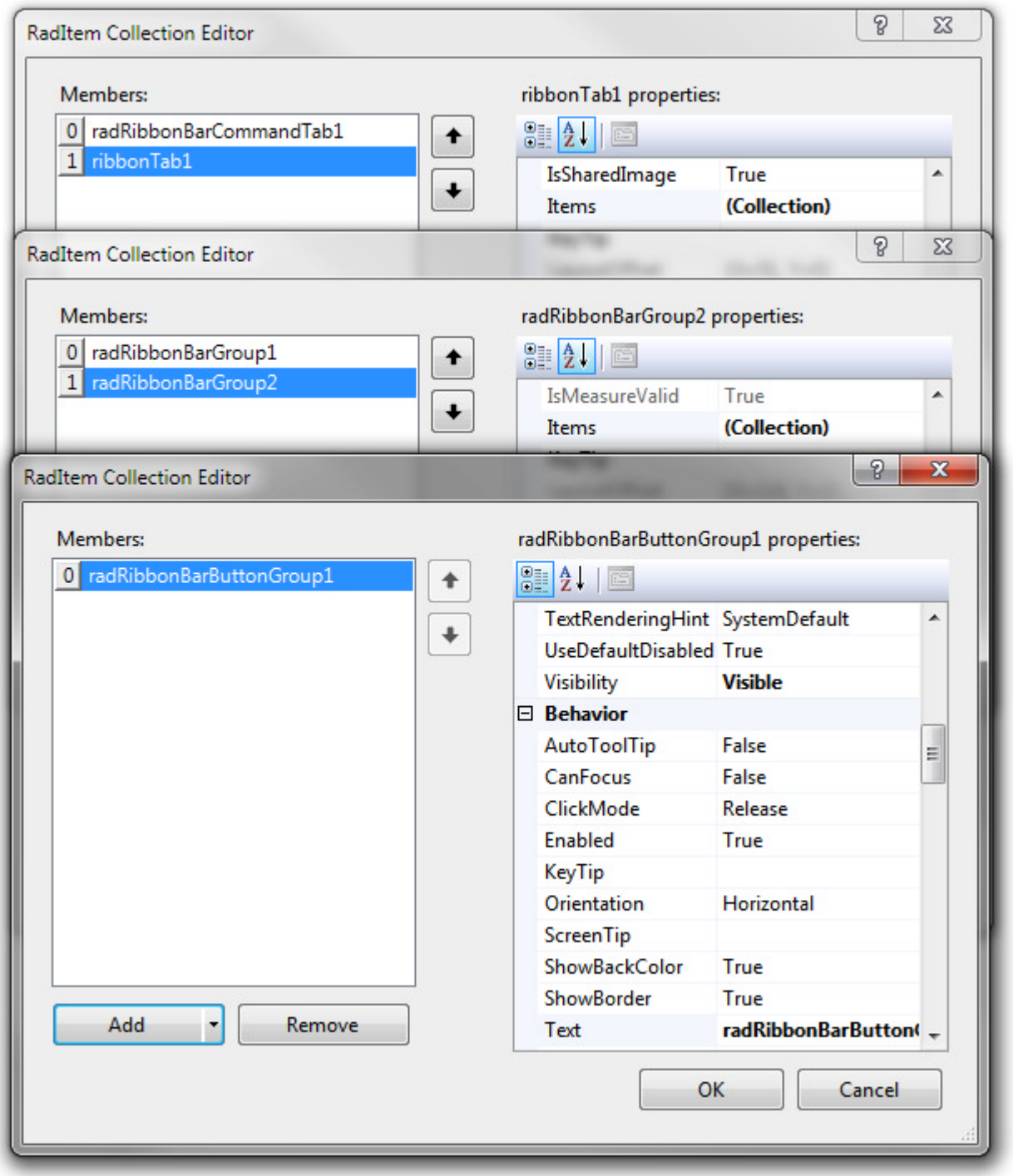
- **Button Groups**: Groups may contain one or more button groups. A button group is a container of nested button groups or individual functional elements. Button group **Orientation** can be Horizontal (containing a horizontal row of elements) or Veritical (containing a vertical column of elements). You can add button groups from the group Smart Tag...



...from the group's context menu **Add an item** option:



...and by editing the group's **Items** collection:

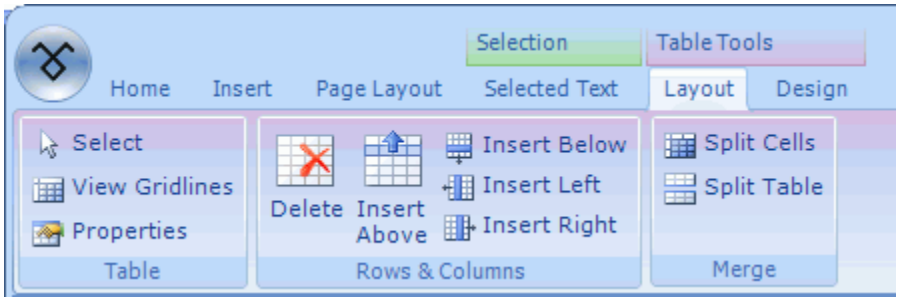


- **Elements**: Elements are individual functional pieces of the RadRibbonBar. Elements can be contained directly within groups or within button groups. Groups and button groups can contain a variety of elements. You can add elements using the context menu or Items collection of either a group or button group.

There are two other special ways to organize user choices, Context Tabs and Galleries.

Contextual Tabs

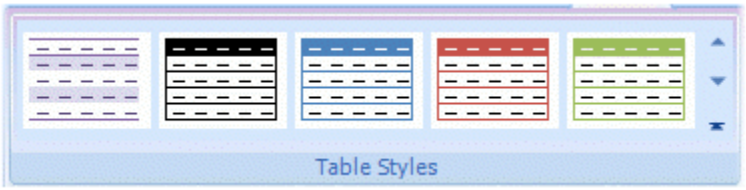
Contextual tabs (also called tab groups) provide a way to organize related tabs. They are often used to group tabs that apply to a specific object in your application.



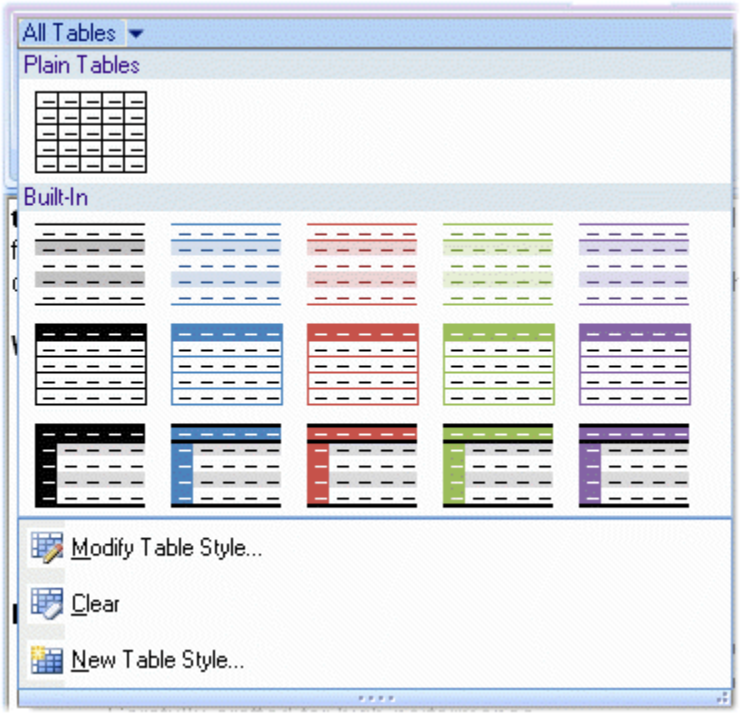
In the screenshot above, "Selection" and "Table Tools" are contextual tabs. Their width indicates the tabs that they contain, so that the Selection contextual tab includes the Selected Text tab, and the Table Tools contextual tab includes the Layout and Design tabs. Color is also used to unify a contextual tab with its contained group's. The tabs that belong to contextual tabs are always placed to the right of other tabs on the Telerik RadRibbonBar. You can create new context tab groups using the **Add New Group...** prompt in the designer, or editing the **ContextualTabGroups** property of the ribbon bar control. To add a tab to the group, drag the tab to the group and drop it there.

Galleries

A gallery is a special type of element that is designed to allow the user to select visually from among a number of choices. A gallery may be displayed in either collapsed or expanded view. In its default collapsed view, a gallery shows a single row of choices, as well as up and down arrows for scrolling to other rows of choices and a drop-down arrow for switching to expanded view.



In its expanded view, a gallery shows all of its choices at one time, a filter selection bar at the top, and tools at the bottom.

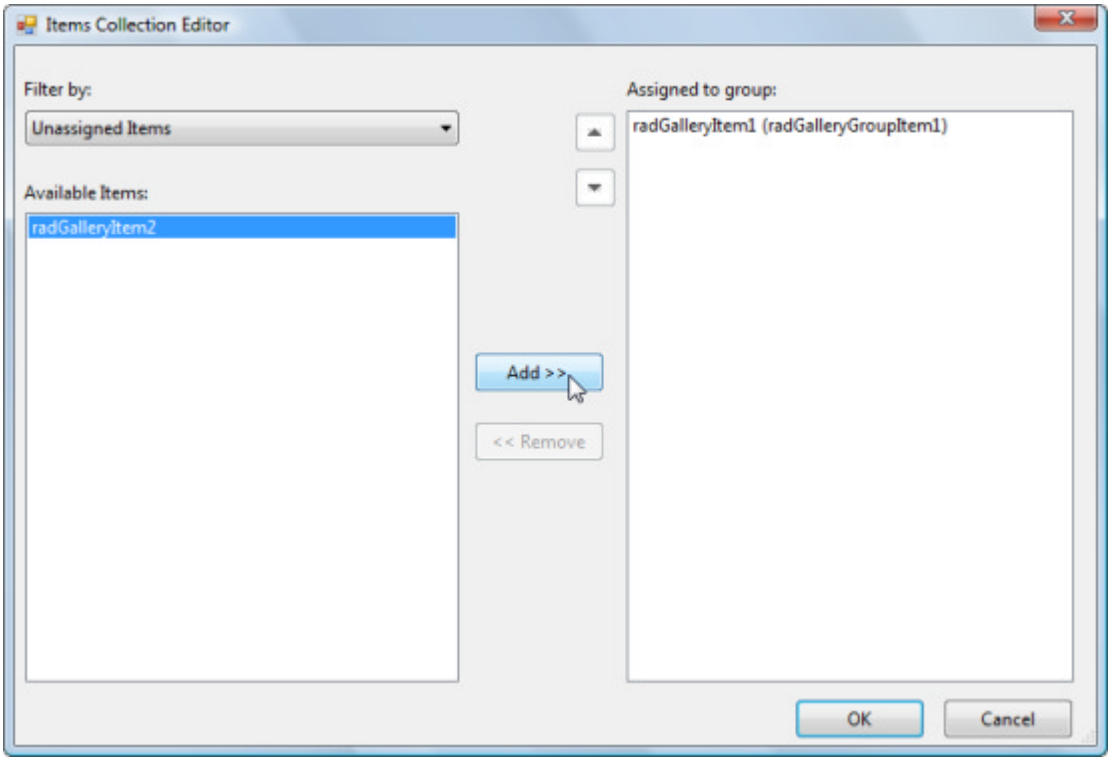


To create a gallery, right-click a group on a RadRibbonBar control in design view. Select **Add an Item**, and then **RadGalleryElement**. To add items to the gallery, select the RadGalleryElement control and edit the **Items** collection. You can set the text and an image for each RadGalleryItem in the collection.

Adding Groups

Groups organize the items within a gallery into distinct sections. In the screenshot of the expanded gallery above, Plain Tables and Built-In are groups. To add a group:

1. Select the RadGalleryElement control.
2. Click in the **Groups** property, and then click the ellipsis button.
3. In the RadItem Collection Editor, click **Add** to add a new group.
4. Set the **Text** property to the caption of the new group.
5. Click in the **Items** property of the RadGalleryGroupItem, and then click the ellipsis button.
6. Click **Add** to add an item to the group. Each time you click Add, one of the items that you already created will be added to the group.



7. Click **OK** when you are done adjusting item membership in the group.
8. Click **OK** when you are done creating groups.

Creating Filters

Filters let you select which groups to display. For example, the All Tables filter in the screenshot above includes the Plain Tables group and the Built-In group.

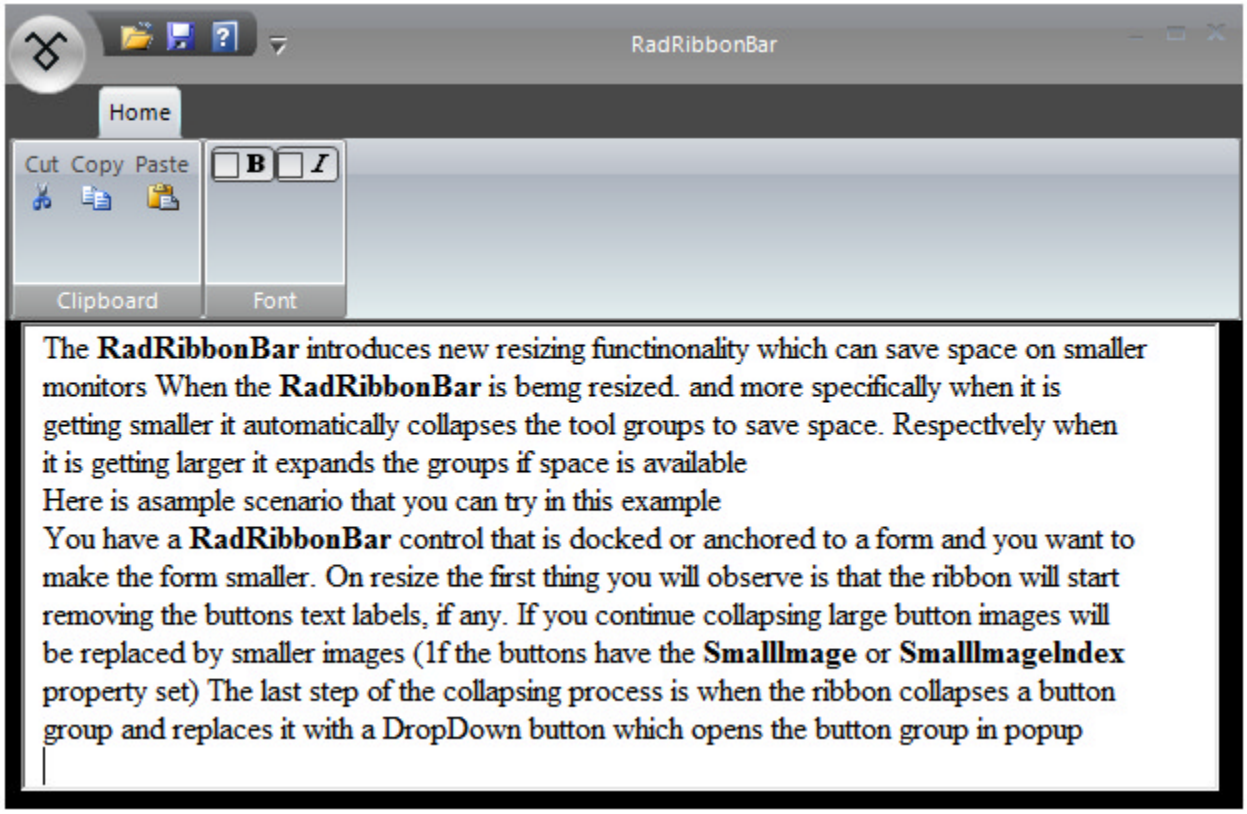
1. Select the RadGalleryElement control.
2. Click in the **Filters** property, and then click the ellipsis button.
3. In the RadItem Collection Editor, click **Add** to add a new filter.
4. Set the **Text** property to the caption of the new filter.
5. Click in the **Items** property of the RadGalleryGroupFilter, and then click the ellipsis button.
6. Click **Add** to add a group to the filter. Each time you click Add, one of the groups that you already created will be added to the filter.
7. Click **OK** when you are done adjusting group membership in the filter.
8. Click **OK** when you are done creating filters.

Creating Tools

You can add tools to be displayed beneath the gallery when it is in an expanded mode. Select the RadGalleryElement and edit the **Tools** collection.

6.7 RadRibbonBar Walk Through

In this walk through you will extend the "Getting Started" application by replacing the RadMenu with a RadRibbonBar.



Preparing the Project



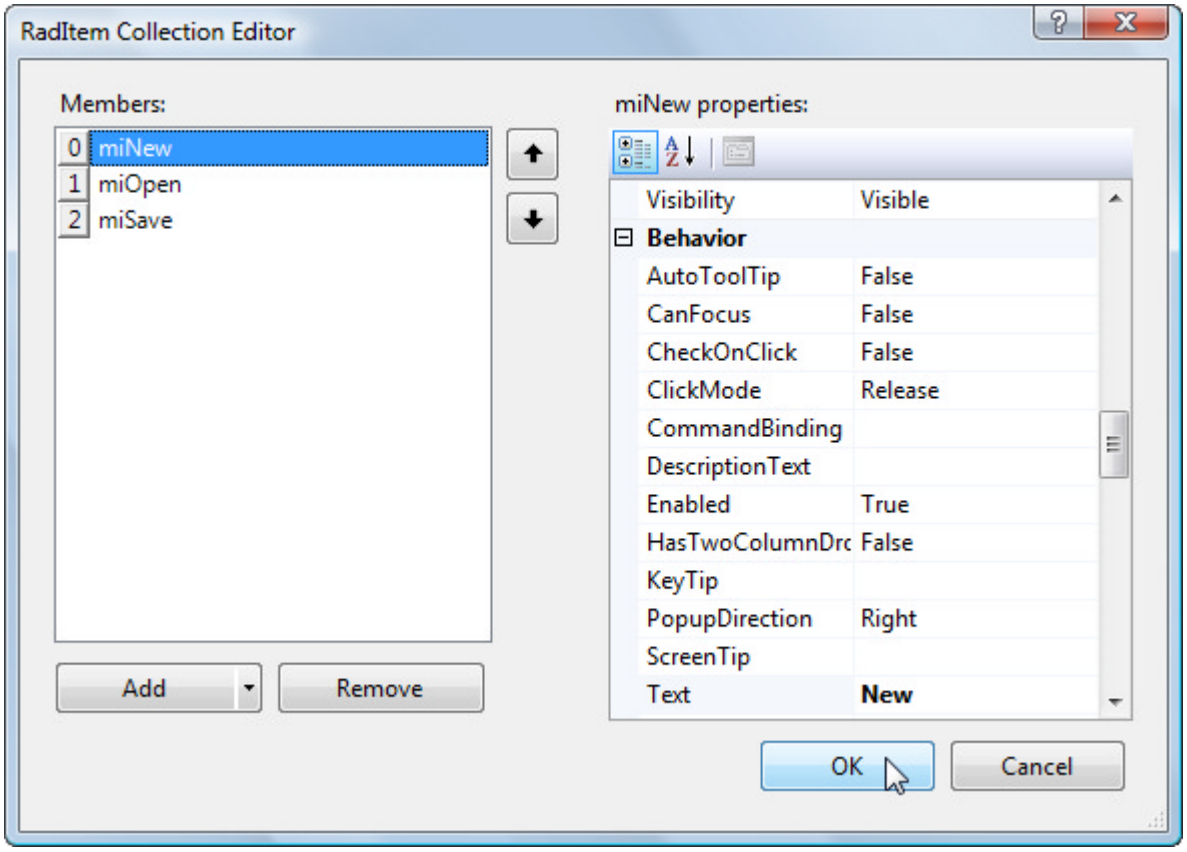
You can find the complete source for this project at:

\Menus\<VB|CS>\RibbonBar

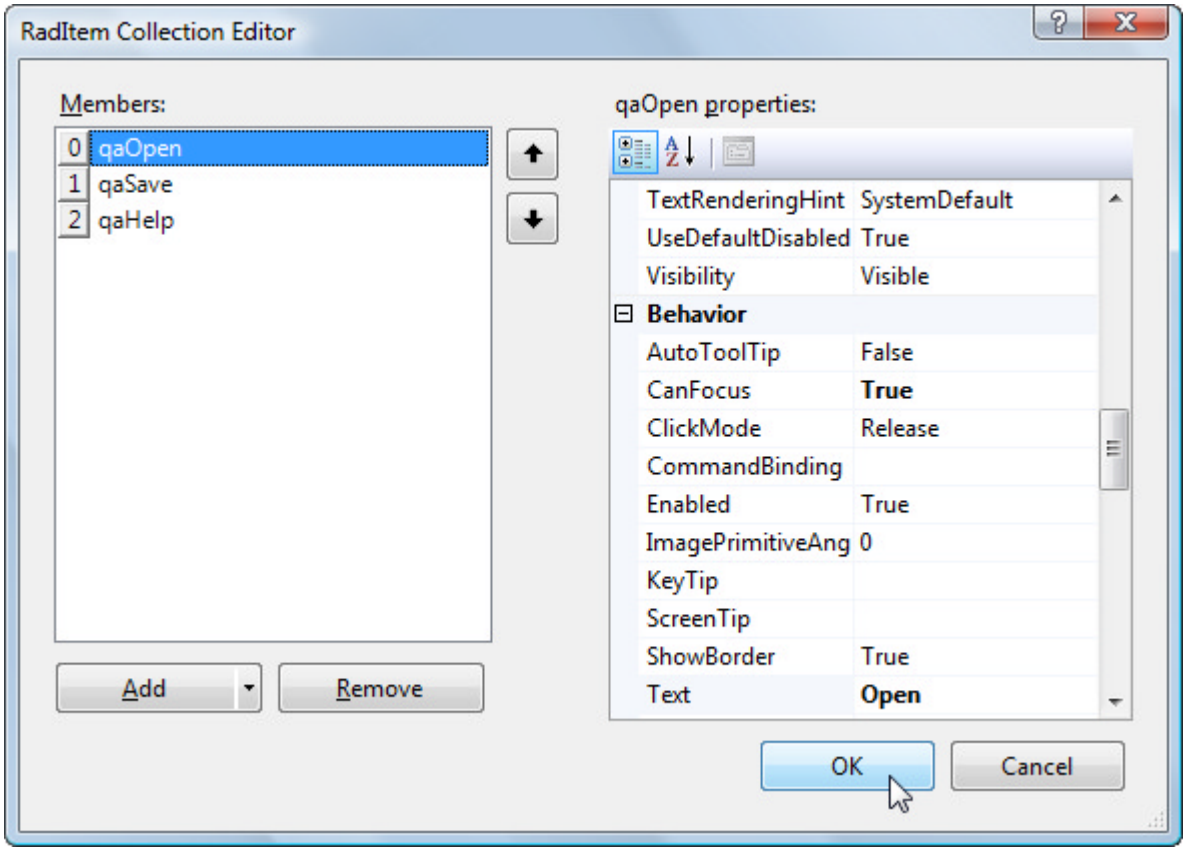
1. Using the "Getting Started" project or a copy, open the form.
2. Remove the RadMenu from the form and size the RichTextBox to make room for the ribbon bar.
3. Drop a RadRibbonBar on the form.
4. Drop a Office2007BlackTheme component on the form.
5. Set the form and RadRibbonBar ThemeName to "Office2007Black".
6. Set the RadRibbonBar ImageList property to point at the ImageList component already in the component tray.

Adapt the Menu to the Ribbon Bar

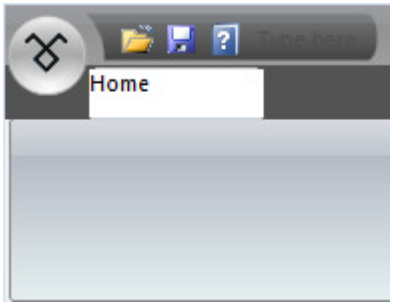
1. In the Properties window for the RadRibbonBar, locate the StartMenuItems and click the ellipses. Add three RadMenuItems:
   - **Name** = "miNew", **Text** = "New", **ImageKey** = <key of the "New" image>
   - **Name** = "miOpen", **Text** = "Open", **ImageKey** = <key of the "Open" image>
   - **Name** = "miSave", **Text** = "Save", **ImageKey** = <key of the "Save" image>



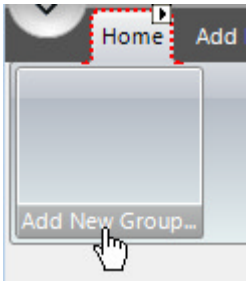
2. Open the RadRibbonBar **QuickAccessToolBarItems** property and add the following RadButtonElements:
   - **Name** = "qaOpen", **DisplayStyle** = Image, **ImageKey** = <key of the "Open" image>
   - **Name** = "qaSave", **DisplayStyle** = Image, **ImageKey** = <key of the "Save" image>
   - **Name** = "qaHelp", **DisplayStyle** = Image, **ImageKey** = <key of the "Help" image>



3. Click the **Add New Tab...** prompt in the designer and enter "Home" for the tab text.

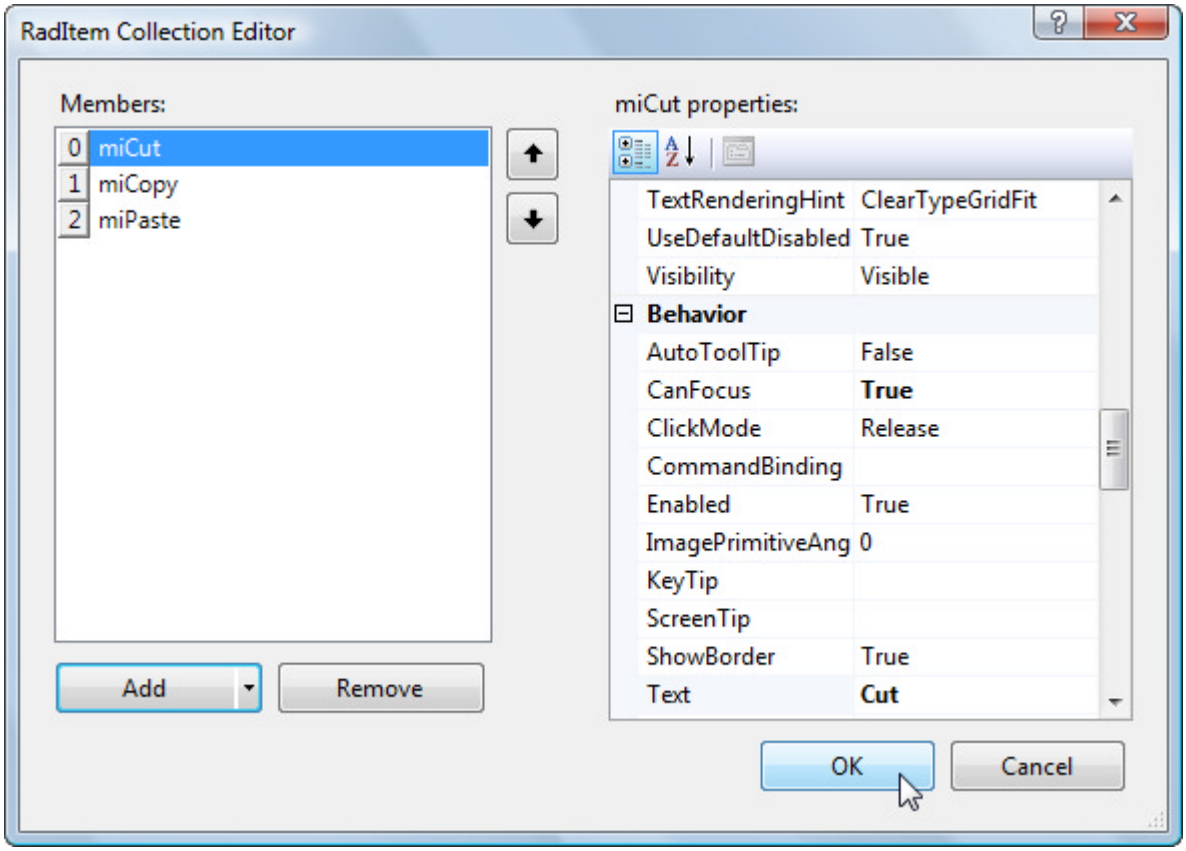


4. Click the **Add New Group...** prompt. Select the new group and set the Text property to "Clipboard".



5. Open the group's **Items** property and add three RadButtonElements:
   - **Name** = "miCut", **Text** = "Cut", **ImageIndex** = <index of the "Cut" image>
   - **Name** = "miCopy", **Text** = "Copy", **ImageIndex** = <index of the "Copy" image>

- **Name** = "miPaste", **Text** = "Paste", **ImageIndex** = <index of the "Paste" image>



6. Add a second group with **Text** = "Font". Add two RadCheckBoxElements:
   - **Name** = "cbBold", **Text** = "Bold", **DisplayStyle** = "Image", **ImageIndex** = <index of the "Bold" image>
   - **Name** = "cbItalic", **Text** = "Italic", **DisplayStyle** = "Image", **ImageIndex** = <index of the "Italic" image>
7. In the Properties window for the RadRibbonBar, locate the **OptionsButton** property and expand the item. Set the **Visibility** property to "Collapsed". Note: there are two buttons on the bottom right of the start menu, "Options" and "Exit". We will hide the Options and implement the exit to replace our "Quit" menu item.
8. In the Properties window Events tab, locate the **ExitButton** property and expand the item. Double-click the Click event to create an event handler. Move your "this.Close()" code to this handler.
9. In the Properties window Events tab, hook up the remaining menu item Click or ToggleStateChanged event handlers for each element.

6.8 Summary

In this chapter you learned how to add drop-down and context menus to your application using RadMenu, RadContextMenu and RadContextMenuManager controls. We began with a tour of usability and feature highlights, then learned how to build menus at design-time using the Menu Designer. You learned common menu item collection programmatic tasks such as adding and removing menu items, locating and modifying menu items. You learned how to programmatically pop up a context menu in a specific screen location. Finally, the chapter introduces the RadRibbonBar and RadApplicationMenu controls and explored their use.

7 Telerik Presentation Foundation

7.1 Objectives

- Learn some of the basic capabilities, architecture and class hierarchy of the Telerik Presentation Foundation.
- Become familiar with the classes used directly in the controls, i.e. RadElement and RadControl.
- See how elements are accessed within RadControls.
- See how elements can be added to RadControls.
- Learn how to add any Control descendant using a RadHostItem as a wrapper.

7.2 Introduction

The Telerik Presentation Foundation (TPF) offers Windows Presentation Foundation (WPF) features within classic Windows Forms applications. TPF is the foundation for all RadControls for Winforms controls and objects.

- All controls have access to common services provided by TPF such as theming, animation, layout and property binding. No matter how complex a control may be, a control is built up from a few simple types of elements. By working with these elements, you can customize any control on a very granular level. The uniform nature of the control architecture makes it possible to nest controls within each other for more flexible and modern user interfaces.
- All elements in a control element tree inherit common property values from their parent elements unless these values are set locally. This feature helps each element use less memory while still providing fine-grained customization options.

Control Architecture

Controls in the RadControls for WinForms suite have both a general shared architecture and a specific internal structure.

Shared Control Architecture

Every control in the Telerik RadControls for WinForms suite is derived from the RadControl class, which in turn descends from System.Windows.Forms.ScrollableControl. This relationship ties the controls into the main Windows Forms architecture, and allows you to place them on forms, participate in drag-and-drop operations, and so on. The controls are simple wrappers and do not implement detailed functionality. Rather, each control has as a **RootRadElement**. The RootRadElement wraps the actual functionality of the control and acts to transfer events to and from the RadControl class. In turn, the RootRadElement has a descendant tree of RadElement classes that implement the actual logic required by the control. In most cases the RootRadElement has a single descendant, though some complex controls such as the Telerik RadRibbonBar require more than one child for the RootRadElement.

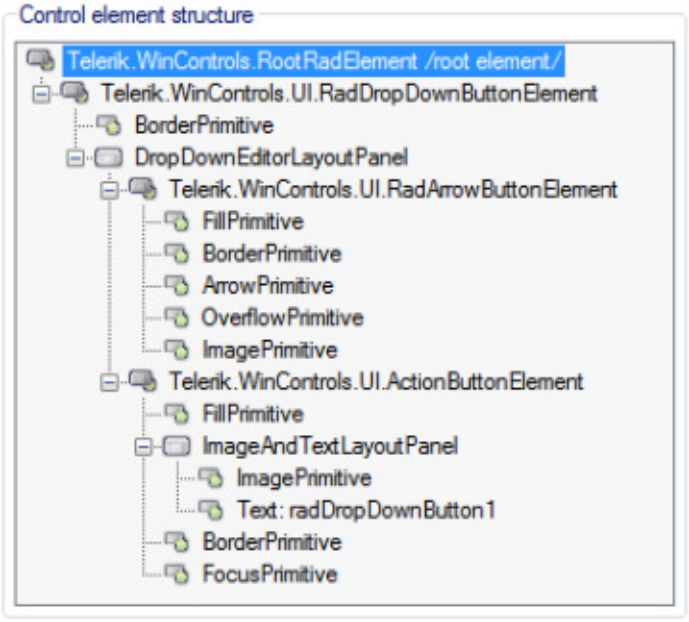
For example, schematically, the overall architecture for a RadDropDownButton looks as follows:

```
RadDropDownButton
  RootRadElement
    RadDropDownButtonElement
      tree of RadElement instances
```

Specific Internal Structure

Any control in the Telerik RadControls for WinForms suite is composed of RadElement instances arranged in a

logical tree. You can inspect this tree using the Control Spy or by clicking the Smart Tag and selecting Edit UI Elements. For example, the screenshot below shows the internal structure of a RadDropDownButton control. In this case, you can see the RootRadElement with the RadDropDownButtonElement as its only child. The RadDropDownButtonElement in turn contains a DropDownEditorLayoutPanel which in turn holds its elements - RadArrowButtonElement and ActionButtonElement . At the leaf level of the tree, various primitives are the atomic pieces that make up the element. For example, a ActionButtonElement contains a fill, an focus and border primitives.



TPF Based Controls

RadControls are built by composing RadElement trees instead of overriding the OnPaint method. Elements can take the form of lightweight primitive elements that draw the individual parts of a control to the form, layout elements that are responsible for arrange primitives on the form, and component elements that handle user input and are responsible for creating layout and primitive elements. Elements can be nested to any arbitrary level of complexity.

- A number of pre-built primitives are available for building themable user interfaces quickly including text, gradient fills, borders, checks, arrows, lines, radio buttons, images and trackbars.
- Elements combine primitives to form the basis of a functioning control. For example, RadLabelElement combines a BorderPrimitive, FillPrimitive and TextPrimitive, then adds specific label properties TextAlignment and BorderVisible.
- Layout elements descending from LayoutPanel manage the arrangement of child elements. For example, StackLayoutPanel arranges elements in a horizontal or vertical stack that may wrap to new lines. StripLayoutPanel orders elements in a single vertical or horizontal line.
- RadControls are typically thin wrappers around a component element, handle any direct communication with Windows and establish communication to the RadElement by creating a root element and adding elements to the root element. RadLabel for example simply creates a root element and then creates and adds a RadLabelElement to the root element.

7.3 Getting Started

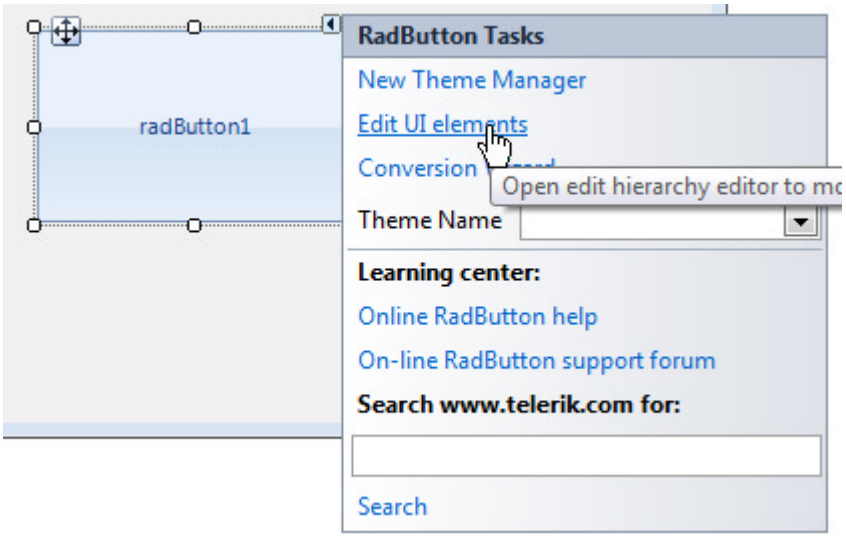
In a typical development scenario you may want to access some aspect of a RadControl and make minor changes. At design time you can access the element tree and get at any control element properties.



You can find the complete source for this project at:

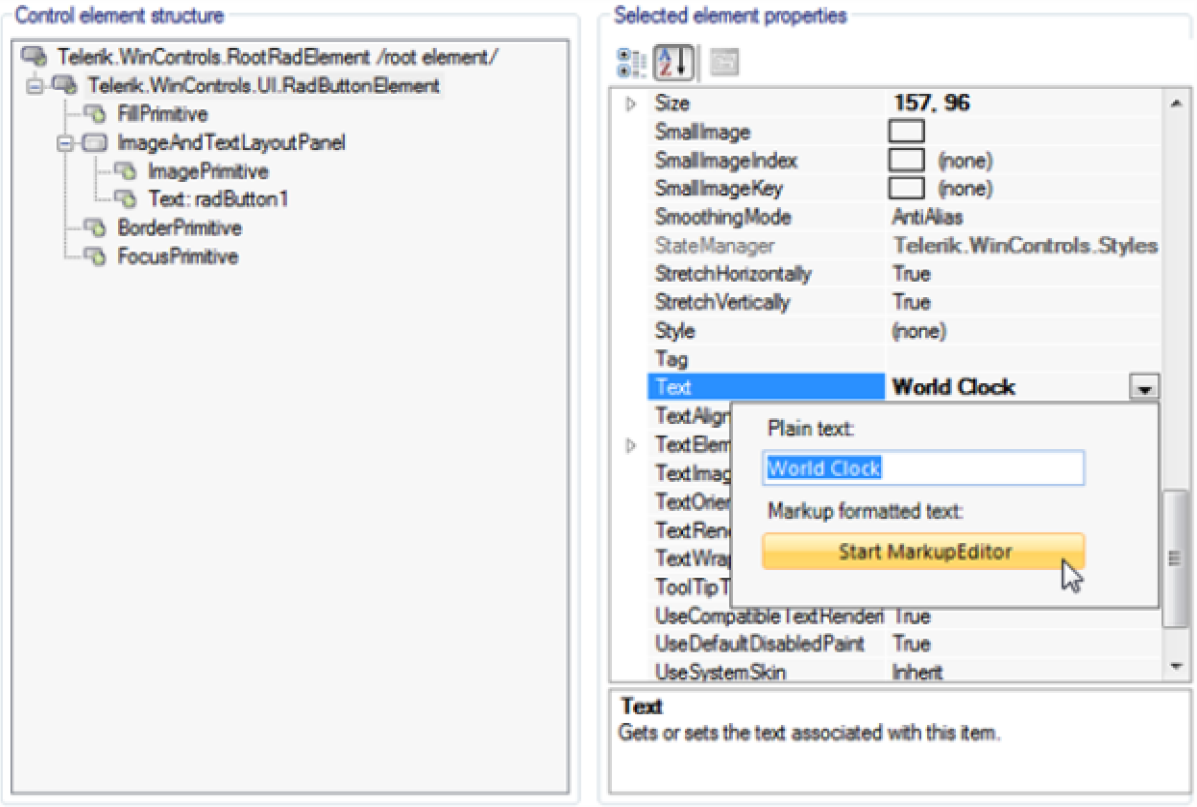
\TPF\<VB|CS>\GettingStarted

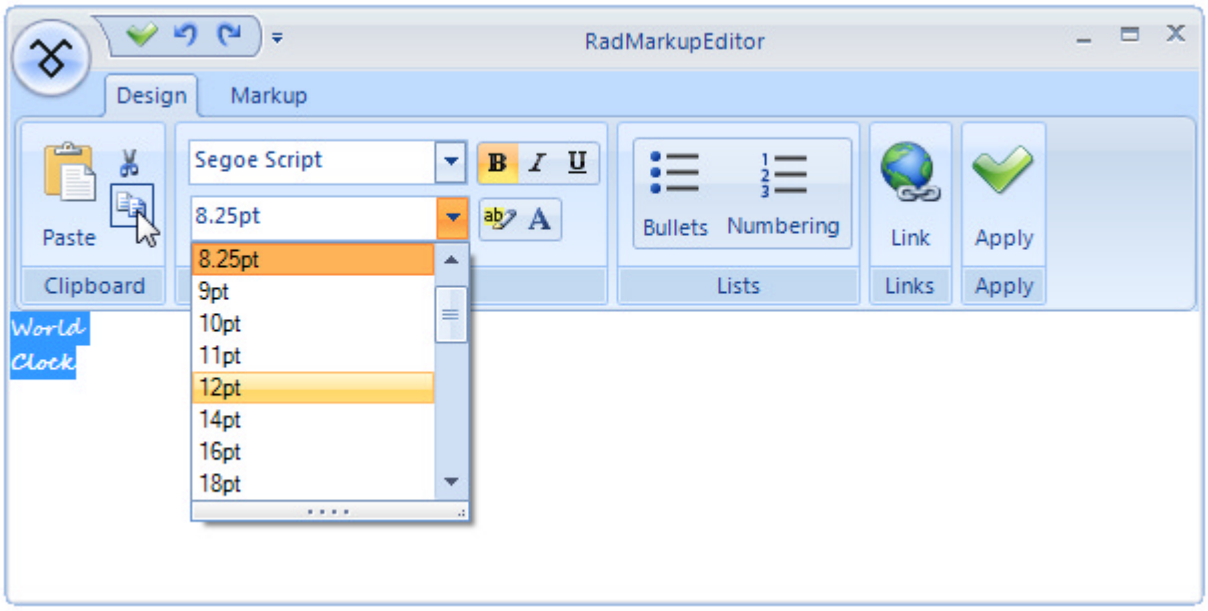
1. In a new Windows Forms Application, drop a RadButton to the default form.
2. Open the Smart Tag and click the Edit UI Elements link.



3. This brings up the Element Hierarchy Editor dialog. Notice the Control element structure on the left side of the dialog. The top level element is called RootRadElement and contains a RadButtonElement. The constituent parts of the RadButtonElement are the FillPrimitive, a layout panel to arrange image and text, BorderPrimitive and FocusPrimitive.

   Select the "RadButtonElement" node of the element tree and change properties: **TextImageRelation** = TextBeforeImage, **Text** = "World Clock", **Padding** = "10, 10, 10, 10". When setting the Text property, use the drop down arrow button to open RadMarkUpEditor, where you can make different modifications. Enter "World", hit enter, enter "Clock", then choose Font "Segoe Script" and make the text Bold. Click Apply to confirm the changes.





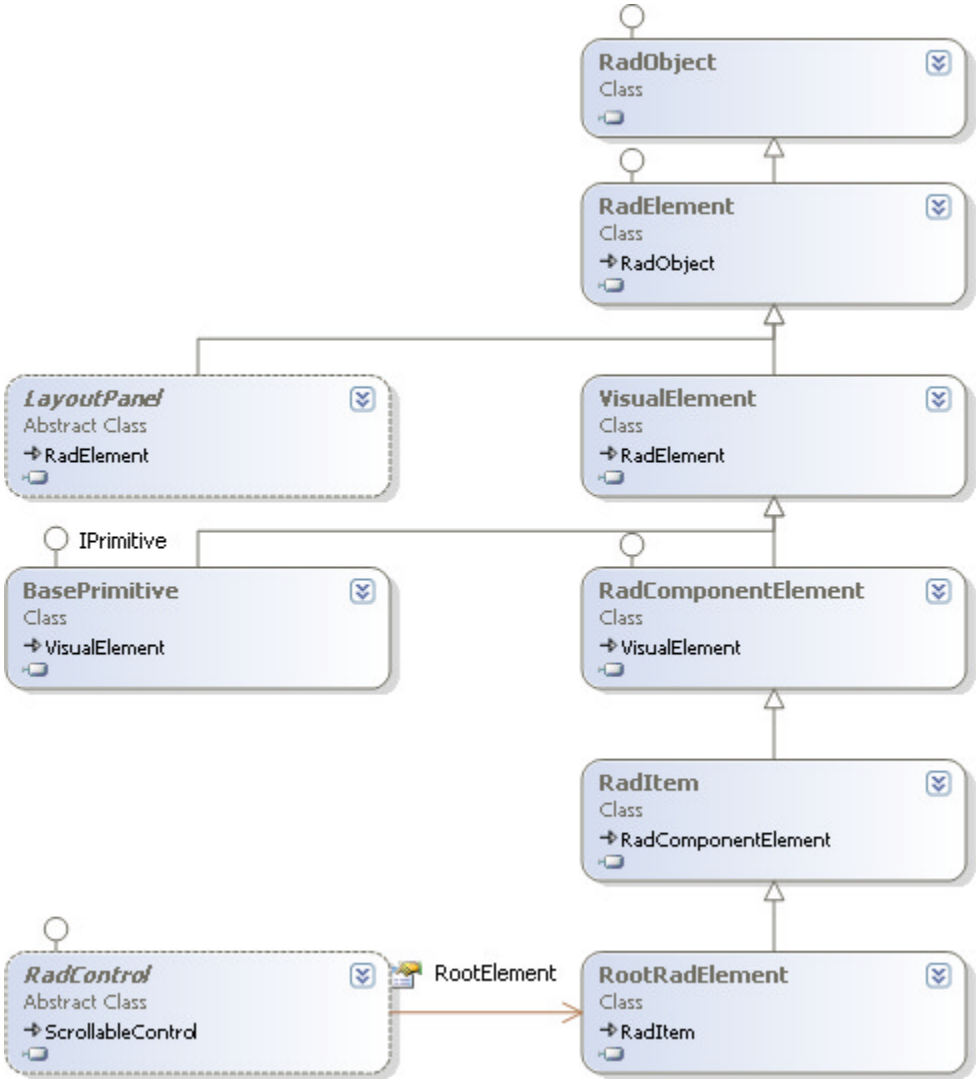
4. Select the "FillPrimitive" node and and set properties: **BackColor** = "Blue", **BackColor2** = "LightBlue", **BackColor3** = "Lavender", **BackColor4** = "Purple", **GradientStyle** = "OfficeGlass".
5. Select the "ImagePrimitive" node and set properties: **Image** = <the image of a globe, or any small image (around 32 x 32 pixels)>, **Opacity** = "0.5".
6. Select the "BorderPrimitive" node and set the Visibility property to Collapsed in order to hide the border around the button
7. Click the **Close** button to view the button.



Properties in primitives are often surfaced to higher level objects in the element tree. Using the Hierarchy editor makes it possible to get at properties that haven't been surfaced yet. The point of this exercise was to use some of the lower level primitives and make the button look much different than the default version. In a more typical development situation, look for the highest level where the property appears.

7.4 Class Hierarchy

This is an abbreviated class diagram of the class inheritance starting with RadObject and building to RadControl. RadProperty and RadPropertyMetaData are not shown.



- **RadProperty** encapsulates properties used in RadObject. Each RadProperty has a name, type, owner and

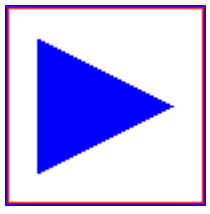
metadata, i.e. information about the property.

- **RadPropertyMetadata** describes a property and de-couples the information about a property from the property itself.
- **RadObject** represents the base class for all RadElement objects and has the ability to store all property values in a single structure.
- **RadElement** is a key class that represents the smallest unit in a RadControl that can be painted or laid out on the screen. RadElement is the base class of all elements that take advantage of TPF features, e.g. property inheritance, layouts and styling. RadElement implements a tree-like structure of RadObjects, property value inheritance and automatic layout and display invalidation based on the options of the PropertyMetadata for each RadProperty. RadElement has a size and location specified in coordinates in relation to its parent.
- **VisualElement** adds visual properties common to all elements that will be painted, e.g. back color, fore color, font, default size, smoothing mode, opacity.
- **LayoutPanel** is an abstract class descending from RadElement. LayoutPanel descendants coordinate sizing and position for a number of elements.
- **RadComponentElement** implements IComponent and introduces the ability to be contained.
- **BasePrimitive** is a VisualElement descendant that adds a virtual PaintPrimitive() method that draws a primitive to the screen.
- **RadItem** represents a visual element which could be added to an ItemsCollection and can be selected, deleted or moved at design time. By default this class knows how to handle user input.
- **RadControl** is the abstract base class for all RadControls. Each RadControl contains a tree of RadElements. The tree has a root, the RadControl.RootElement (type RootRadElement shown in the diagram) and children of the root element. RadControl acts as a bridge between traditional Windows forms controls and the RadElement tree. RadControl also defines properties common across all controls e.g. ThemeName, Style, ImageList, image scaling properties, minimum and maximum sizes.

7.5 Primitives

Primitives are the elements in the control element tree that are painted to the screen when Windows displays a form containing a Telerik RadControl. All primitives are derived from BasePrimitive, a subclass of VisualElement. BasePrimitive adds the virtual PaintPrimitive() method to VisualElement. The descendants of BasePrimitive override this virtual function to draw themselves. Individual primitive classes expose additional properties. For example, the TextPrimitive class includes a Text property, which specifies the text that it will place on the screen.

Browse through these available primitive types to see what you can include in your own RadControl customizations:

Primitive	Example	Description
ArrowPrimitive		The ArrowPrimitive draws a filled triangular polygon. Orientation is controlled by the Direction property with possible ArrowDirection enumeration values of Up, Down, Left or Right. The arrow is filled with the ForeColor property value. This example draws a border primitive and an ArrowPrimitive.

BorderPrimitive



The BorderPrimitive class is responsible for drawing the border around an area on the screen. BorderPrimitive is automatically sized to the inner edges of the parent element. The BoxStyle property controls display characteristics and can be set to SingleBorder, FourBorders and OuterInnerBorders.

All four sides of the border are drawn with the same properties except color properties. Color is controlled by the GradientStyles property. To get all colors in the border to display the same, use the GrandientStyles.Solid enumeration member.

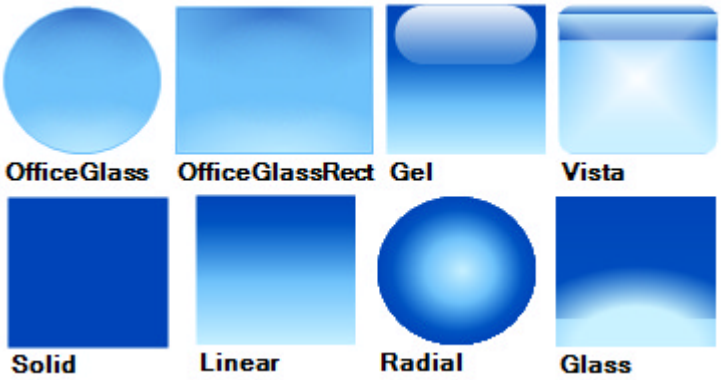
CheckPrimitive



The CheckPrimitive is painted as three sets of angled lines and is colored using the ForeColor property. The three sets of lines are easier to see when the primitive is scaled up. This example creates two CheckPrimitives, one at the default size and the second scaled up by a factor of 5.

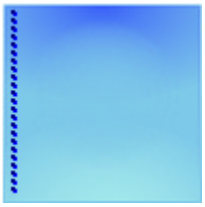
FillPrimitive

The FillPrimitive class is responsible for filling in an area on the screen. GradientStyle is a pivitol property that allows you to select from solid, linear, radial, glass, office glass, office glass rectangle, gel, and vista styles. GradientStyle characterizes how the other properties are interpreted: NumberOfColors, GradientAngle,



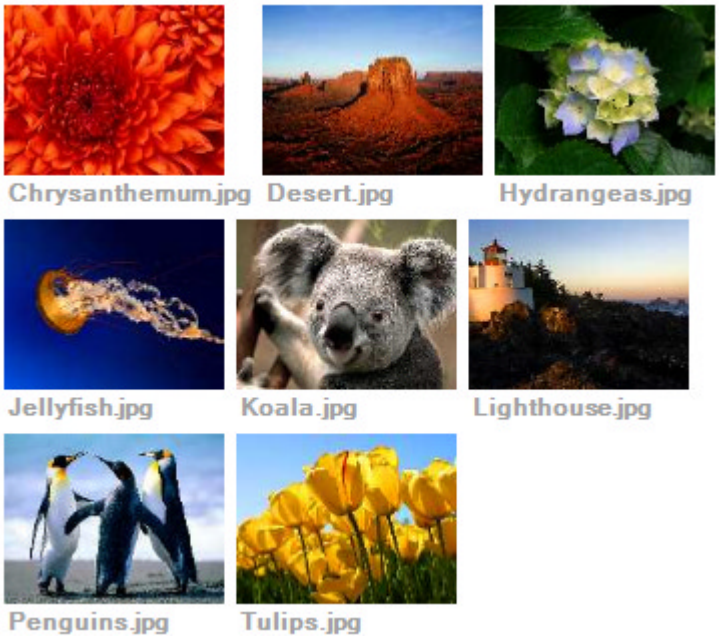
GradientPercentage, BackColor, BackColor2, BackColor3 and BackColor4.

GripPrimitive



GripPrimitive displays a vertical row of dots as a visual cue to users that this is an area that can be gripped and dragged. GripPrimitive is used in the RadToolStrip to display on the left hand side of each strip. Grip dots are displayed using BackColor and BackColor2 colors. BackColor2 represents a shadow to set the dots off the background, particularly when the dots and the background contain some of the same colors.

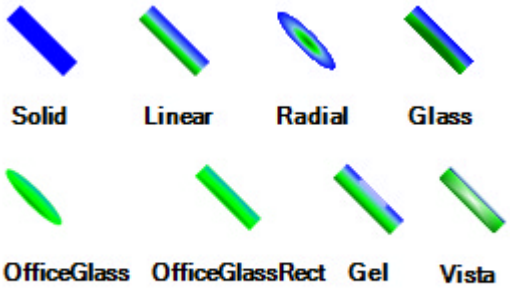
ImagePrimitive



Use the ImagePrimitive class whenever you need to draw an image as part of rendering a control. The properties specific to displaying images are Image, ImageKey, ImageIndex and ImageScaling. Note: the ImageList property is introduced in the RadControl class.

LinePrimitive is actually a variant of a filled

LinePrimitive



rectangle. The shadow of the fill is set by the BackColor, BackColor2, BackColor3 and BackColor4 properties. The way these four colors are used is determined by the GradientStyle and GradientAngle properties. LineWidth sets the thickness of the line. Some GradientStyle settings will not be visible if the LineWidth is too narrow.

OverflowPrimitive



The OverflowPrimitive is a visual indicator that more choices are available. The OverflowPrimitive consists of a triangle and a line just under the base of the triangle. The triangle and line are colored using the ForeColor property. Depending on the Direction property of the arrow, BackColor and ShadowColor may be used.

ProgressBarPrimitive



ProgressBarPrimitive is a specialized descendant of FillPrimitive that indicates the state of a long running process. The fill area is controlled by the GradientStyle and the BackColor..BackColor4 properties. The fill covers the percentage area between Minimum and Maximum specified by the Value1 property. Value1 is rendered using up to all four colors. Value2 displays in the BackColor at 25% opacity. Orientation controls the starting point of the fill and can originate from Left, Right, Top or Bottom.

The example shows a Linear fill for Value1 (set to 20), and a pale Aqua for Value2 (set to 70).

RadioPrimitive



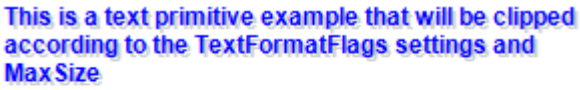
RadioPrimitive provides the basic visual layout of a radio button background. Typically you would use a higher level object, i.e. the RadRadioButtonElement to include in your control. RadRadioButtonElement handles the toggling of other peer level radio buttons

RectanglePrimitive



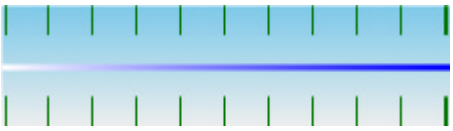
RectanglePrimitive is a lightweight primitive that draws a rectangle using ForeColor only. Unlike BorderPrimitive, RectanglePrimitive does not have GradientStyle, BoxStyle and other color properties.

TextPrimitive



Use the TextPrimitive class whenever you need to display text as part of rendering a control. Color for the font in the primitive is set by the ForeColor property. Other text related properties are Text, TextAlignment, TextFormatFlags, TextOrientation, Shadow.

TrackBarPrimitive



TrackBarPrimitive represents the basic UI of a trackbar, without the trackbar thumb. Typically you would use a higher level object, i.e. the RadTrackBarElement to include in your control. RadTrackBarElement handles the position of the track bar thumb and the various events that go into making the track

FocusPrimitive		bar useful. FocusPrimitive supplies user interface cues that an element has received focus. The constructor for FocusPrimitive takes a BorderPrimitive that provides the basic dimensions. Fore colors and inner colors may be set explicitly or default control colors are used.

7.6 Elements

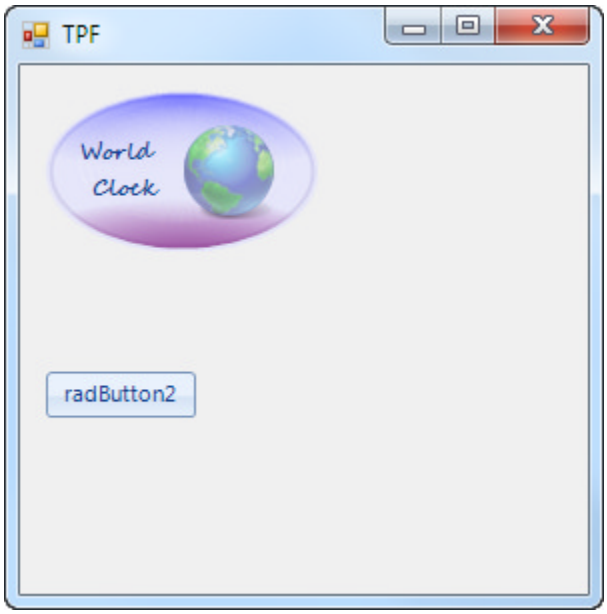
Elements fall into three categories, depending on their base class:

- **Layout** elements derive from LayoutPanel. Each LayoutPanel arranges child items in a particular manner.
- **Painted** elements derive from BasePrimitive. They override the OnPaint method to draw figures on the controls graphic surface.
- **Component** elements derive from RadItem. RadItem descendants handle user input and can be worked with in the design environment. A component element overrides the virtual CreateChildElements() method to create one or more layout elements and primitives. Typically TPF based controls are simple wrappers around RadItem descendant classes, and the RadItem descendant classes define logic and user interface.

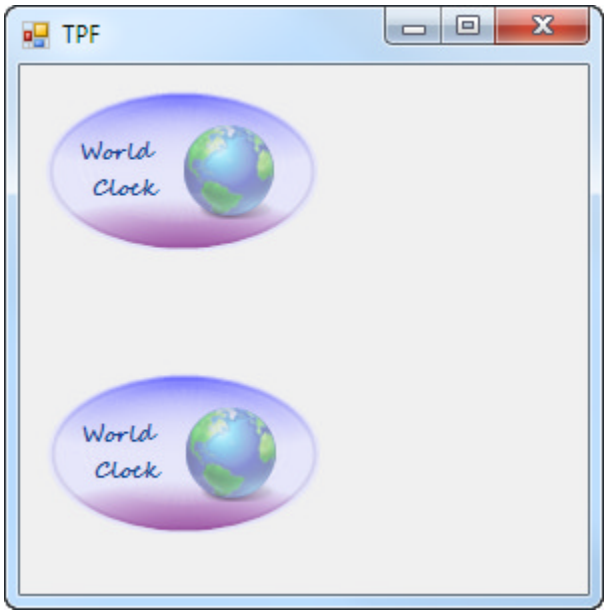
7.7 Programming the Telerik Presentation Foundation

Accessing Elements

Now that you know how to get at elements that make up RadControls at design time using the Hierarchy Editor dialog, you can access these same properties in code. At design time we can start with two RadButtons, the first styled like the example in "Getting Started", the second is the stock RadButton with no changes.



The code below will match the second button to look like the first.



As you will see, the trick is to traverse the Children RadElementCollection, locate the correct element and cast it to the appropriate type. The example below first gets the RadButtonElement and uses that to access the other elements.

> This top level element is often surfaced as a property, e.g. you can reference "radButton1.ButtonElement" instead of "radButton1.RootElement.Children".

[VB] Setting Button Element Properties

```
Public Sub New()
 InitializeComponent()
 ' get the "ButtonElement", just under the root element,
 ' set the padding and text
 Dim buttonElement As RadButtonElement = TryCast(radButton2.RootElement.Children(0),
RadButtonElement)
 buttonElement.TextImageRelation = TextImageRelation.TextBeforeImage
 buttonElement.Padding = New Padding(10)
 buttonElement.Text = "World" + Environment.NewLine + "Clock"
 ' access the fill primitive, use "OfficeGlass" style and set coloring
 Dim fillPrimitive As FillPrimitive = DirectCast(buttonElement.GetChildrenByType(GetType
(FillPrimitive))(0), FillPrimitive)
 fillPrimitive.GradientStyle = Telerik.WinControls.GradientStyles.OfficeGlass
 fillPrimitive.BackColor = Color.Blue
 fillPrimitive.BackColor2 = Color.LightBlue
 fillPrimitive.BackColor3 = Color.Lavender
 fillPrimitive.BackColor4 = Color.Purple
 ' get the image primitive and set image. Set the opacity to be 50% transparent.
 Dim imagePrimitive As ImagePrimitive = TryCast(DirectCast(buttonElement.Children
(1).Children(0), ImagePrimitive), ImagePrimitive)
 imagePrimitive.Image = Properties.Resources.Globe
 imagePrimitive.Opacity = 0.5
 ' get the text primitive and set the font to use a bold, script
 Dim textPrimitive As TextPrimitive = TryCast(DirectCast(buttonElement.Children(1).Children
(1), TextPrimitive), TextPrimitive)
 textPrimitive.Font = New Font("Segoe Script", 8.25F, FontStyle.Bold)
 ' get the border primitive and hide it.
```

```
 Dim borderPrimitive As BorderPrimitive = DirectCast(buttonElement.GetChildrenByType(GetType
(BorderPrimitive))(0), BorderPrimitive)
 borderPrimitive.Visibility = Telerik.WinControls.ElementVisibility.Hidden
 ' match the button sizes
 radButton2.Size = radButton1.Size
End Sub
```

[C#] Setting Button Element Properties

```
public Form1()
{
 InitializeComponent();
 // get the "ButtonElement", just under the root element,
 // set the padding and text
 RadButtonElement buttonElement = radButton2.RootElement.Children[0] as RadButtonElement;
 buttonElement.TextImageRelation = TextImageRelation.TextBeforeImage;
 buttonElement.Padding = new Padding(10);
 buttonElement.Text = "World" + Environment.NewLine + "Clock";
 // access the fill primitive, use "OfficeGlass" style and set coloring
 FillPrimitive fillPrimitive =
   (FillPrimitive)buttonElement.GetChildrenByType(typeof(FillPrimitive))[0];
 fillPrimitive.GradientStyle = Telerik.WinControls.GradientStyles.OfficeGlass;
 fillPrimitive.BackColor = Color.Blue;
 fillPrimitive.BackColor2 = Color.LightBlue;
 fillPrimitive.BackColor3 = Color.Lavender;
 fillPrimitive.BackColor4 = Color.Purple;
 // get the image primitive and set image. Set the opacity to be 50% transparent.
 ImagePrimitive imagePrimitive =
   (ImagePrimitive)buttonElement.Children[1].Children[0] as ImagePrimitive;
 imagePrimitive.Image = Properties.Resources.Globe;
 imagePrimitive.Opacity = 0.5;
 // get the text primitive and set the font to use a bold, script
 TextPrimitive textPrimitive =
   (TextPrimitive)buttonElement.Children[1].Children[1] as TextPrimitive;
 textPrimitive.Font = new Font("Segoe Script", 8.25F, FontStyle.Bold);
 // get the border primitive and hide it.
 BorderPrimitive borderPrimitive =
   (BorderPrimitive)buttonElement.GetChildrenByType(typeof(BorderPrimitive))[0];
 borderPrimitive.Visibility = Telerik.WinControls.ElementVisibility.Hidden;
 // match the button sizes
 radButton2.Size = radButton1.Size;
}
```



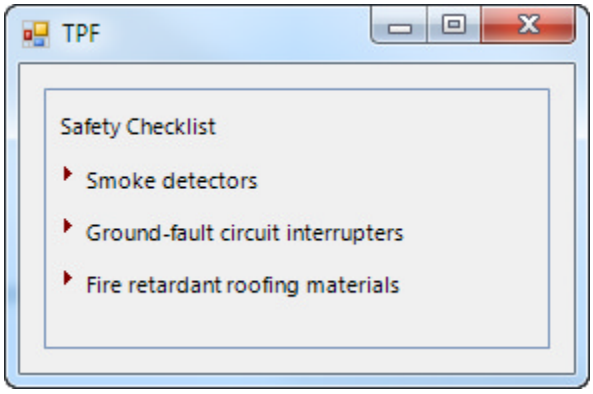
If you're having trouble getting the correct syntax to access a given property, try setting it using the Hierarchy Editor dialog at design time and look at the generated code, i.e. Form1.Designer.cs. The code may be a little "wordy" compared with hand-written code but you can get a better look at how the hierarchy is accessed and what types are being used.

```
//
// radButton1
//
this.radButton1.Image = global::TestMyTPF.Properties.Resources._1403_Globe;
this.radButton1.Location = new System.Drawing.Point(12, 12);
this.radButton1.Name = "radButton1";
this.radButton1.Size = new System.Drawing.Size(138, 82);
this.radButton1.TabIndex = 0;
this.radButton1.Text = "World\r\nClock";
this.radButton1.TextImageRelation = System.Windows.Forms.TextImageRelation.TextBeforeImage;
this.radButton1.Click += new System.EventHandler(this.radButton1_Click);
((Telerik.WinControls.UI.RadButtonElement)(this.radButton1.GetChildAt(0))).Image = global::
((Telerik.WinControls.UI.RadButtonElement)(this.radButton1.GetChildAt(0))).TextImageRelati
((Telerik.WinControls.UI.RadButtonElement)(this.radButton1.GetChildAt(0))).Text = "World\r\
((Telerik.WinControls.UI.RadButtonElement)(this.radButton1.GetChildAt(0))).Padding = new S
((Telerik.WinControls.Primitives.FillPrimitive)(this.radButton1.GetChildAt(0).GetChildAt(0)
((Telerik.WinControls.Primitives.FillPrimitive)(this.radButton1.GetChildAt(0).GetChildAt(0
((Telerik.WinControls.Primitives.FillPrimitive)(this.radButton1.GetChildAt(0).GetChildAt(0)
((Telerik.WinControls.Primitives.FillPrimitive)(this.radButton1.GetChildAt(0).GetChildAt(0)
((Telerik.WinControls.Primitives.FillPrimitive)(this.radButton1.GetChildAt(0).GetChildAt(0)
((Telerik.WinControls.Primitives.ImagePrimitive)(this.radButton1.GetChildAt(0).GetChildAt(
((Telerik.WinControls.Primitives.ImagePrimitive)(this.radButton1.GetChildAt(0).GetChildAt(
((Telerik.WinControls.Primitives.TextPrimitive)(this.radButton1.GetChildAt(0).GetChildAt(1
((Telerik.WinControls.Primitives.BorderPrimitive)(this.radButton1.GetChildAt(0).GetChildAt
((Telerik.WinControls.Primitives.BorderPrimitive)(this.radButton1.GetChildAt(0).GetChildAt
```

Adding Elements to RadControls

Elements can be added to existing RadControls to produce unique display and behavior. The typical use would be to make some tweak in an existing control if the existing makeup of the control doesn't produce what you want to see.

This next example adds elements to a RadPanel. A StackLayoutPanel is added to the RadPanel's Children collection. The Orientation of the layout panel is Vertical, so that elements added to its Children collection will be displayed from top to bottom. A label with a title "Safety Checklist" is added to the layout panel, then a series of "bullet points". Each bullet point is made up of its own StackLayoutPanel with the Orientation set to Horizontal so that the "bullet point" and text will be displayed left to right.





You can find the complete source for this project at:

\TPF\<VB>|CS>\AddingElements

[VB] Composing Elements

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 ' Create a stack panel to contain the title and bullet points
```

```
 Dim layoutPanel As New StackLayoutPanel()
 layoutPanel.Orientation = Orientation.Vertical
 Dim label As New RadLabelElement()
 ' create the "title"
 label.Text = "Safety Checklist"
 label.Padding = New Padding(5, 10, 0, 0)
 ' add the title and bullet points
 layoutPanel.Children.Add(label)
 layoutPanel.Children.Add(GetBulletPoint("Smoke detectors"))
 layoutPanel.Children.Add(GetBulletPoint("Ground-fault circuit interrupters"))
 layoutPanel.Children.Add(GetBulletPoint("Fire retardant roofing materials"))
 ' add the layout panel element to the panel's children
 radPanel1.RootElement.Children.Add(layoutPanel)
End Sub
Private Function GetBulletPoint(text As String) As StackLayoutPanel
 ' create a layout panel for a single "bullet point"
 Dim layoutPanel As New StackLayoutPanel()
 layoutPanel.Orientation = Orientation.Horizontal
 layoutPanel.Margin = New Padding(10, 10, 0, 0)
 ' add the "bullet point" graphic and add to the layout panel
 Dim arrowPrimitive As New ArrowPrimitive()
 arrowPrimitive.Direction = Telerik.WinControls.ArrowDirection.Right
 arrowPrimitive.Margin = New Padding(0, 0, 5, 0)
 arrowPrimitive.ForeColor = Color.Maroon
 layoutPanel.Children.Add(arrowPrimitive)
 ' add the text for the bullet point and add to the layout panel
 Dim textPrimitive As New TextPrimitive()
 textPrimitive.Text = text
 layoutPanel.Children.Add(textPrimitive)
 Return layoutPanel
End Function
```

[C#] Composing Elements

```
private void Form1_Load(object sender, EventArgs e)
{
 // Create a stack panel to contain the title and bullet points
 StackLayoutPanel layoutPanel = new StackLayoutPanel();
 layoutPanel.Orientation = Orientation.Vertical;
 RadLabelElement label = new RadLabelElement();
 // create the "title"
 label.Text = "Safety Checklist";
 label.Padding = new Padding(5, 10, 0, 0);
 // add the title and bullet points
 layoutPanel.Children.Add(label);
 layoutPanel.Children.Add(GetBulletPoint("Smoke detectors"));
 layoutPanel.Children.Add(GetBulletPoint("Ground-fault circuit interrupters"));
 layoutPanel.Children.Add(GetBulletPoint("Fire retardant roofing materials"));
 // add the layout panel element to the panel's children
 radPanel1.RootElement.Children.Add(layoutPanel);
}
private StackLayoutPanel GetBulletPoint(string text)
{
 // create a layout panel for a single "bullet point"
 StackLayoutPanel layoutPanel = new StackLayoutPanel();
 layoutPanel.Orientation = Orientation.Horizontal;
```

```
 layoutPanel.Margin = new Padding(10, 10, 0, 0);
 // add the "bullet point" graphic and add to the layout panel
 ArrowPrimitive arrowPrimitive = new ArrowPrimitive();
 arrowPrimitive.Direction = Telerik.WinControls.ArrowDirection.Right;
 arrowPrimitive.Margin = new Padding(0, 0, 5, 0);
 arrowPrimitive.ForeColor = Color.Maroon;
 layoutPanel.Children.Add(arrowPrimitive);
 // add the text for the bullet point and add to the layout panel
 TextPrimitive textPrimitive = new TextPrimitive();
 textPrimitive.Text = text;
 layoutPanel.Children.Add(textPrimitive);
 return layoutPanel;
}
```



There are easier ways to get this particular task done; this is simply an example showing how you can mix-and-match elements to build controls of arbitrary arrangement and complexity. The general rule is to look for the highest level control or element that will suit your "must haves". If you opt for lower-level control you can fine tune the results, but you will also have to manage interaction between elements and perform more extensive testing for unexpected behavior.

Adding Controls to Elements

If you need to host an existing control where you would normally place a RadElement, you can do it using **RadHostItem**, a little adapter between standard Windows controls and the inner world of TPF elements. This assumes there are no elements that already fit the bill for what you want to do. Perhaps you have a unique control that would take some work to replicate using RadElements and simply want to add it to a RadControl. The general steps are:

- Create an instance of the control.
- Create a RadHostItem and pass the control instance in the constructor.
- Add the RadHostItem to the Children collection of your control or item.

Let's say we have a RadListControl and want to add to its items a GroupBox. For this scenario, in the CreatingVisualListItem event of RadListControl, we are going to create a GroupBox and RadHostItem and pass the GroupBox in the constructor of RadHostItem. Then add the RadHostItem to the Children collection of an instance of RadListVisualItem and assign the newly created visual item to the visual item from the event arguments.



You can find the complete source for this project at:

\TPF\<VB|CS>AddingControlsToElements

[VB] Adding and Configuring RadHostItems

```
Private Sub RadForm1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
RadListControl1.Items.Add("Item 1")
RadListControl1.Items.Add("Item 2")
RadListControl1.Items.Add("Item 3")
End Sub

Private Sub RadListControl1_CreatingVisualListItem(ByVal sender As Object, ByVal args As
Telerik.WinControls.UI.CreatingVisualListItemEventArgs) Handles
RadListControl1.CreatingVisualListItem
'make the items autosize
```

```
RadListControl1.AutoSizeItems = True
' create and configure the groupbox
Dim groupBox As New GroupBox()
' create the RadHostItem with the groupbox as hosted control
Dim host As New RadHostItem(groupBox)
' add host item to the list box item
Dim visualItem As New RadListVisualItem()
visualItem.Children.Insert(0, host)
visualItem.Padding = New Padding(50)
args.VisualItem = visualItem
End Sub
```

[C#] Adding and Configuring RadHostItems

```
private void RadForm1_Load(object sender, EventArgs e)
{
radListControl1.Items.Add(new RadListDataItem("Item 1"));
radListControl1.Items.Add(new RadListDataItem("Item 2"));
radListControl1.Items.Add(new RadListDataItem("Item 3"));
radListControl1.CreatingVisualListItem += new CreatingVisualListItemEventHandler
(radListControl1_CreatingVisualListItem);
radListControl1.VisualItemFormatting += new VisualListItemFormattingEventHandler
(radListControl1_VisualItemFormatting);
}

void radListControl1_CreatingVisualListItem(object sender, CreatingVisualListItemEventArgs
args)
{
//make the items autosize
radListControl1.AutoSizeItems = true;
// create and configure the groupbox
GroupBox groupBox = new GroupBox();
// create the RadHostItem with the groupbox as hosted control
RadHostItem host = new RadHostItem(groupBox);
// add host item to the list box item
RadListVisualItem visualItem = new RadListVisualItem();
visualItem.Children.Insert(0, host);
visualItem.Padding = new Padding(50);
args.VisualItem = visualItem;
}
```

Once the control is hosted, you can refer to it by accessing the RadHostItem's HostedControl property and casting it to the appropriate control type. Refer to the following example where we make modifications to the item - taking the text from the real data item, assigning it to the accessed group box and removing it from the visual item. Here is the place where you can also make modifications to the appearance of the item. All this is achieved in the VisualItemFormatting event.

[VB] Accessing the Hosted Control

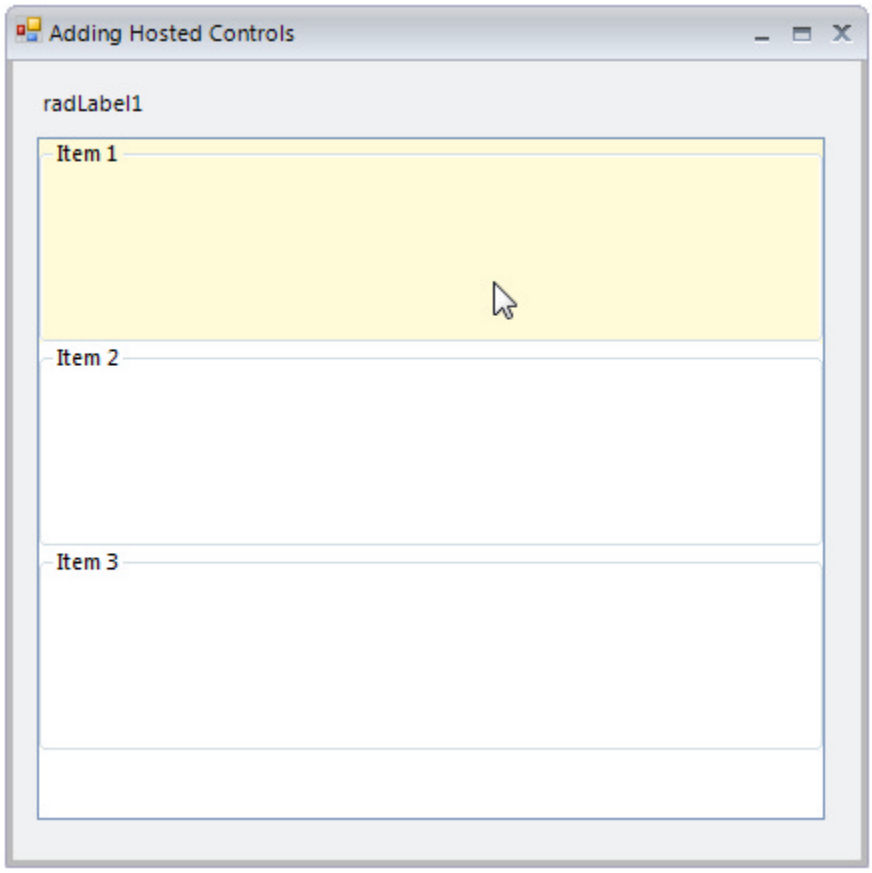
```
Private Sub RadListControl1_VisualItemFormatting(ByVal sender As Object, ByVal args As
Telerik.WinControls.UI.VisualItemFormattingEventArgs) Handles
RadListControl1.VisualItemFormatting
'get the host item from the visual item childred collection
Dim hostItem As RadHostItem = DirectCast(args.VisualItem.Children(0), RadHostItem)
'get the group box which is hosted by the host item
Dim groupBox As GroupBox = DirectCast(hostItem.HostedControl, GroupBox)
'set the text of the group box to eqals the data item text
groupBox.Text = args.VisualItem.Data.Text
```

```
'remove the visual item default text
args.VisualItem.Text = ""
End Sub
```

[C#] Accessing the Hosted Control

```
void radListControl1_VisualItemFormatting(object sender, VisualItemFormattingEventArgs args)
{
//get the host item from the visual item childred collection
RadHostItem hostItem = ((RadHostItem)args.VisualItem.Children[0]);
//get the group box which is hosted by the host item
GroupBox groupBox = (GroupBox)hostItem.HostedControl;
//set the text of the group box to eqals the data item text
groupBox.Text = args.VisualItem.Data.Text;
//remove the visual item default text
args.VisualItem.Text = "";
}
```

The running project looks something like this:



Animation

You can animate aspects of RadElements based on a property change. The animation can be another property that is smoothly transitioned between states. The screenshot below shows the "FishEyeBar" project underway. When the mouse moves over an element, the element is scaled up to a larger size. As the mouse moves off the item, the item smoothly scales back down to its original size.





You can find the complete source for this project at:

\TPF\<VB|CS>\FishEyeBar

Examine the code for the FishEyeBar project. Most of the code we're interested in occurs in the WelcomeScreenBottom.cs user control. The starting point is to define a descendant of PropertyChangedBehavior. Notice that the constructor base() takes a reference to a RadProperty. This is the property that will trigger the behavior. Try changing this to a IsMouseDown property and the behavior will not happen on mouse over but only when you click the mouse button.

[VB] Inheriting from PropertyChangeBehavior

```
Public Class WelcomeScreenMouseOverBehavior
 Inherits PropertyChangeBehavior
 Public Sub New()
  MyBase.New(RadItem.IsMouseOverProperty)
 End Sub
 '...
End Class
```

[C#] Inheriting from PropertyChangeBehavior

```
public class WelcomeScreenMouseOverBehavior : PropertyChangeBehavior
{
 public WelcomeScreenMouseOverBehavior()
   : base(RadItem.IsMouseOverProperty)
   {
   }
   //...
}
```

Another critical piece for implementing the PropertyChangeBehavior is to override the OnPropertyChange method. In this method you define what animations will take place. This is done by creating an AnimatedPropertySetting and applying that setting to the element you want to animate. The AnimatedPropertySetting constructor lets you define the property to be animated and several overloads include:

- Properties for controlling the smoothness and speed of the animation, i.e. number of frames, step and interval between frames.
- Start and end property values.

In this example the ScaleTransformProperty is used to scale the element to a larger size.

[VB] Appling animation

```
Private oldCursor As Cursor = Nothing
Public Overrides Sub OnPropertyChange(ByVal element As RadElement, ByVal e As
RadPropertyChangedEventArgs)
For Each childItem As RadElement In element.Parent.Children
Dim item As RadItem = TryCast(childItem, RadItem)
If item IsNot Nothing Then
Dim animation As AnimatedPropertySetting
Dim scale As SizeF
If CBool(e.NewValue) = False Then
scale = New SizeF(1.0F, 1.0F)
If Me.oldCursor IsNot Nothing AndAlso element.ElementTree IsNot Nothing Then
element.ElementTree.Control.Cursor = Me.oldCursor
End If
Else
If item.Equals(element) Then
scale = New SizeF(1.1F, 1.1F)
Else
scale = New SizeF(0.9F, 0.9F)
End If
If element.ElementTree IsNot Nothing Then
Me.oldCursor = element.ElementTree.Control.Cursor
element.ElementTree.Control.Cursor = Cursors.Hand
End If
End If
animation = New AnimatedPropertySetting(RadElement.ScaleTransformProperty, scale, 10, 33)
animation.SkipToEndValueOnReplace = False
animation.ApplyValue(item)
End If
Next
End Sub
```

[C#] Appling animation

```
private Cursor oldCursor = null;
public override void OnPropertyChange(RadElement element, RadPropertyChangedEventArgs e)
{
foreach (RadElement childItem in element.Parent.Children)
{
RadItem item = childItem as RadItem;
if (item != null)
{
AnimatedPropertySetting animation;
SizeF scale;
if ((bool)e.NewValue == false)
{
scale = new SizeF(1F, 1F);
if (this.oldCursor != null && element.ElementTree != null)
{
element.ElementTree.Control.Cursor = this.oldCursor;
}
}
else
{
scale = item == element ? new SizeF(1.1F, 1.1F) : new SizeF(.9F, .9F);
if (element.ElementTree != null)
{
```

```
this.oldCursor = element.ElementTree.Control.Cursor;
element.ElementTree.Control.Cursor = Cursors.Hand;
}
}
animation = new AnimatedPropertySetting(RadElement.ScaleTransformProperty, scale, 10, 33);
animation.SkipToEndValueOnReplace = false;
animation.ApplyValue(item);
}
}
}
```

7.8 Summary

In this chapter you learned some of the capabilities, architecture and class hierarchy of the Telerik Presentation Foundation. We paid special attention to the classes used directly in the controls, i.e. RadElement and RadControl. We took a brief look at the structure, WPF resemblance and how the elements interact. We saw how the elements could be accessed within RadControls and could be combined to create new unique controls. You also learned how to add any Control descendant using RadHostItem as a wrapper.

8 PageView

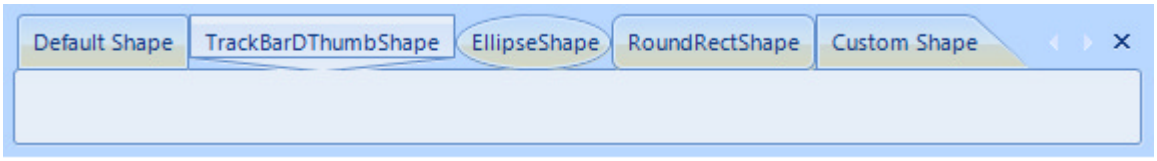
8.1 Objectives

- Tour the usability and feature highlights of RadPageView.
- Explore the tabbed interface and manage controls within associated content panels.
- Perform common Items collection programmatic tasks such as adding, removing, iterating, locating and selecting pages.
- Handle page layout and dimensions.
- Handle conditions where the number of pages overflow the available space.
- Work with page images, text and custom controls.
- Page drag and drop

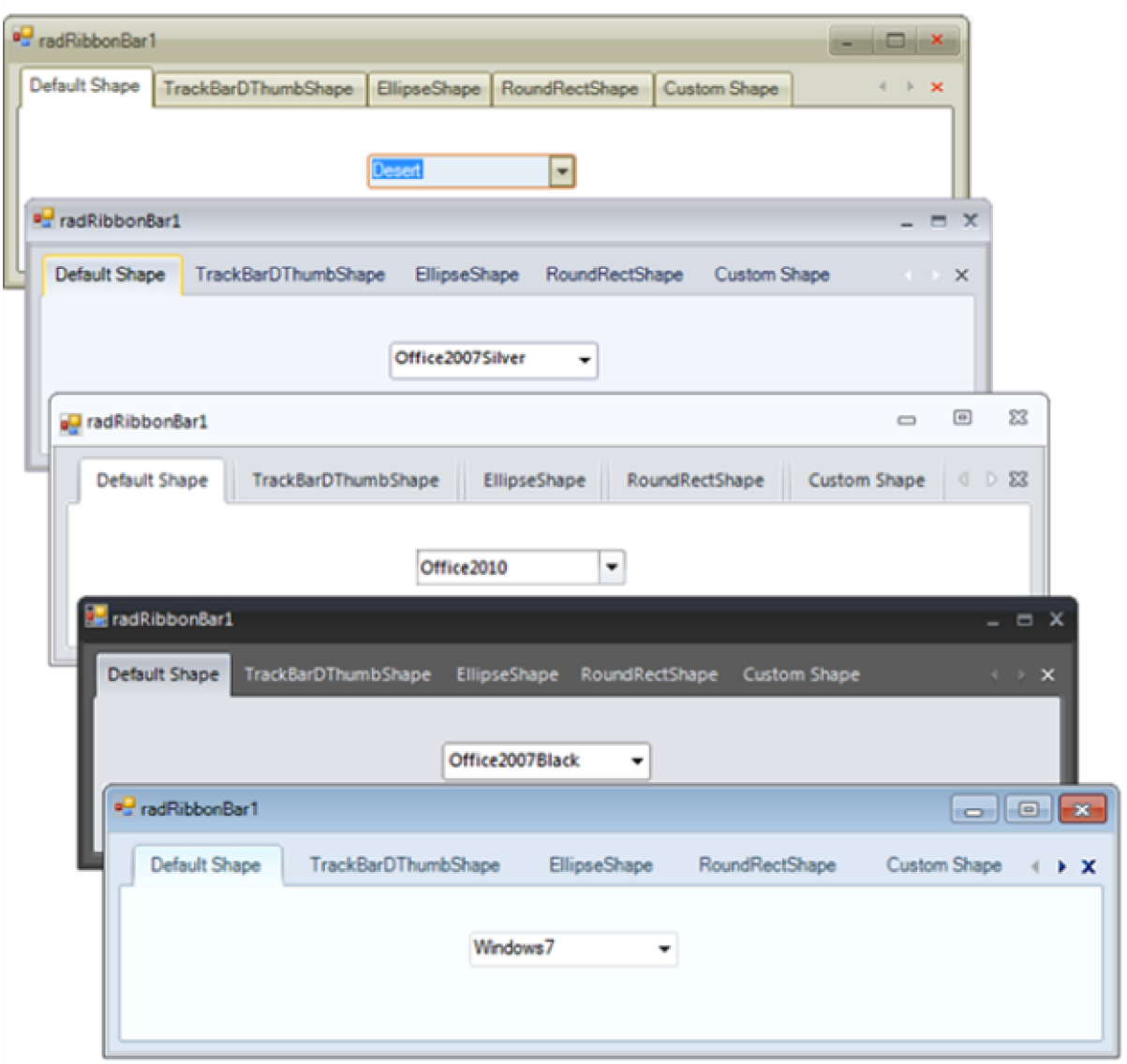
8.2 Introduction

A tabbed interface can supply intuitive navigation between application areas. RadPageView supplies the basis for building tabbed interfaces with rich formatting and behavior. Page items can have any shape you can imagine, be rearranged with drag-and-drop, edited, and automatically manage other controls on associated content panels. Some RadPageView highlights:

- Assign any shape to individual page items. The shape can be one of a predefined list or you can create a custom shape in the Shape Designer tool and assign it to the page item.

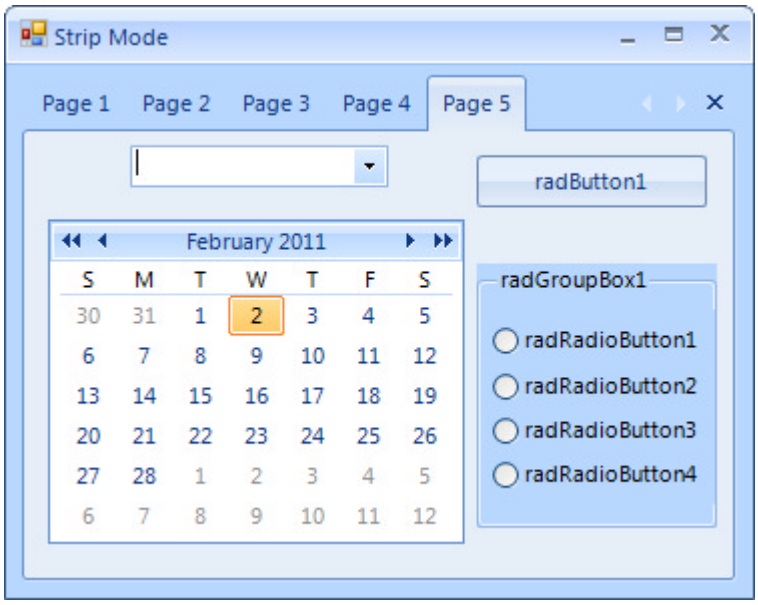


- Use themes to customize the appearance of RadPageView, including changing appearance on selection or mouse over.

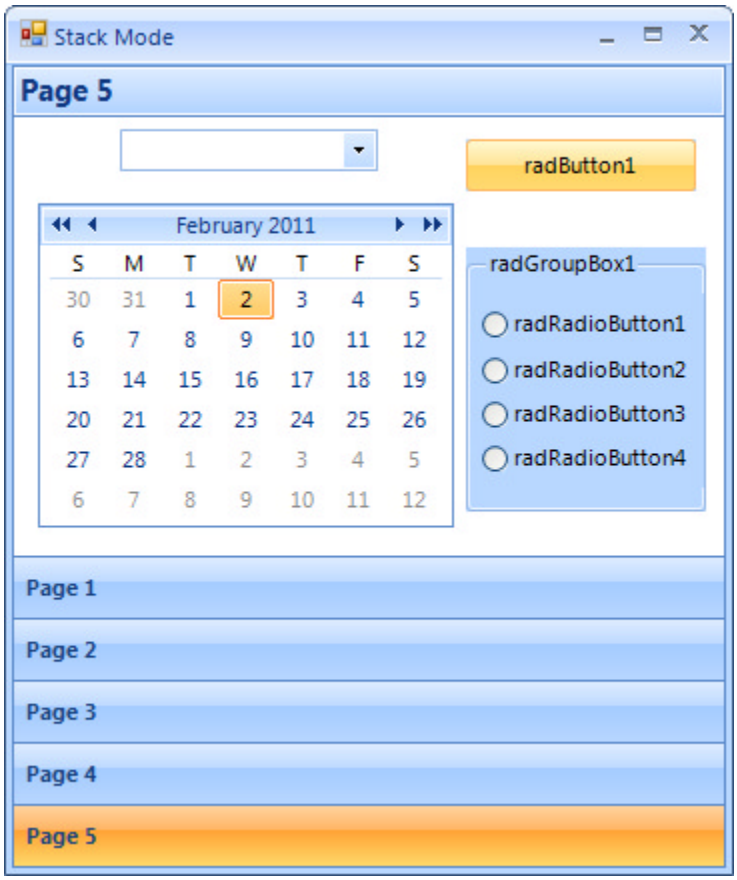


- Take advantage of the different ViewModes:

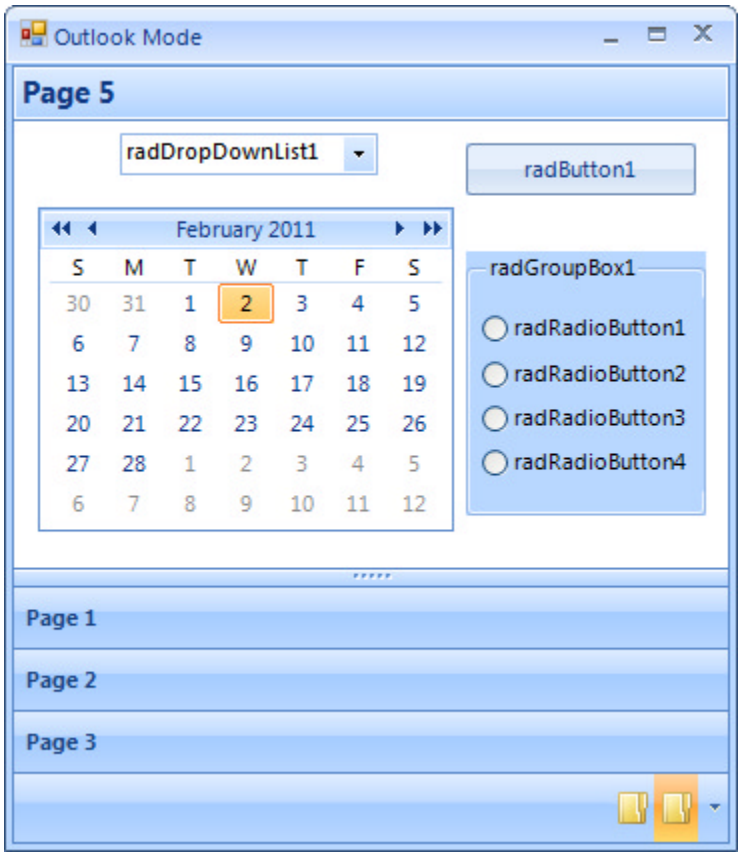
- Strip Mode



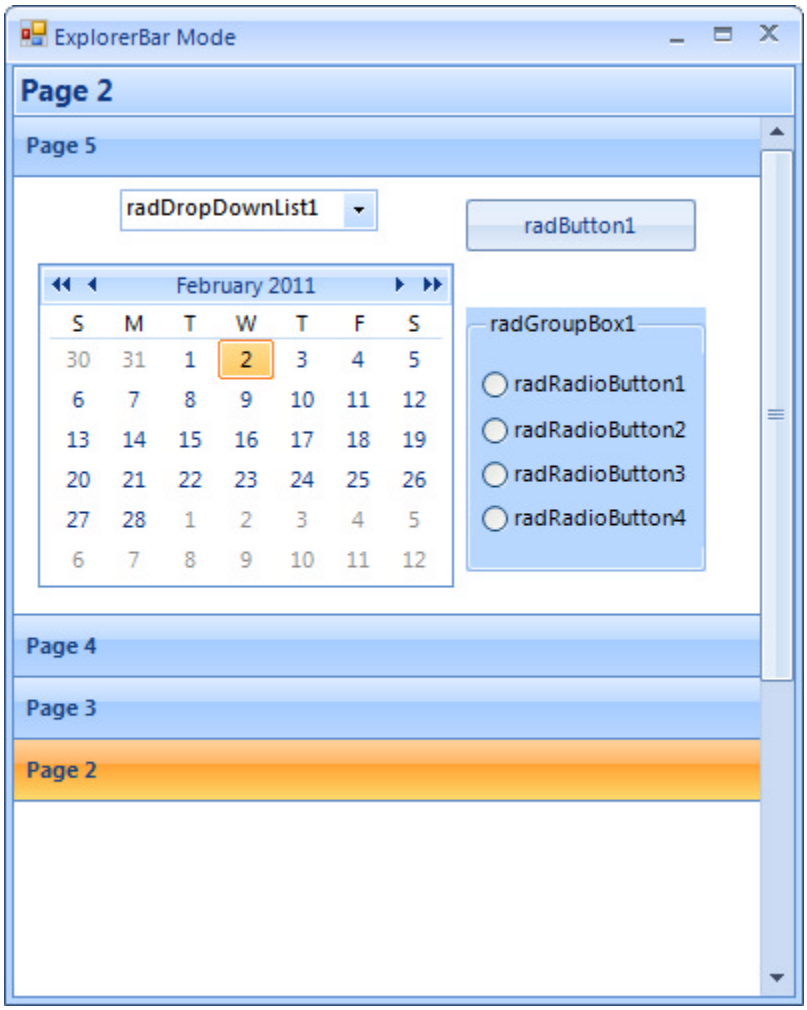
- Stack Mode



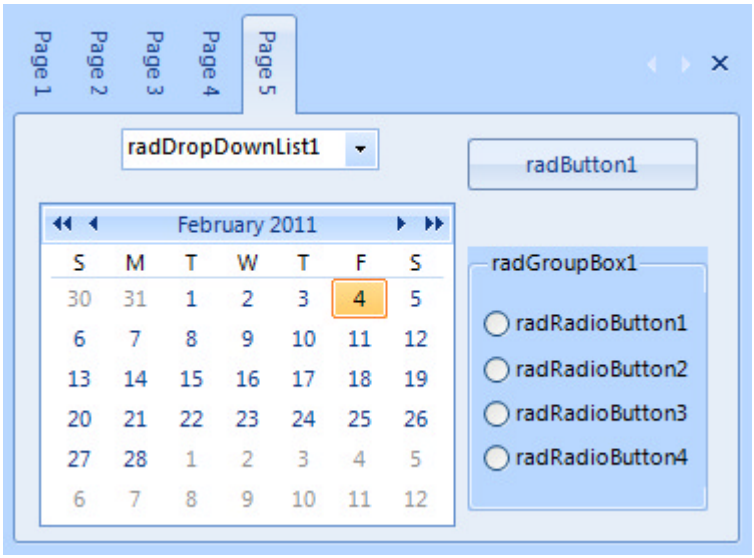
- Outlook Mode

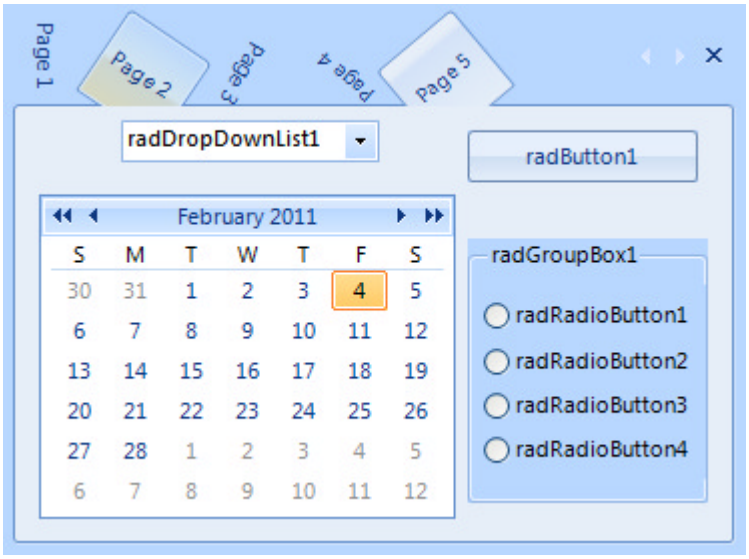


- ExplorerBar Mode

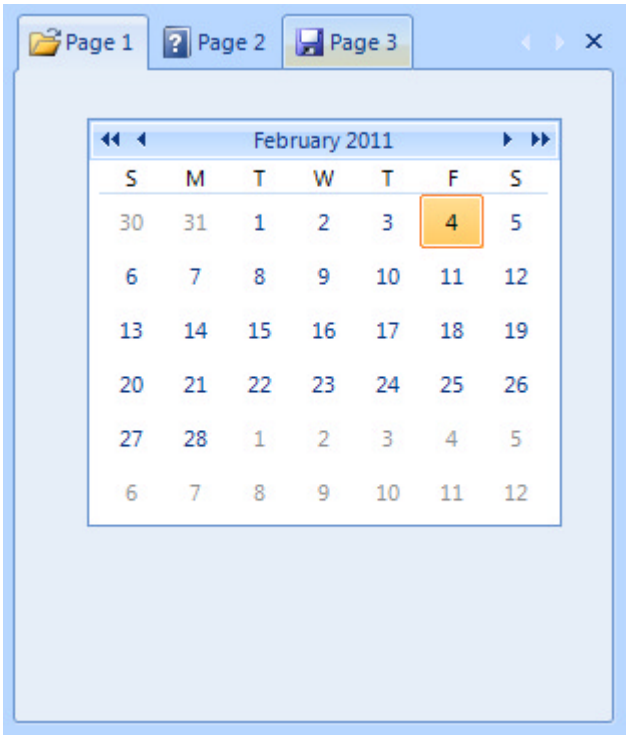


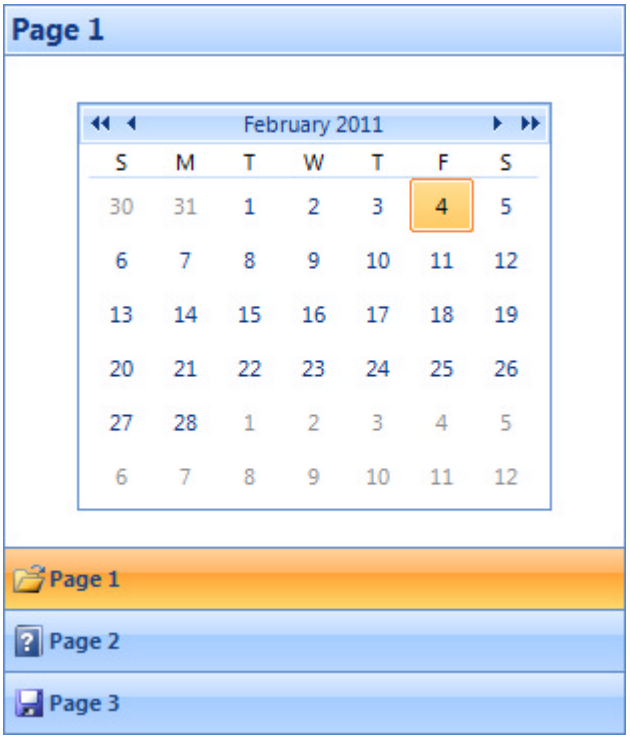
- Get fine grain control over the layout of the page. Display text in horizontal or vertical orientation, or at any angle. Anchor page items on any side of the control.

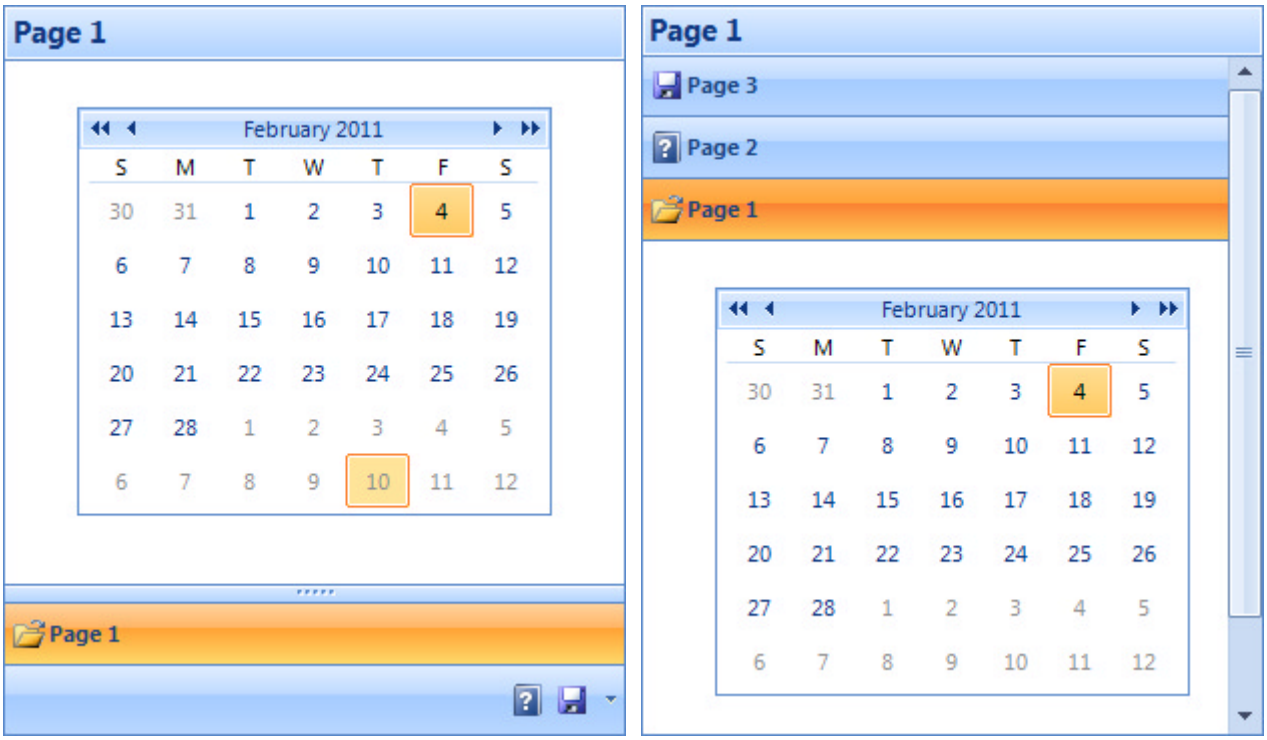




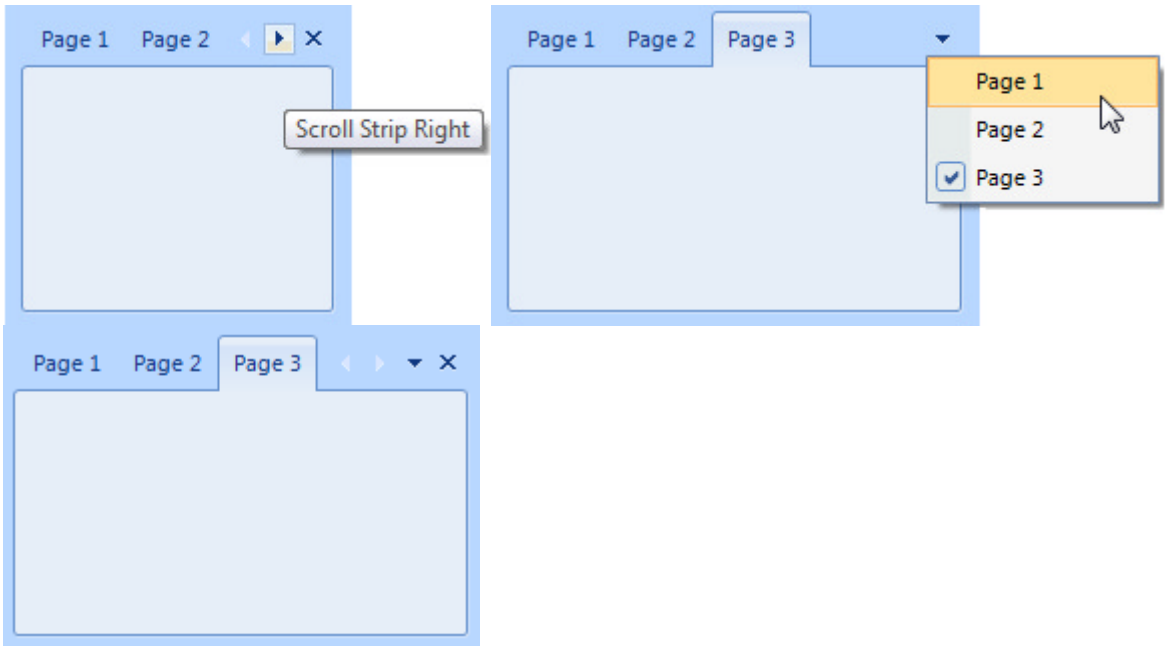
- Display images and text on page items. You can control the layout of the picture relationship to the text.



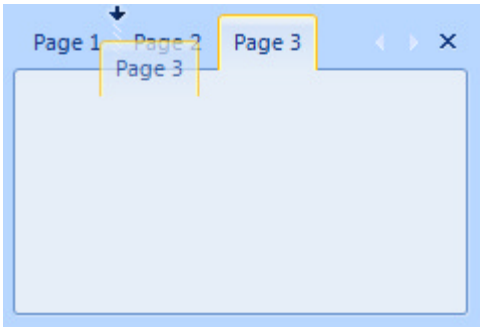




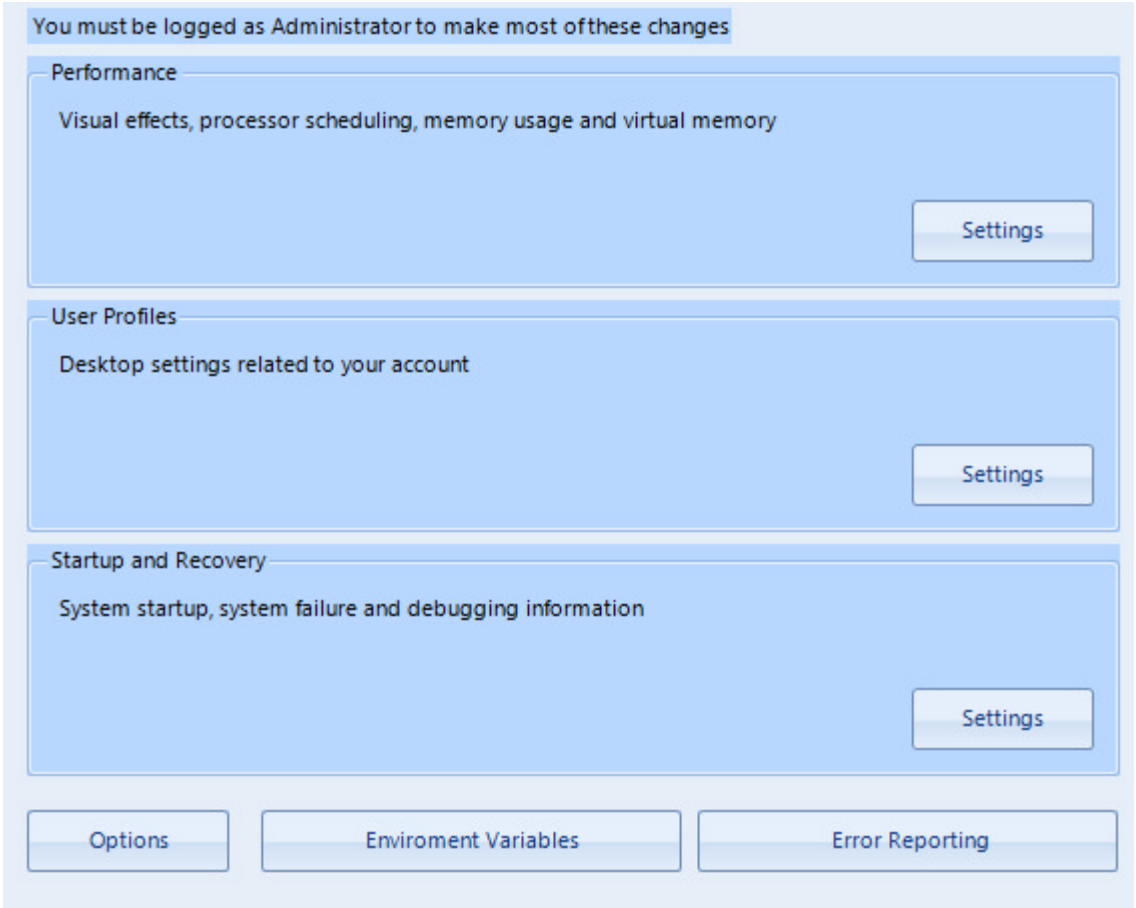
- Allow page scrolling and overflow management if there are more pages than can be displayed at one time.



- Enable runtime page rearrangement via drag-and-drop.

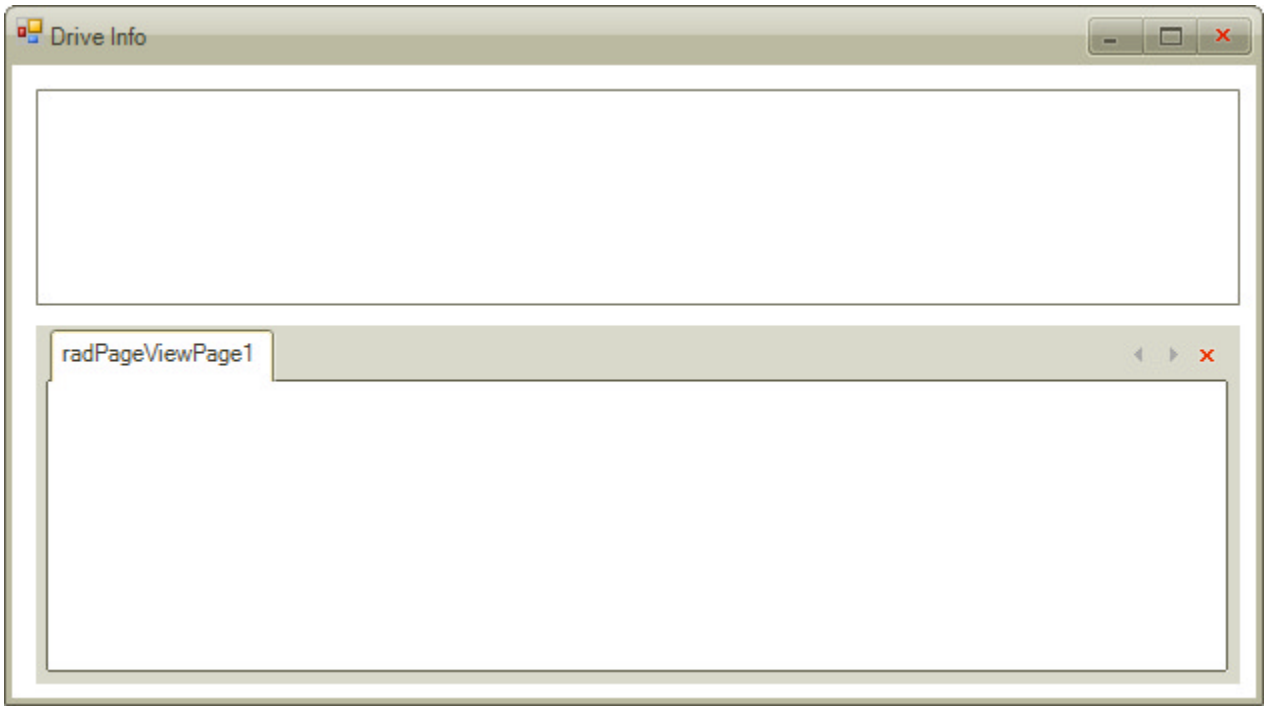


- You can also add different controls to the pages, arrange them just like in a form or panel.



8.3 Getting Started

In this project a RadPageView will show pages for several drive types. The "C:\" drive page content panel hosts a RadListControl populated with a list of files on the C:\ drive.



Preparing the Project



You can find the complete source for this project at:

\Tabs\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.

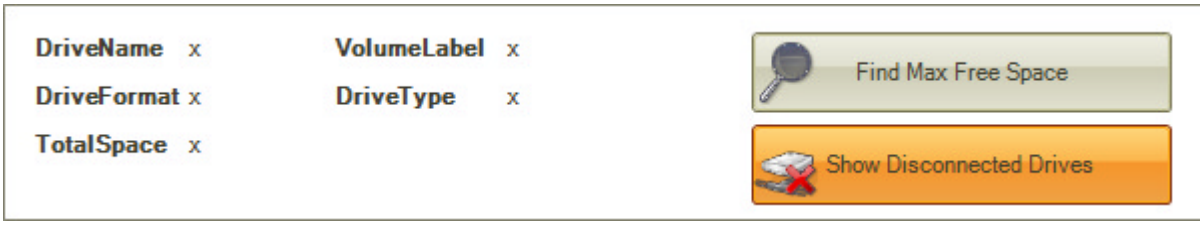
Adding Controls to the RadForm

1. Drag **DesertTheme** and **ImageList** components to the RadForm. This step will add the components to the component tray below the form designer.
2. Click the ImageList Smart Tag "Choose Images" option and add images to represent the types of drives listed below.
    - Unknown
    - NoRootDirectory
    - Removable
    - Fixed
    - Network
    - CDRom
    - Ram

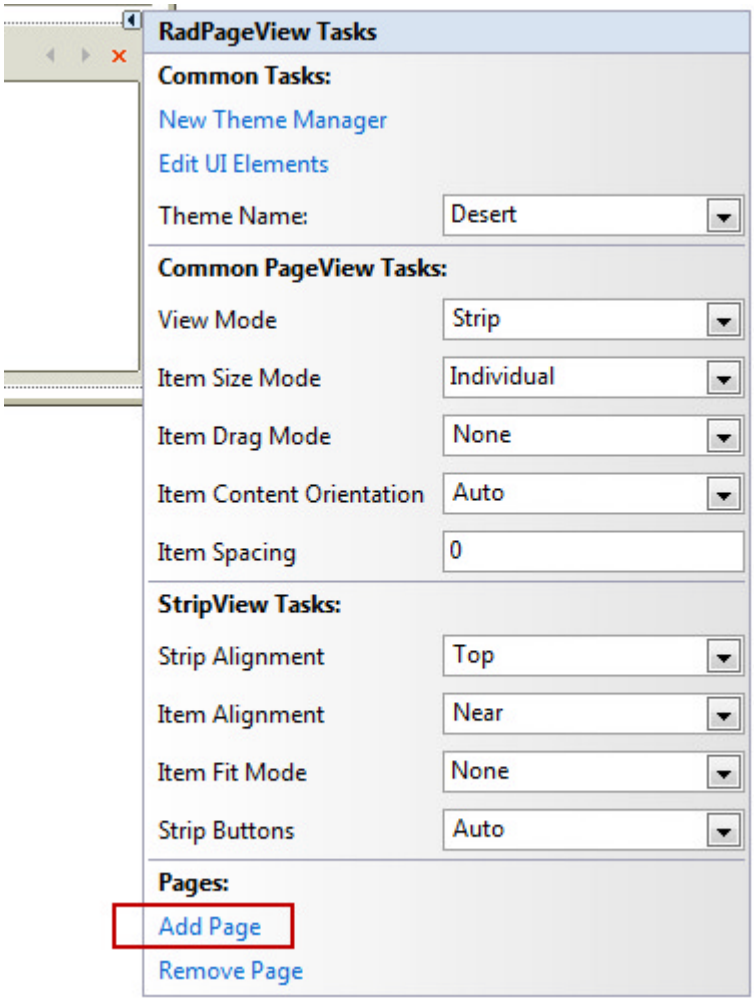
- Search
- Disconnected Drive

> You can typically find images in the Visual Studio "Common7\VS2008ImageLibrary\1033" directory in "VS2008ImageLibrary.zip".

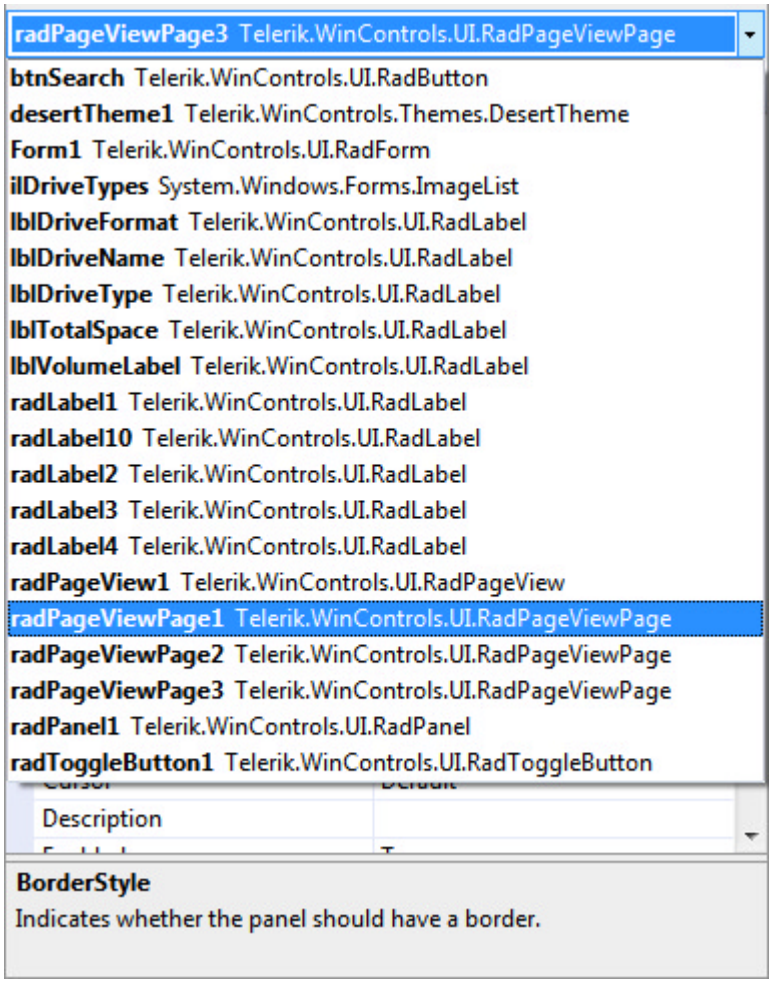
> The images for the screenshots used in this example were found in the zipped image library under "VS2008ImageLibrary\Objects\png_format\WinVista" directory.



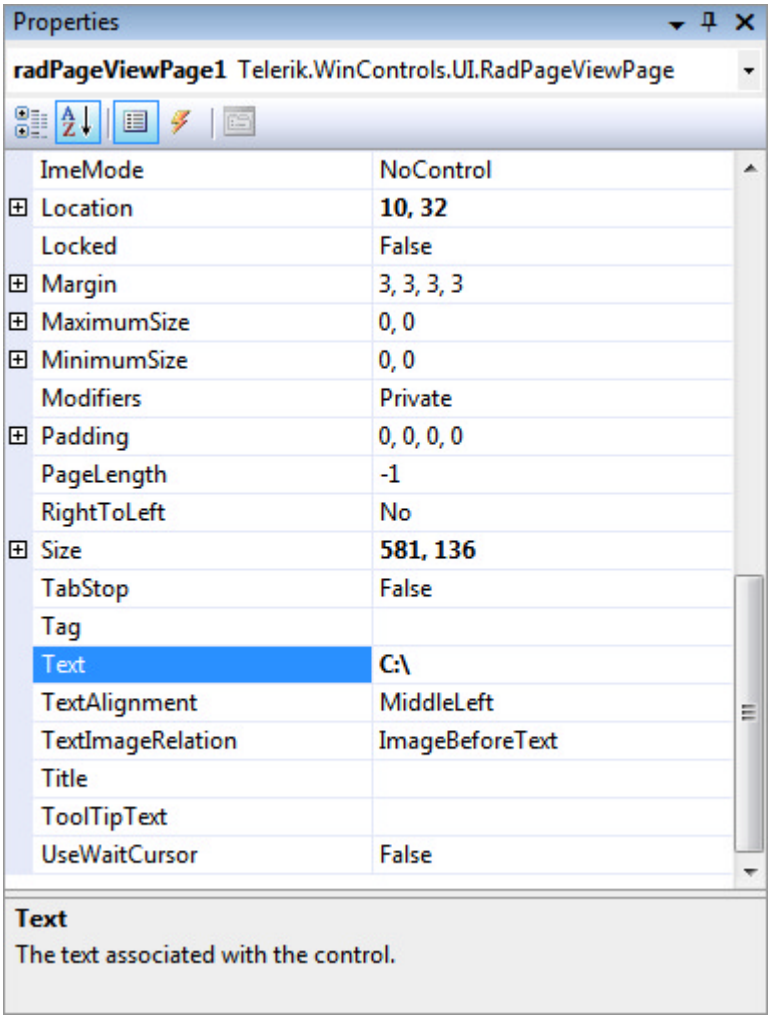
3. In the Properties window set the RadForm Properties:
   - **Size** = 630, 350
   - **MinimumSize** = 630, 350
   - **Text** = "Drive Info"
   - **ThemeName** = "Desert"
4. Drop a **RadPanel** and a **RadPageView** on the form and arrange them to roughly the same proportions as the example below.
   - RadPanel: **Text** = "", **ThemeName** = "Desert", **Anchor** = "Top, Left, Right".
   - RadPageView: **ThemeName** = "Desert", **Anchor** = "Top, Bottom, Left, Right".



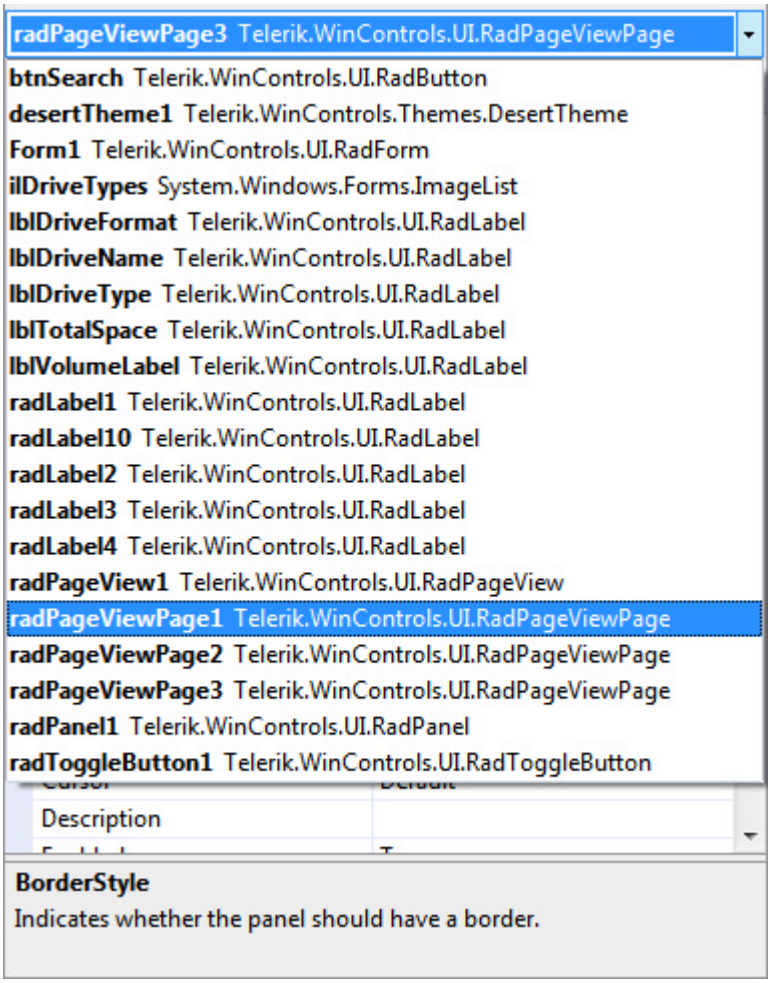
5. Add controls to the RadPanel and set properties:
   - RadLabel: **Text** = "Drive Name:"
   - RadLabel: **Name** = "lblDriveName", **Text** = "x".
   - RadLabel: **Text** = "Drive Format".
   - RadLabel: **Name** = "lblDriveFormat", **Text** = "x".
   - RadLabel: **Text** = "Total Space:".
   - RadLabel: **Name** = "lblTotalSpace", **Text** = "x".
   - RadLabel: **Text** = "Volume Label:".
   - RadLabel: **Name** = "lblVolumeLabel", **Text** = "x".
   - RadLabel: **Text** = "Drive Type:".
   - RadLabel: **Name** = "lblDriveType", **Text** = "x".
   - RadButton: **Name** = "btnSearch", **Text** = "Find Max Free Space", **ThemeName** = "Desert", **Anchor** = "Top, Right", **ImageList** = "ilDriveTypes", **ImageKey** = <the key for the "search" image>".
   - RadToggleButton: **Name** = "btnShow", **Text** = "Show Disconnected Drives", **ThemeName** = "Desert", **Anchor** = "Top, Right", **ImageList** = "ilDriveTypes", **ImageKey** = <the key for the "disconnected drive" image>, **IsChecked** = True.

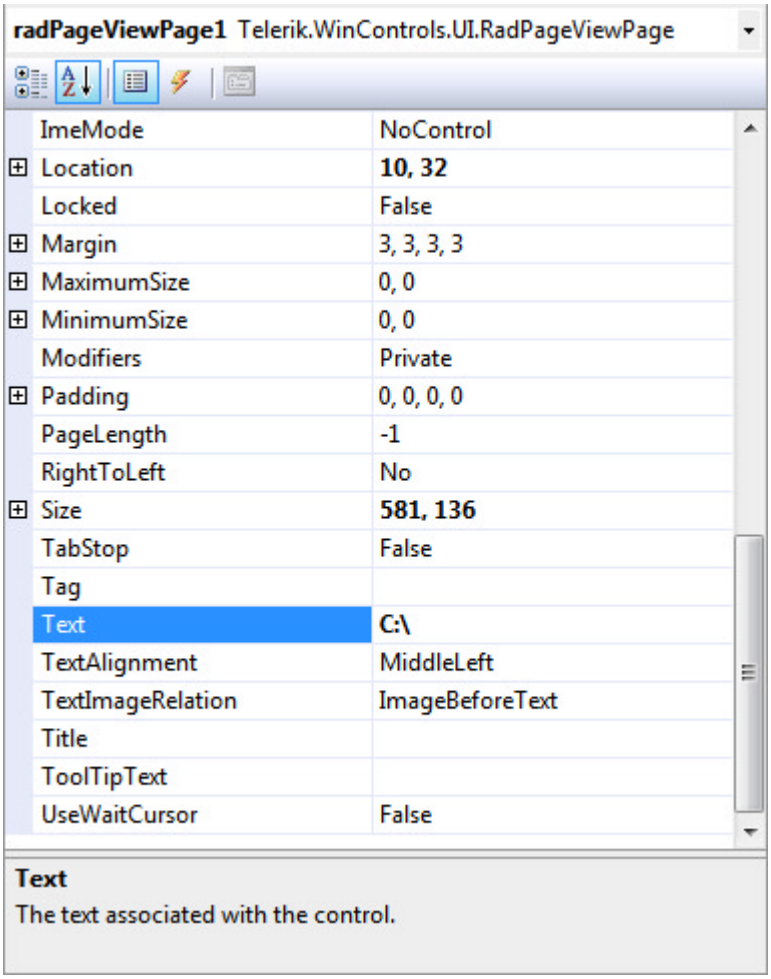


6. Add two more pages from the Smart Tag(to get a total of three)

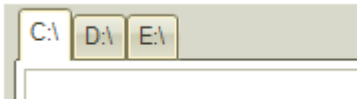


7. Set their names - the first page to "C:\", the second to "D:\" and the last one to "E:\".


radPageViewPage3 Telerik.WinControls.UI.RadPageViewPage
btnSearch Telerik.WinControls.UI.RadButton
desertTheme1 Telerik.WinControls.Themes.DesertTheme
Form1 Telerik.WinControls.UI.RadForm
ilDriveTypes System.Windows.Forms.ImageList
lblDriveFormat Telerik.WinControls.UI.RadLabel
lblDriveName Telerik.WinControls.UI.RadLabel
lblDriveType Telerik.WinControls.UI.RadLabel
lblTotalSpace Telerik.WinControls.UI.RadLabel
lblVolumeLabel Telerik.WinControls.UI.RadLabel
radLabel1 Telerik.WinControls.UI.RadLabel
radLabel10 Telerik.WinControls.UI.RadLabel
radLabel2 Telerik.WinControls.UI.RadLabel
radLabel3 Telerik.WinControls.UI.RadLabel
radLabel4 Telerik.WinControls.UI.RadLabel
radPageView1 Telerik.WinControls.UI.RadPageView
radPageViewPage1 Telerik.WinControls.UI.RadPageViewPage
radPageViewPage2 Telerik.WinControls.UI.RadPageViewPage
radPageViewPage3 Telerik.WinControls.UI.RadPageViewPage
radPanel1 Telerik.WinControls.UI.RadPanel
radToggleButton1 Telerik.WinControls.UI.RadToggleButton
Description
BorderStyle
Indicates whether the panel should have a border.



The result is:



8. Assign the appropriate images from the ImageList to the pages of RadPageView in code:

[VB] Set the images to RadPageView pages

```
RadPageViewPage1.Image = ilDriveTypes.Images("Hard_Drive.png")
RadPageViewPage2.Image = ilDriveTypes.Images("CD_ROM.png")
RadPageViewPage3.Image = ilDriveTypes.Images("Network.png")
```

[C#] Set the images to RadPageView pages

```
radPageViewPage1.Image = ilDriveTypes.Images["Hard_Drive.png"];
radPageViewPage2.Image = ilDriveTypes.Images["CD_ROM.png"];
radPageViewPage3.Image = ilDriveTypes.Images["Network.png"];
```

9. Click the "C:\" page and drop a RadListControl into the content area, just below the page button itself. Set the **Name** property of the list box to "lbFiles", **Dock** = "Fill", **ThemeName** = "Desert".

For more detail about RadListControl, please see the upcoming ListControl chapter.

10. Add **System.IO** and **Telerik.WinControls.UI** references to your "Imports" (VB) or "uses" (C#) clause.
11. Double-click the form to create a form Load event handler. Add the code below.

    This code will use the System.IO Directory.GetDirectories() method to retrieve a list of directories from the "C:\" drive and populate the list control.

    [VB] Populating the List Box

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 Dim directories As String() = Directory.GetDirectories("C:\")
 For Each directory As String In directories
  lbFiles.Items.Add(New RadListDataItem(directory))
 Next
End Sub
```

    [C#] Populating the List Box

```
private void RadForm1_Load(object sender, EventArgs e)
{
string[] directories = Directory.GetDirectories("C:\\");
foreach (string directory in directories)
{
lbFiles.Items.Add(new RadListDataItem(directory));
}
}
```

12. Set the **DriveName** label (lblDriveName) Text to the initially loaded page text:
13. In the Properties window events, locate the **SelectedPageChanged** event for RadPageView. Double-click the event to create an event handler and add the code below.

    The code here retrieves the selected page text and assigns it to the label in the panel above when page change is performed.

    [VB] Handling the TabSelected Event

```
Private Sub radPageView1_SelectedPageChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles radPageView1.SelectedPageChanged
lblDriveName.Text = radPageView1.SelectedPage.Text
End Sub
```

    [C#] Handling the TabSelected Event

```
private void radPageView1_SelectedPageChanged(object sender, EventArgs e)
{
lblDriveName.Text = radPageView1.SelectedPage.Text;
}
```

14. Press **Ctrl-F5** to run the application. Notice the behavior of the tabs and content panel area as you click between the tabs. Also notice how "lblDriveName" reflects the currently selected tab.

8.4 Using the Design Time Interface

RadPageView Smart Tag

The items of the RadPageView Smart Tag include Common tasks, Common PageView tasks which apply to all ViewModes, specific StripView tasks and Pages.

Common Tasks

From the Smart Tag Tasks menu you can select New Theme Manager to add a Theme Manager component to the

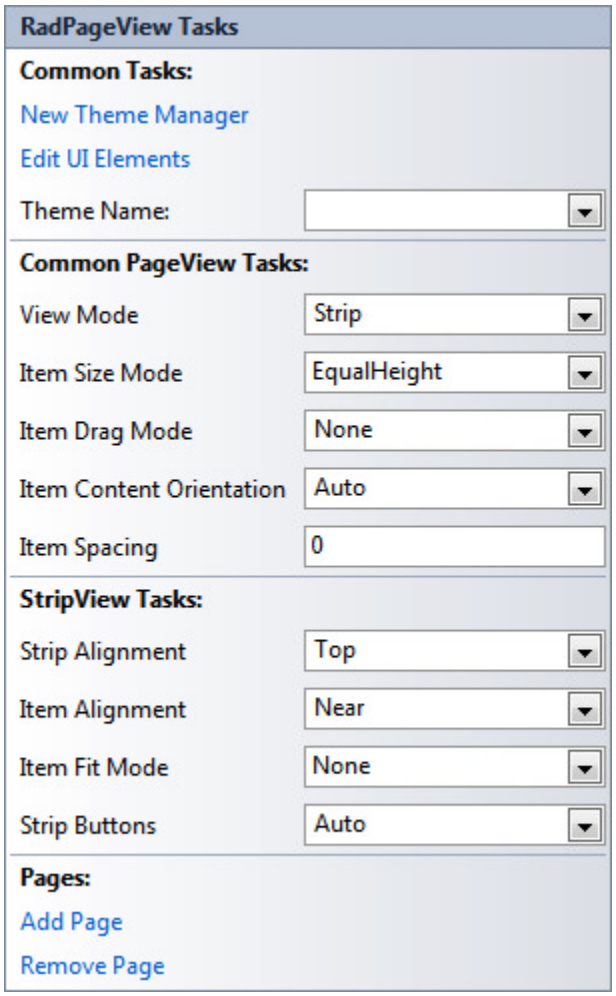
component tray, Edit UI elements to browse all of the elements of the editor and to change properties for any element. The Theme Name drop down lets you pick an existing theme to style your control.

Common PageView Tasks:

- **ViewMode** lets you change the mode in which RadPageView will be displayed - *Strip*, *Stack*, *Outlook* or *ExplorerBar* mode
- **Item Size Mode** determines whether the items should have equal size. The supported modes are EqualHeight, EqualWidth, EqualSize and Individual.
- **Item Drag Mode** determines the drag mode of the items. The supported modes are Preview and Immediate.
- **Item Content Orientation** determines the orientation of the item content (image, text, and buttons' panel). The supported orientations are Auto, Horizontal, Horizontal180, Vertical90 and Vertical270.
- **Item Spacing** determines the spacing between the items in pixels.

StripView tasks:

- **Strip Alignment** determines the position of the tab items in RadPageView in relation to the content area. It can be Top, Left, Bottom or Right.
- **Item Alignment** determines the alignment of items within the strip layout.
- **Item Fit Mode** determines the fit mode of the items. The available modes are None, Fill, Shrink and ShrinkAndFill.
- **Strip Buttons** determines the style of the navigation buttons. The available styles are None, Auto, LeftScroll, RightScroll, Scroll, VS2005Style, Close, ItemList, VS2008Style, All.



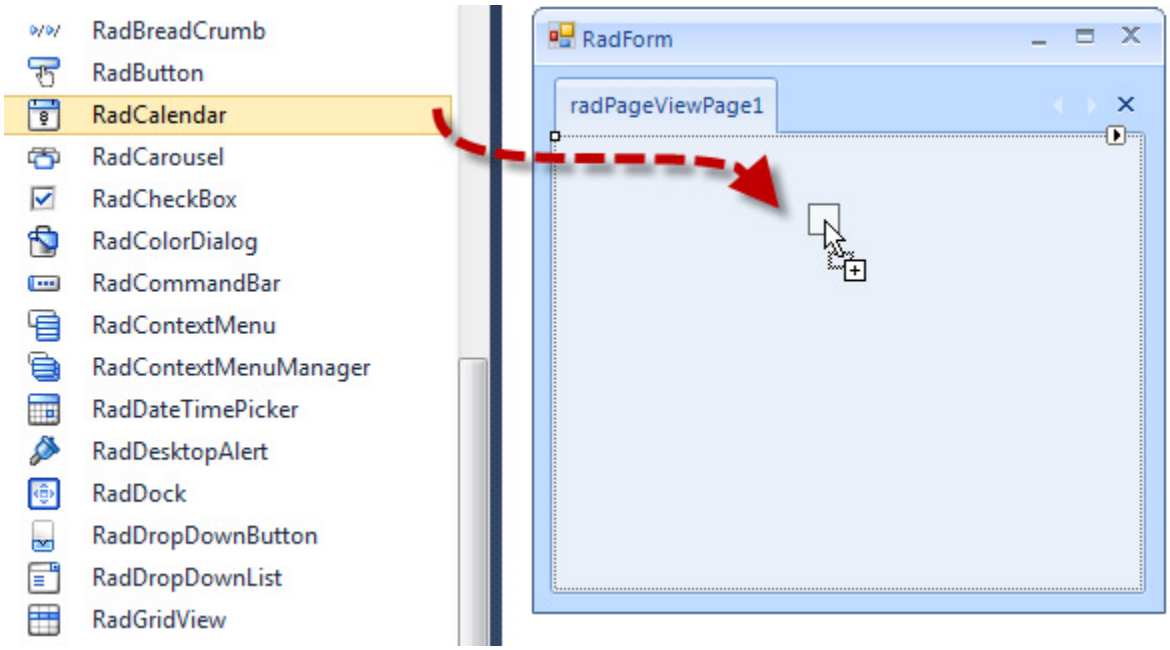
Pages

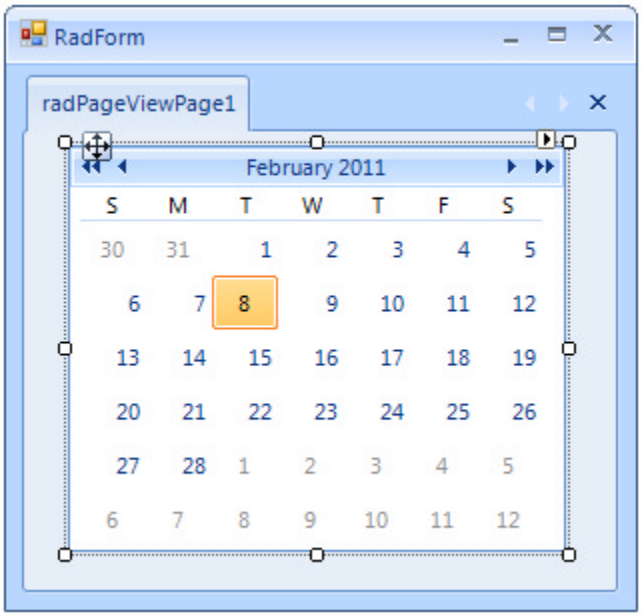
- **Add Page** allows you to add new page to RadPageView
- **Remove Page** - removes the last added page.

8.5 Working with RadPageView

Add controls to the Contect Area?

Adding controls to the content area can be done in both design and run-time. Adding controls in design time is achieved by adding new RadPageViewPage to RadPageView and then adding the desired control from the toolbox to the page:





You can also add controls to RadPageView content area programmatically:

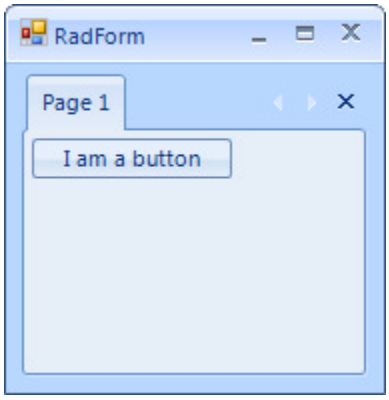
[VB] Adding Controls to RadPageViewPage

```
Dim page1 As New RadPageViewPage()
page1.Text = "Page 1"
radPageView1.Pages.Add(page1)
Dim button As New RadButton()
button.Text = "I am a button"
button.Size = New System.Drawing.Size(100, 20)
page1.Controls.Add(button)
```

[C#] Adding Controls to RadPageViewPage

```
RadPageViewPage page1 = new RadPageViewPage();
page1.Text = "Page 1";
```

```
radPageView1.Pages.Add(page1);
RadButton button = new RadButton();
button.Text = "I am a button";
button.Size = new System.Drawing.Size(100,20);
page1.Controls.Add(button);
```





Gotcha!

You can also access RadPageViewPage by index or the Text of the page into the page view Pages collection,

[VB] Accessing Items Collection

```
radPageView1.Items(0).ContentPanel.Controls.Add(button)
radPageView1.Items("Page 1").ContentPanel.Controls.Add(button)
```

[C#] Accessing Items Collection

```
radPageView1.Items[0].ContentPanel.Controls.Add(button);
radPageView1.Items["Page 1"].ContentPanel.Controls.Add(button);
```

Page Scrolling and Overflow

RadPageView has the ability to scroll or display a drop-down list to handle tabs that overflow the available space. The StripButtons property of RadPageView in Strip View mode allows the user to scroll or get a drop-down list of the tabs that overflow the available space. There is also a close button which, when pressed, closes the currently selected page. The StripButtons property is available at PadPageViewStripElement which can be accessed through the ViewElement property when RadPageView is in Strip mode, for example:

[VB] Accessing the StripButtons property

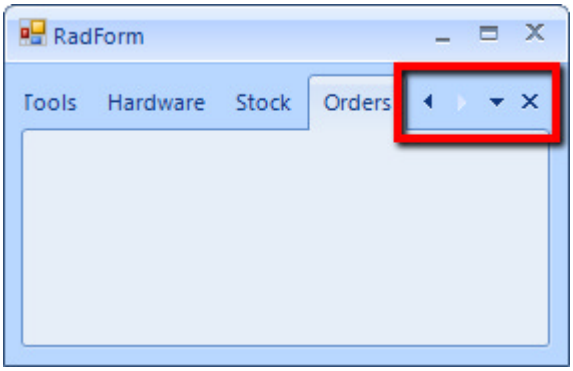
```
Dim stripElement As RadPageViewStripElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewStripElement)
stripElement.StripButtons = StripViewButtons.All
```

[C#] Accessing the StripButtons property

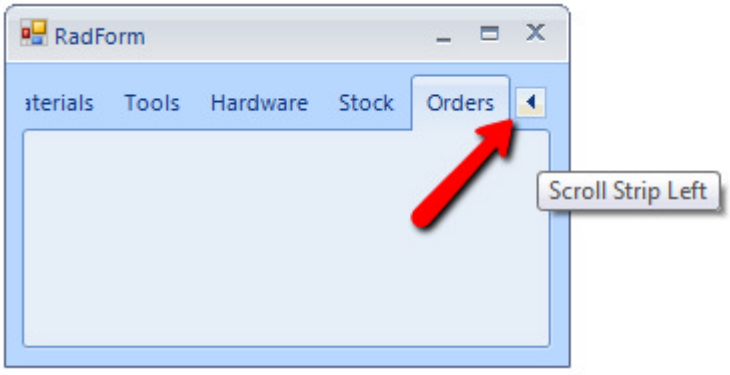
```
RadPageViewStripElement stripElement = (RadPageViewStripElement)
this.radPageView1.ViewElement;
stripElement.StripButtons = StripViewButtons.All;
```

Strip mode of RadPageView supports the following **StripButtons** modes:

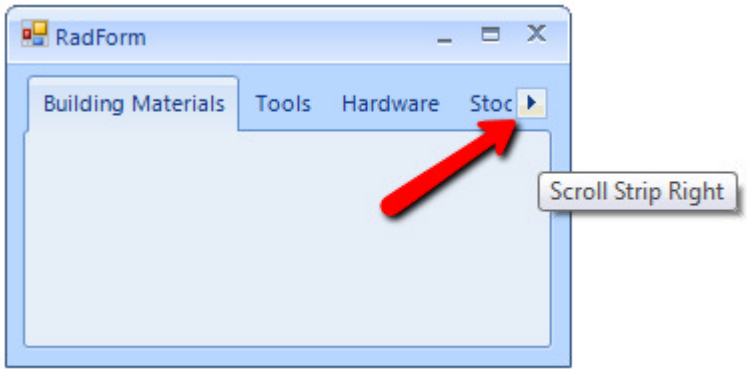
- **All**. When this option is selected. RadPageView will display all the available strip buttons - *left scroll button, right scroll button, overflow button* and *close button:*



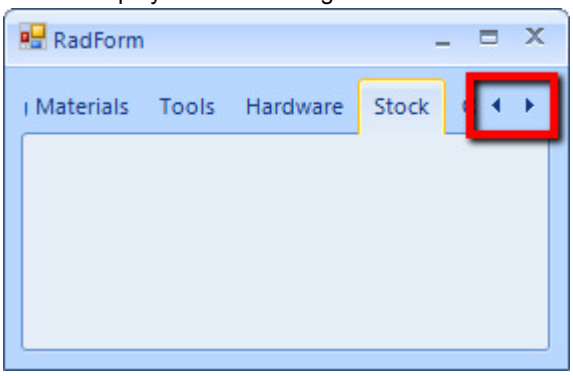
- **LeftScroll**. This option displays only the left scroll button:



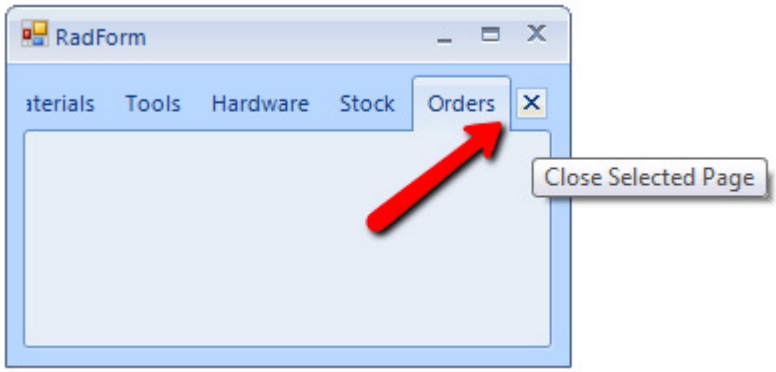
- **RightScroll**. This option displays only the right scroll button:



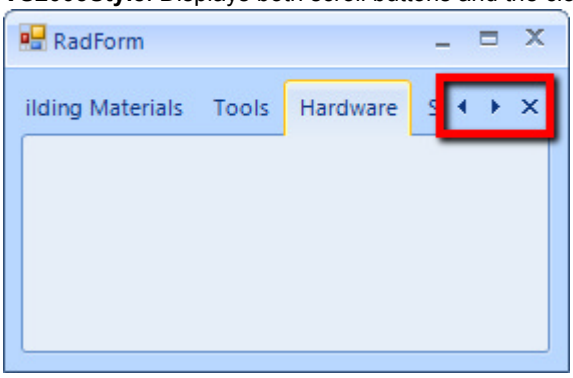
- **Scroll**. Display both left and right scroll buttons:



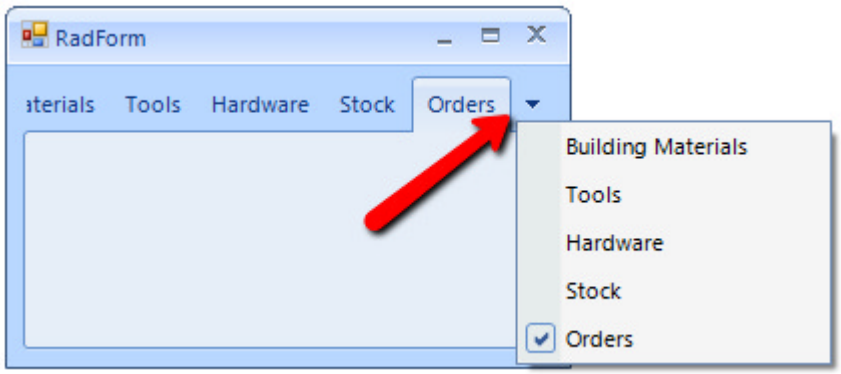
- **Close**. Displays only the close strip button.



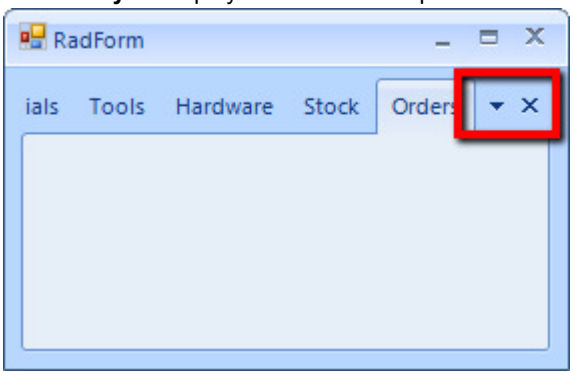
- **VS2005Style**: Displays both scroll buttons and the close button**:**



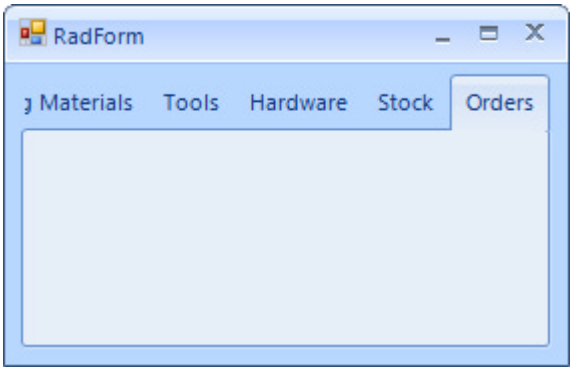
- **ItemList**. Displays a drop-down button that invokes a drop-down list of the items:



- **VS2008Style.** Displays the item list drop-down button and the close button:

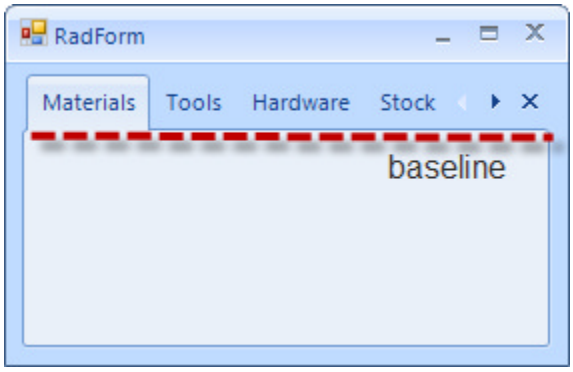


- **None**. Displays no strip buttons:



Layout

By default, when you drop RadPageView on the Form, the **StripAlignment** property is set to Top, and its member tabs are inserted with their **TextOrientation** property set to Horizontal. These default settings cause the tabs to appear above the RadPageView's baseline, and the text is oriented parallel to the baseline:



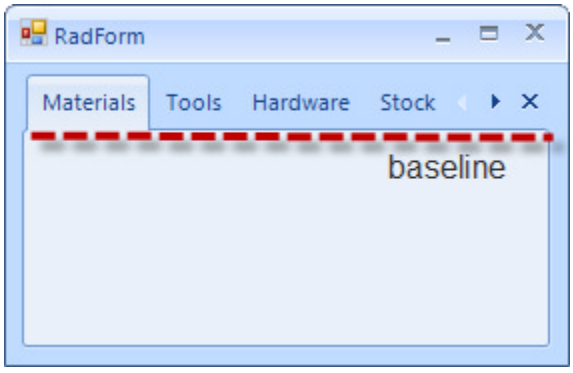
The RadPageView StripAlignment property can be set to **Top**, **Bottom**, **Left**, or **Right**:

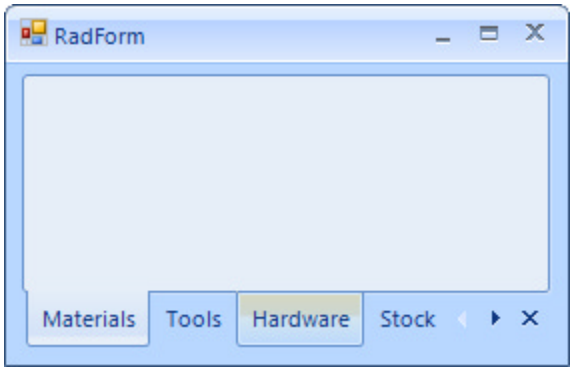
[VB] Set the strip alignment

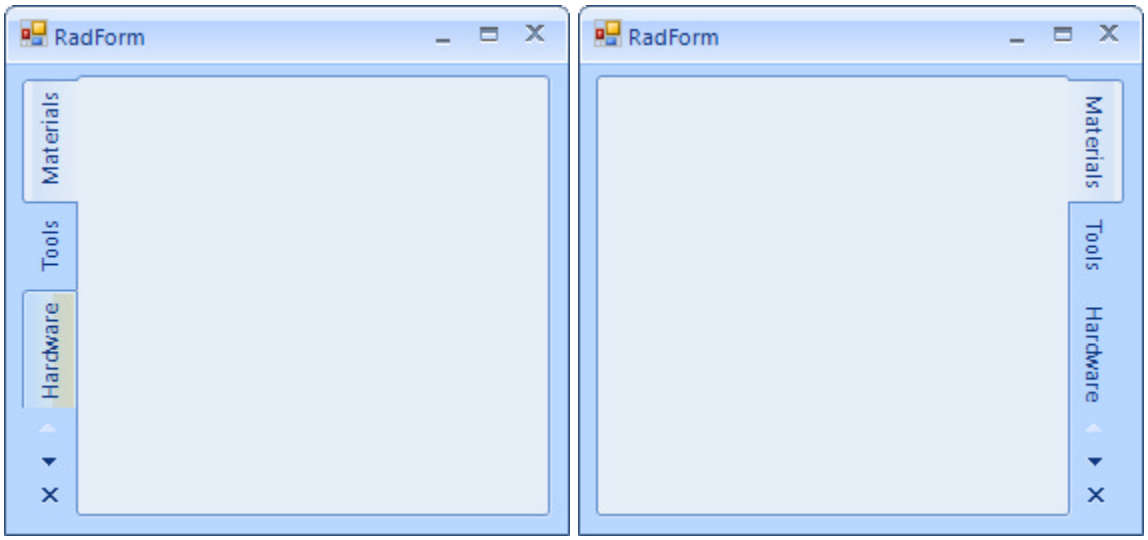
```
Dim element As RadPageViewStripElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewStripElement)
element.StripAlignment = StripViewAlignment.Left
```

[C#] Set the strip alignment

```
RadPageViewStripElement element = (this.radPageView1.ViewElement as
RadPageViewStripElement);
element.StripAlignment = StripViewAlignment.Left;
```







TextOrientation can be **Vertical** or **Horizontal** to the current baseline. The example below has tabs positioned to the left and TextOrientation is Vertical (i.e. perpendicular to the baseline):

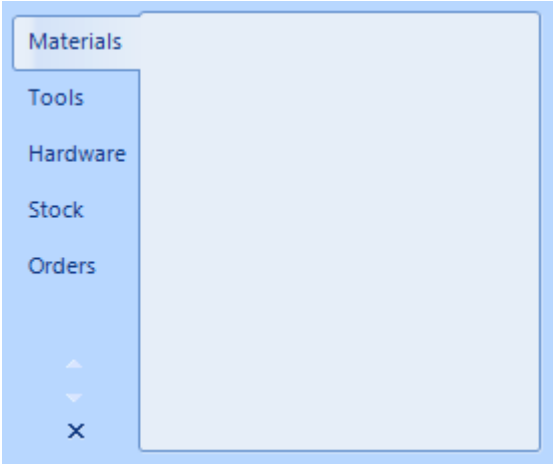
[VB] Set the strip alignment

```
Dim element As RadPageViewStripElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewStripElement)

element.TextOrientation = Orientation.Vertical;
```

[C#] Set the strip alignment

```
RadPageViewStripElement element = (this.radPageView1.ViewElement as
RadPageViewStripElement);
element.TextOrientation = Orientation.Vertical
```



Item Size Mode has four options to choose from:

- **EqualHeght** - makes the height for all tabs the same, even when images or multiple lines of text make the content of some tabs larger. Otherwise, each tab height is automatically sized to fit the content.

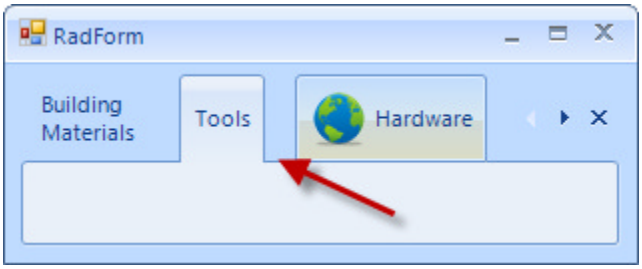


Multiple Tab Lines

You can get multiple lines of text in a tab by either appending Environment.NewLine programmatically to the Text property string.

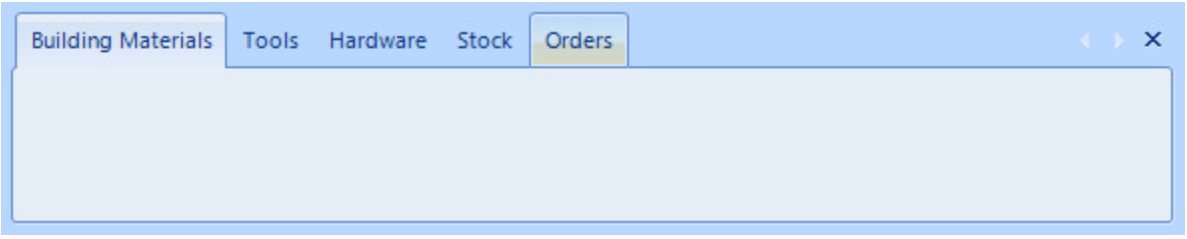
- **EqualWidth** respectively makes the width of all tabs the same
- **EqualSize** makes both width and height for all tabs the same
- **Individual** this mode autosizes all tabs according to their content (text or image) individually

ItemSpacing set the spacing between the different tabs. The following image demonstrates this property set to 15 pixels.

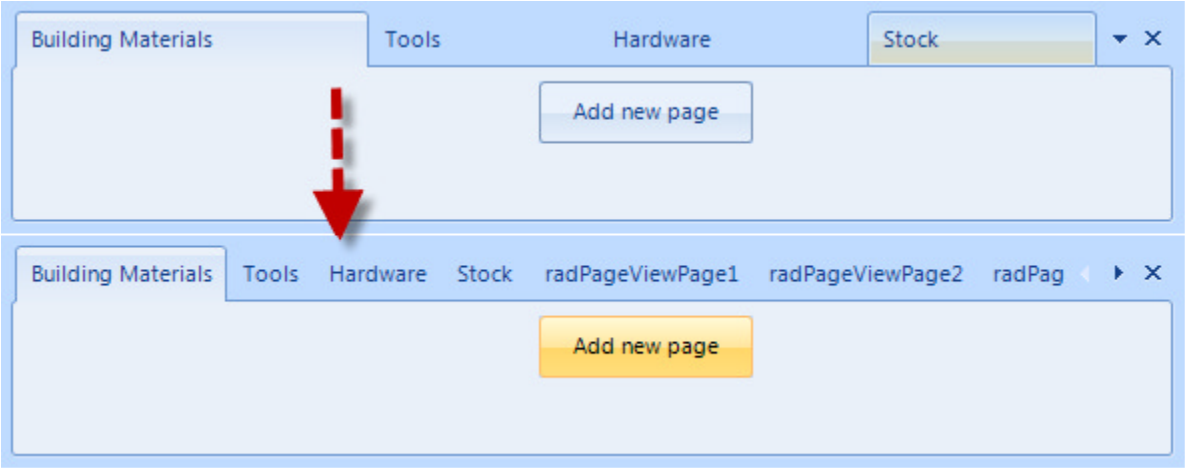


ItemFitMode

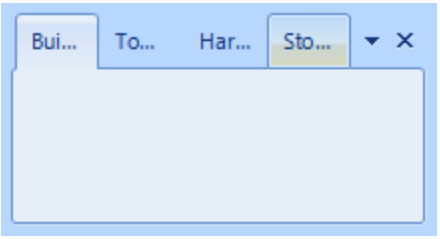
By default, tab width is sized to according to its content - **ItemFitMode** is **None:**



When the **ItemFitMode** property is set to **Fill**, tab width takes up all the available horizontal space. As more tabs are added, they shrink (up to the moment when the tabs content can be visible) to accomodate new arrivals. The screenshot below shows page view with ItemFitMode set to Fill and four initial tabs. After pressing a button that adds new tabs programmatically, the tabs shrink to fit all tabs until there is enough room for them without disturbing the content of the tabs.



In the case where **ItemFitMode** is set to **Shrink** the items are shrinking when there is not enough room for them, without considering the tab content:



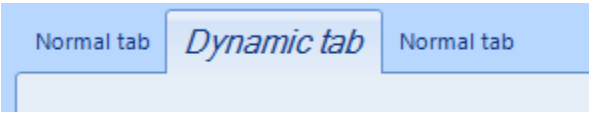
The last option is **ShrinkAndGrow** and as its name speaks it combines the Fill and the Shrink modes - the tabs are filling the entire space (when such exists) and they shrink when there is no enough room to fit them.

8.6 Working with RadPageViewPages

In addition to layout properties, each RadPageViewPage object in RadPageView Pages collection is highly configurable.

Text and Font

In the code and screenshot below, TabItem object **Text** and **Font** properties control the text appearance:



[VB] Setting Text and Font

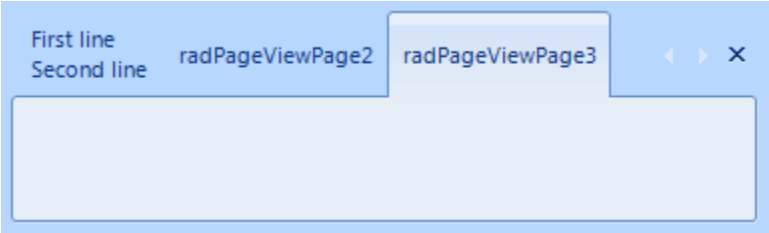
```
radPageViewPage2.Text = "Dynamic Tab"
radPageViewPage2.Item.Font = New System.Drawing.Font("Microsoft Sans Serif", 12F,
System.Drawing.FontStyle.Italic)
```

[C#] Setting Text and Font

```
radPageViewPage2.Text = "Dynamic tab";
radPageViewPage2.Item.Font = new System.Drawing.Font("Microsoft Sans Serif", 12F,
System.Drawing.FontStyle.Italic);
```

Multi Line Text

To assign multiple lines to the Text property, separate each line with an Environment.Newline character.



[VB] Adding Multiple Lines Programmatically

```
radPageViewPage1.Text = "First line" + Environment.NewLine & "Second line"
```

[C#] Adding Multiple Lines Programmatically

```
radPageViewPage1.Text = "First line" + Environment.NewLine + "Second line";
```

Text Orientation

You can set the text orientation for each tab individually by assigning TextOrientation to Vertical or Horizontal (the default). To grab fine grained control you can use the AngleTransform property instead.



[VB] Setting the TextOrientation

```
radPageViewPage1.Item.TextOrientation = Orientation.Vertical
radPageViewPage2.Item.AngleTransform = 45
radPageViewPage3.Item.AngleTransform = 120
radPageViewPage4.Item.AngleTransform = 220
radPageViewPage5.Item.AngleTransform = -35
```

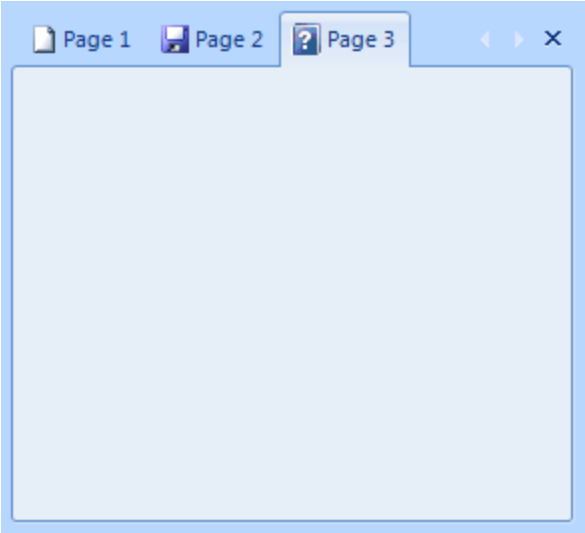
[C#] Setting the TextOrientation

```
radPageViewPage1.Item.TextOrientation = Orientation.Vertical;
radPageViewPage2.Item.AngleTransform = 45;
radPageViewPage3.Item.AngleTransform = 120;
radPageViewPage4.Item.AngleTransform = 220;
radPageViewPage5.Item.AngleTransform = -35;
```

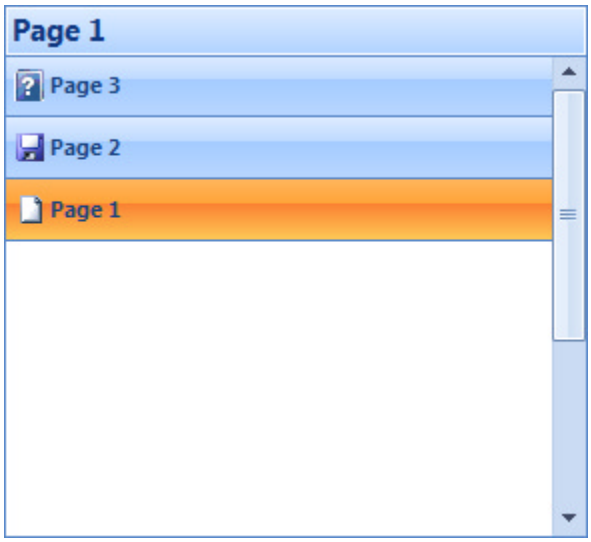
Images

You can display images for each page item. Assign the Image property of the desired page either from the ImageList, loading from disk or assigning an image resource.

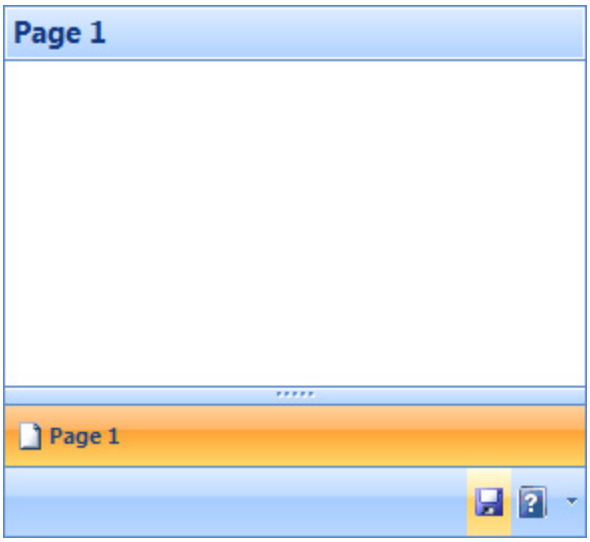
Strip view:



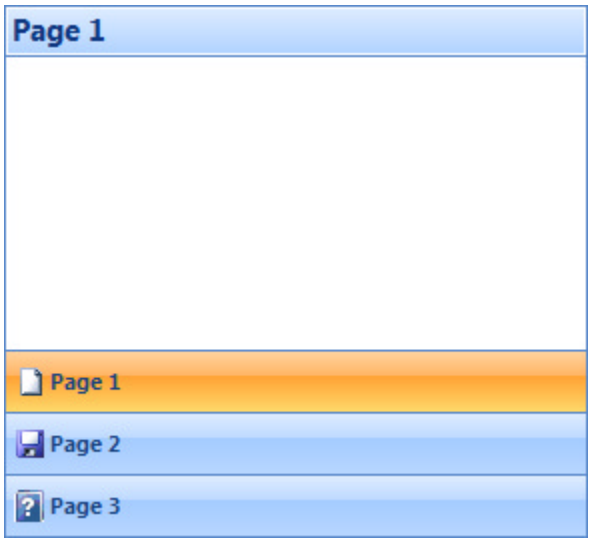
ExplorerBar mode:



Outlook mode:



Stack mode:



[VB] Methods to Assign Tab Images

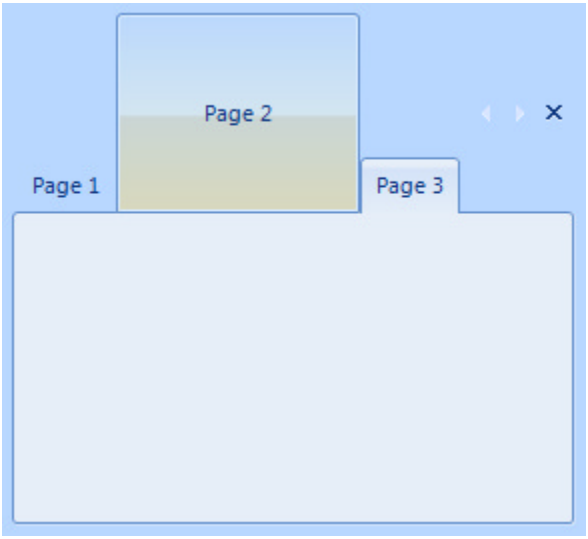
```
radPageViewPage1.Image = Image.FromFile("C:\image.png")
radPageViewPage2.Image = Resources.disk
radPageViewPage3.Image = imageList1.Images("help.png")
```

[C#] Methods to Assign Tab Images

```
radPageViewPage1.Image = Image.FromFile(@"C:\image.png");
radPageViewPage2.Image = Resources.disk;
radPageViewPage3.Image = imageList1.Images["help.png"];
```

Padding and Margin

Margin is the extra space that can be applied to the outside of the page item, Padding is applied to the inside of the page item. Both use the Padding object and can be initialized with a single number that adds space to all four edges at once, or you can specify each edge individually. This example sets page item padding for all four edges to 40 pixels. The two assignments shown in the code are equivalent.



[VB] Assigning Padding

```
Dim element As RadPageViewStripElement = TryCast(Me.radPageView1.ViewElement,
RadPageViewStripElement)
element.ItemSizeMode = PageViewItemSizeMode.Individual
radPageViewPage2.Item.Padding = New System.Windows.Forms.Padding(40)
radPageViewPage2.Item.Padding = New System.Windows.Forms.Padding(40, 40, 40, 40)
```

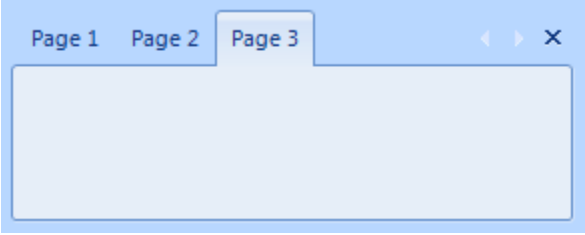
[C#] Assigning Padding

```
RadPageViewStripElement element = (this.radPageView1.ViewElement as
RadPageViewStripElement);
element.ItemSizeMode = PageViewItemSizeMode.Individual;
radPageViewPage2.Item.Padding = new System.Windows.Forms.Padding(40);
radPageViewPage2.Item.Padding = new System.Windows.Forms.Padding(40, 40, 40, 40);
```

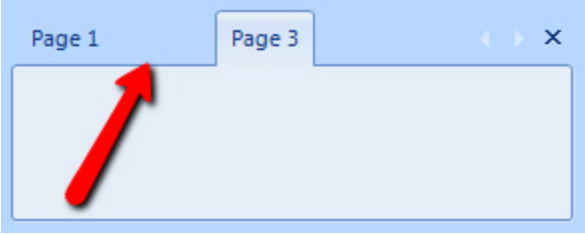
If **ItemSizeMode** is not explicitly set to **Individual** for the current view element, then the height for all tabs will be equal to the tab with the greatest height.

Visibility

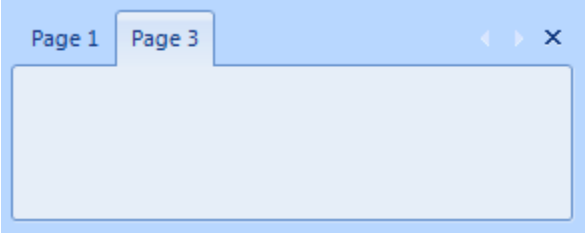
By default, the page item **Visibility** property is set to **Visible**. This setting means that the tab and its ContentPanel can be seen and manipulated by the end user at run time.



You can also set the Visibility property to **Hidden** where neither the tab nor its ContentPanel are visible, but a space is left open in the RadPageView at run time for the hidden tab.



If you set the page item's Visibility property to **Collapsed**, the tab and its associated ContentPanel are invisible, and the space occupied by the tab is removed.



8.7 Programming RadPageView

Iterating Items

You can iterate every page using the Pages collection. This example removes several pages when the "Simple Menu" checkbox is selected. The pages in the Pages collection have Tag properties set to "simple" or

"advanced". When the "Simple Menu" checkbox is clicked, the pages are iterated and hidden if they are "advanced".



You can find the complete source for this project at:

\Tabs\<VB|CS>\FindItem

The project requires references to Telerik.WinControls, Telerik.WinControls.UI and Telerik.WinControls.Enumerations in the "Imports" (VB) or "uses" (C#) section of code.

[VB] Iterating the Items Collection

```
Private Sub radCheckBox1_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs) Handles RadCheckBox1.ToggleStateChanged
For Each page As RadPageViewPage In RadPageView1.Pages
If (page.Tag.ToString().Equals("advanced") And args.ToggleState.Equals(ToggleState.On)) Then
page.Item.Visibility = ElementVisibility.Collapsed
Else
page.Item.Visibility = ElementVisibility.Visible
End If
Next page
End Sub
```

[C#] Iterating the Items Collection

```
private void radCheckBox1_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
foreach (RadPageViewPage page in radPageView1.Pages)
{
bool isAdvanced = page.Tag.ToString().Equals("advanced");
page.Item.Visibility = isAdvanced & args.ToggleState == ToggleState.On
? ElementVisibility.Collapsed : ElementVisibility.Visible;
}
}
```

Finding Items

The RadPageView Pages collection has a series of extension methods that implement functions for finding and performing other operations on page items. Include a reference to System.Linq so that these extension methods will show up.

Here is an example that duplicates the effect of the previous "Iterating items" example. The code uses the Where() method to select all page items with a Tag property of "advanced". The returned IEnumerable collection is iterated and the visibility is set to Collapsed if the checkbox is selected.



You can find the complete source for this project at:

\Tabs\<VB|CS>\FindItem

[VB] Finding Tab Items

```
Dim advanced As IEnumerable(Of RadPageViewPage) = RadPageView1.Pages.Where(Function(item)
item.Tag.ToString().Equals("advanced"))
For Each page As RadPageViewPage In advanced
page.Item.Visibility = If(args.ToggleState = ToggleState.[On], ElementVisibility.Collapsed,
ElementVisibility.Visible)
Next
```

[C#] Finding Tab Items

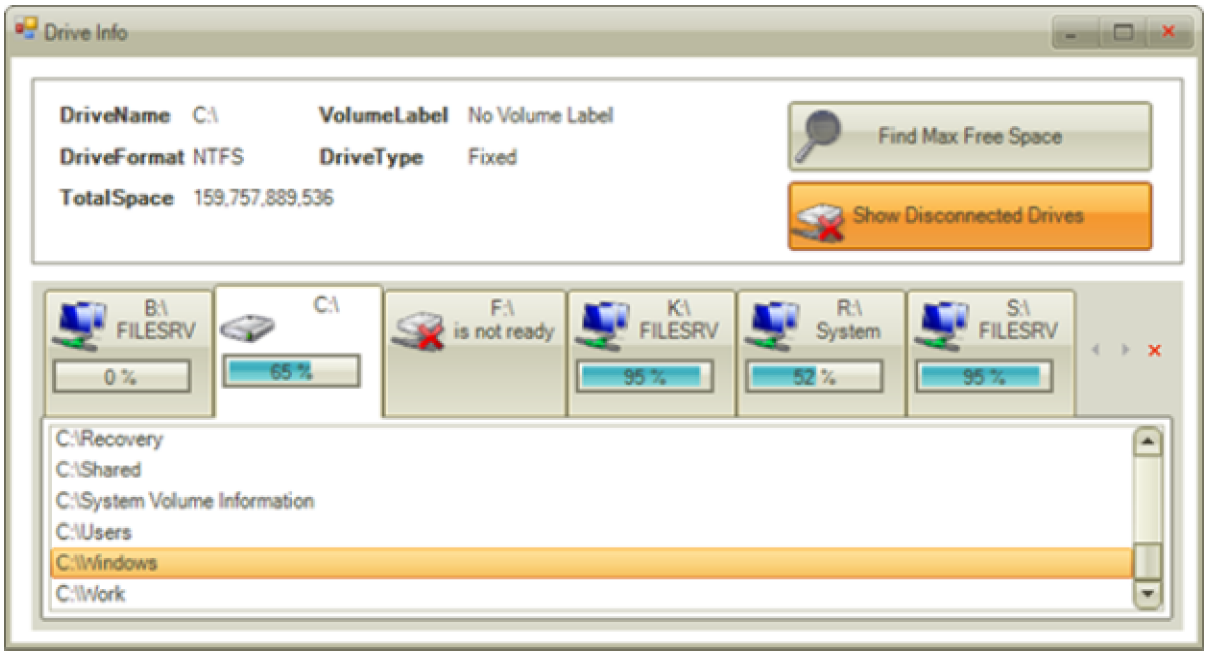
```
IEnumerable<RadPageViewPage> advanced = radPageView1.Pages.Where(item => item.Tag.ToString
().Equals("advanced"));
foreach (RadPageViewPage page in advanced)
{
page.Item.Visibility = args.ToggleState == ToggleState.On ? ElementVisibility.Collapsed :
ElementVisibility.Visible;
}
```

Custom Elements Walkthrough

All the visual elements of the project have already been setup in the "Getting Started" project. This walkthrough will show:

- Extending the RadPageViewItem (in this case RadPageViewStripItem) with RadProgressBarElements in it.
- Adding pages dynamically.
- Adding controls dynamically to the pages.
- Iterating pages.
- Associating images with page items.
- Layout and position of text, images and custom elements.

Use the Environment.GetLogicalDrives() method to retrieve a list of drive name strings (e.g. "C:\", "D:\", etc.), then create a DriveInfo object, passing the drive name in the constructor. The DriveInfo object provides the data displayed in the panel at the top of the form, the type of drive used to index into the images used to represent each drive and the amount of total and free space used to format progress bars displayed in each page item. The System.IO Directory.GetDirectories() method returns an array of directories that are listed in the page below each page item.



Preparing the Project

You can find the complete source for this project at:

\PageView\<VB|CS>\CustomElementsWalkthrough

1. Start with the "GettingStarted" project or a copy.
2. Select the page view Smart tag and remove all pages from there.
3. In the code for the form, add references to the "Imports" (VB) or "uses" (C#) section of code:
   - Telerik.WinControls
   - Telerik.WinControls.Enumerations

> The project should already have System.IO and Telerik.WinControls.UI references from the "Getting Started" project.

Populate RadPageView

1. In the form constructor after the InitializeComponent call, add a call to LoadTabStrip(). LoadTabStrip() is a private method we will write in upcoming steps that populates the page view with pages for each drive on the local machine and content panels listing the directories for each drive.

[VB] Calling LoadTabStrip

```
Public Sub New()
InitializeComponent()
LoadPageView()
End Sub
```

[C#] Calling LoadTabStrip

```
public RadForm1()
{
```

```
InitializeComponent();
LoadPageView();
}
```

2. Add a series of helper methods to the form code that will populate the page items. The code does the following:
   - Creates a new RadPageViewPage that stores the associated DriveInfo object in the Tag property.
   - Creates a new RadListControl and populates it with the directories.
   - Calculates the amount of used drive space.

[VB] Adding Helper Methods

```
#Region "helper methods"
' Create and configure a single RadPageViewPage and populate the
' tag with the DriveInfo passed
Private Function GetDrivePageItem(ByVal driveInfo As DriveInfo) As RadPageViewPage
Dim page As New RadPageViewPage()
page.TextImageRelation = TextImageRelation.ImageBeforeText
page.TextAlignment = ContentAlignment.TopCenter
page.ImageAlignment = ContentAlignment.TopLeft
page.Tag = driveInfo
Return page
End Function
' Create, configure and populate a RadListControl with files from the
' drive root.
Private Function GetListControl(ByVal drive As String) As RadListControl
Dim listControl As New RadListControl()
listControl.Dock = DockStyle.Fill
listControl.ThemeName = "Desert"
Dim directories As String() = Directory.GetDirectories(drive.ToString())
For Each directory As String In directories
listControl.Items.Add(New RadListDataItem(directory))
Next
Return listControl
End Function
' Get the percent of used space on a drive
Private Function GetUsedPercent(ByVal driveInfo As DriveInfo) As Integer
Return CInt(Math.Truncate((CDbl(driveInfo.TotalSize) - CDbl(driveInfo.AvailableFreeSp
CDbl(driveInfo.TotalSize) * 100))
End Function
#End Region
```

[C#] Adding Helper Methods

```
#region helper methods
// Create and configure a single RadPageViewPage and populate the
// tag with the DriveInfo passed
private RadPageViewPage GetDrivePageItem(DriveInfo driveInfo)
{
RadPageViewPage page = new RadPageViewPage();
page.TextImageRelation = TextImageRelation.ImageBeforeText;
page.TextAlignment = ContentAlignment.TopCenter;
page.ImageAlignment = ContentAlignment.TopLeft;
page.Tag = driveInfo;
return page;
}
```

```
// Create, configure and populate a RadListControl with files from the
// drive root.
private RadListControl GetListControl(string drive)
{
RadListControl listControl = new RadListControl();
listControl.Dock = DockStyle.Fill;
listControl.ThemeName = "Desert";
string[] directories = Directory.GetDirectories(drive.ToString());
foreach (string directory in directories)
{
listControl.Items.Add(new RadListDataItem(directory));
}
return listControl;
}
// Get the percent of used space on a drive
private int GetUsedPercent(DriveInfo driveInfo)
{
return (int)((
(double)driveInfo.TotalSize -
(double)driveInfo.AvailableFreeSpace) / (double)driveInfo.TotalSize * 100);
}
#endregion helper methods
```

3. Create a new class called RadPageViewProgressBarItem that inherits from RadPageViewStripItem (since we are using Strip mode). This class will create a new RadPageViewStripItem which will be customized to contain RadProgressBarElement.
   - Declare new RadProgressBarElement field.
   - Override the CreateChildElements method where we are going to instantiate the progress bar element and set its properties - Margin, Padding, Min and MaxSize, NotifyParentOnMouseInput, ShowProgressBarIndicator.
   - Subscribing to the progress bar TextElement Click event is necessary in order to select the underlying page when the progress bar of a specific page item is clicked.
   - At the end of this method add the progress bar instance to the Children collection of the page item.
   - Create a new property that returns the instance of the progress bar in order to be able to access it.
   - Override the ArrageChildren method in order to add the progress bar to the arrange logic of the item
   - Finally override the ThemeEffectiveType in order to allow theming of the custom item.

   [VB] Create the custom page item

```
'Create custom page item
Class RadPageViewProgressBarItem
Inherits RadPageViewStripItem
Private m_progressBar As RadProgressBarElement
Protected Overrides Sub CreateChildElements()
MyBase.CreateChildElements()
' Create and configure a RadProgressBarElement.
' Hook up a click event to pass through clicks made
' to the progress to the page item underneath
m_progressBar = New RadProgressBarElement()
m_progressBar.Margin = New Padding(5, 40, 5, 5)
m_progressBar.Padding = New Padding(5)
m_progressBar.MinSize = New Size(80, 20)
m_progressBar.MaxSize = New Size(80, 20)
AddHandler m_progressBar.Click, AddressOf TextElement_Click
m_progressBar.TextElement.NotifyParentOnMouseInput = True
m_progressBar.ShowProgressIndicator = True
Me.Children.Add(m_progressBar)
```

```
End Sub
'expose the progressbar in order to access it directly from the page item
Public ReadOnly Property ProgressBar() As RadProgressBarElement
Get
Return Me.m_progressBar
End Get
End Property
'select the corresponding page when the progress bar is clicked
Private Sub TextElement_Click(ByVal sender As Object, ByVal e As EventArgs)
DirectCast(Me.Owner.ElementTree.Control, RadPageView).SelectedPage = Me.Page
End Sub
'arrange the item children
Protected Overrides Sub ArrangeChildren(ByVal available As SizeF)
m_progressBar.Arrange(New RectangleF(PointF.Empty, available))
MyBase.ArrangeChildren(available)
End Sub
'make the item themeable
Protected Overrides ReadOnly Property ThemeEffectiveType() As Type
Get
Return GetType(RadPageViewStripItem)
End Get
End Property
End Class
```

[C#] Create the custom page item

```
//Create custom page item
class RadPageViewProgressBarItem : RadPageViewStripItem
{
RadProgressBarElement progressBar;
protected override void CreateChildElements()
{
base.CreateChildElements();
// Create and configure a RadProgressBarElement.
// Hook up a click event to pass through clicks made
// to the progress to the page item underneath
progressBar = new RadProgressBarElement();
progressBar.Margin = new Padding(5, 40, 5, 5);
progressBar.Padding = new Padding(5);
progressBar.MinSize = new Size(80, 20);
progressBar.MaxSize = new Size(80, 20);
progressBar.TextElement.NotifyParentOnMouseInput = true;
progressBar.TextElement.Click += new EventHandler(TextElement_Click);
progressBar.ShowProgressIndicator = true;
this.Children.Add(progressBar);
}
//expose the progressbar in order to access it directly from the page item
public RadProgressBarElement ProgressBar
{
get
{
return this.progressBar;
}
}
//select the corresponding page when the progress bar is clicked
```

```
void TextElement_Click(object sender, EventArgs e)
{
((RadPageView)this.Owner.ElementTree.Control).SelectedPage = this.Page;
}
//arrange the item children
protected override void ArrangeChildren(SizeF available)
{
progressBar.Arrange(new RectangleF(PointF.Empty, available));
base.ArrangeChildren(available);
}
//make the item themeable
protected override Type ThemeEffectiveType
{
get
{
return typeof(RadPageViewStripItem);
}
}
}
```

4. Add the LoadPageView method to populate the page items with drive information.
- Before creating the instance of RadPageViewPage subscribe to the ItemCreating event in order to be able to replace the default page item with the custom one. In the event handler access the currently created item from the event arguments and replace it with a new instance of our custom item:

[VB] Replace the default page item with the custom one

```
'Replace the default page item with the custom one that contains RadProgressBarElement
Private Sub radPageView1_ItemCreating(ByVal sender As Object, ByVal e As
RadPageViewItemCreatingEventArgs)
e.Item = New RadPageViewProgressBarItem()
End Sub
```

[C#] Replace the default page item with the custom one

```
//Replace the default page item with the custom one that contains RadProgressBarElement
void radPageView1_ItemCreating(object sender, RadPageViewItemCreatingEventArgs e)
{
e.Item = new RadPageViewProgressBarItem();
}
```

- Create the actual page and add it to the Pages collection of RadPageView
- Set the page Text, Image and also the progress bar value according to the drive

[VB] Populating Tabs

```
Private Sub LoadPageView()
For Each drive As String In Environment.GetLogicalDrives()
Dim driveInfo As New DriveInfo(drive)
AddHandler radPageView1.ItemCreating, AddressOf radPageView1_ItemCreating
Dim page As RadPageViewPage = GetDrivePageItem(driveInfo)
radPageView1.Pages.Add(page)
Dim progressBar As RadProgressBarElement = DirectCast(page.Item,
RadPageViewProgressBarItem).ProgressBar
If driveInfo.IsReady Then
page.Text = driveInfo.Name + Environment.NewLine + driveInfo.VolumeLabel
page.Image = ilDriveTypes.Images(CInt(driveInfo.DriveType))
progressBar.Value1 = GetUsedPercent(driveInfo)
```

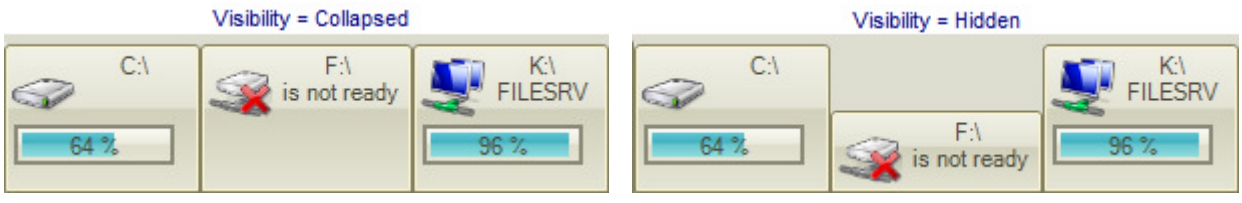
```
page.Controls.Add(GetListControl(drive))
Else
page.Text = drive + Environment.NewLine & " is not ready"
page.Image = ilDriveTypes.Images("DisconnectedDrive.png")
progressBar.Visibility = ElementVisibility.Hidden
End If
Next
End Sub
```

[C#] Populating Tabs

```
private void LoadPageView()
{
foreach (string drive in Environment.GetLogicalDrives())
{
DriveInfo driveInfo = new DriveInfo(drive);
radPageView1.ItemCreating += new EventHandler<RadPageViewItemCreatingEventArgs>
(radPageView1_ItemCreating);
RadPageViewPage page = GetDrivePageItem(driveInfo);
radPageView1.Pages.Add(page);
RadProgressBarElement progressBar = ((RadPageViewProgressBarItem)page.Item).ProgressBar;
if (driveInfo.IsReady)
{
page.Text = driveInfo.Name + Environment.NewLine + driveInfo.VolumeLabel;
page.Image = ilDriveTypes.Images[(int)driveInfo.DriveType];
progressBar.Value1 = GetUsedPercent(driveInfo);
page.Controls.Add(GetListControl(drive));
}
else
{
page.Text = drive + Environment.NewLine + " is not ready";
page.Image = ilDriveTypes.Images["DisconnectedDrive.png"];
progressBar.Visibility = ElementVisibility.Hidden;
}
}
}
```

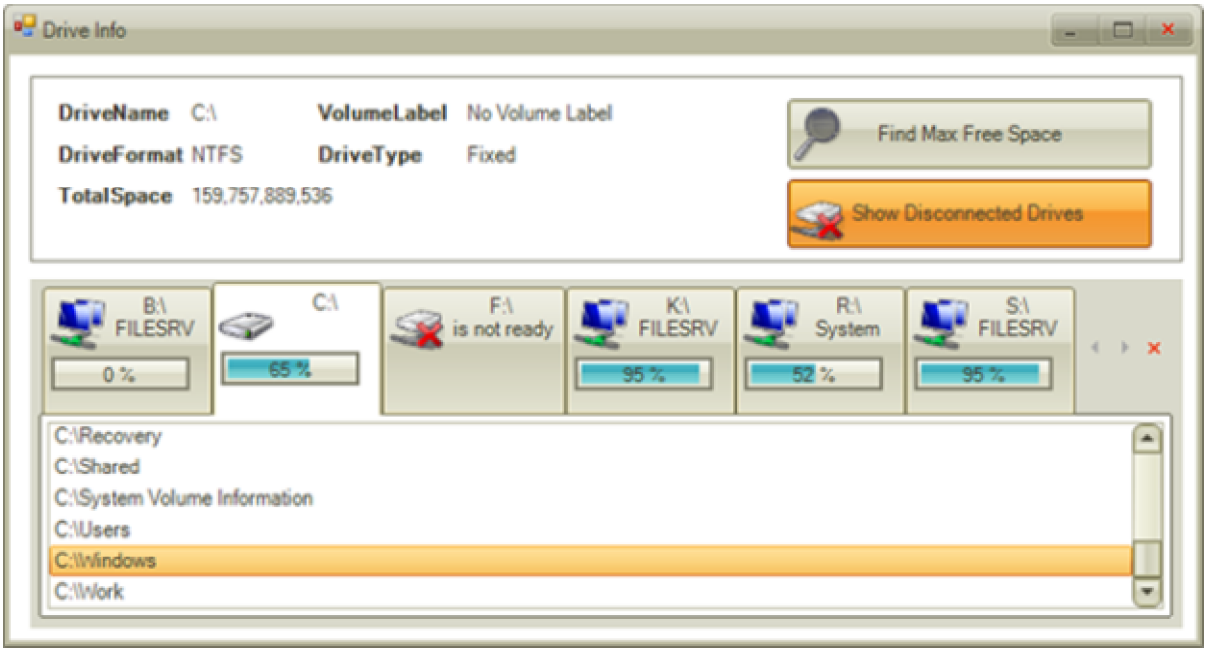
If the drive is ready (e.g. the DVD is in the drive), the name and volume label are displayed in the page item text, and the appropriate image for the type of drive (e.g. hard drive, CD ROM, network, etc.) is assigned from the ImageList. When you set up the ImageList in the "Getting Started" project, the order of images matches the members of the DriveInfo.DriveType enumeration, so the DriveType can be used to index into the ImageList. If the drive is ready, the progress bar value is assigned the percentage amount used on the drive. If the drive is not ready, the text describes that condition, displays the "Disconnected Drive" image and hides the progress bar element.

Notice that when the drive is disconnected, the progress bar Visibility is set to "Hidden", not "Collapsed" so that the progress bar still takes up the same amount of space in the page item.



Press **Ctrl-F5** to run the application. You may first want to map a network drive or any USB drives you have available so that you have more drives to test against. Try running the application with your DVD or CD in

the drive and without.



5. Double-click the "Find Max Free Space" button and add the following code to the Click event handler.

The code iterates the pages comparing the DriveInfo drive space data looking for the greatest available free space. After the iteration, the page presenting the drive with the greatest free space is selected.

[VB] Handing the "Find Max Free Space" Button Click Event

```
Private Sub btnSearch_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnSearch.Click
Dim selectedPage As RadPageViewPage = Nothing
For Each page As RadPageViewPage In radPageView1.Pages
' no drive info, skip
If page.Tag Is Nothing Then
Continue For
End If
' save first available tab with drive info
If selectedPage Is Nothing Then
selectedPage = page
End If
' drive isn't ready, skip
If Not TryCast(page.Tag, DriveInfo).IsReady Then
Continue For
End If
' save tab with greatest free space
selectedPage = If(TryCast(selectedPage.Tag, DriveInfo).TotalFreeSpace > TryCast(page.Tag,
DriveInfo).TotalFreeSpace, selectedPage, page)
Next
' select tab with greatest free space
radPageView1.SelectedPage = selectedPage
End Sub
```

[C#] Handing the "Find Max Free Space" Button Click Event

```
private void btnSearch_Click(object sender, EventArgs e)
{
```

```
RadPageViewPage selectedPage = null;
foreach (RadPageViewPage page in radPageView1.Pages)
{
// no drive info, skip
if (page.Tag == null)
continue;
// save first available tab with drive info
if (selectedPage == null)
selectedPage = page;
// drive isn't ready, skip
if (!(page.Tag as DriveInfo).IsReady)
continue;
// save tab with greatest free space
selectedPage =
(selectedPage.Tag as DriveInfo).TotalFreeSpace >
(page.Tag as DriveInfo).TotalFreeSpace ?
selectedPage : page;
}
// select tab with greatest free space
radPageView1.SelectedPage = selectedPage;
}
```

6. Double-click the "Show Disconnected Drives" RadToggleButton to create a ToggleStateChanged event handler. Add the code below to the event handler.

   This code iterates the tab items in the strip examining the DriveInfo IsReady flag and collapses any unready drive tabs.

   [VB] Hiding Unready Drives

```
' Iterate the tabs DriveInfo and collapse any that aren't ready
Private Sub btnShow_ToggleStateChanged(ByVal sender As Object, ByVal args As
StateChangedEventArgs) Handles btnShow.ToggleStateChanged
For Each page As RadPageViewPage In radPageView1.Pages
If Not TryCast(page.Tag, DriveInfo).IsReady Then
page.Item.Visibility = If(args.ToggleState = ToggleState.[On], ElementVisibility.Visible,
ElementVisibility.Collapsed)
End If
Next
End Sub
```

   [C#] Hiding Unready Drives

```
// Iterate the tabs DriveInfo and collapse any that aren't ready
private void btnShow_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
foreach (RadPageViewPage page in radPageView1.Pages)
{
if (!(page.Tag as DriveInfo).IsReady)
{
page.Item.Visibility = args.ToggleState == ToggleState.On
? ElementVisibility.Visible : ElementVisibility.Collapsed;
}
}
}
```

   Using the Properties window Events tab, add a SelectedPageChanging event handler for the page view.

   This event handler prevents the user from navigating to a tab that has a drive that isn't ready. Instead, a

RadMessageBox displays a message and the event is canceled. The DriveInfo object is used to populate all the labels for the panel at the top of the form.

[VB] Handling the SelectedPageChanging Event

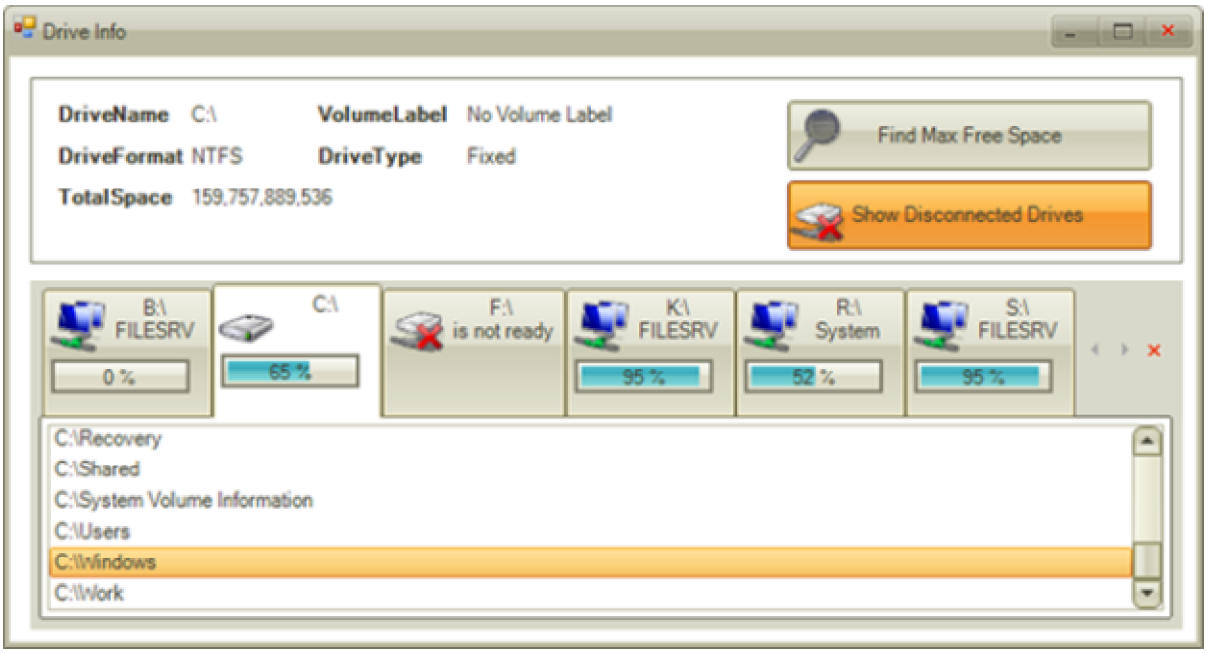
```
Private Sub radPageView1_SelectedPageChanging(ByVal sender As Object, ByVal e As
RadPageViewCancelEventArgs) Handles radPageView1.SelectedPageChanging
'Fill the labels with data
If e.Page.Tag IsNot Nothing Then
Dim driveInfo As DriveInfo = DirectCast(e.Page.Tag, DriveInfo)
If driveInfo.IsReady Then
lblDriveName.Text = driveInfo.Name
lblVolumeLabel.Text = If(driveInfo.VolumeLabel.Equals([String].Empty), "No Volume Label",
driveInfo.VolumeLabel)
lblDriveFormat.Text = driveInfo.DriveFormat
lblDriveType.Text = driveInfo.DriveType.ToString()
lblTotalSpace.Text = driveInfo.TotalSize.ToString("n0")
End If
End If
'Show message box for the drives that are not ready
If e.Page Is Nothing Then
Return
End If
If Not TryCast(e.Page.Tag, DriveInfo).IsReady Then
Dim tabItem As RadPageViewItem = TryCast(e.Page.Item, RadPageViewItem)
RadMessageBox.SetThemeName(Me.ThemeName)
Dim message As [String] = [String].Format("{0}", tabItem.Text)
RadMessageBox.Show(message, "Drive Unavailable", MessageBoxButtons.OK, RadMessageIcon.
[Error])
e.Cancel = True
End If
End Sub
```

[C#] Handling the SelectedPageChanging Event

```
void radPageView1_SelectedPageChanging(object sender, RadPageViewCancelEventArgs e)
{
//Fill the labels with data
if (e.Page.Tag != null)
{
DriveInfo driveInfo = (DriveInfo)e.Page.Tag;
if (driveInfo.IsReady)
{
lblDriveName.Text = driveInfo.Name;
lblVolumeLabel.Text = driveInfo.VolumeLabel.Equals(String.Empty) ? "No Volume Label" :
driveInfo.VolumeLabel;
lblDriveFormat.Text = driveInfo.DriveFormat;
lblDriveType.Text = driveInfo.DriveType.ToString();
lblTotalSpace.Text = driveInfo.TotalSize.ToString("n0");
}
}
//Show message box for the drives that are not ready
if (e.Page == null)
return;
if (!(e.Page.Tag as DriveInfo).IsReady)
{
RadPageViewItem tabItem = e.Page.Item as RadPageViewItem;
```

```
RadMessageBox.SetThemeName(this.ThemeName);
String message = String.Format("{0}", tabItem.Text);
RadMessageBox.Show(message, "Drive Unavailable", MessageBoxButtons.OK,
RadMessageIcon.Error);
e.Cancel = true;
}
}
```

7. Press **Ctrl-F5** to run the application. Test the "Free Space" and "Show Disconnected" button functionality, notice how the labels change as you select tabs, try clicking on a disconnected drive tab to get the message box and notice the free space displayed in the progress bar.



Drag and Drop

Enabling the drag and drop functionality of RadPageView pages is achieved by making use of the **ItemDragMode** property of the current ViewMode. This property provides enumeration with the following values:

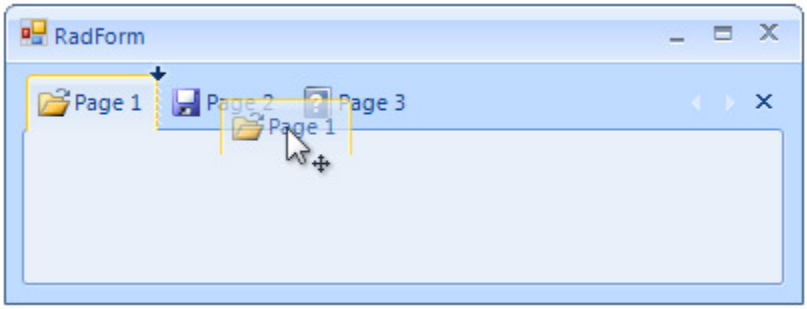
- **PageViewItemDragMode.None** - which is the default setting - disables the dragging functionality
- **PageViewItemDragMode.Immediate** - which applies the dragging operation without preview
- **PageViewItemDragMode.Preview** - which displays a preview of the dragging operation.

[VB] Setting Preview drag mode to RadPageView

```
Dim element as New RadPageViewStripElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewStripElement)
element.ItemDragMode = PageViewItemDragMode.Preview
```

[C#] Setting Preview drag mode to RadPageView

```
RadPageViewStripElement element = (this.radPageView1.ViewElement as
RadPageViewStripElement);
element.ItemDragMode = PageViewItemDragMode.Preview;
```

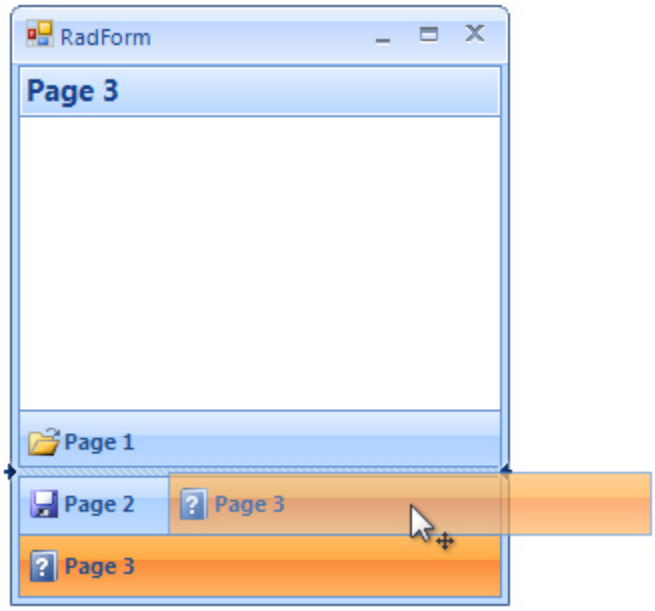


[VB] Setting Preview drag mode to RadPageView

```
Dim element as New RadPageViewStackElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewStackElement)
element.ItemDragMode = PageViewItemDragMode.Preview
```

[C#] Setting Preview drag mode to RadPageView

```
RadPageViewStackElement element = (this.radPageView1.ViewElement as
RadPageViewStackElement);
element.ItemDragMode = PageViewItemDragMode.Preview;
```

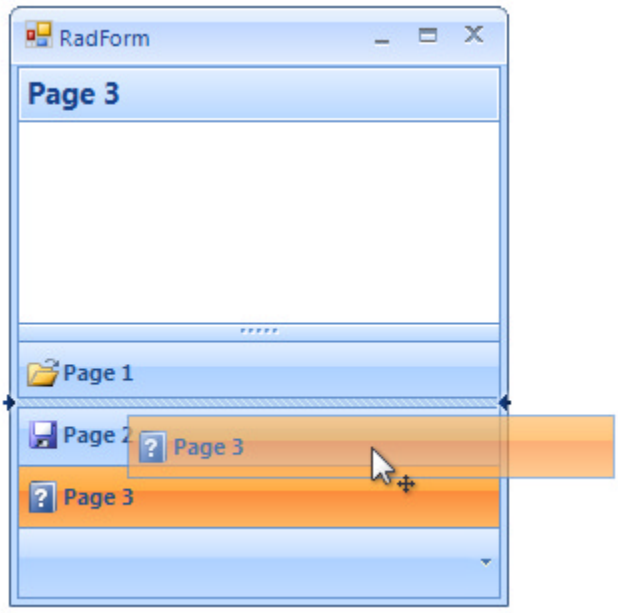


[VB] Setting Preview drag mode to RadPageView

```
Dim element as New RadPageViewOutlookElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewOutlookElement)
element.ItemDragMode = PageViewItemDragMode.Preview
```

[C#] Setting Preview drag mode to RadPageView

```
RadPageViewOutlookElement element = (this.radPageView1.ViewElement as
RadPageViewOutlookElement);
element.ItemDragMode = PageViewItemDragMode.Preview;
```

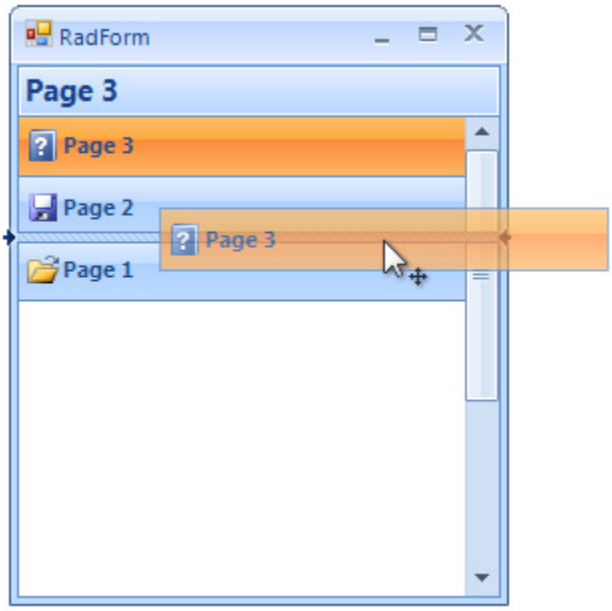


[VB] Setting Preview drag mode to RadPageView

```
Dim element as New RadPageViewExplorerBarElement = DirectCast(Me.radPageView1.ViewElement,
RadPageViewExplorerBarElement)
element.ItemDragMode = PageViewItemDragMode.Preview
```

[C#] Setting Preview drag mode to RadPageView

```
RadPageViewExplorerBarElement element = (this.radPageView1.ViewElement as
RadPageViewExplorerBarElement);
element.ItemDragMode = PageViewItemDragMode.Preview;
```



8.8 Summary

In this chapter you explored creating tabbed interfaces and managing controls within associated content panels. The chapter starts with a tour of usability and feature highlights, then moves onto common Pages collection programmatic tasks such as adding and removing pages , iterating, locating and selecting tabs. The chapter also explains how to make use of the tab drag and drop functionality, handle layout and dimensions, tab images and text.

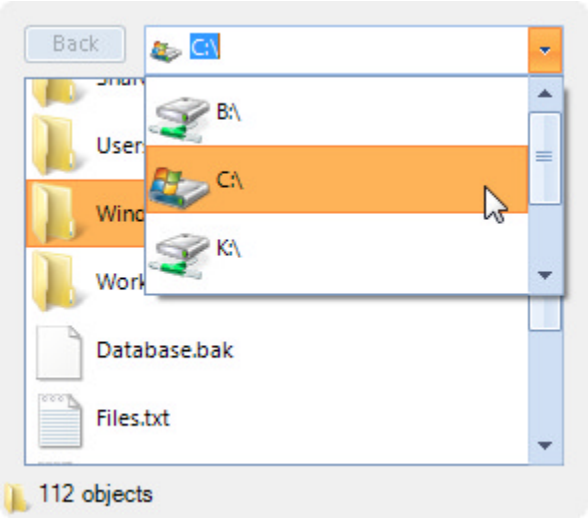
9 ListControl

9.1 Objectives

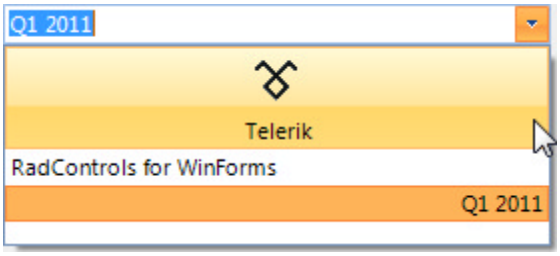
- Learn how RadListControl and RadDropDownList are used to display data in list form for user selection.
- Learn common programmatic tasks: adding and removing items, iterating, locating and modifying items.
- Learn how to embed controls within items.
- Learn how to handle item layout, dimensions, images and text.
- Learn how to move items between list controls and drag and drop multiple items between list controls.

9.2 Introduction

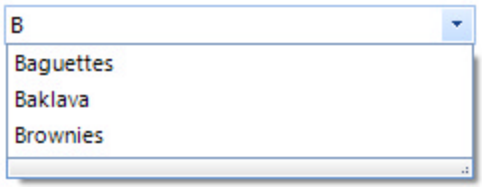
RadListControl and RadDropDownList are enhanced, themable alternatives to the corresponding standard Windows Forms controls. Both controls are based on the Telerik Presentation Foundation and have a great deal of flexibility in what content can be included in theirs lists and how the content is arranged.



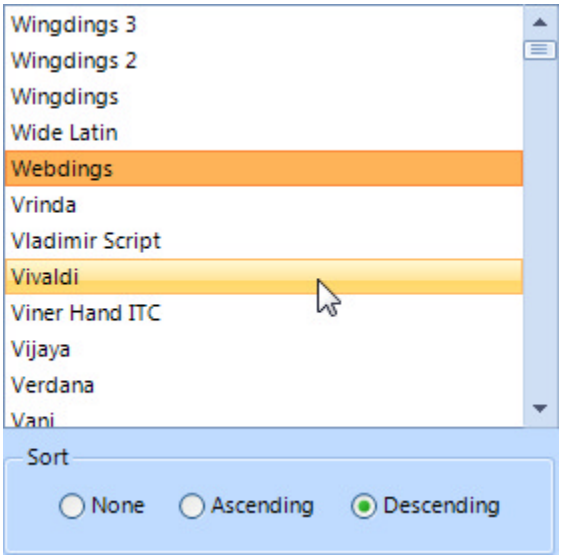
You can associate images and text descriptions with individual items in the drop-down list. Item and description fonts can be configured separately. The relationship of the text and image is configurable so that image or text can be on top or to the left. Images can have transparency and can be of any size.



Item text and image can be customized at design-time and run-time. Customization can be triggered based on the content of database data. Different image and text alignments are possible.



RadDropDownList provides flexible auto-completion options that suggest and append text from choices in the list as the user types.



Lists can be automatically sorted in ascending or descending order. At any time the list can be restored to the original order.

9.3 Getting Started

Using RadDropDownList and RadListControl in tandem you can create master/detail relationships that react to user input. In this example we create a rudimentary file explorer. We will expand on this project in later exercises to display applications icons, display small icons in the edit portion of the dropdown list, open files in response to double-clicking the file list and display custom tool tips for each file item.

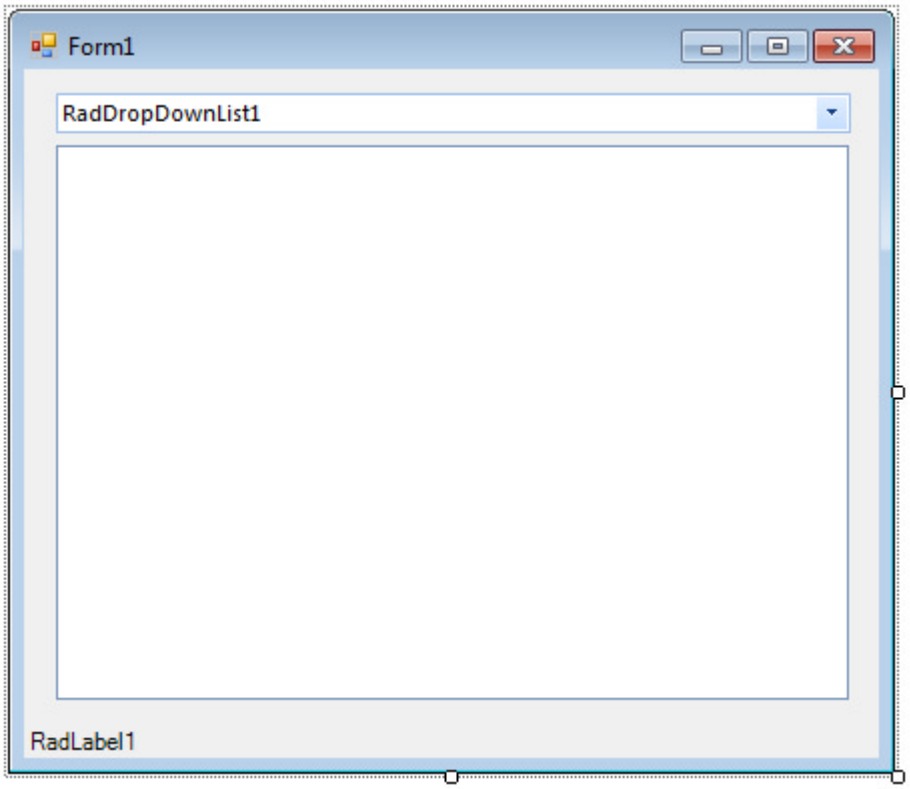


You can find the complete source for this project at:

\ListControl\<VB|CS>\GettingStarted

Form Design

1. Create a new Windows Forms application.
2. Add the following controls to the form and set properties. Arrange the controls to look something like the screenshot below.
   - **RadButton**: **Name** = "btnBack", **Text** = "Back"
   - **RadDropDownList**: **Name** = "ddlDrives", **Anchor** = Top, Left, Right
   - **RadListControl**: **Name** = "lcFiles", **Anchor** = Top, Bottom, Left, Right
   - **RadLabel**: **Name** = "lblStatus", **AutoSize** = False, **Dock** = "Bottom", **Size.Height** = "20"



Adding Code

1. Add references to "Telerik.WinControls.UI" and "System.IO" to the "Imports" (VB) or "using" (C#) section of code.
2. Double-click the form to create a Load event handler. Add the code below.

The event handler iterates the DriveInfo array returned by the static System.IO GetDrives() method. So that we don't leave the dropdown list text as "radDropDownList1", the SelectedIndex is set to the first entry in the list. In the next step we will include the AddDirectoryToDropDownList() method.

[VB] Iterating DriveInfo Array

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handle
MyBase.Load
' load the dropdown list with drive names and select the first one
For Each driveInfo__1 As DriveInfo In DriveInfo.GetDrives()
If driveInfo__1.IsReady Then
Dim info As New DirectoryInfo(driveInfo__1.RootDirectory.FullName)
AddDirectoryToDropDownList(info, ddlDrives)
End If
Next
ddlDrives.SelectedIndex = 0
End Sub
```

[C#] Iterating DriveInfo Array

```
private void Form1_Load(object sender, EventArgs e)
{
// load the dropdown list with drive names and select the first one
foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
{
```

```
if (driveInfo.IsReady)
{
DirectoryInfo info = new DirectoryInfo(driveInfo.RootDirectory.FullName);
AddDirectoryToDropDownList(info, ddlDrives);
}
}
ddlDrives.SelectedIndex = 0;
}
```

3. Include the code for the AddDirectoryToDropDownList() method.

   In this preliminary version of the method, AddDirectoryToDropDownList simply creates a RadListDataItem with the FullName of the directory as the Text and adds the DirectoryInfo object to the Value argument of the constructor.

   [VB] Including the AddDirectoryToDropDownList() Method

```
Private Sub AddDirectoryToDropDownList(ByVal info As DirectoryInfo, ByVal dropDownList As
RadDropDownList)
' display the path and save the "info" as the Value property
Dim item As New RadListDataItem(info.FullName, info)
dropDownList.Items.Add(item)
End Sub
```

   [C#] Including the AddDirectoryToDropDownList() Method

```
private void AddDirectoryToDropDownList(DirectoryInfo info, RadDropDownList dropDownList)
{
// display the path and save the "info" as the Value property
RadListDataItem item = new RadListDataItem(info.FullName, info);
dropDownList.Items.Add(item);
}
```

4. Create an event handler for the "ddlDrives" SelectedIndexChanged event. Add the code below.

   Here we get the DirectoryInfo stored in the dropdown list SelectedItem Value property. Use the DirectoryInfo GetFileSystemInfos to populate the list control with the collection of FileSystemInfo.

   [VB] Handling the RadDropDownList SelectedIndexChanged Event

```
Private Sub ddlDrives_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles ddlDrives.SelectedIndexChan
' get the dropdown list that triggered this event,
' get the currently selected item in the dropdown lsit and
' extract the DirectoryInfo
Dim dropDownListElement As RadDropDownListElement = TryCast(sender, RadDropDownListEleme
Dim item As RadListDataItem = dropDownListElement.SelectedItem
Dim directoryInfo As DirectoryInfo = TryCast(item.Value, DirectoryInfo)
lcFiles.Items.Clear()
' get a list of all directories and files
For Each info As FileSystemInfo In directoryInfo.GetFileSystemInfos()
lcFiles.Items.Add(New RadListDataItem(info.Name, info))
Next
lcFiles.SelectedIndex = 0
End Sub
```

   [C#] Handling the RadDropDownList SelectedIndexChanged Event

```
private void ddlDrives_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
```

```
// get the dropdown list that triggered this event,
// get the currently selected item in the dropdown lsit and
// extract the DirectoryInfo
RadDropDownListElement dropDownListElement = sender as RadDropDownListElement;
RadListDataItem item = dropDownListElement.SelectedItem;
DirectoryInfo directoryInfo = item.Value as DirectoryInfo;
lcFiles.Items.Clear();
// get a list of all directories and files
foreach (FileSystemInfo info in directoryInfo.GetFileSystemInfos())
{
lcFiles.Items.Add(new RadListDataItem(info.Name, info));
}
lcFiles.SelectedIndex = 0;
}
```

5. Create an event handler for the "lcFiles" SelectedIndexChanged event. Add the code below.

   The SelectedItem can become null when the list is being reloaded, so first check that the SelectedItem property is not null. Extract the FileSystemInfo object stored in the Value property and use the Name to populate "lblStatus". In later versions of the project we use FileSystemInfo to provide more complete feedback to the UI.

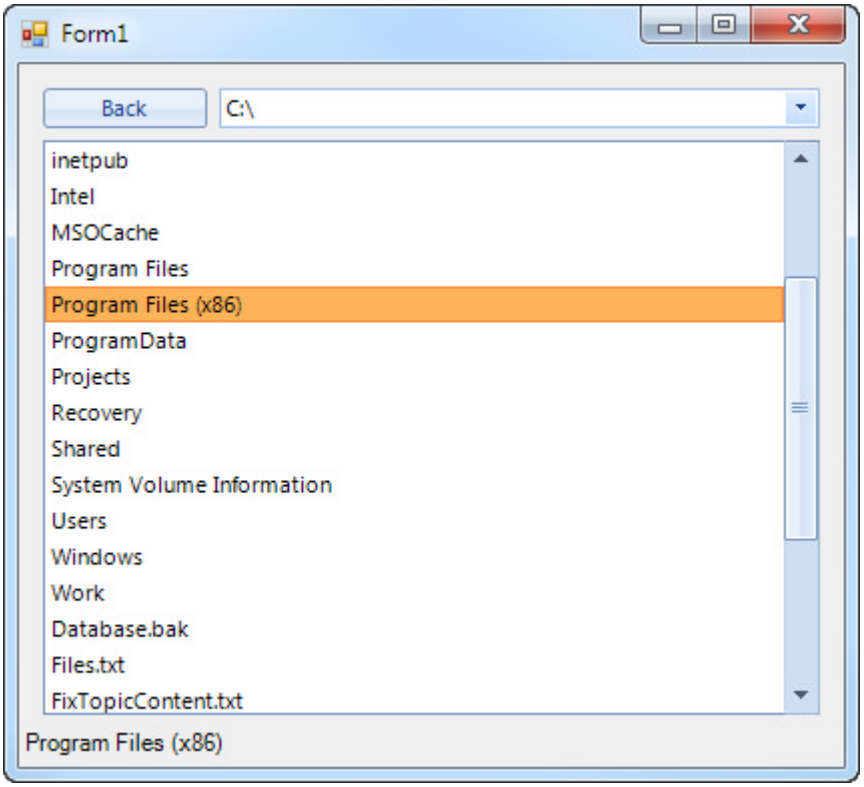
   [VB] Handling the SelectedIndexChanged Event

```
Private Sub lcFiles_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles lcFiles.SelectedIndexChange
If TryCast(sender, RadListElement).SelectedItem IsNot Nothing Then
Dim item As RadListDataItem = TryCast(sender, RadListElement).SelectedItem
Dim info As FileSystemInfo = TryCast(item.Value, FileSystemInfo)
lblStatus.Text = info.Name
End If
End Sub
```

   [C#] Handling the SelectedIndexChanged Event

```
private void lcFiles_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
if ((sender as RadListElement).SelectedItem != null)
{
RadListDataItem item = (sender as RadListElement).SelectedItem;
FileSystemInfo info = item.Value as FileSystemInfo;
lblStatus.Text = info.Name;
}
}
```

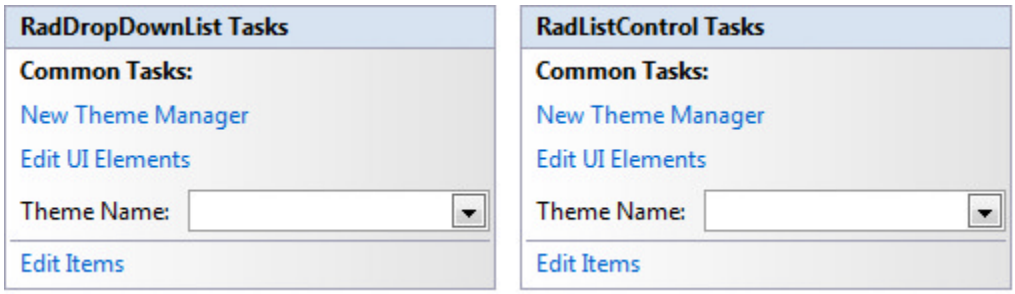
6. Press **Ctrl-F5** to run the application. Test selecting drives directories from the dropdown list and select files in the list.



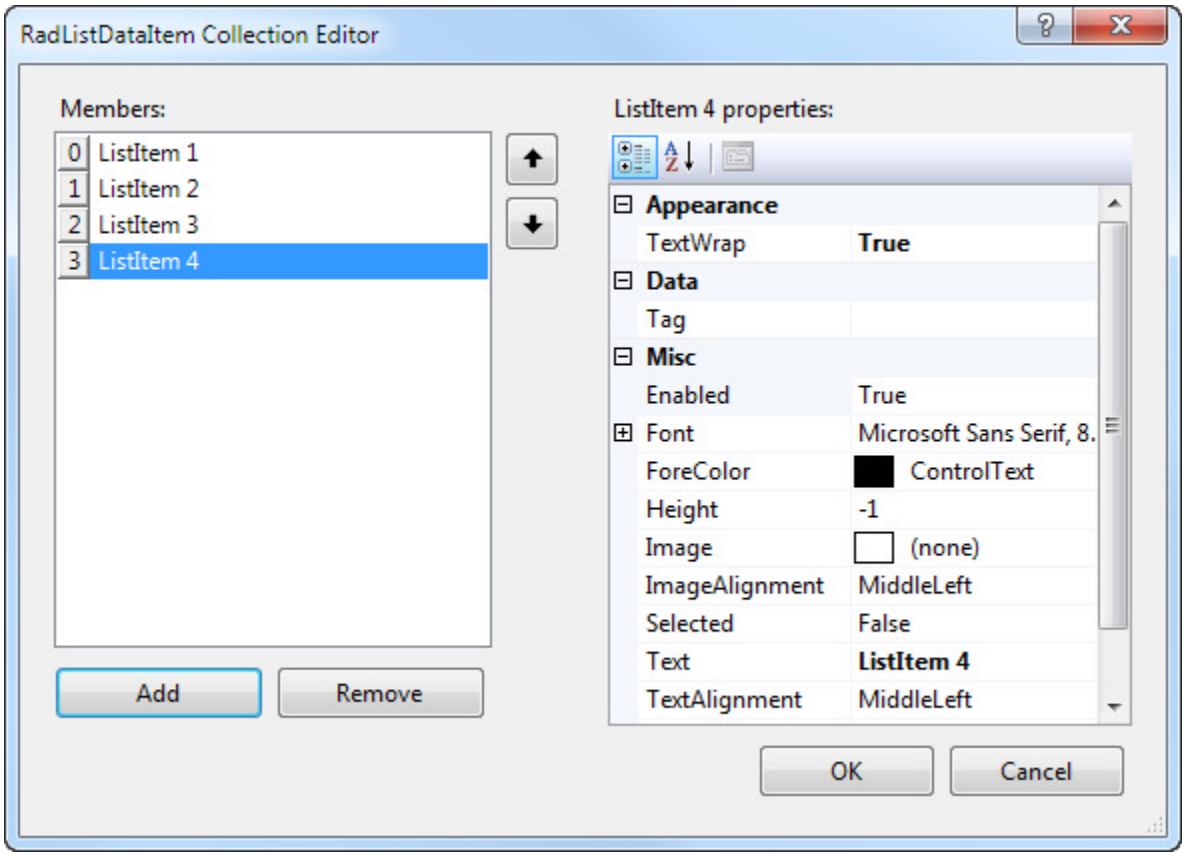
9.4 Using the Design Time Interface

Smart Tag

The Smart Tag interface for RadListControl and RadDropDownList are completely identical.



The key item here is the "Edit Items" link used to display the RadListDataItem Editor dialog where you can add such items to the Items collection of both controls.



9.5 Programming ListControl and DropDownList

RadListControls and RadDropDownList Common Tasks

Virtualization

Both **RadDropDownList** and **RadListControl** use "*UI virtualization*" so that items can be added and removed quickly.Virtualization impacts two other properties. The virtualization mechanism creates only a few visual items, just enough to fill the area that RadListControl occupies on a form and to allow scrolling. For example if a RadListControl instance is 100 pixels tall and the list item height has been set to be 20, RadListControl will create only 5 visual items. But how can 5 items visually represent possibly thousands of logical RadListDataItems? The virtualization mechanism does this by simply mapping the visual items to the logical items taking into account the scroll bar value. For example if you scroll down by one item, all visual items will be mapped to one below the previous logical item that they were mapped to initially. In essence, the logical items shift under the visual items and the visual items update accordingly. This means that updating the UI is a constant operation and is instantaneous in the majority of use cases. RadListVisualItem is the second major point of extensibility. RadListControl allows creating custom visual items, just like the logical items which allows the user to visually represent her custom data as close as possible.

Working with Items

The key property for both RadListControl and RadDropDownList is the **Items** collection of **RadListDataItem** objects. Minimally, to add new items programmatically you need to set the **Text** property; the Text property can be set in the item constructor. The example below creates a RadListDataItem for each in a collection of installed fonts.



You can find the complete source for this project at:

\ListControl\<VB|CS>\Programming

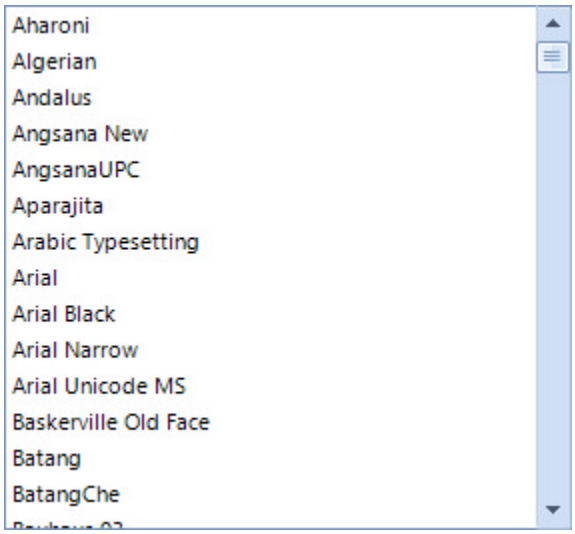
[VB] Adding Items

```
Dim families As FontFamily() = New InstalledFontCollection().Families
For Each family As FontFamily In families
 Dim item As New RadListDataItem (family.Name)
 radListControl1.Items.Add(item)
Next
```

[C#] Adding Items

```
FontFamily[] families = new InstalledFontCollection().Families;
foreach (FontFamily family in families)
{
 RadListDataItem item = new RadListDataItem (family.Name);
 radListControl1.Items.Add(item);
}
```

FontFamily can be found in the System.Drawing namespace.



If you want to add items at a specific location within the list, use the **Insert()** method and pass the index where the item should be included.

[VB] Inserting to the Collection

```
radListControl1.Items.Insert(0, New RadListDataItem("put me at the top of the list"))
```

[C#] Inserting to the Collection

```
radListControl1.Items.Insert(0, new RadListDataItem("put me at the top of the list"));
```

If you have the data for all the items you want to add up front, you can include them at one once using the **AddRange()** method.

[VB] Using AddRange()

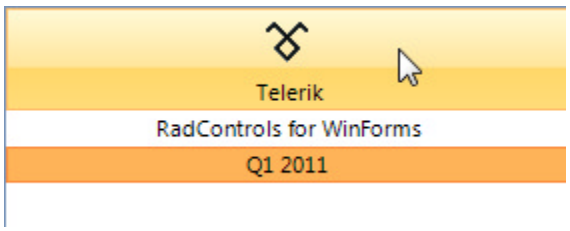
```
radListControl1.Items.AddRange(New RadListDataItem() {New RadListDataItem("One"), New
RadListDataItem("Two"), New RadListDataItem("Three")})
```

[C#] Using AddRange()

```
radListControl1.Items.AddRange(new RadListDataItem[] {
```

```
 new RadListDataItem("One"),
 new RadListDataItem("Two"),
 new RadListDataItem("Three")
});
```

Like many of the RadItem descendants, we can assign an image to it and you can set the text and the image alignment. The AutoSizeItems property is used to size each item according to its content:



[VB] Adding and Arranging Images

```
Dim item As New RadListDataItem("Telerik")
item.Image = Image.FromFile("C:\Program Files (x86)\Telerik\RadControls for WinForms Q1 2011
\Examples\QuickStart\Resources\telerikLOGO.png")
item.TextImageRelation = TextImageRelation.ImageAboveText
item.TextAlignment = ContentAlignment.BottomCenter
radListControl1.Items.Add(item)

Dim item1 As New RadListDataItem("RadControls for WinForms")
item1.TextAlignment = ContentAlignment.MiddleLeft
radListControl1.Items.Add(item1)

Dim item2 As New RadListDataItem("Q1 2011")
item2.TextAlignment = ContentAlignment.MiddleRight
radListControl1.Items.Add(item2)

radListControl1.AutoSizeItems = True
```

[C#] Adding and Arranging Images

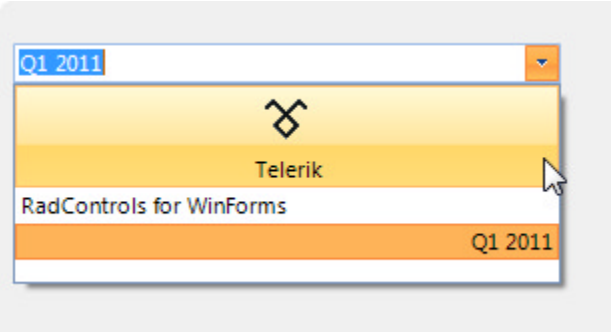
```
RadListDataItem item = new RadListDataItem("Telerik");
item.Image = Image.FromFile(@"C:\Program Files (x86)\Telerik\RadControls for WinForms Q1
2011\Examples\QuickStart\Resources\telerikLOGO.png");
item.TextImageRelation = TextImageRelation.ImageAboveText;
item.TextAlignment = ContentAlignment.BottomCenter;
radListControl1.Items.Add(item);

RadListDataItem item1 = new RadListDataItem("RadControls for WinForms");
item1.TextAlignment = ContentAlignment.MiddleLeft;
radListControl1.Items.Add(item1);

RadListDataItem item2 = new RadListDataItem("Q1 2011");
item2.TextAlignment = ContentAlignment.MiddleRight;
radListControl1.Items.Add(item2);

radListControl1.AutoSizeItems = true;
```

By substituting RadDropDownList for RadListControl and adding to a RadListDataItems collection, you can reuse all the code above. Here are the results in a RadDropDownList.



Locating Items

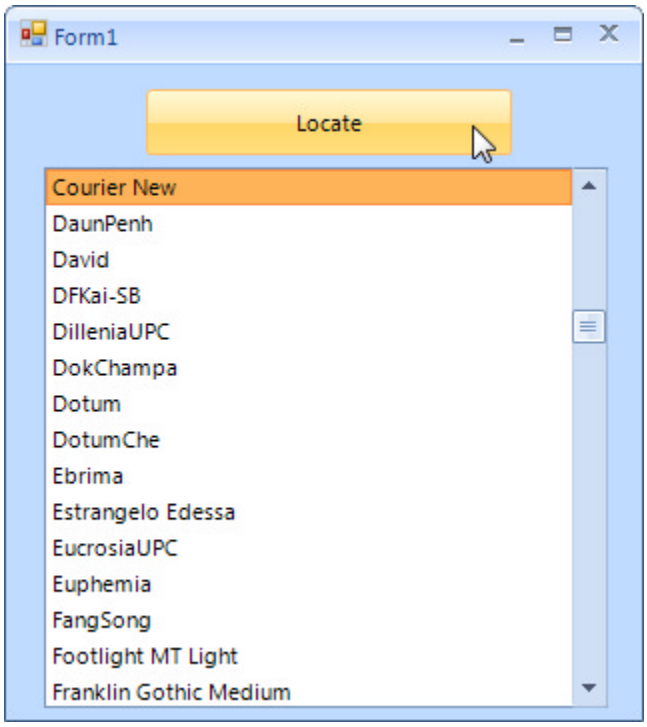
You can use the FindString method to get the index of an item with specified text. If the index eqals "-1", this means that no items with the specified text is found. For example, the code below finds the "Courier New" font family in the Items collection, scrolls to the item and selects it.

[VB] Locating, Scrolling to and Selecting an Item

```
Private Sub btnLocate_Click(sender As Object, e As EventArgs)
 Dim index As Integer = radListControl1.FindString("Courier New")
 Dim item As RadListDataItem = radListControl1.Items(index)
 If item IsNot Nothing Then
  radListControl1.ListElement.ScrollToItem(item)
  radListControl1.SelectedItem = item
End If
End Sub
```

[C#] Locating, Scrolling to and Selecting an Item

```
private void btnLocate_Click(object sender, EventArgs e)
{
int index = radListControl1.FindString("Courier New");
RadListDataItem item = radListControl1.Items[index];
if (item != null)
{
radListControl1.ListElement.ScrollToItem(item);
radListControl1.SelectedItem = item;
}
}
```



Removing Items

To remove an item use the item collection Remove() method and pass a RadListDataItem that should be removed. If you know the ordinal position of the item, call RemoveAt() and pass the index. To remove all items at once, call the Clear() method.

Sorting

Both RadListControl and RadDropDownList have **SortStyle** properties that can be **None**, **Ascending** or **Descending**. The sort is performed against the Text property.

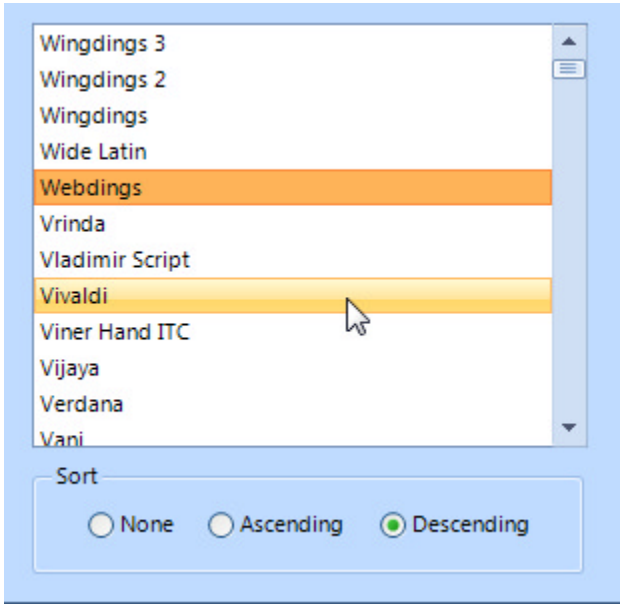
[VB] Setting the Sort

```
Private Sub Sort_ToggleStateChanged(sender As Object, args As StateChangedEventArgs)
        If sender = rbNone Then
                radListControl1.SortStyle = Telerik.WinControls.Enumerations.SortStyle.None
        ElseIf sender = rbAscending Then
                radListControl1.SortStyle =
Telerik.WinControls.Enumerations.SortStyle.Ascending
        ElseIf sender = rbDescending Then
                radListControl1.SortStyle =
Telerik.WinControls.Enumerations.SortStyle.Descending
        End If
End Sub
```

[C#] Setting the Sort

```
private void Sort_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
if (sender == rbNone)
radListControl1.SortStyle = Telerik.WinControls.Enumerations.SortStyle.None;
else if (sender == rbAscending)
radListControl1.SortStyle = Telerik.WinControls.Enumerations.SortStyle.Ascending;
else if (sender == rbDescending)
```

```
radListControl1.SortStyle = Telerik.WinControls.Enumerations.SortStyle.Descending;
}
```



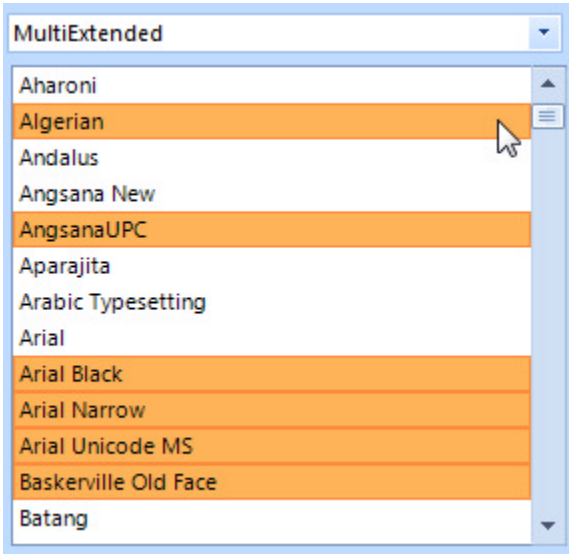
RadListControls

Selecting Items

The **SelectionMode** property determines if items in the list can be selected, if multiple items can be selected and the selection behavior when selecting multiple items.

- A value of **None** prevents items from being selected.
- A value of **One** allows one item at a time to be selected. Selecting an item removes previous selections.
- **MultiSimple** allows the user to hold down the control or shift key and select multiple items, one at a time.
- **MultiExtended** allows the user to hold down the control key to select multiple items or to hold down the shift key and select multiple items in a range.

Run the sample project and select from the dropdown list to experiment with the selection behavior.





You can find the complete source for this project at:

\ListControl\<VB|CS>\Programming

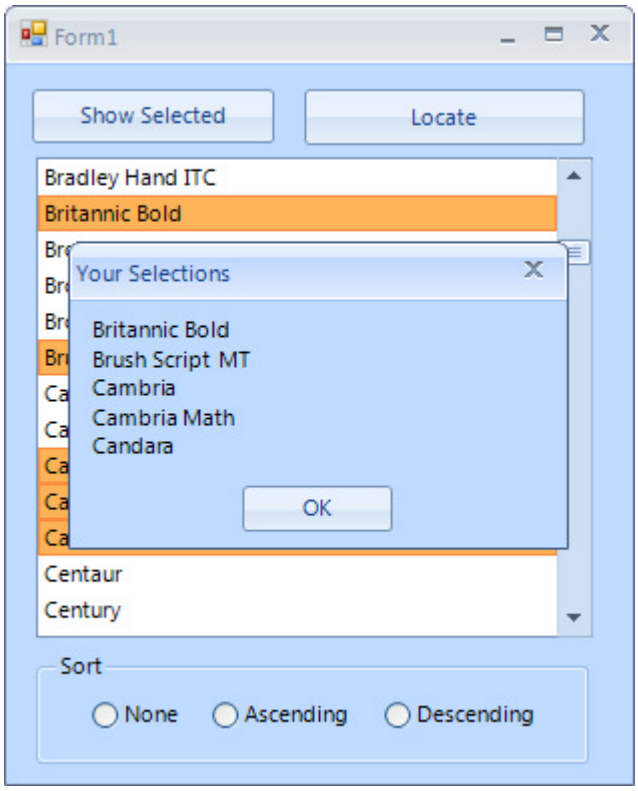
To retrieve multiple selected items, iterate the RadListControl **SelectedItems** collection.

[VB] Iterating the SelectedItems Collection

```
Private Sub btnShowSelected_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnShowSelected.Click
Dim builder As New StringBuilder()
For Each item As RadListDataItem In RadListControl1.SelectedItems
builder.Append(item.Text)
builder.Append(Environment.NewLine)
Next
RadMessageBox.SetThemeName(Me.ThemeName)
RadMessageBox.Show(builder.ToString(), "Your Selections")
End Sub
```

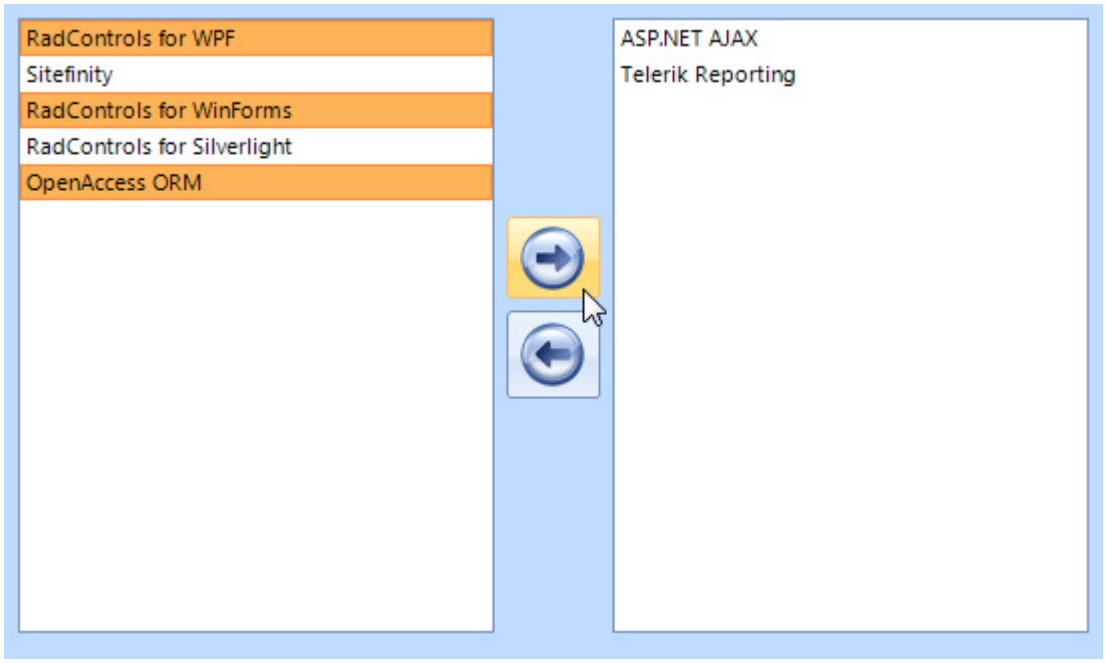
[C#] Iterating the SelectedItems Collection

```
private void btnShowSelected_Click(object sender, EventArgs e)
{
StringBuilder builder = new StringBuilder();
foreach (RadListDataItem item in radListControl1.SelectedItems)
{
builder.Append(item.Text);
builder.Append(Environment.NewLine);
}
RadMessageBox.SetThemeName(this.ThemeName);
RadMessageBox.Show(builder.ToString(), "Your Selections");
}
```



Moving Files Between List Controls

You may need to move items between list controls when you have some set of possible items and the user needs to select a subset. When moving items between two RadListControls once you add the item to the target list control it is automatically removed from the source list control; an item can belong only to one RadListDataItemCollection at a time. In the example below, the SelectionMode property for both list controls is MutiExtended.



[VB] Moving Items Between List Controls

```
Private Sub MoveItems(source As RadListControl, target As RadListControl)
        Dim itemsToMove As New List(Of RadListDataItem)()
        For Each item As RadListDataItem In source.SelectedItems
                itemsToMove.Add(item)
        Next

        For Each item As RadListDataItem In itemsToMove
                target.Items.Add(item)
        Next
End Sub

Private Sub btnLeft_Click(sender As Object, e As EventArgs)
        MoveItems(lcRight, lcLeft)
End Sub

Private Sub btnRight_Click(sender As Object, e As EventArgs)
        MoveItems(lcLeft, lcRight)
End Sub
```

[C#] Moving Items Between List Controls

```
private void MoveItems(RadListControl source, RadListControl target)
{
List<RadListDataItem> itemsToMove = new List<RadListDataItem>();
foreach (RadListDataItem item in source.SelectedItems)
{
itemsToMove.Add(item);
}
foreach (RadListDataItem item in itemsToMove)
{
target.Items.Add(item);
}
}
private void btnLeft_Click(object sender, EventArgs e)
{
MoveItems(lcRight, lcLeft);
}
private void btnRight_Click(object sender, EventArgs e)
{
MoveItems(lcLeft, lcRight);
}
```

Drag and Drop Between ListControls



You can find the complete source for this project at:

\ListControl\<VB|CS>\MovingItems

To implement drag and drop you minimally need to:

- Set the **AllowDrop** property of the target control to true.
- Define a **MouseDown** event handler for the source control where you call the **DoDragDrop**() method. You can pass an object in this method that identifies what is being dragged and specify what kind of drag operation you want to perform (move, copy, etc).
- Define a **DragEnter** event handler for the target control where you determine if the drag drop will be permitted. Arguments to this event handler provide the object being dragged and the drag operation being

performed.

- Define a **DragDrop** event handler for the target control that performs the actual work suggested by the type of drag drop operation requested (i.e. move, copy, etc). Here we can reuse the MoveItems() method created in the previous exercise.

This works well when a single object is being dragged at a time, but with list controls we may want to select multiple items and drag them. To make this work, add these steps:

- On **MouseDown**, Don't call DoDragDrop(). Instead, create a rectangle with the mouse position at its center. Store this rectangle in a private variable for use within other event handlers. Typically this rectangle is about 4 pixels wide and is defined by **SystemInformation.DragSize**.
- On **MouseMove**, check to see if the mouse has moved outside the rectangle, and if so, call **DoDragDrop()** here.
- On **MouseUp**, clear the variable holding the rectangle.

Thats enough to allow your user to select multiple items in a list control and drag them to another list control.



Gotcha!

There's a small usability issue that crops up. Item selection only occurs after MouseUp. So if you click an item and directly begin to drag, the selection hasn't happened yet and the drag operation doesn't see the item. So if you have two items already selected and click a third and drag, the first two are dropped but the last item gets missed.

To work around this you need to detect the RadListDataItem under the mouse on MouseDown but before MouseUp. Save a reference to this item and add it to the SelectedItems collection before calling MoveItems(). That way you can keep your MoveItems() logic undisturbed and all the items will be dragged and dropped.

1. Set the RadListControl **SelectionMode** and **AllowDrop** properties either at design time or in code. **Note**: you don't need to set SelectionMode to make drag and drop work, but we want to show multiple files being dragged.

   [VB] Set RadListControls Properties

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
lcLeft.SelectionMode = SelectionMode.MultiExtended
lcLeft.AllowDrop = True
lcRight.SelectionMode = SelectionMode.MultiExtended
lcRight.AllowDrop = True
End Sub
```

   [C#] Set RadListControls Properties

```
private void Form1_Load(object sender, EventArgs e)
{
lcLeft.SelectionMode = SelectionMode.MultiExtended;
lcLeft.AllowDrop = true;
lcRight.SelectionMode = SelectionMode.MultiExtended;
lcRight.AllowDrop = true;
}
```

2. Add a private variable to hold the "drag rectangle" that defines the zone the mouse must travel outside of before the drag begins. Also add a variable to hold the RadListDataItem that is directly under the mouse on MouseDown.

   [VB] Variables for the Drag Rectangle and Current RadListDataItem

```
Private _dragRectangle As Rectangle = Rectangle.Empty
Private _currentItem As RadListDataItem
```

[C#] Variables for the Drag Rectangle and Current RadListDataItem

```
private Rectangle _dragRectangle = Rectangle.Empty;
private RadListDataItem _currentItem;
```

3. Handle the MouseDown event.
   - Get a reference to the source RadListControl.
   - Use the RadListControl ElementTree **GetElementAtPoint()** method to extract the RadListDataItem that is directly under the mouse and save it in "_currentItem".
   - Create a Rectangle around the current mouse cursor position. Do this my creating a point above and to the left of the mouse cursor, where the distance is half the size of your rectangle dimensions. The rectangle dimensions are supplied by the **SystemInformation.DragSize** (typically 4 pixels in both dimensions).

   [VB] Handling the MouseDown Event

```
Private Sub lcLeft_MouseDown(ByVal sender As Object, ByVal e As MouseEventArgs) Handl
lcLeft.MouseDown
' get the listcontrol that items are being dragged from
Dim sourceListControl As RadListControl = TryCast(sender, RadListControl)
' get the list control item directly under the mouse
Dim element As RadElement = sourceListControl.ElementTree.GetElementAtPoint(e.Locatic
Dim visualItem As RadListVisualItem = TryCast(element, RadListVisualItem)
If visualItem IsNot Nothing Then
_currentItem = visualItem.Data
' create a "drag rectangle" with mouse in center
Dim dragPoint As New Point(e.X - (SystemInformation.DragSize.Width / 2), e.Y -
(SystemInformation.DragSize.Height / 2))
_dragRectangle = New Rectangle(dragPoint, SystemInformation.DragSize)
End If
End Sub
```

   [C#] Handling the MouseDown Event

```
private void lcLeft_MouseDown(object sender, MouseEventArgs e)
{
// get the listcontrol that items are being dragged from
RadListControl sourceListControl = sender as RadListControl;
// get the list control item directly under the mouse
RadElement element = sourceListControl.ElementTree.GetElementAtPoint(e.Location);
RadListVisualItem visualItem = element as RadListVisualItem;
if (visualItem != null)
{
_currentItem = visualItem.Data;
// create a "drag rectangle" with mouse in center
Point dragPoint = new Point(
e.X - (SystemInformation.DragSize.Width / 2),
e.Y - (SystemInformation.DragSize.Height / 2));
_dragRectangle = new Rectangle(dragPoint, SystemInformation.DragSize);
}
}
```

4. Handle the MouseMove event. First check that we have at least one item to drop, then determine if the mouse has moved outside the "drag rectangle" by using the Rectangle Contains() method. If the mouse has moved outside, then call the source RadListControl DoDragDrop() method. Pass a reference to the source RadListControl itself and a **DragDropEffects** flag that indicates that this is a Move operation.

[VB] Handle the MouseMove Event

```
Private Sub lcLeft_MouseMove(ByVal sender As Object, ByVal e As MouseEventArgs) Handles
lcLeft.MouseMove
Dim sourceListControl As RadListControl = TryCast(sender, RadListControl)
' we have at least one item to drop
If _currentItem IsNot Nothing Then
' we have moved outside the rectangle, so start the drag drop
If Not _dragRectangle.Contains(e.X, e.Y) Then
sourceListControl.DoDragDrop(sourceListControl, DragDropEffects.Move)
End If
End If
End Sub
```

[C#] Handle the MouseMove Event

```
private void lcLeft_MouseMove(object sender, MouseEventArgs e)
{
RadListControl sourceListControl = sender as RadListControl;
// we have at least one item to drop
if (_currentItem != null)
{
// we have moved outside the rectangle, so start the drag drop
if (!_dragRectangle.Contains(e.X, e.Y))
{
sourceListControl.DoDragDrop(sourceListControl, DragDropEffects.Move);
}
}
}
```

5. Handle the **MouseUp** event, clear the _currentItem and the _dragRectangle members:

[VB] Handling the MouseUp Event

```
Private Sub lcLeft_MouseUp(ByVal sender As Object, ByVal e As MouseEventArgs) Handles
lcLeft.MouseUp
' we're no longer dragging, so clear these temporary variables
_dragRectangle = Rectangle.Empty
_currentItem = Nothing
End Sub
```

[C#] Handling the MouseUp Event

```
private void lcLeft_MouseUp(object sender, MouseEventArgs e)
{
// we're no longer dragging, so clear these temporary variables
_dragRectangle = Rectangle.Empty;
_currentItem = null;
}
```

6. Handle the **DragEnter** event. The purpose of this event handler is to set the DragEventArgs Effect flag to accept the drag or not. First call the DragEventArgs Data.GetDataPresent() method and pass the RadListControl type. This verifies that is an attempt to drop a RadListControl. Then get a reference to the source RadListControl that was sent and compare it with RadListControl in "sender". This is to verify that we're not dropping items within the same RadListControl . If all these checks pass then set the Effect property to DragDropEffects.Move.

[VB] Handle the DragEnter Event

```
Private Sub lcRight_DragEnter(ByVal sender As Object, ByVal e As DragEventArgs) Handles
lcRight.DragEnter
```

```
' verify that the drag contains the RadListControl and that
' we're not dragging onto the same list control.
If e.Data.GetDataPresent(GetType(RadListControl)) Then
Dim sourceListControl As RadListControl = DirectCast(e.Data.GetData(GetType
(RadListControl)), RadListControl)
If sourceListControl IsNot sender Then
e.Effect = DragDropEffects.Move
End If
End If
End Sub
```

[C#] Handle the DragEnter Event

```
private void lcRight_DragEnter(object sender, DragEventArgs e)
{
// verify that the drag contains the RadListControl and that
// we're not dragging onto the same list control.
if (e.Data.GetDataPresent(typeof(RadListControl)))
{
RadListControl sourceListControl =
(RadListControl)e.Data.GetData(typeof(RadListControl));
if (sourceListControl != (sender as RadListControl))
e.Effect = DragDropEffects.Move;
}
}
```

7. Handle the **DragDrop** event. For this event to even fire there is at least one item to drop in _currentItem, then call MoveItems(). Finally, clear the _dragRectangle and _currentItem members now that we've completed the drop.

[VB] Handle the DragDrop Event

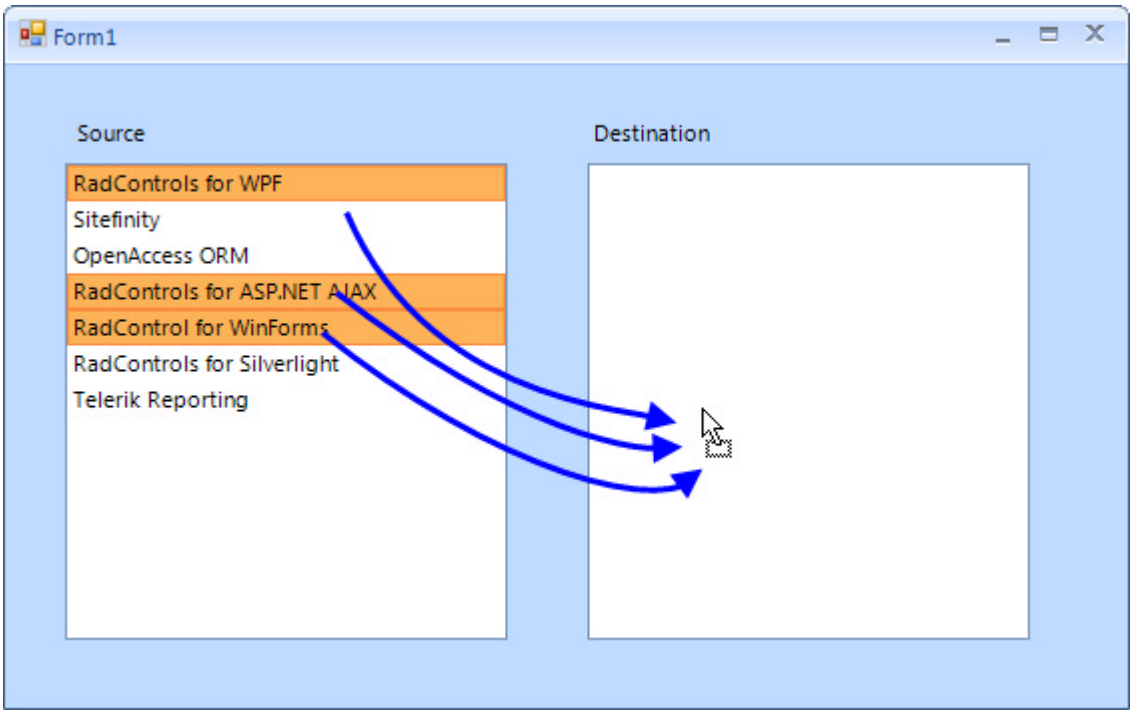
```
Private Sub lcRight_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs) Handles
lcRight.DragDrop
' get a reference to the source list control,
' and move the items.
Dim sourceListControl As RadListControl = DirectCast(e.Data.GetData(GetType
(RadListControl)), RadListControl)
If _currentItem.Selected = False Then
_currentItem.Selected = True
End If
MoveItems(sourceListControl, TryCast(sender, RadListControl))
_dragRectangle = Rectangle.Empty
_currentItem = Nothing
End Sub
```

[C#] Handle the DragDrop Event

```
private void lcRight_DragDrop(object sender, DragEventArgs e)
{
// get a reference to the source list control,
// and move the items.
RadListControl sourceListControl =
(RadListControl)e.Data.GetData(typeof(RadListControl));
if (!_currentItem.Selected)
{
_currentItem.Selected = true;
}
MoveItems(sourceListControl, (sender as RadListControl));
```

```
_dragRectangle = Rectangle.Empty;
_currentItem = null;
}
```

8. Make sure that the Mouse and DragDrop event handlers are defined for both list controls.
9. Now when you run the application you can select multiple items and drag them between lists.



RadDropDownList

RadDropDownList is essentially a RadListControl in a dropdown. It shares many of the behaviors of RadListControl, but has some additional aspects particular to the dropping down of the list and the text entry.

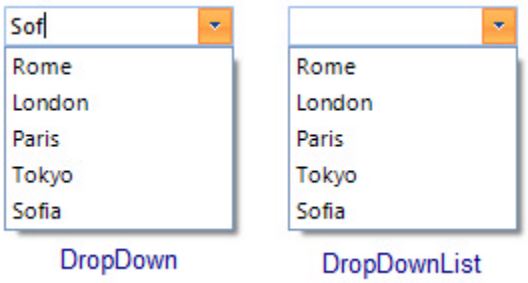
Drop Down Dimensions

DropDownSizingMode controls what drag bars are available to resize the drop down list portion of the control:

- **None** to disallow resizing entirely.
- **UpDownAndRightBottom** to allow both horizontal and vertical resizing.
- **UpDown** to allow only vertical resizing.
- **RightBottom** to allow only horizontal resizing.

Editing

DropDownStyle determines if the text area at the top of the control can be edited. A setting of **DropDown** (the default) allows editing and the **DropDownList** setting shows the text area as read-only.



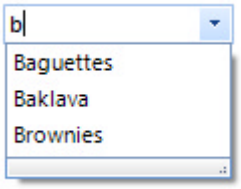
AutoComplete

RadDropDownList can automatically "suggest", or navigate to the closest match in the list as the user types and can append the closest choice to the entry in the textbox portion of the dropdown list. **AutoCompleteMode** controls this behavior:

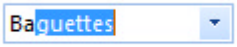
- **None**: Nothing happens when a user begins to type into the text box portion of the control. If the user types the whole text of an item and presses Enter, the item is selected.



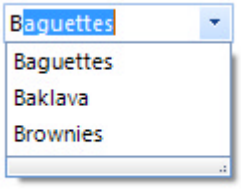
- **Suggest**: As the user types an entry into the text box, the drop-down portion of the control is shown and the items are filtered by the input text.



- **Append**: As the user types, the next item in the list that matches the user input is automatically appended to the characters the user has already typed. The drop-down list is not shown without the user clicking the arrow.



- **SuggestAppend**: Similar to the Append setting, but the drop-down list is shown and the filtered items according the input are shown.



Selecting Items

Unlike RadListControls, RadDropDownList has a single item selected at any time. Set the **SelectedIndex** to an ordinal value or set the **SelectedItem** property to a RadListDataItem instance.

[VB] Setting the Selected item

```
radDropDownList1.SelectedIndex = 2
radDropDownList1.SelectedItem = radDropDownList1.Items(3)
```

[C#] Setting the SelectedIndex item

```
radDropDownList1.SelectedIndex = 2;
radDropDownList1.SelectedItem = radDropDownList1.Items[3];
```

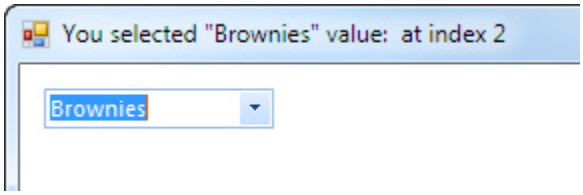
Respond to changes in selection by handling the **SelectedIndexChanged** event. SelectedIndexChanged event fires when the index of the current list item changes. The index may change as a result of user selection with a mouse or keyboard navigation or programmatic assignment of the index. Note that with a data-bound dropdown list, SelectedIndexChanged will be hit as the data is being bound to the control. When the data binding is done through the designer using the property window, the SelectedIndex will change again as the form is unloaded.

[VB] Handling the SelectedIndexChanged Event

```
Private Sub RadDropDownList1_SelectedIndexChanged(sender As Object, e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles
RadDropDownList1.SelectedIndexChanged
Me.Text = "You selected \"" + radDropDownList1.SelectedText + " \ " value: " +
RadDropDownList1.SelectedValue + " at index " + RadDropDownList1.SelectedIndex.ToString()
End Sub
```

[C#] Handling the SelectedIndexChanged Event

```
void radDropDownList1_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
this.Text = "You selected \"" + radDropDownList1.SelectedText + "\" value: " +
radDropDownList1.SelectedValue + " at index " + radDropDownList1.SelectedIndex.ToString();
}
```



Programmatically Opening and Closing the Drop Down

Open and close the drop down using the ShowDropDown() and CloseDropDown() methods respectively. In this example we locate the "Departures" item in the list, select it and open the drop down.

[VB] Showing the Drop Down

```
Private Sub radButton1_Click(sender As Object, e As EventArgs)
radDropDownList1.Items(radDropDownList1.FindString("Brownies")).Selected = True
radDropDownList1.ShowDropDown()
End Sub
```

[C#] Showing the Drop Down

```
private void radButton1_Click(object sender, EventArgs e)
{
radDropDownList1.Items[radDropDownList1.FindString("Brownies")].Selected = true;
radDropDownList1.ShowDropDown();
}
```

Walkthrough

In this guided walk-through we will enhance the original "Getting Started" project to fine-tune and add features:

- Display applications icons using the Windows Shell API
- Display a small icon in the edit portion of the dropdown list
- Open files in response to double-clicking the file list
- Display custom tool tips for each file item

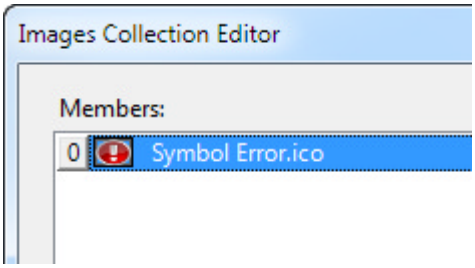
Preparing the Project



You can find the complete source for this project at:

\ListControl\<VB|CS>\Walkthrough

- Start with the "GettingStarted" project or a copy.
- In the designer, add a standard ImageList component. Using the Smart Tag **Choose Images** link, add an image to represent "Error". In this application the image key will be "Symbol Error.ico".



- In the designer, select the "ddDrives" RadDropDownList and in the Properties window Events tab, remove the SelectedIndexChanged event. Later we will connect this event programmatically.
- In the Solution Explorer, add a new class file "Shell32.cs". Add the code below.

  This class is a wrapper for the Shell32.dll that contains Windows Shell API functions, including a function to retrieve a bitmap from any file that contains an icon for itself. For more information, consult the online MSDN for articles on the Shell API and the System.Runtime.InteropServices namespace.

  [VB] Defining the Shell32 API Wrapper Class

```
Imports System.Runtime.InteropServices
' Declares types and methods that can be called from the DLL
Public Class Shell32
Public Shared Function GetFileBitmap(ByVal fileName As String) As Bitmap
Dim result As Bitmap = Nothing
Dim shFileInfo As New SHFILEINFO()
Dim returnCode As IntPtr = SHGetFileInfo(fileName, 0, shFileInfo, Marshal.SizeOf
(shFileInfo), SHGFI_ICON)
If shFileInfo.hIcon <> IntPtr.Zero Then
Dim icon As System.Drawing.Icon = System.Drawing.Icon.FromHandle(shFileInfo.hIcon)
result = icon.ToBitmap()
DestroyIcon(shFileInfo.hIcon)
icon.Dispose()
```

```
End If
Return result
End Function
<StructLayout(LayoutKind.Sequential)> _
Public Structure SHFILEINFO
Public hIcon As IntPtr
Public iIcon As IntPtr
Public dwAttributes As UInteger
<MarshalAs(UnmanagedType.ByValTStr, SizeConst:=260)> _
Public szDisplayName As String
<MarshalAs(UnmanagedType.ByValTStr, SizeConst:=80)> _
Public szTypeName As String
End Structure
Public Const SHGFI_ICON As UInteger = &H100
Public Const SHGFI_LARGEICON As UInteger = &H0
<DllImport("shell32.dll")> _
Public Shared Function SHGetFileInfo(ByVal path As String, ByVal dwFileAttributes As
UInteger, ByRef shfi As SHFILEINFO, ByVal cbSizeFileInfo As UInteger, ByVal uFlags As
UInteger) As IntPtr
End Function
<DllImport("user32.dll")> _
Public Shared Function DestroyIcon(ByVal hIcon As IntPtr) As Int32
End Function
End Class
```

[C#] Defining the Shell32 API Wrapper Class

```
using System.Runtime.InteropServices;
// Declares types and methods that can be called from the DLL
public class Shell32
{
public static Bitmap GetFileBitmap(string fileName)
{
Bitmap result = null;
SHFILEINFO shFileInfo = new SHFILEINFO();
IntPtr returnCode = SHGetFileInfo(fileName, 0, ref shFileInfo,
(uint)Marshal.SizeOf(shFileInfo), SHGFI_ICON);
if (shFileInfo.hIcon != IntPtr.Zero)
{
System.Drawing.Icon icon = System.Drawing.Icon.FromHandle(shFileInfo.hIcon);
result = icon.ToBitmap();
```

```
DestroyIcon(shFileInfo.hIcon);
icon.Dispose();
}
return result;
}
[StructLayout(LayoutKind.Sequential)]
public struct SHFILEINFO
{
public IntPtr hIcon;
public IntPtr iIcon;
public uint dwAttributes;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 260)]
public string szDisplayName;
[MarshalAs(UnmanagedType.ByValTStr, SizeConst = 80)]
public string szTypeName;
};
public const uint SHGFI_ICON = 0x100;
public const uint SHGFI_LARGEICON = 0x0;
[DllImport("shell32.dll")]
public static extern IntPtr SHGetFileInfo(string path,
uint dwFileAttributes,
ref SHFILEINFO shfi,
uint cbSizeFileInfo,
uint uFlags);
[DllImport("user32.dll")]
public static extern Int32 DestroyIcon(IntPtr hIcon);
}
```

- Make sure the following namespace references exist in the "Imports" (VB) or "uses" (C#) clause. These references will support the Windows Shell API methods and other features as they are added.

[VB] Adding Namespace References

```
Imports System
Imports System.Drawing
Imports System.IO
Imports System.Text
Imports System.Windows.Forms
Imports Telerik.WinControls.Primitives
Imports Telerik.WinControls.UI
```

[C#] Adding Namespace References

```
using System;
using System.Drawing;
using System.IO;
```

```
using System.Text;
using System.Windows.Forms;
using Telerik.WinControls.Primitives;
using Telerik.WinControls.UI;
```

- Add helper methods to work with FileSystemInfo and DirectoryInfo objects.

[VB] Directory Helper Methods

```
Private Function IsRootDirectory(ByVal info As DirectoryInfo) As Boolean
Return info.Root.FullName.Equals(info.FullName)
End Function
Private Function IsDirectory(ByVal info As FileSystemInfo) As Boolean
Return (info.Attributes And FileAttributes.Directory) = FileAttributes.Directory
End Function
Private Function IsHidden(ByVal info As FileSystemInfo) As Boolean
Return (info.Attributes And FileAttributes.Hidden) = FileAttributes.Hidden
End Function
Private Function GetParentDirectory(ByVal info As FileSystemInfo) As DirectoryInfo
Return New DirectoryInfo(Path.GetDirectoryName(info.FullName))
End Function
```

[C#] Directory Helper Methods

```
private bool IsRootDirectory(DirectoryInfo info)
{
return info.Root.FullName.Equals(info.FullName);
}
private bool IsDirectory(FileSystemInfo info)
{
return (info.Attributes & FileAttributes.Directory) == FileAttributes.Directory;
}
private bool IsHidden(FileSystemInfo info)
{
return (info.Attributes & FileAttributes.Hidden) == FileAttributes.Hidden;
}
private DirectoryInfo GetParentDirectory(FileSystemInfo info)
{
return new DirectoryInfo(Path.GetDirectoryName(info.FullName));
}
```

- Add private members that will be used to reference an image element in the dropdown list text area and a reference to the list item under the mouse during the last MouseDown.

[VB] Add Private Members

```
' reference to element for image in dropdown list text area
Private _imagePrimitive As ImagePrimitive
' stores a reference to item under last mouse down
Private _currentItem As RadListDataItem
```

[C#] Add Private Members

```
// reference to element for image in dropdown list text area
private ImagePrimitive _imagePrimitive;
// stores a reference to item under last mouse down
private RadListDataItem _currentItem;
```

- Add status reporting helper methods.

There are several overloads for the ShowStatus() method that take a message/image, an exception or a RadListDataItem. GetFileSystemDescription returns a string with information about a FileSystemInfo object passed in. The DescriptionDetail enumeration passed in to the second parameter can be "Full" to include statistics on file creation and modify dates.

[VB] Adding Status Reporting Helper Methods

```
' display a status message and accompanying image 16px thumbnail
Private Sub ShowStatus(ByVal message As String, ByVal image As Image)
lblStatus.Text = message
lblStatus.TextImageRelation = TextImageRelation.ImageBeforeText
If image IsNot Nothing Then
lblStatus.Image = image.GetThumbnailImage(16, 16, Nothing, New IntPtr())
End If
End Sub
' display exception message in status label with error icon 16px thumbnail
Private Sub ShowStatus(ByVal ex As Exception)
ShowStatus(ex.Message, imageList1.Images("symbol error.ico"))
End Sub
' show file/directory description as status message and 16px icon
Private Sub ShowStatus(ByVal selectedItem As RadListDataItem)
If selectedItem IsNot Nothing Then
Dim item As RadListDataItem = TryCast(selectedItem, RadListDataItem)
Dim info As FileSystemInfo = TryCast(item.Value, FileSystemInfo)
Dim message As String = GetFileSystemDescription(info, DescriptionDetail.[Partial])
ShowStatus(message, item.Image)
End If
End Sub
' scope of description
Private Enum DescriptionDetail
[Partial]
Full
End Enum
' build a string with number of objects/bytes in directory/file.
' if DescriptionDetail is "Full", add "Created"/"Updated" statistics
Private Function GetFileSystemDescription(ByVal info As FileSystemInfo, ByVal detail As
DescriptionDetail) As String
Dim builder As New StringBuilder()
If TypeOf info Is DirectoryInfo Then
Dim dirInfo As DirectoryInfo = TryCast(info, DirectoryInfo)
Dim size As Long = dirInfo.GetDirectories().Length + dirInfo.GetFiles().Length
builder.Append(size.ToString())
builder.Append(" objects")
builder.Append(System.Environment.NewLine)
Else
Dim fileInfo As FileInfo = TryCast(info, FileInfo)
builder.Append(fileInfo.Length.ToString())
builder.Append(" bytes")
builder.Append(System.Environment.NewLine)
End If
If detail = DescriptionDetail.Full Then
builder.Append("Created on: ")
builder.Append(info.CreationTime.ToString())
builder.Append(System.Environment.NewLine)
builder.Append("Updated on: ")
builder.Append(info.LastWriteTime.ToString())
```

```
End If
Return builder.ToString()
End Function
```

[C#] Adding Status Reporting Helper Methods

```
// display a status message and accompanying image 16px thumbnail
private void ShowStatus(string message, Image image)
{
lblStatus.Text = message;
lblStatus.TextImageRelation = TextImageRelation.ImageBeforeText;
if (image != null)
{
lblStatus.Image = image.GetThumbnailImage(16, 16, null, new IntPtr());
}
}
// display exception message in status label with error icon 16px thumbnail
private void ShowStatus(Exception ex)
{
ShowStatus(ex.Message, imageList1.Images["symbol error.ico"]);
}
// show file/directory description as status message and 16px icon
private void ShowStatus(RadListDataItem selectedItem)
{
if (selectedItem != null)
{
RadListDataItem item = selectedItem as RadListDataItem;
FileSystemInfo info = item.Value as FileSystemInfo;
string message = GetFileSystemDescription(info, DescriptionDetail.Partial);
ShowStatus(message, item.Image);
}
}
// scope of description
private enum DescriptionDetail { Partial, Full };
// build a string with number of objects/bytes in directory/file.
// if DescriptionDetail is "Full", add "Created"/"Updated" statistics
private string GetFileSystemDescription(FileSystemInfo info, DescriptionDetail detail)
{
StringBuilder builder = new StringBuilder();
if (info is DirectoryInfo)
{
DirectoryInfo dirInfo = info as DirectoryInfo;
long size = dirInfo.GetDirectories().Length + dirInfo.GetFiles().Length;
builder.Append(size.ToString());
builder.Append(" objects");
builder.Append(System.Environment.NewLine);
}
else
{
FileInfo fileInfo = info as FileInfo;
builder.Append(fileInfo.Length.ToString());
builder.Append(" bytes");
builder.Append(System.Environment.NewLine);
}
if (detail == DescriptionDetail.Full)
{
```

```
builder.Append("Created on: ");
builder.Append(info.CreationTime.ToString());
builder.Append(System.Environment.NewLine);
builder.Append("Updated on: ");
builder.Append(info.LastWriteTime.ToString());
}
return builder.ToString();
}
```

- Add helper methods to load dropdown list and list control. AddDirectoryToDropDownList() already exists in the "GettingStarted" project, so replace it with the code below.

  Both methods AddDirectoryToDropDownList() and LoadFilesListControl() take a DirectoryInfo and a reference to the control (list control or dropdown list) to load. AddDirectoryToDropDownList() loads a single directory path and image for the directory. LoadFilesListControl() iterates the FileSystemInfo objects within a directory and appends a list item for each. Notice that the call to GetFileSystemDescription() traps UnauthorizedAccessException and disables the item.

  [VB] Adding DropDownList and ListControl Helper Methods

```
' Add path and DirectoryInfo object to RadDropDownList
Private Sub AddDirectoryToDropDownList(ByVal info As DirectoryInfo, ByVal dropDownList As
RadDropDownList)
If dropDownList.FindStringExact(info.FullName) = -1 Then
Dim item As New RadListDataItem(info.FullName, info)
item.Text = info.FullName
item.TextImageRelation = TextImageRelation.ImageBeforeText
' use Shell32 api to get appication bitmap
item.Image = Shell32.GetFileBitmap(info.FullName)
dropDownList.Items.Add(item)
End If
dropDownList.SelectedItem = dropDownList.Items(dropDownList.FindStringExact(info.FullName
End Sub
' load a RadListControl with the files in a directory
Private Sub LoadFilesListControl(ByVal directoryInfo As DirectoryInfo, ByVal listControl
RadListControl)
listControl.Items.Clear()
For Each info As FileSystemInfo In directoryInfo.GetFileSystemInfos()
If Not IsHidden(info) Then
' add Text and Value in constructor
Dim item As New RadListDataItem(info.Name, info)
item.TextImageRelation = TextImageRelation.ImageBeforeText
' use the Shell32 API to get bitmap for file
item.Image = Shell32.GetFileBitmap(info.FullName)
listControl.Items.Add(item)
Try
GetFileSystemDescription(DirectCast(item.Value, FileSystemInfo), DescriptionDetail.Full)
' if directory cannot be accessed, disable item and include error message
Catch ex As UnauthorizedAccessException
item.Enabled = False
End Try
End If
Next
If listControl.Items.Count > 0 Then
listControl.SelectedIndex = 0
End If
```

```
End Sub
```

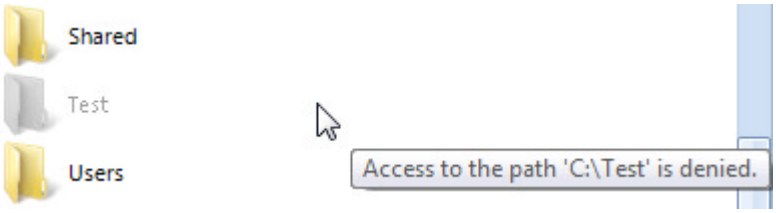
[C#] Adding DropDownList and ListControl Helper Methods

```
// Add path and DirectoryInfo object to RadDropDownList
private void AddDirectoryToDropDownList(DirectoryInfo info, RadDropDownList dropDownList)
{
if (dropDownList.FindStringExact(info.FullName) == -1)
{
RadListDataItem item = new RadListDataItem(info.FullName, info);
item.Text = info.FullName;
item.TextImageRelation = TextImageRelation.ImageBeforeText;
// use Shell32 api to get appication bitmap
item.Image = Shell32.GetFileBitmap(info.FullName);
dropDownList.Items.Add(item);
}
dropDownList.SelectedItem = dropDownList.Items[dropDownList.FindStringExact(info.FullName
}
// load a RadListControl with the files in a directory
private void LoadFilesListControl(DirectoryInfo directoryInfo, RadListControl listControl
{
listControl.Items.Clear();
foreach (FileSystemInfo info in directoryInfo.GetFileSystemInfos())
{
if (!IsHidden(info))
{
// add Text and Value in constructor
RadListDataItem item = new RadListDataItem(info.Name, info);
item.TextImageRelation = TextImageRelation.ImageBeforeText;
// use the Shell32 API to get bitmap for file
item.Image = Shell32.GetFileBitmap(info.FullName);
listControl.Items.Add(item);
try
{
GetFileSystemDescription((FileSystemInfo)item.Value, DescriptionDetail.Full);
}
// if directory cannot be accessed, disable item and include error message
catch (UnauthorizedAccessException ex)
{
item.Enabled = false;
}
}
}
if (listControl.Items.Count > 0)
{
listControl.SelectedIndex = 0;
}
}
```

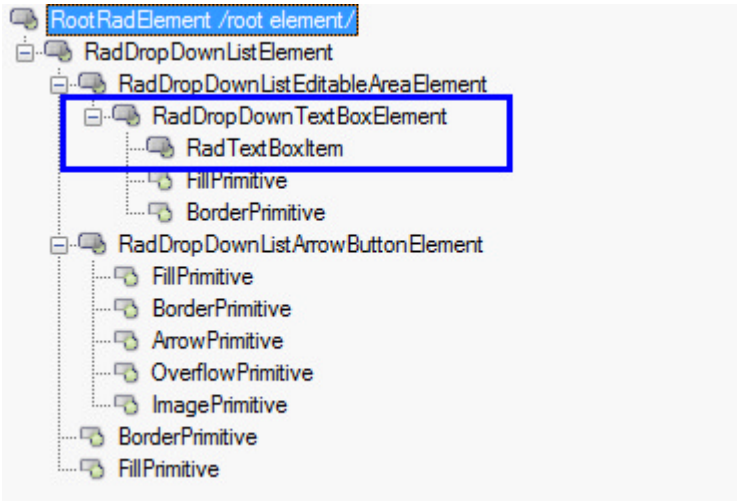
Now that the helper methods are all in place we can rewrite the event handling methods to implement the new features.

- Replace the form Load event with the code below.

The code that adds directories to the dropdown list is identical to the "GettingStarted" project. The purpose of code that follows it is to add an icon just to the left of the item text area. Additionally we subscribe to the ToolTipNeeded event of the control, where we display either the file information or the exception message if UnauthorizedAccessException is thrown.



To display the image on the left side of the text use the element tree from the SmartTag >> Edit UI Element to locate the textbox element. This element is needed because we are going to add the image for the text area in the textbox element children collection.



[VB] Handling the Form Load and ToolTipTextNeeded Events

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles Me.Load
' load the dropdown list with drive names and select the first one
For Each driveInfo__1 As DriveInfo In DriveInfo.GetDrives()
If driveInfo__1.IsReady Then
Dim info As New DirectoryInfo(driveInfo__1.RootDirectory.FullName)
AddDirectoryToDropDownList(info, ddlDrives)
End If
Next
' get references to the textbox element and the textbox item within the element
Dim textBoxElement As RadTextBoxElement = ddlDrives.DropDownListElement.TextBox
Dim textBoxItem As RadTextBoxItem = textBoxElement.TextBoxItem
' Add extra padding on the left for the new image
textBoxItem.Margin = New Padding(20, 1, 1, 1)
' pad the element so textboxitem doesn't overwrite the
' border on the right side
textBoxElement.Padding = New Padding(2, 2, 20, 2)
' save a reference
_imagePrimitive = New ImagePrimitive()
AddHandler lcFiles.ToolTipTextNeeded, AddressOf lcFiles_ToolTipTextNeeded
AddHandler ddlDrives.SelectedIndexChanged, AddressOf ddlDrives_SelectedIndexChanged
'insert the ImagePrimitive to the text box element
textBoxElement.Children.Insert(0, _imagePrimitive)
'hide the FillPrimitive so the image to be displayed
textBoxElement.Fill.Visibility = Telerik.WinControls.ElementVisibility.Collapsed
'select the first item in the dropdown
ddlDrives.SelectedIndex = 0
```

```
'make the items size according to their content
ddlDrives.AutoSizeItems = True
lcFiles.AutoSizeItems = True
End Sub
Private Sub lcFiles_ToolTipTextNeeded(ByVal sender As Object, ByVal e As
Telerik.WinControls.ToolTipTextNeededEventArgs) Handles lcFiles.ToolTipTextNeeded
Dim visualItem As RadListVisualItem = TryCast(sender, RadListVisualItem)
If visualItem IsNot Nothing Then
Try
e.ToolTipText = GetFileSystemDescription(DirectCast(visualItem.Data.Value, FileSystemInfo),
DescriptionDetail.Full)
' if directory cannot be accessed, display error message
Catch ex As UnauthorizedAccessException
e.ToolTipText = ex.Message
End Try
End If
End Sub
```

[C#] Handling the Form Load and ToolTipTextNeeded Events

```
private void Form1_Load(object sender, EventArgs e)
{
// load the dropdown list with drive names and select the first one
foreach (DriveInfo driveInfo in DriveInfo.GetDrives())
{
if (driveInfo.IsReady)
{
DirectoryInfo info = new DirectoryInfo(driveInfo.RootDirectory.FullName);
AddDirectoryToDropDownList(info, ddlDrives);
}
}
// get references to the textbox element and the textbox item within the element
RadTextBoxElement textBoxElement = ddlDrives.DropDownListElement.TextBox;
RadTextBoxItem textBoxItem = textBoxElement.TextBoxItem;
// Add extra padding on the left for the new image
textBoxItem.Margin = new Padding(20, 1, 1, 1);
// pad the element so textboxitem doesn't overwrite the
// border on the right side
textBoxElement.Padding = new Padding(2, 2, 20, 2);
// save a reference
_imagePrimitive = new ImagePrimitive();
lcFiles.ToolTipTextNeeded += new Telerik.WinControls.ToolTipTextNeededEventHandler
(lcFiles_ToolTipTextNeeded);
ddlDrives.SelectedIndexChanged += new
Telerik.WinControls.UI.Data.PositionChangedEventHandler(ddlDrives_SelectedIndexChanged);
//insert the ImagePrimitive to the text box element
textBoxElement.Children.Insert(0, _imagePrimitive);
//hide the FillPrimitive so the image to be displayed
textBoxElement.Fill.Visibility = Telerik.WinControls.ElementVisibility.Collapsed;
//select the first item in the dropdown
ddlDrives.SelectedIndex = 0;
//make the items size according to their content
ddlDrives.AutoSizeItems = true;
lcFiles.AutoSizeItems = true;
}
void lcFiles_ToolTipTextNeeded(object sender, Telerik.WinControls.ToolTipTextNeededEventArgs
```

```
e)
{
RadListVisualItem visualItem = sender as RadListVisualItem;
if (visualItem != null)
{
try
{
e.ToolTipText = GetFileSystemDescription((FileSystemInfo)visualItem.Data.Value,
DescriptionDetail.Full);
}
// if directory cannot be accessed, display error message
catch (UnauthorizedAccessException ex)
{
e.ToolTipText = ex.Message;
}
}
}
```

- Replace the SelectedIndexChanged event with the code below.

  You'll see this pattern of code in several places where the cursor is set to the "WaitCursor", if there's an exception, the ShowStatus() method displays the exception message and finally the cursor is restored to its default. The code inside the Try{} block is much the same as the "GettingStarted" example except now the "Back" button is disabled if we're already at the root. Also, here's where you set the custom image for the RadDropDownList text area.

  [VB] Handling the SelectedIndexChanged Event

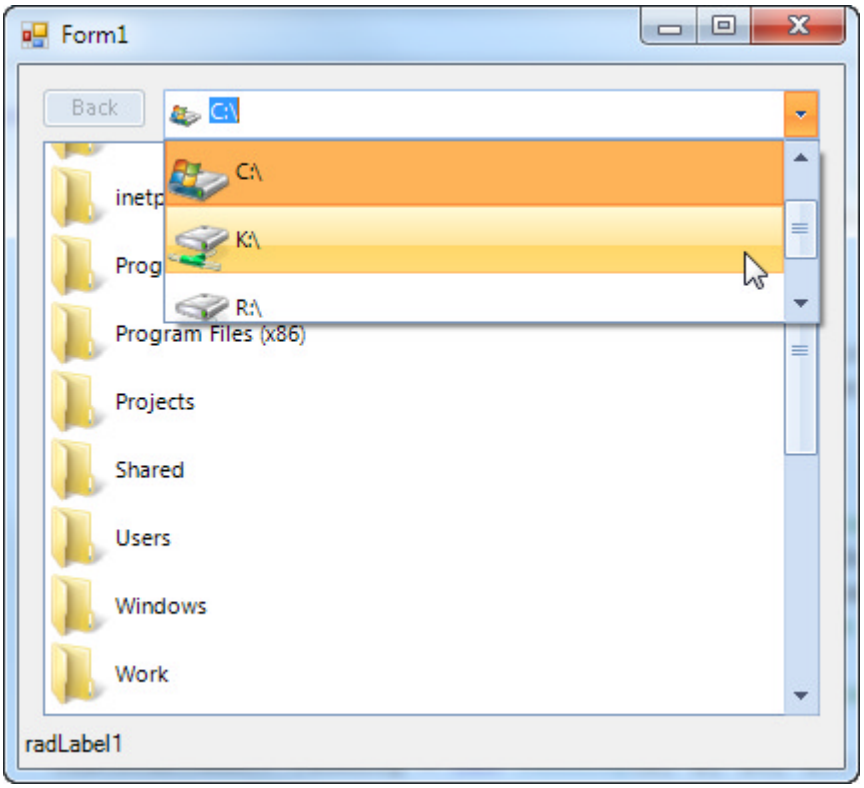
```
Private Sub ddlDrives_SelectedIndexChanged(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs)
Cursor.Current = Cursors.WaitCursor
Try
Dim directoryInfo As DirectoryInfo = TryCast(ddlDrives.SelectedValue, DirectoryInfo)
btnBack.Enabled = Not IsRootDirectory(directoryInfo)
LoadFilesListControl(directoryInfo, lcFiles)
Dim selectedImage As Image = TryCast(ddlDrives.SelectedItem, RadListDataItem).Image
_imagePrimitive.Image = selectedImage.GetThumbnailImage(16, 16, Nothing, New IntPtr())
Catch ex As UnauthorizedAccessException
ShowStatus(ex)
Finally
Cursor.Current = Cursors.[Default]
End Try
End Sub
```

  [C#] Handling the SelectedIndexChanged Event

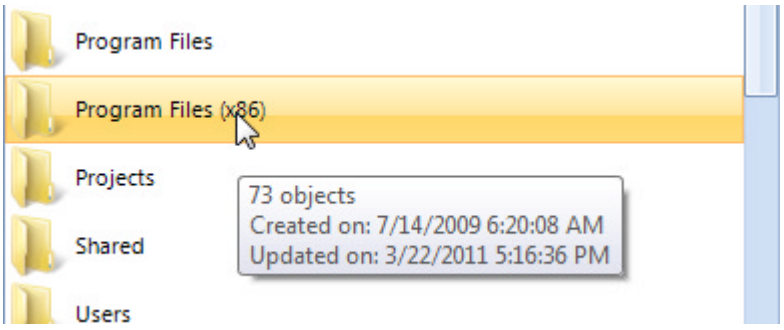
```
void ddlDrives_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
Cursor.Current = Cursors.WaitCursor;
try
{
DirectoryInfo directoryInfo = ddlDrives.SelectedValue as DirectoryInfo;
btnBack.Enabled = !IsRootDirectory(directoryInfo);
LoadFilesListControl(directoryInfo, lcFiles);
Image selectedImage = (ddlDrives.SelectedItem as RadListDataItem).Image;
_imagePrimitive.Image = selectedImage.GetThumbnailImage(16, 16, null, new IntPtr());
```

```
}
catch (UnauthorizedAccessException ex)
{
ShowStatus(ex);
}
finally
{
Cursor.Current = Cursors.Default;
}
}
```

- Press **Ctrl-F5** to run the application so far. Icons should display next to all the drives in the drop down and in the list control as well. The textbox portion of the RadDropDownList should have a thumbnail representation of the drive icon.



The tool tip should display timestamp statistics and the number of bytes/objects in each folder/file.



- Add methods to handle "lcFiles" RadListControl events.

The SelectedIndexChanged event handler simply calls the ShowStatus overload that accepts a RadListDataItem.

The MouseDown, MouseUp and DoubleClick events require a little more explanation. If a directory is inaccessible (i.e. throws a UnauthorizedAccessException) then the item can't be selected. We want the user to be able to double-click a directory or file and open it, but if we open the selected item and double-click an inaccessible item, the selected item gets opened. To fix this, you can get the RadListDataItem at the mouse location on MouseDown and save it. When the user double-clicks, check that the saved RadListDataItem is not null and use it instead of the selected item.

The main logic occurs in the DoubleClick event where the directory is opened or if its a file, the file is passed to System.Diagnostics.Process.Start().

> ⚠ *When testing this, be careful what you click on. System.Diagnostics.Process.Start() will run whatever executable you double-click.*

[VB] Handling RadListControl Events

```
Private Sub lcFiles_SelectedIndexChanged(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles lcFiles.SelectedIndexChange
If lcFiles.SelectedItem IsNot Nothing AndAlso lcFiles.SelectedItem.Enabled Then
ShowStatus(lcFiles.SelectedItem)
End If
End Sub
Private Sub lcFiles_MouseDown(ByVal sender As Object, ByVal e As MouseEventArgs) Handles
lcFiles.MouseDown
Dim item As RadListVisualItem = TryCast(lcFiles.ElementTree.GetElementAtPoint(e.Location)
RadListVisualItem)
If item IsNot Nothing Then
_currentItem = item.Data
End If
End Sub
Private Sub lcFiles_MouseUp(ByVal sender As Object, ByVal e As MouseEventArgs) Handles
lcFiles.MouseUp
_currentItem = Nothing
End Sub
Private Sub lcFiles_DoubleClick(ByVal sender As Object, ByVal e As EventArgs) Handles
lcFiles.DoubleClick
Cursor.Current = Cursors.WaitCursor
If _currentItem IsNot Nothing AndAlso _currentItem.Enabled Then
Dim info As FileSystemInfo = TryCast(_currentItem.Value, FileSystemInfo)
If TypeOf info Is DirectoryInfo Then
AddDirectoryToDropDownList(TryCast(info, DirectoryInfo), ddlDrives)
Else
System.Diagnostics.Process.Start(info.FullName)
End If
End If
Cursor.Current = Cursors.[Default]
End Sub
```

[C#] Handling RadListControl Events

```
private void lcFiles_MouseDown(object sender, MouseEventArgs e)
{
RadListVisualItem item = lcFiles.ElementTree.GetElementAtPoint(e.Location) as
RadListVisualItem;
if (item != null)
{
```

```
_currentItem = item.Data;
}
}
private void lcFiles_MouseUp(object sender, MouseEventArgs e)
{
_currentItem = null;
}
private void lcFiles_DoubleClick(object sender, EventArgs e)
{
Cursor.Current = Cursors.WaitCursor;
if (_currentItem != null && _currentItem.Enabled)
{
FileSystemInfo info = _currentItem.Value as FileSystemInfo;
if (info is DirectoryInfo)
AddDirectoryToDropDownList(info as DirectoryInfo, ddlDrives);
else
System.Diagnostics.Process.Start(info.FullName);
}
Cursor.Current = Cursors.Default;
}
```

- In the Properties window Events tab, hook up "lcFiles" with the new MouseDown, MouseUp and DoubleClick event handlers.
- Handle the "Back" button click event.

  The logic here gets the parent of the current directory and calls AddDirectoryToDropDownList. This has the effect of adding that directory to the dropdown list, making that directory the current item in the dropdown list and loading all the files for that directory into the "lcFiles" list control.

  [VB] Handling the Click Event

```
Private Sub btnBack_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnBack.Click
Cursor.Current = Cursors.WaitCursor
Dim directoryInfo As DirectoryInfo = TryCast(ddlDrives.SelectedValue, DirectoryInfo)
AddDirectoryToDropDownList(GetParentDirectory(directoryInfo), ddlDrives)
Cursor.Current = Cursors.[Default]
End Sub
End Class
```

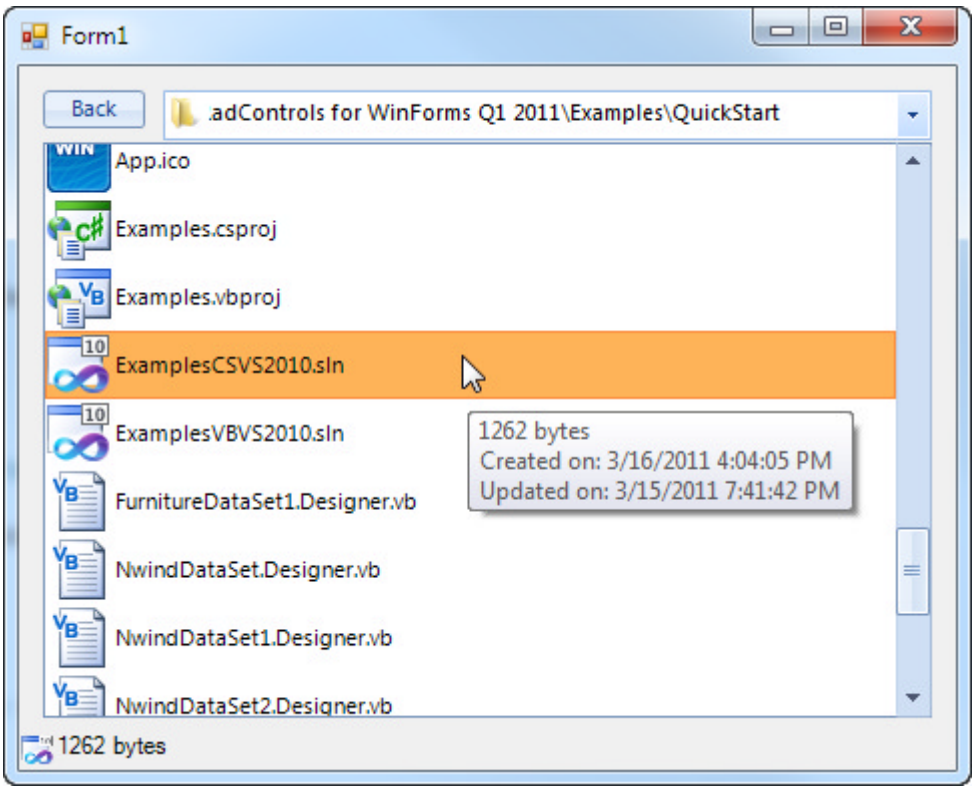
  [C#] Handling the Click Event

```
private void btnBack_Click(object sender, EventArgs e)
{
Cursor.Current = Cursors.WaitCursor;
DirectoryInfo directoryInfo = ddlDrives.SelectedValue as DirectoryInfo;
AddDirectoryToDropDownList(GetParentDirectory(directoryInfo), ddlDrives);
Cursor.Current = Cursors.Default;
}
```

- Press **Ctrl-F5** to run the application again. You should be able to navigate around the directory structure. The "Back" button should be enabled only when you're not at the root level and should move you back one directory level. Double-clicking a folder should drill down into that folder, add it to the RadDropDownList and display all the objects in that directory.

> Again, be careful of what you click on. System.Diagnostics.Process.Start() will run whatever executable you double-click. This is also true of .bat files or any other executable. Just as with the Windows File Explorer, double-clicking files will run whatever program is associated with that file

type.



9.6 Summary

In this chapter you explored how to work with RadListControl and RadDropDownList, to add and remove items from their collections, to customize the aitems appearance and to use the different functionalities that these controls offer. Also, you have learnt how to make a drop down functionality between list controls.

10 ListView

10.1 Objectives

- Get familiar with the RadListView control.
- Learn how to add items, columns and groups at design-time and using code.
- Learn how to group, filter and sort data based on user input.
- Display data in different views.
- Customize the view appearance.

10.2 Introduction

Telerik RadListView control is created as a result of the concord of the powerful data layer used by **RadGridView (Section 21.2)** and **RadListControl (Section 9.2)** , together with the outstanding **Telerik Presentation Framework (Section 7.2)** . The data layer provides very high performance when working with data and also different types of binding options. Additionally, it provides features like grouping, sorting and filtering. Thanks to the Telerik Presentation Framework the control customization is very flexible and intuitive.

RadListView is a bindable control for representing and editing list data with lots of customization abilities. First of all, RadListView will offer three different types of views, which will let you visualize your data the way that you want. You may choose between **SimpleListView**, **IconView** or **DetailView**.

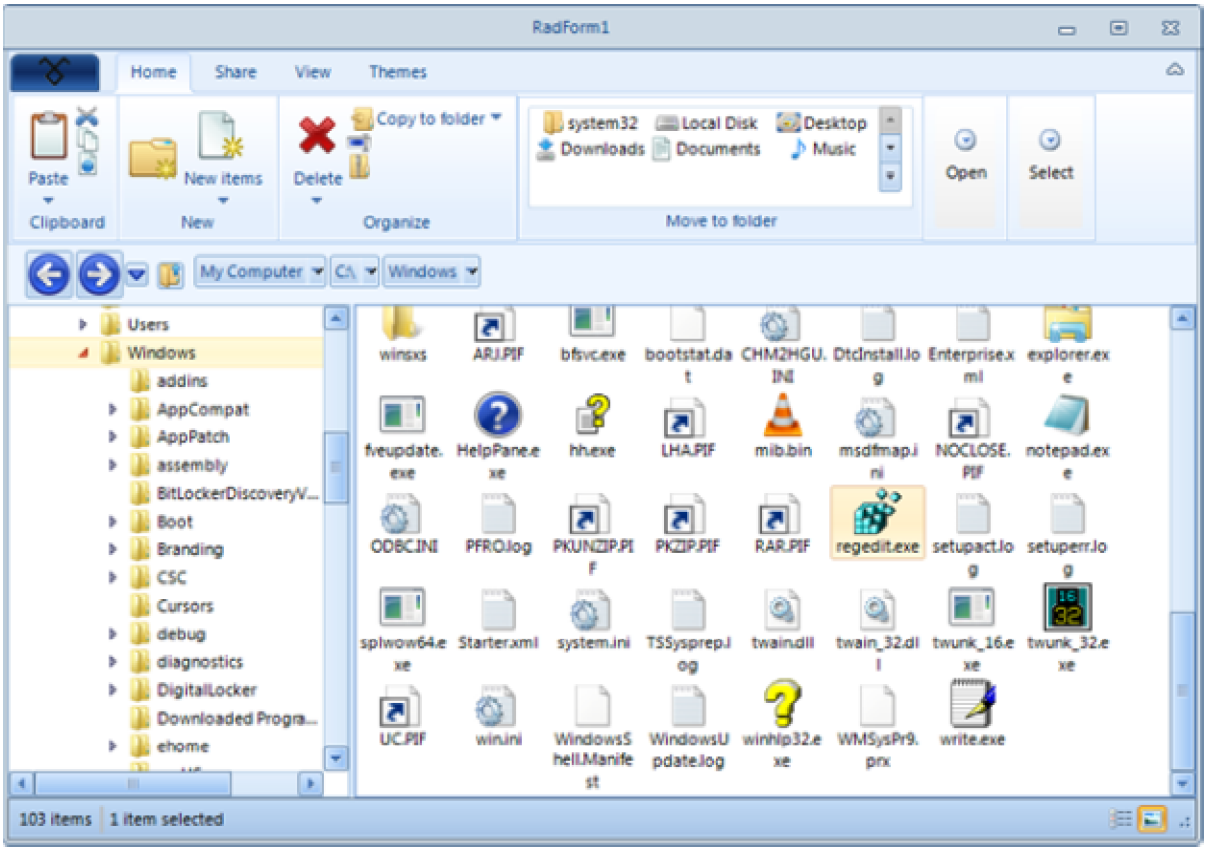
RadListView will also provide a rich and flexible API, which will let the developers easily customize the layout or the behavior of the control. Furthermore, the API will be similar to the API of our existing controls, so it will be easier for those who have already used our products to use the new component. The known family of items' creating/formatting events will let you replace or style the items according to your preferences. Check boxes, resizing and reordering of columns, hot tracking, multiple selection and different sizing modes are another small part of the possibilities provided by the API.

At design time, you will be able to easily choose a data source or populate it with unbound items, groups and columns.

Last, but not least, RadListView will support kinetic scrolling mechanism which is a great advantage when used for applications deployed on Kiosks.

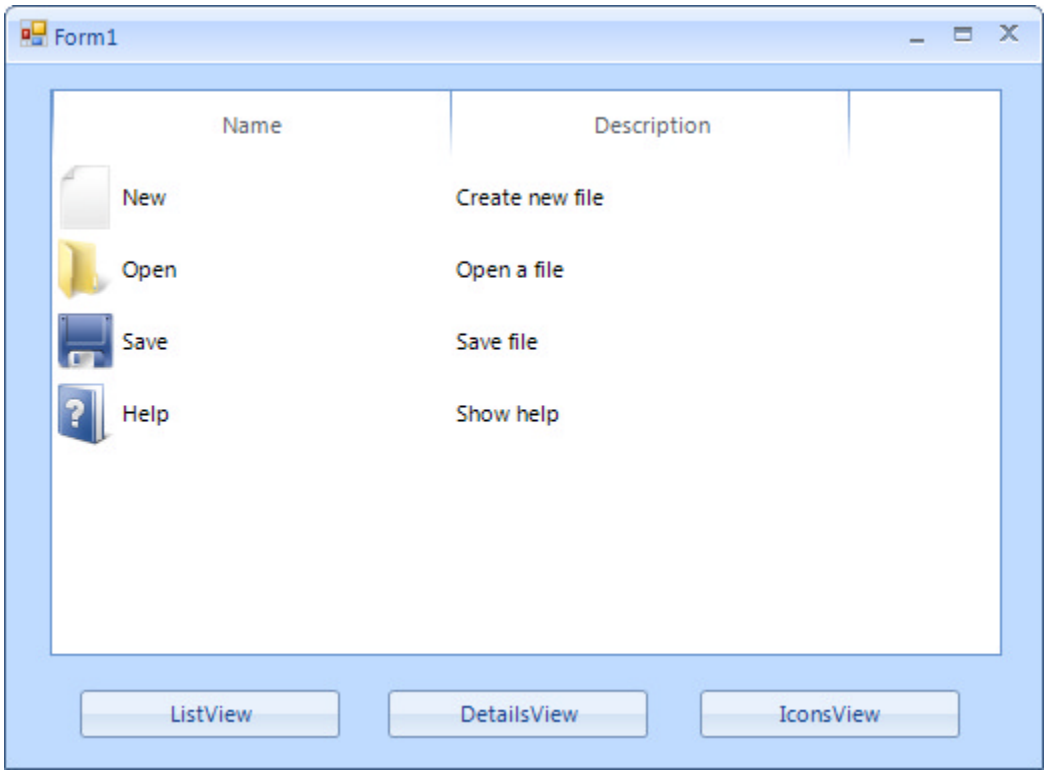
Follows a list of the features that RadListView supports:

- DataBinding
- ViewTypes – ListView, DetailsView and IconsView
- Columns
- Sorting
- Grouping
- Filtering
- Kinetic scrolling
- CheckBoxes
- Multi selection
- Hot tracking



10.3 GettingStarted

This article, will show you how in a few clicks in design time you can setup your RadListView control.

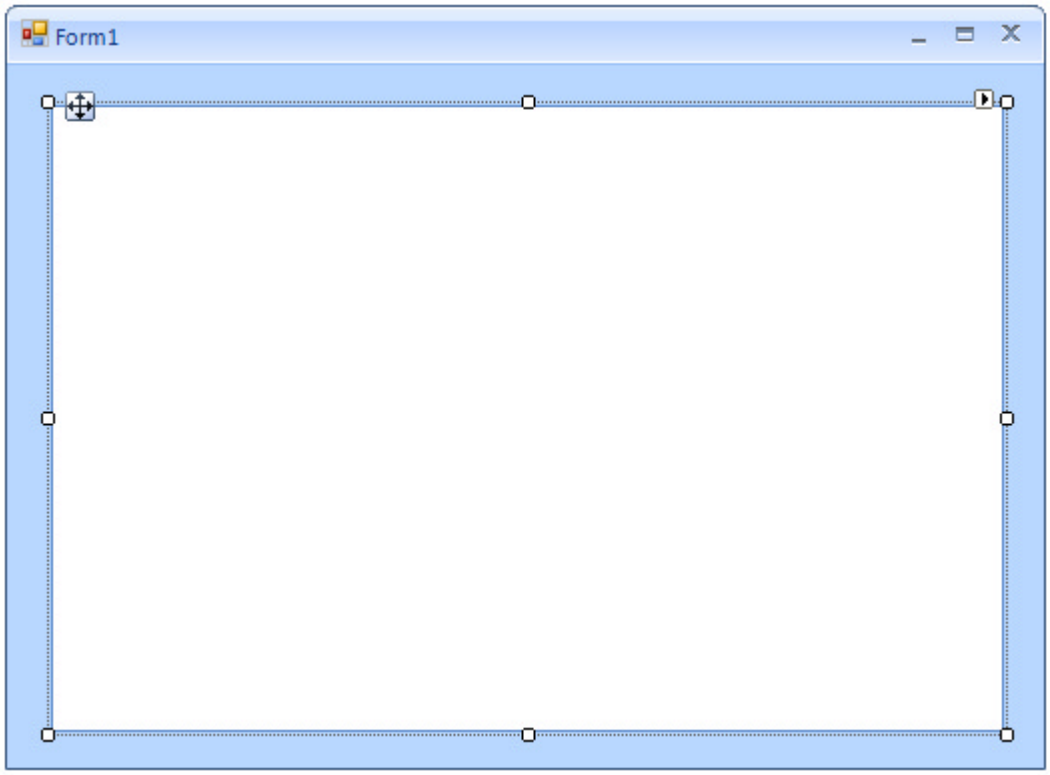


Preparing the Project

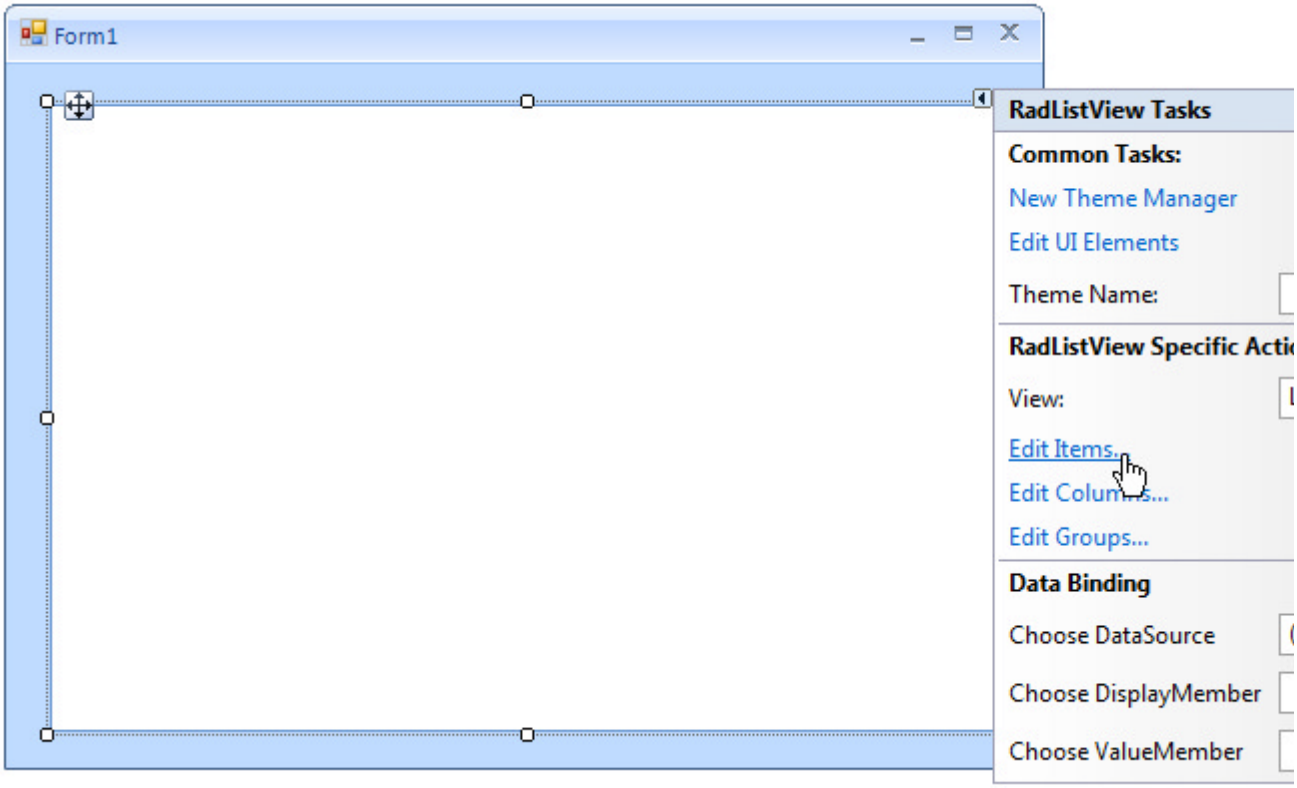
You can find the complete source for this project at:

\ListView\<VB|CS>\GettingStarted

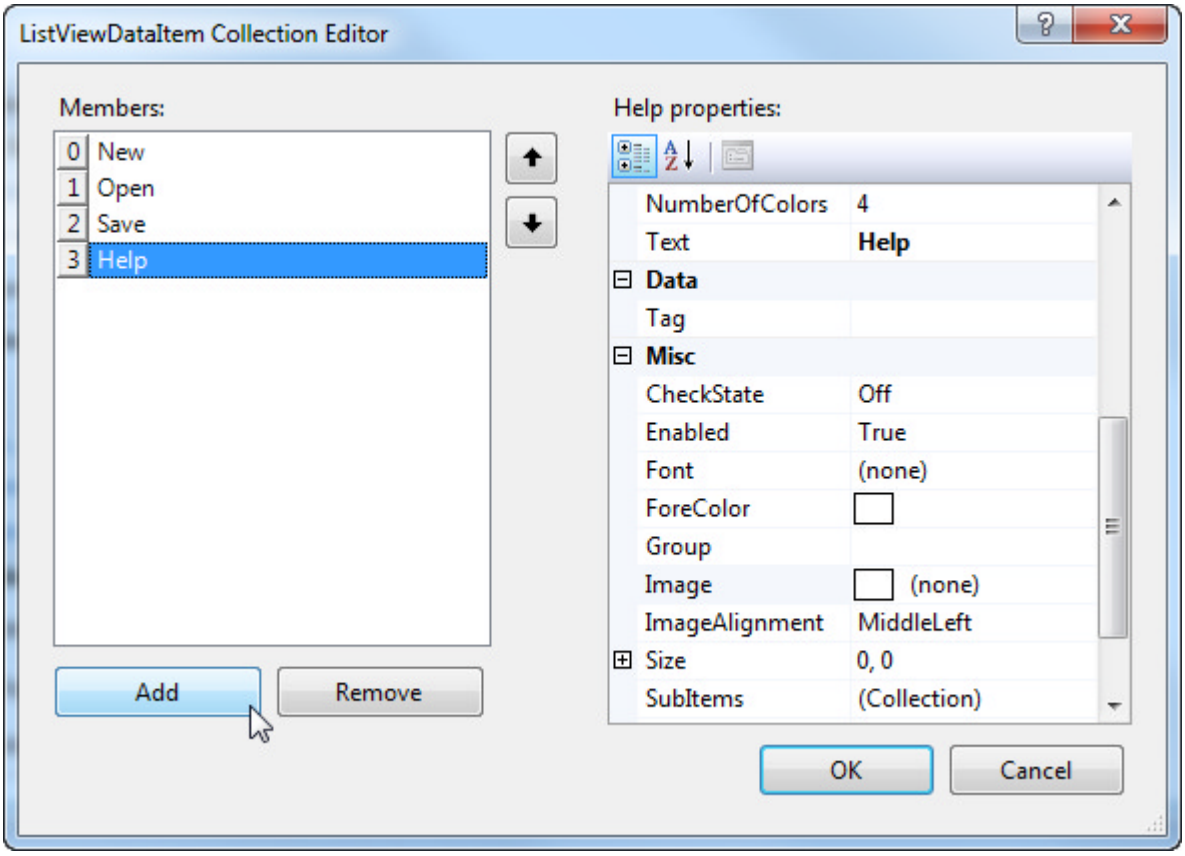
1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. Change the new RadForm1 to be the startup form.
6. Drag RadListView control from the toolbox to the form.



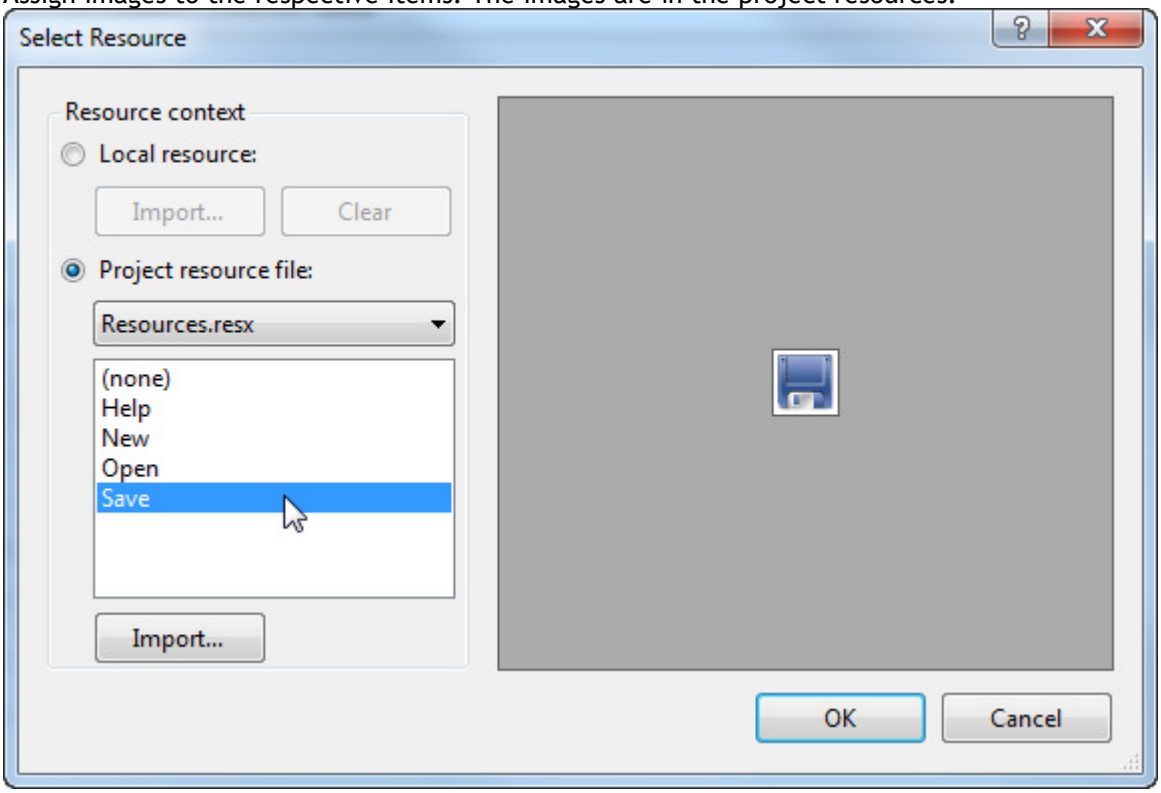
7. From the **Smart Tag**, select the *Add Items*.. option and add a few items to the control.



8. Change the items text to "*New*", "*Open*", "*Save*" and "*Help*".

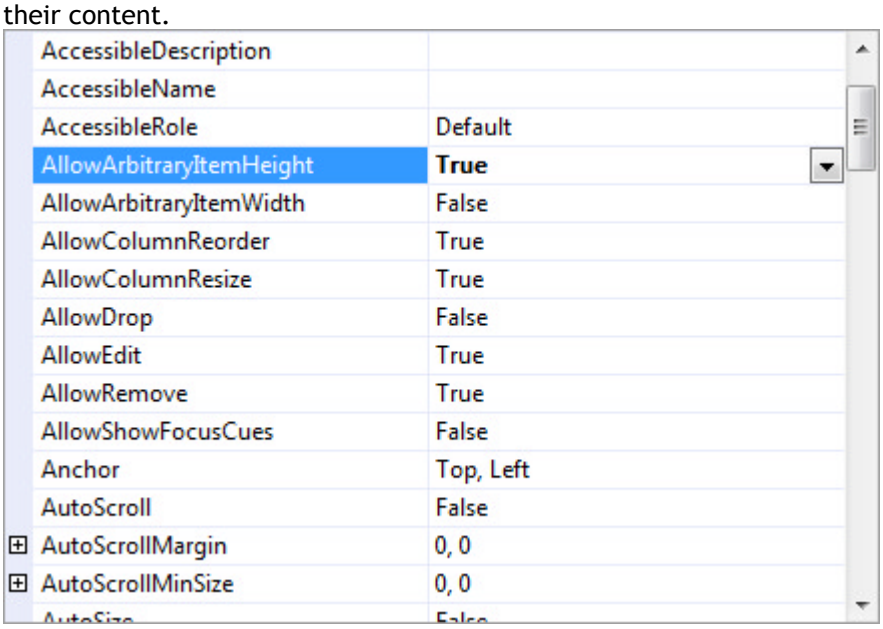


9. Assign images to the respective items. The images are in the project resources.



10. Set the **AllowArbitraryItemHeigh** to *true*, in order to allow items to size themselves in height according to

their content.



11. Add three RadButtons below the form with the following parameters:
    - Name: radButton1 = *listViewButton*; radButton2 = *detailsViewButton*; radButton3 = *iconsViewButton*
    - Text: listViewButton.Text = "*ListView*"; detailsViewButton.Text = "*DetailsView*"; iconsViewButton.Text = "*IconsView*"

12. Subscribe for these buttons click events and set the respective **ViewType** of RadListView in each event handler. Additionally, set the **AllowArbitratyItemHeight** to *true* when changing to **DetailsView** or **ListView** and **AllowArbitratyItemWidth** to *true* for **IconsView**:

[C#] Change ViewType of RadListView

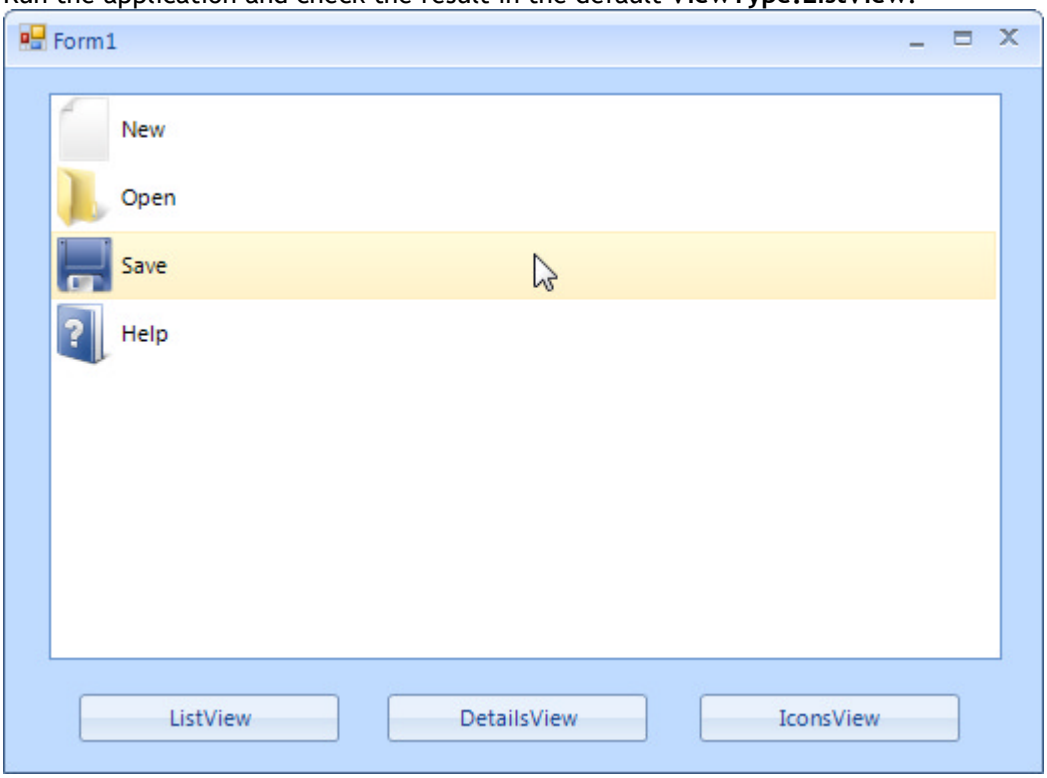
```
private void listViewButton_Click(object sender, EventArgs e)
{
radListView1.ViewType = ListViewType.ListView;
radListView1.AllowArbitraryItemHeight = true;
}
private void detailsViewButton_Click(object sender, EventArgs e)
{
radListView1.ViewType = ListViewType.DetailsView;
radListView1.AllowArbitraryItemHeight = true;
}
private void iconsViewButton_Click(object sender, EventArgs e)
{
radListView1.ViewType = ListViewType.IconsView;
radListView1.AllowArbitraryItemWidth = true;
}
```

[VB.NET] Change ViewType of RadListView

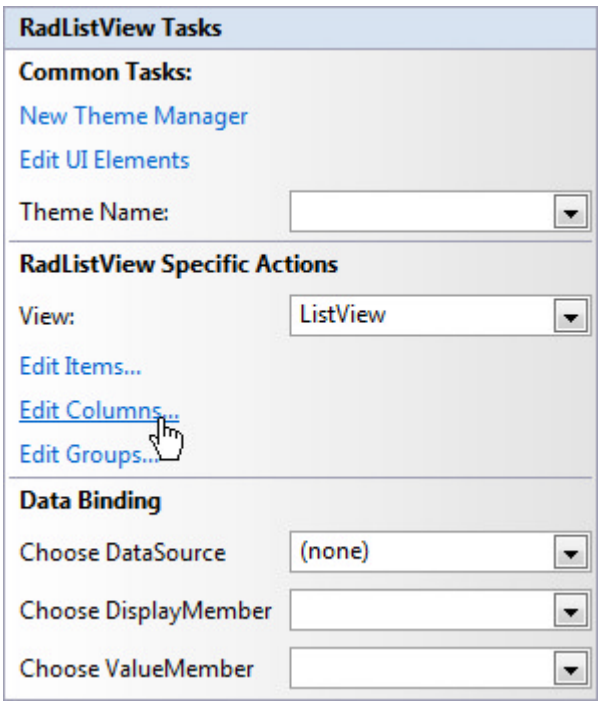
```
Private Sub listViewButton_Click(ByVal sender As System.Object, ByVal e As System.EventAı
Handles listViewButton.Click
RadListView1.ViewType = ListViewType.ListView
RadListView1.AllowArbitraryItemHeight = True
End Sub
```

```
Private Sub detailsViewButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles detailsViewButton.Click
RadListView1.ViewType = ListViewType.DetailsView
RadListView1.AllowArbitraryItemHeight = True
End Sub
Private Sub iconsViewButton_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles iconsViewButton.Click
RadListView1.ViewType = ListViewType.IconsView
RadListView1.AllowArbitraryItemWidth = True
End Sub
```

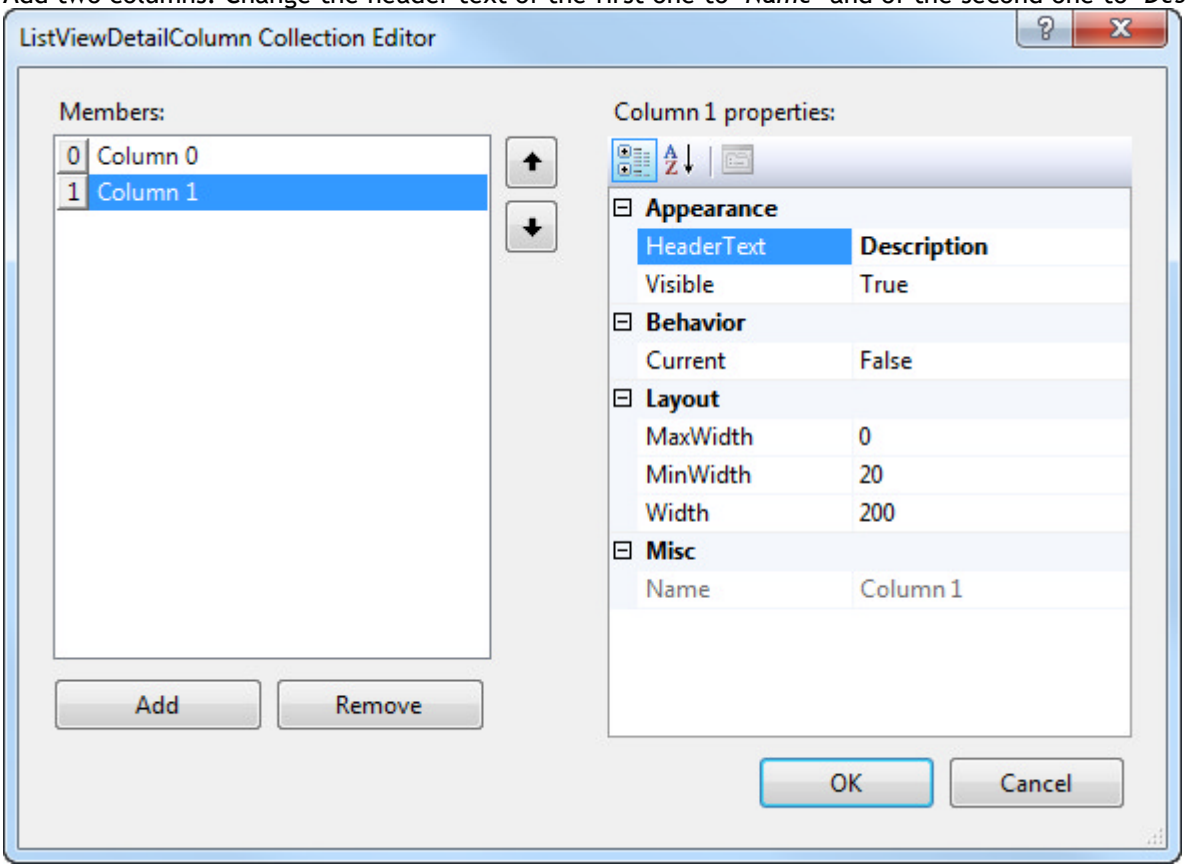
13. Run the application and check the result in the default **ViewType.ListView**.



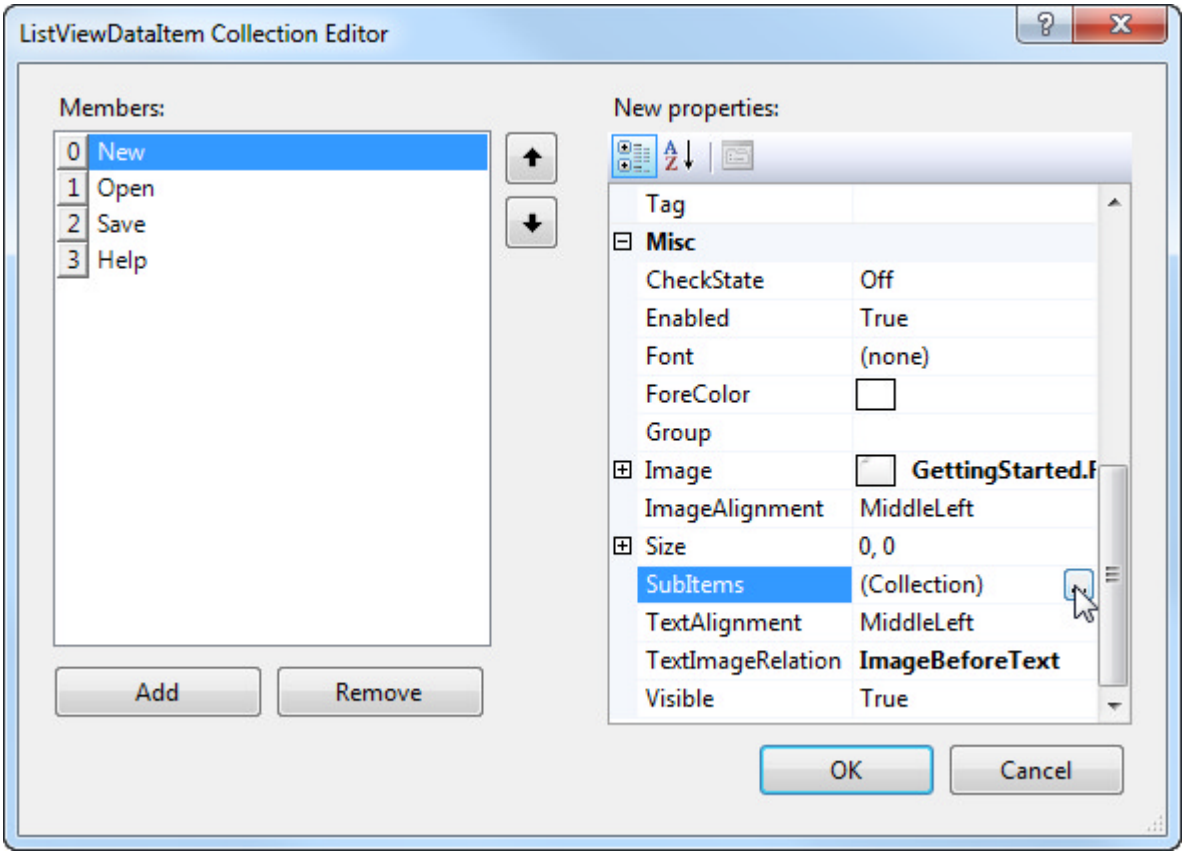
14. In order to setup the **DetailsView**, open the **Smart Tag** of RadListView in design time and select *Add Columns*..



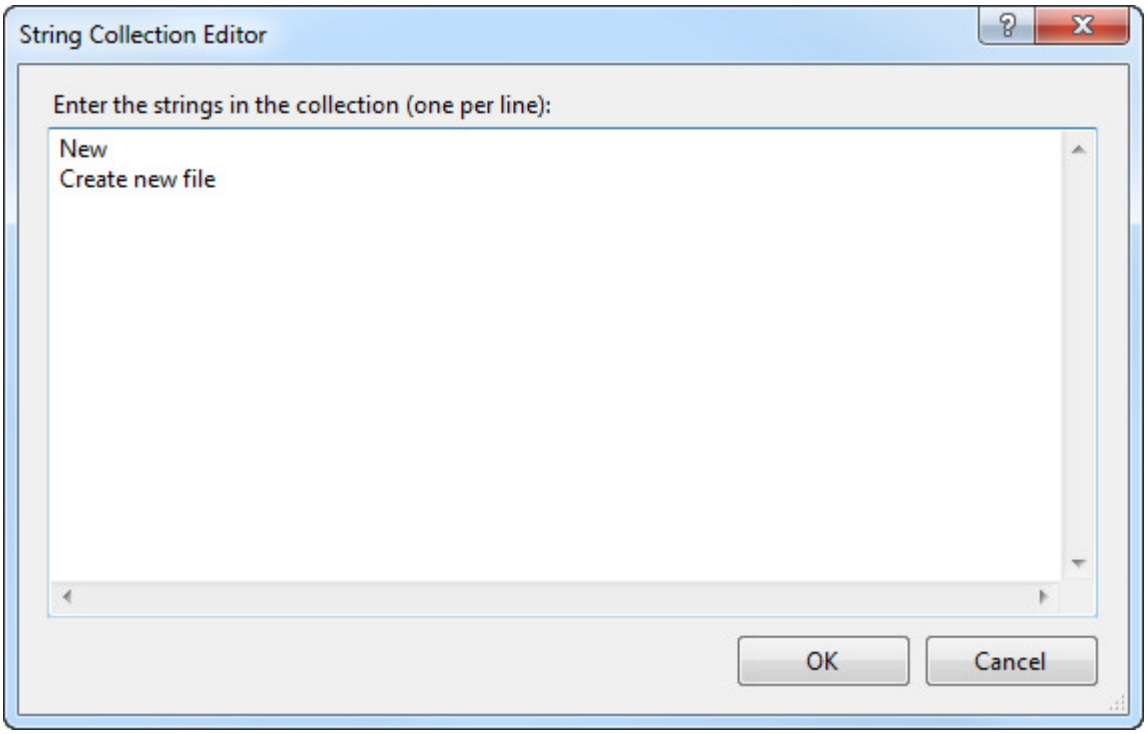
15. Add two columns. Change the header text of the first one to "*Name*" and of the second one to "*Description*"



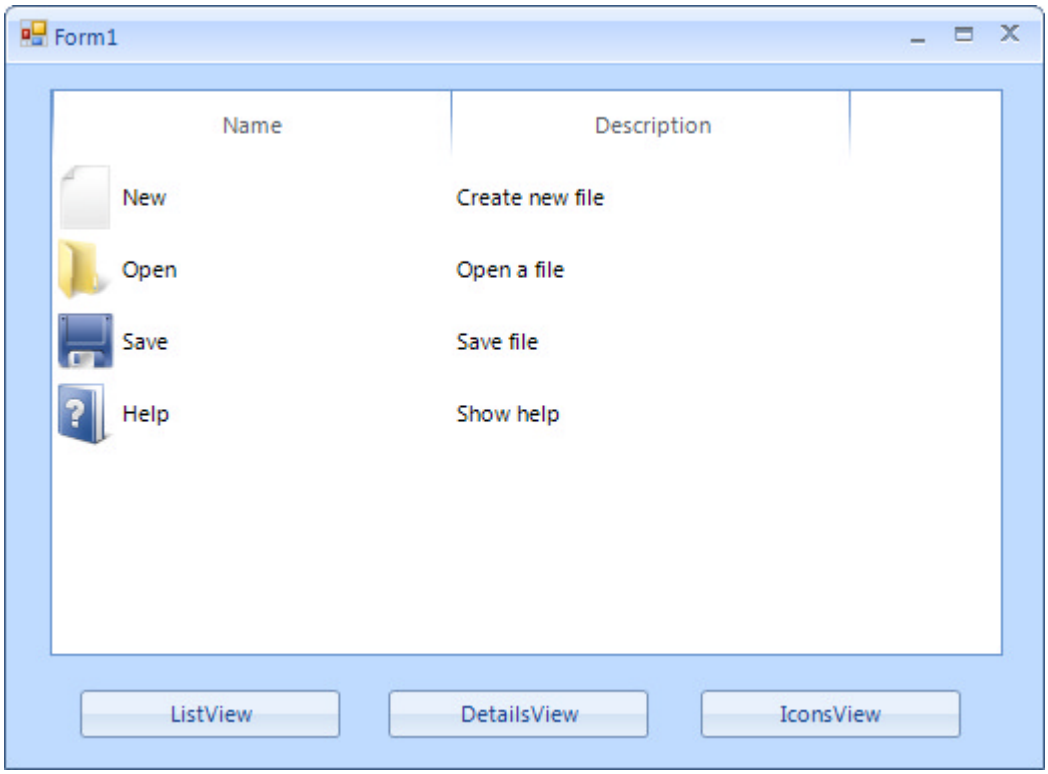
16. In order to assign the item values for a specific column in design time, you can use the SubItems property of RadListDataItem. Open the items collection, and click the SubItems property:



17. Assign some name and description in the StringCollectionEditor. First line adds value for the first column, second line for the second column etc.



18. Thats all. Run the application and review the three views offered by RadListView.

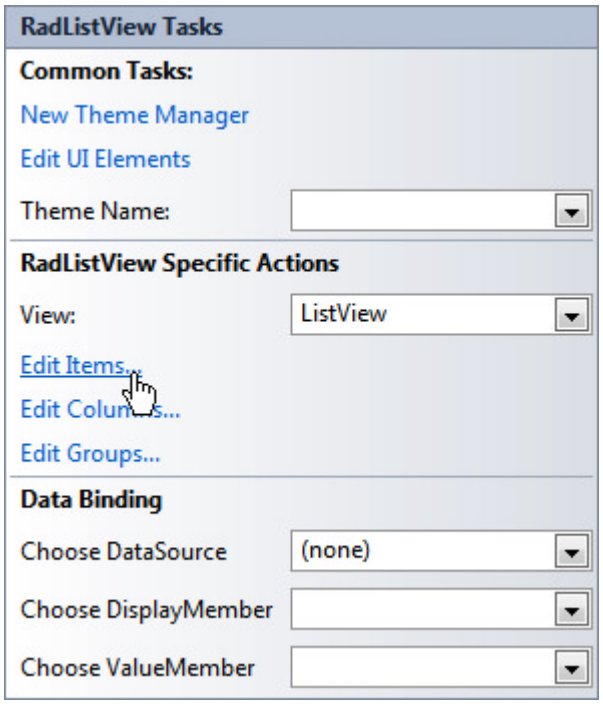


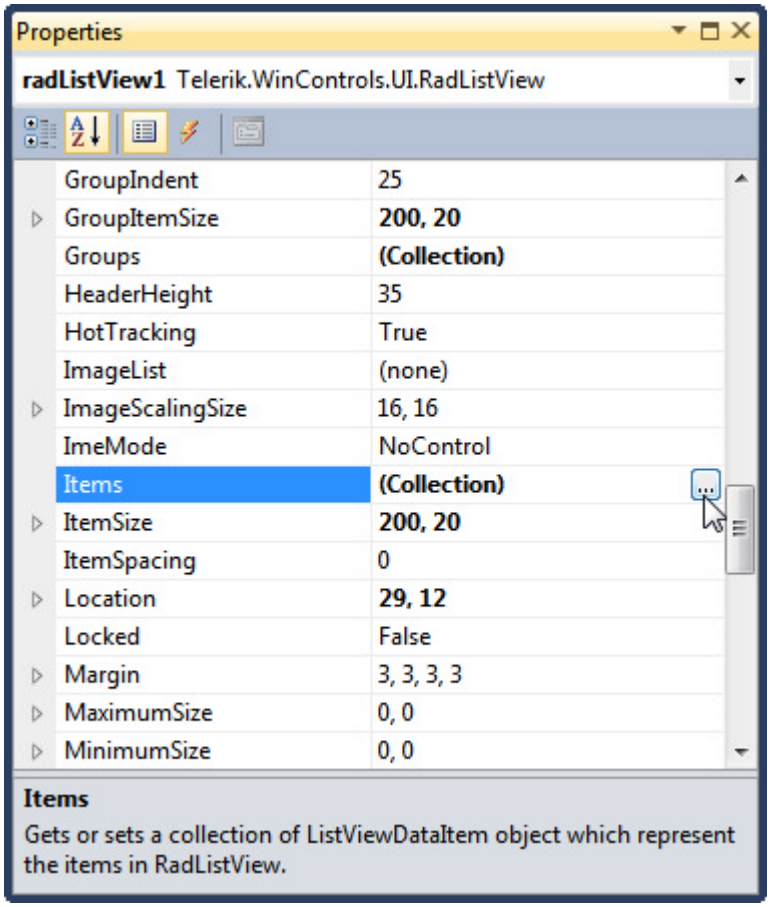
10.4 Using the Design Time Interface

The design time support of RadListView provides you with the ability to add items, columns, groups and also to customize the control properties.

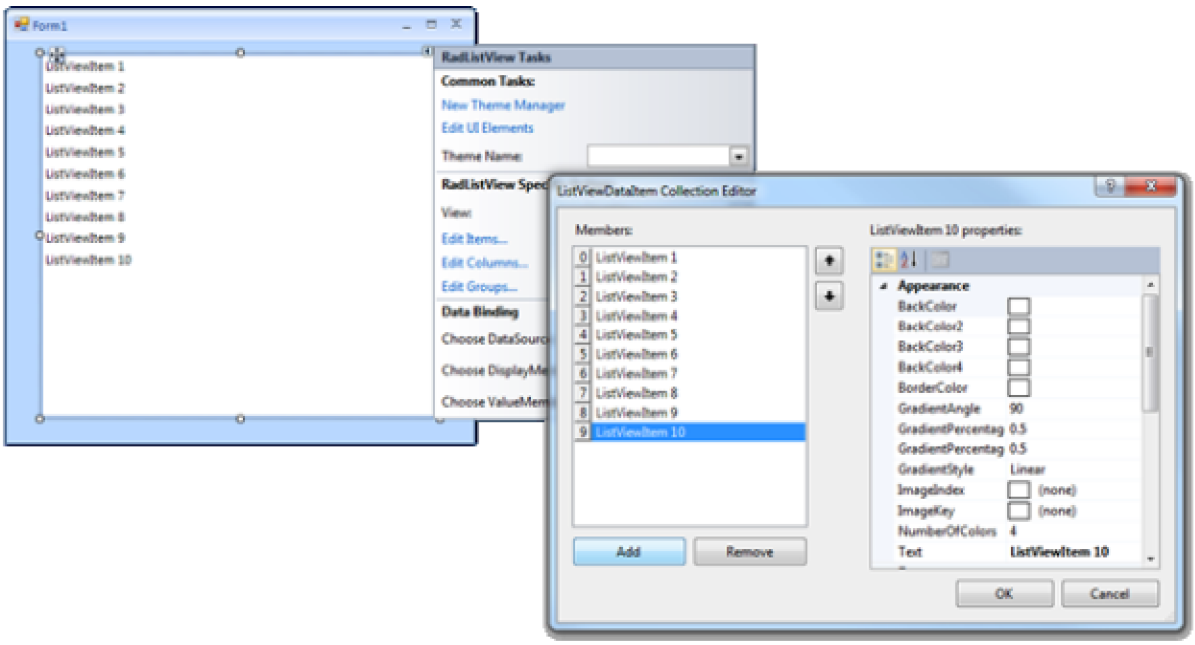
Add Items

Adding items to RadListView at design time is possible through the **Items** collection of the control. This collection is accessible by using the **Smart Tag** or the **Items** collection property in the property grid.

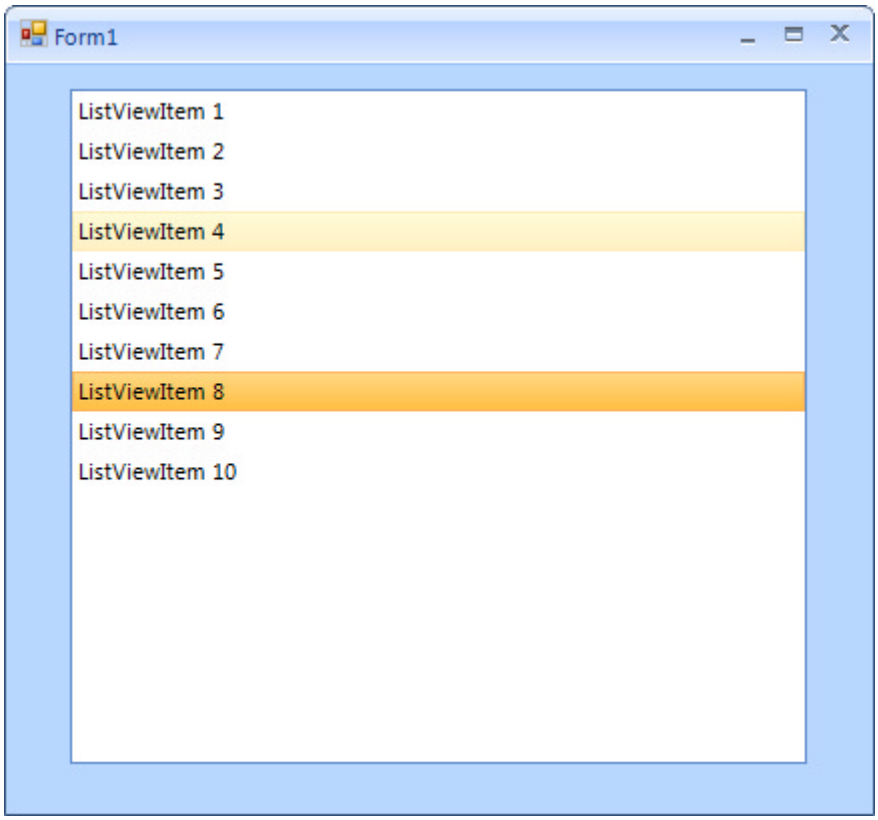




Both of these will open **ListViewDataItemCollectionEditor**, from where, by clicking the **Add** button you can add items to the collection. Additionally, on the right hand side,. you will see a property grid, where you can change the properties of each item individually.

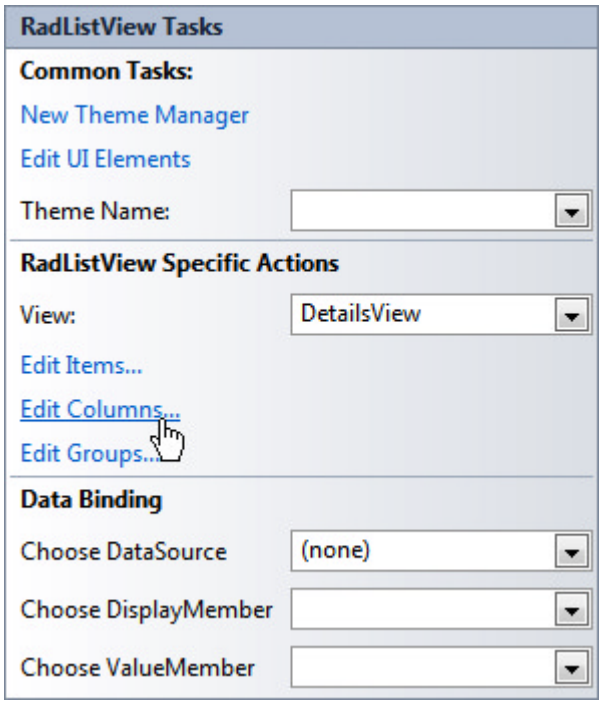


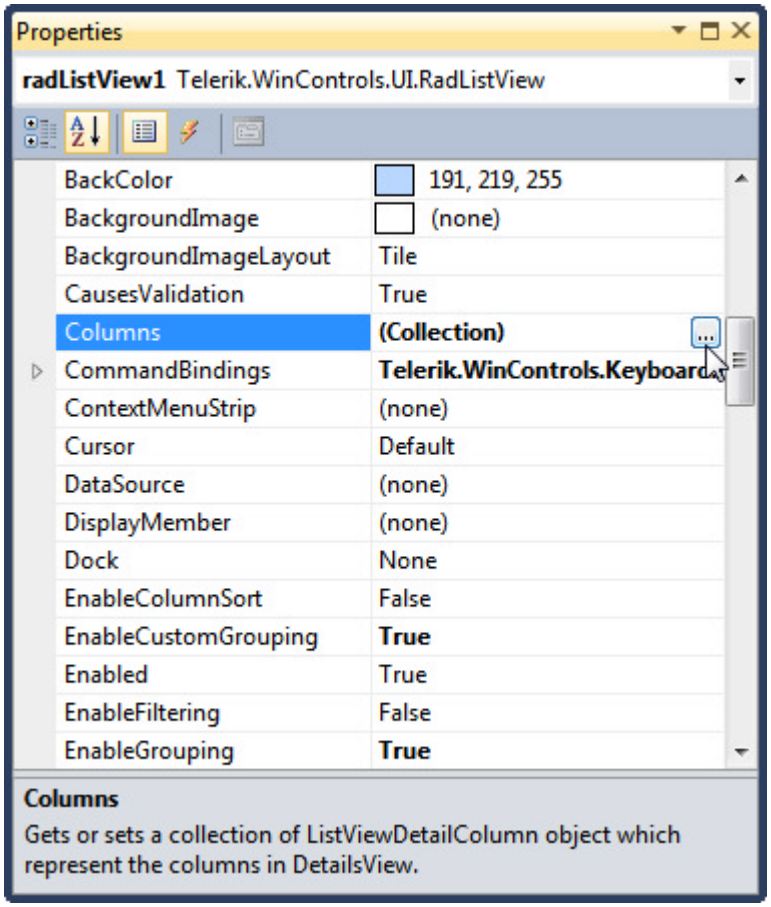
Here is the result of the described approach:



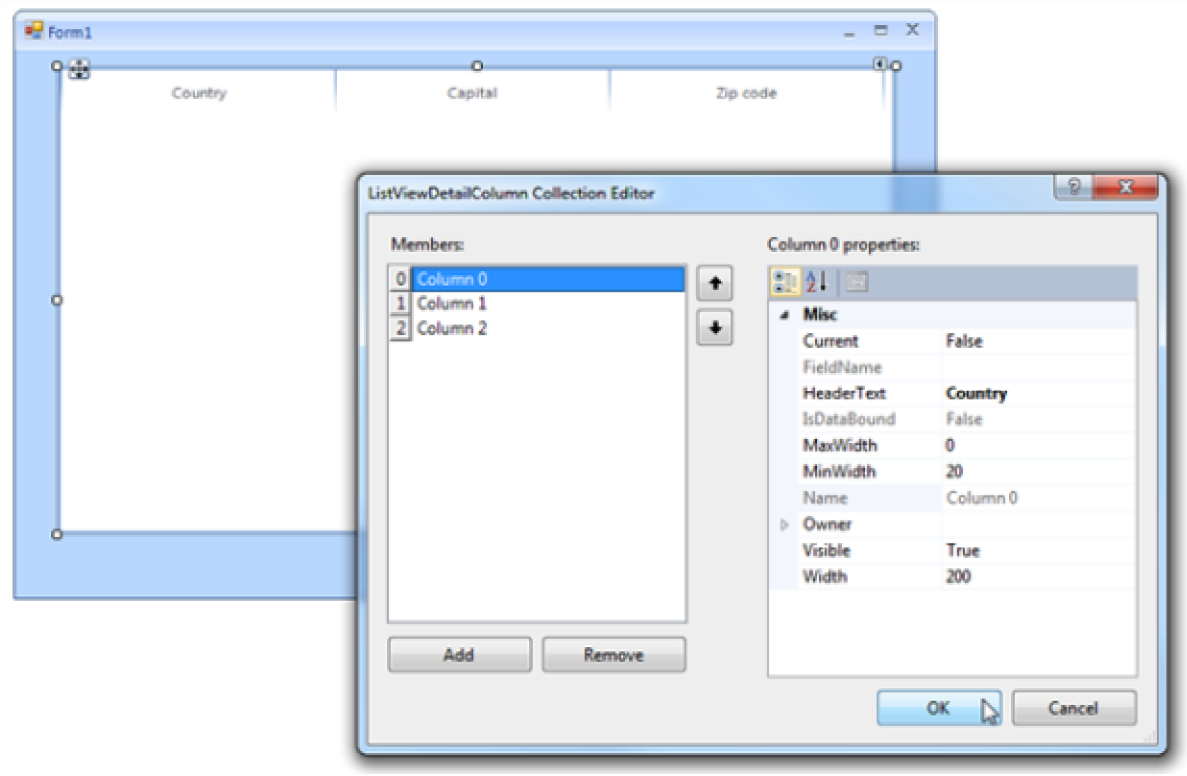
Add Columns

When RadListView **ViewType** is set to **DetailsView**, the data can be displayed in columns view. Adding columns to the control at design time, is done by populating the **Columns** collection of RadListView. This collection can be accessed either through the Columns property in the property grid of the control or through the **Smart Tag** of RadListView.

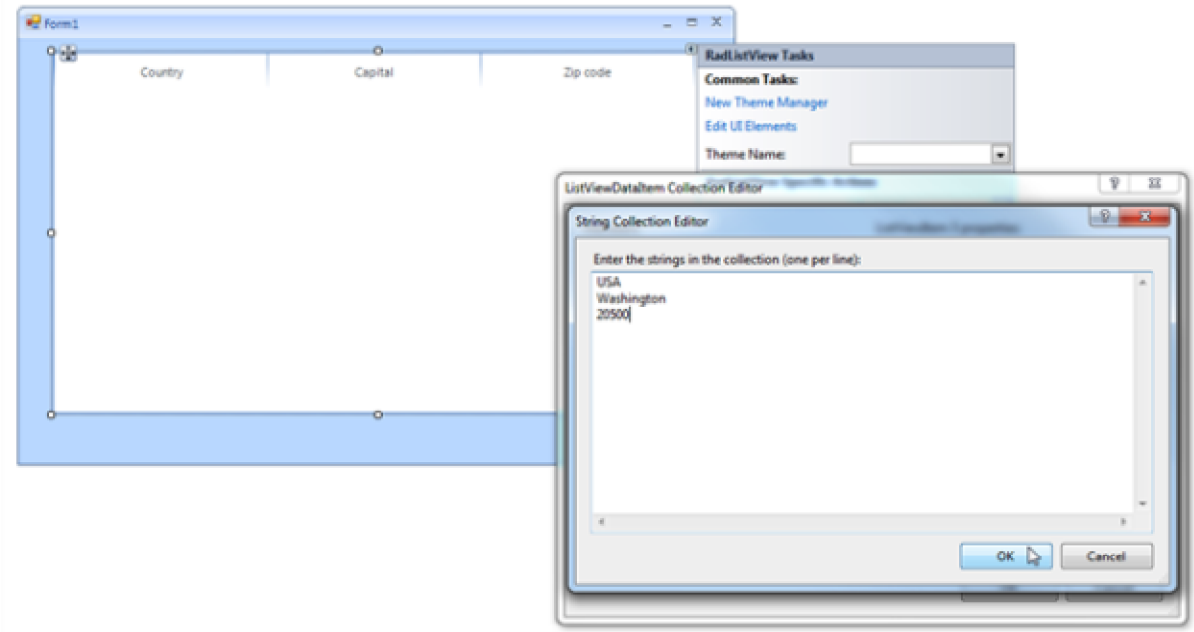




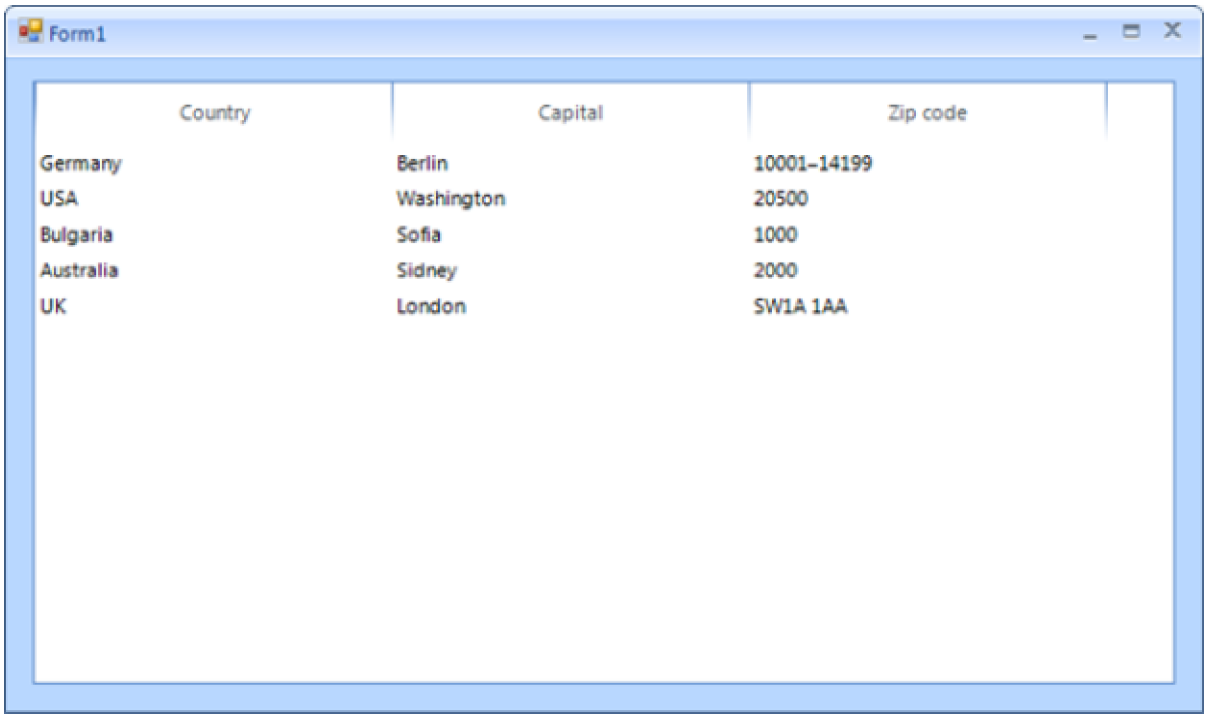
Once **ListViewDetailColumn** collection editor is open, you can add as many columns as you need, and also, you can modify their properties in the property grid.



Once the **Columns** collection is populated, you can assign values for each column of each ListViewDataItem, by making use of the **SubItems** property (this property is available in design-time only):

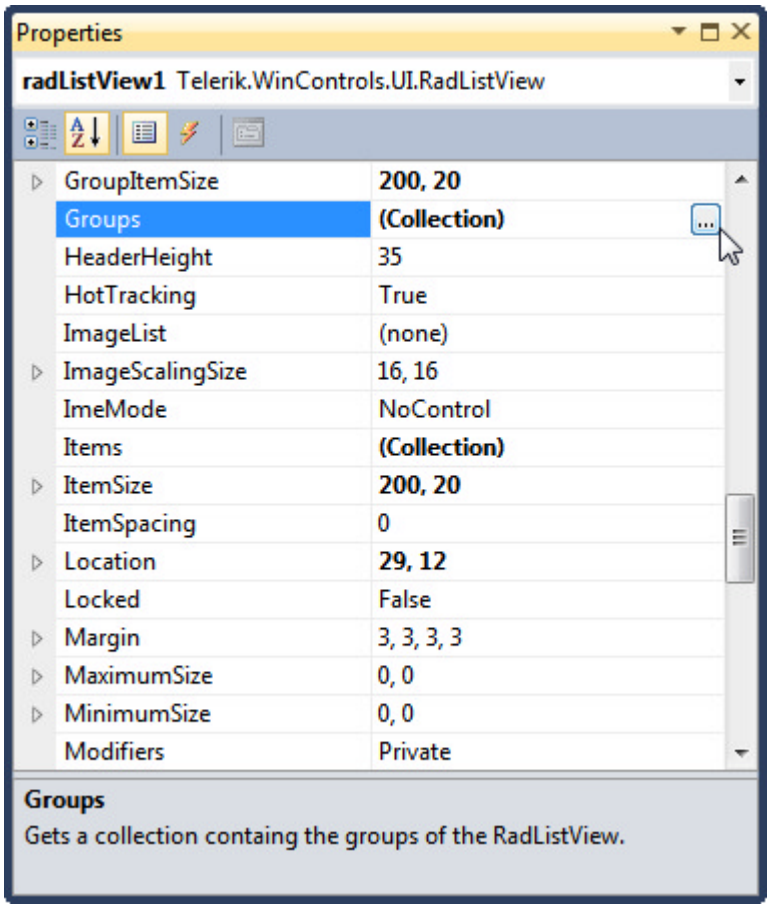


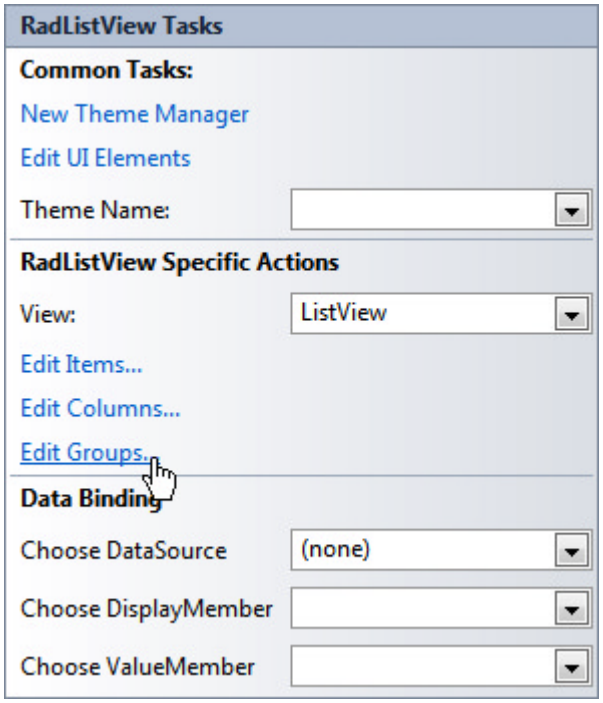
Here is the result of the explained operations:



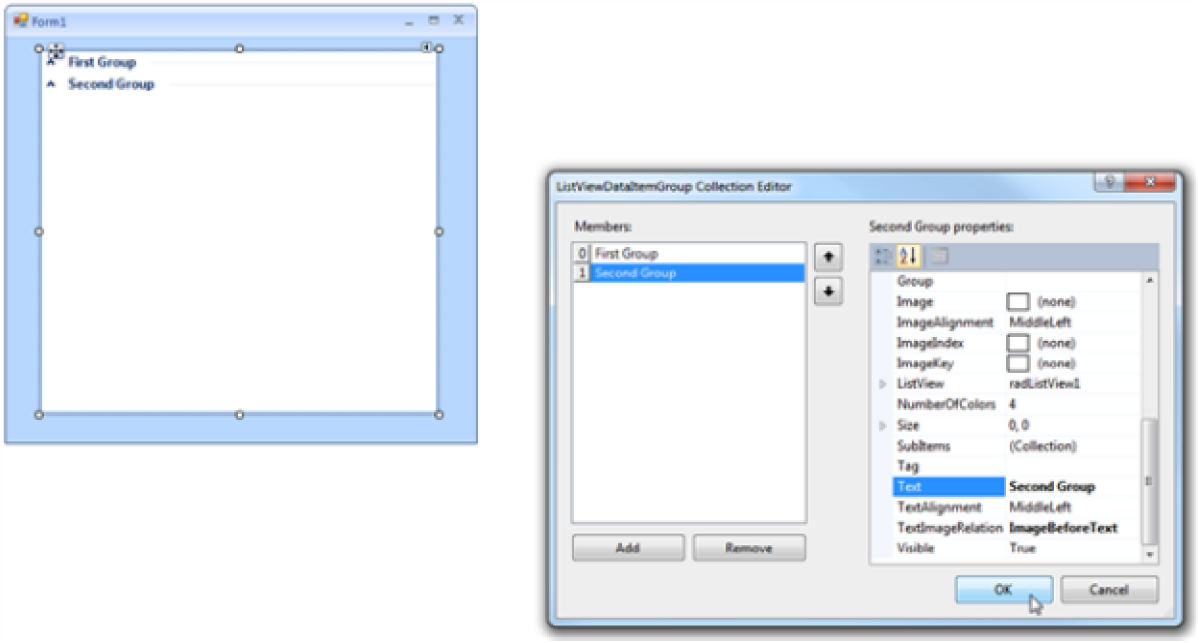
Add Groups

Items in RadListView can be grouped conveniently into different groups, which will enhance your end-user experience when working with RadListView. Grouping the items at design time can be achieved by setting the **EnableCustomGroups** and **ShowGroups** properties of the control to true and, adding the desired groups into the Groups collection of RadListControl. This collection is accessible either through the **Smart Tag** of the control or through the **Groups** collection property located in the property grid.

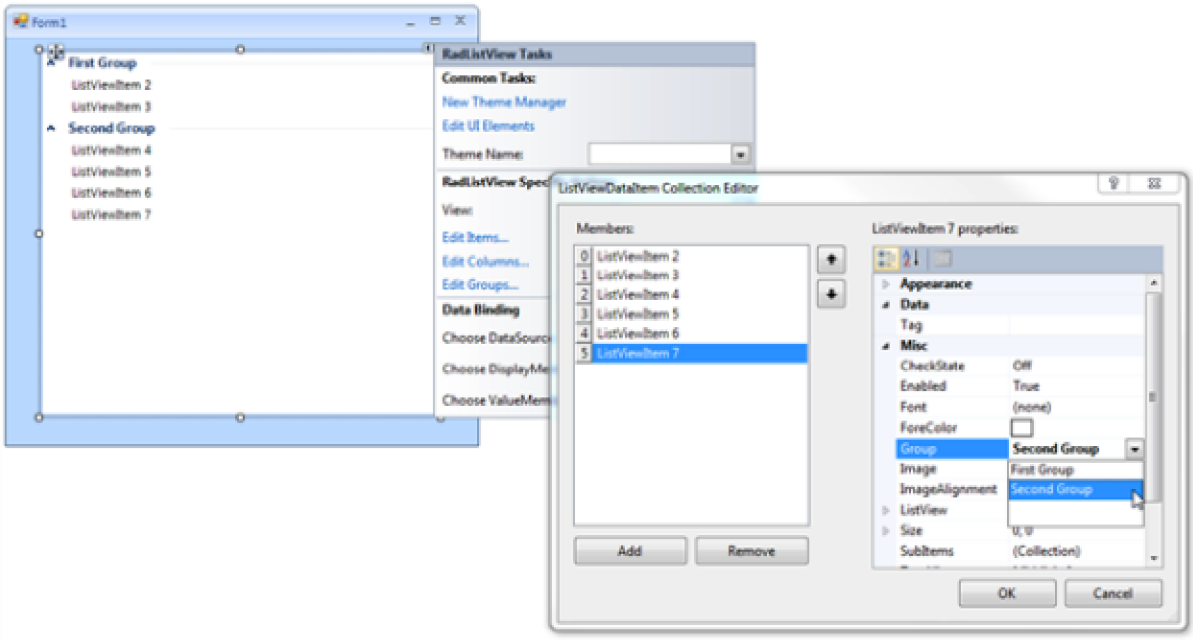




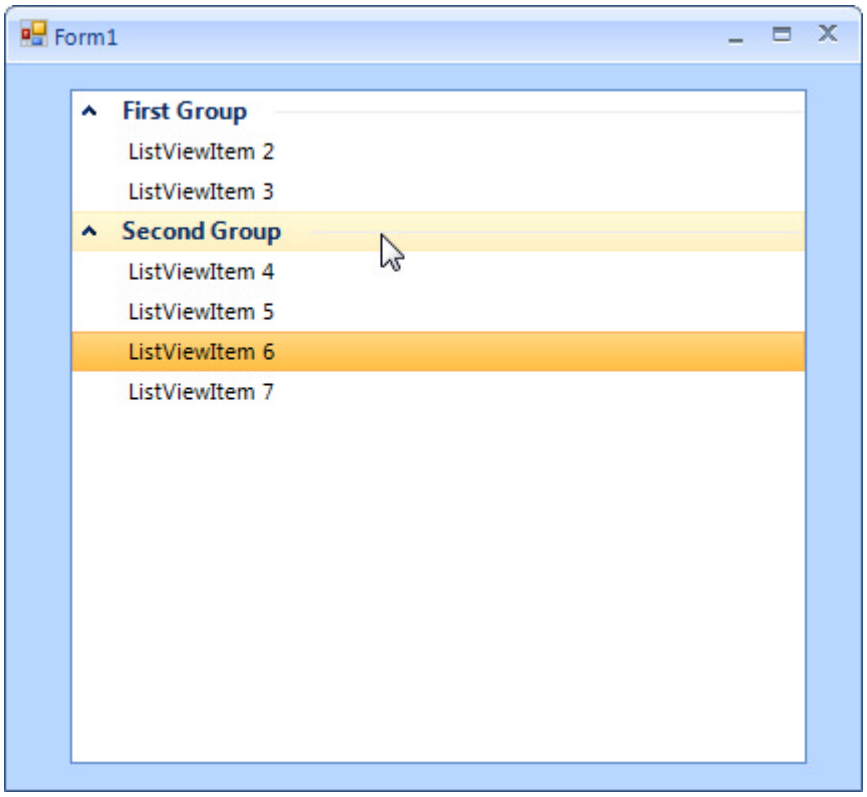
Once the **ListViewDataItemGroup** collection is opened, you can add the desired groups by using the **Add** button on the bottom, and also, you can modify the group settings in the property grid.



After the desired groups are added, you can easily assign each **ListViewDataItem** to the group of your choice by setting the **ListViewDataItem.Group** property:

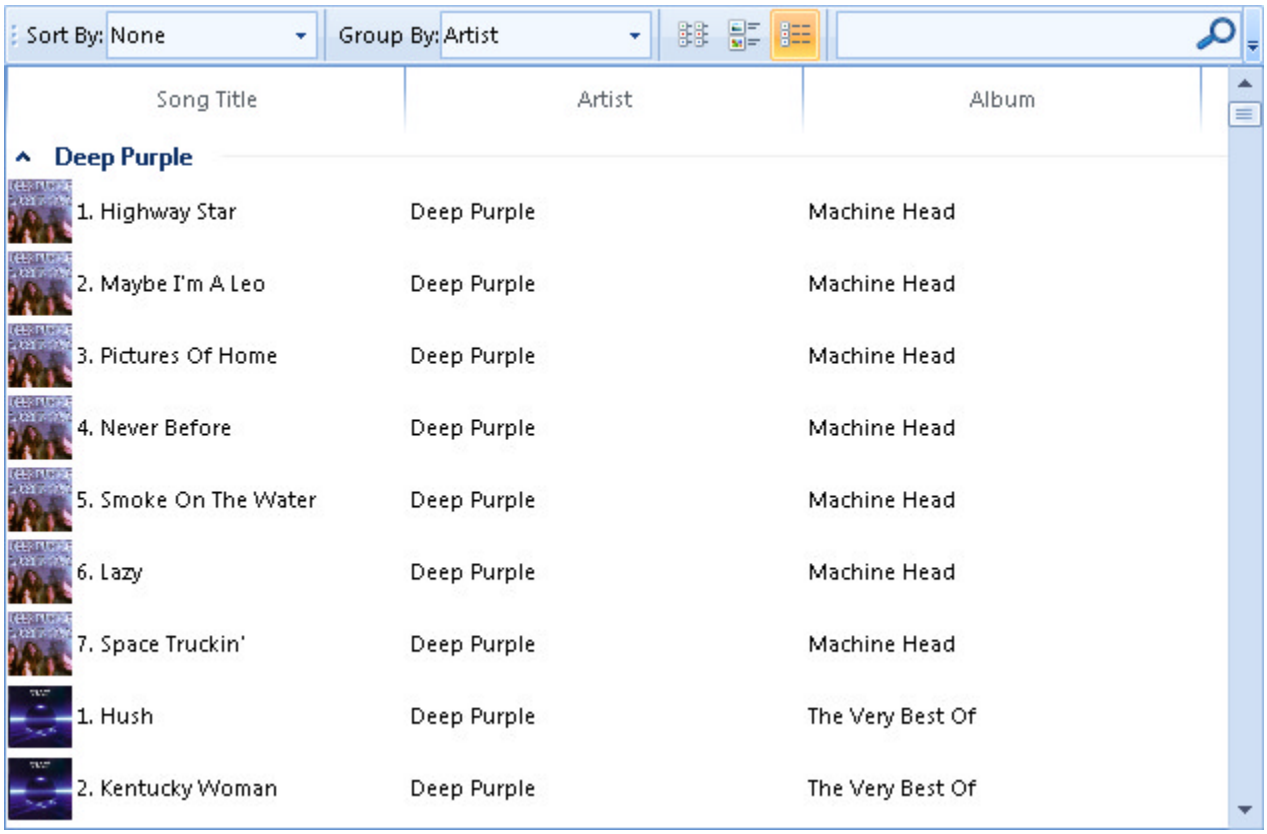


Here is how the grouped items will look like:



10.5 Programming ListView

This article will demonstrate the basic capabilities of RadListView, like filtering, grouping, sorting, customizing items, etc. and it will show you in a step-by-step manner how to start creating applications with RadListView.

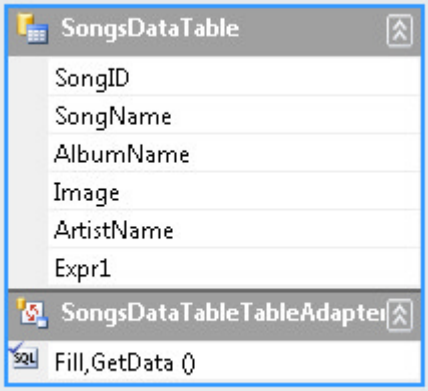




You can find the complete source for this project at:

\ListView\<VB|CS>\Programming

For the matter of this example we are going to use a DataTable containing data about Artists, Albums, Songs and Image.

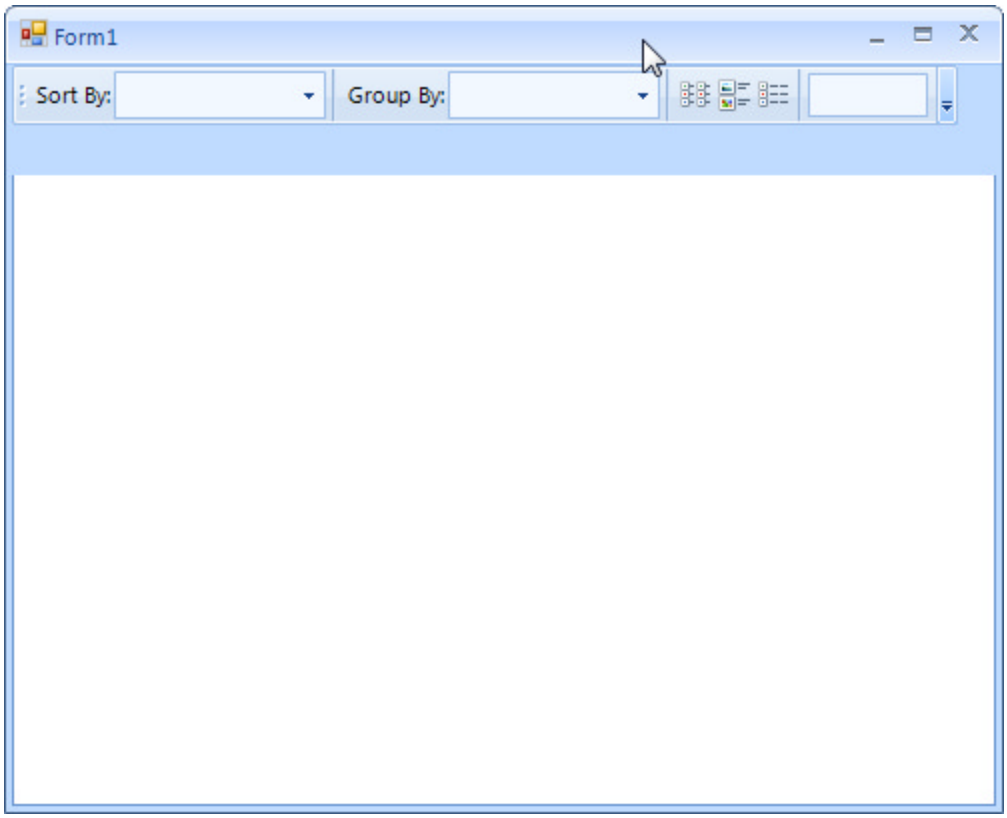


1. First let's create a form with **RadCommandBar (Section 14.2)** docked **Top** and RadListView docked **Fill**. Add a **Strip** to RadCommandBar and populate it with the following items:

CommandBarLabel	• Text: "Sort By:"

CommandBarDropDownList	• Name: commandBarDropDownSort • Text: "" • Items: None, Song Name, Artist, Album
CommandBarSeparatorItem	
CommandBarLabel	• Text: “Group By:”
CommandBarDropDownList	• Name: commandBarDropDownGroup • Text: "" • Items: None, Album, Artist
CommandBarSeparatorItem	
CommandBarToggleButton	• Name: commandBarToggleList • ToolTipText: “ListView” • Image: Some image representing ListView
CommandBarToggleButton	• Name: commandBarToggleTiles • ToolTipText: “IconsView” • Image: Some image representing IconsView
CommandBarToggleButton	• Name: commandBarToggleDetails • ToolTipText: “DetailsView” • Image: Some image representing DetailsView
CommandBarSeparatorItem	
CommandBarTextBox	• Name: commandBarTextBoxFilter • Text: “”

2. At this point the form should like something similar to this:



3. Now let continue with setting the control **DataSource**, allow edit and remove operations and subscribe to the events that we are going to use in this example.

[C#] Initial settings

```
this.radListView1.ItemDataBound += new Telerik.WinControls.UI.ListViewItemEventHandler
(radListView1_ItemDataBound);
this.radListView1.VisualItemFormatting += new
Telerik.WinControls.UI.ListViewVisualItemEventHandler(radListView1_VisualItemFormatting);
this.radListView1.CellFormatting += new
Telerik.WinControls.UI.ListViewCellFormattingEventHandler(radListView1_CellFormatting);
this.radListView1.ColumnCreating += new ListViewColumnCreatingEventHandler
(radListView1_ColumnCreating);
this.radListView1.ViewTypeChanged += new EventHandler(radListView1_ViewTypeChanged);

this.radListView1.AllowEdit = false;
this.radListView1.AllowRemove = false;
this.radListView1.DataSource = this.songsDataTableBindingSource;
this.radListView1.DisplayMember = "SongName";
this.radListView1.ValueMember = "SongID";

this.radListView1.ViewType = ListViewType.IconsView;
```

[VB.NET] Initial settings

```
AddHandler Me.RadListView1.ItemDataBound, AddressOf radListView1_ItemDataBound
AddHandler Me.RadListView1.VisualItemFormatting, AddressOf radListView1_VisualItemFormatting
AddHandler Me.RadListView1.ViewTypeChanged, AddressOf radListView1_ViewTypeChanged
```

```
AddHandler Me.RadListView1.CellFormatting, AddressOf radListView1_CellFormatting
AddHandler Me.RadListView1.ColumnCreating, AddressOf radListView1_ColumnCreating

Me.RadListView1.AllowEdit = False
Me.RadListView1.AllowRemove = False
Me.RadListView1.DataSource = Me.SongsDataTableBindingSource
Me.RadListView1.DisplayMember = "SongName"
Me.RadListView1.ValueMember = "SongID"
```

4. Lets handle those events. In the event handler for the **ItemDataBound** event, we will take the corresponding item image from the data source and we will assign it to the ListViewDataItem.

[C#] Set the item image to the data item

```
void radListView1_ItemDataBound(object sender, Telerik.WinControls.UI.ListViewItemEventArgs
e)
{
   DataRowView row = e.Item.DataBoundItem as DataRowView;
   MusicCollectionDataSet.SongsDataTableRow songRow = row.Row as
MusicCollectionDataSet.SongsDataTableRow;
   e.Item.Image = Image.FromStream(new MemoryStream(songRow.Image), false, false);
}
```

[VB.NET] Set the item image to the data item

```
Private Sub radListView1_ItemDataBound(sender As Object, e As
Telerik.WinControls.UI.ListViewItemEventArgs)
    Dim row As DataRowView = TryCast(e.Item.DataBoundItem, DataRowView)
    Dim songRow As MusicCollectionDataSet.SongsDataTableRow = TryCast(row.Row,
MusicCollectionDataSet.SongsDataTableRow)
    e.Item.Image = Image.FromStream(New MemoryStream(songRow.Image), False, False)
End Sub
```

5. Next lets hand the **VisualItemFormatting** event, where we will set the visual item image. Additionally, for IconsView we will set the visual item text to a html-like combination of the *AlbumName*, *ArtistName* and *SongName*.

[C#] Customize visual item

```
void radListView1_VisualItemFormatting(object sender,
Telerik.WinControls.UI.ListViewVisualItemEventArgs e)
{
   if (e.VisualItem.Data.Image != null)
   {
       e.VisualItem.Image = e.VisualItem.Data.Image.GetThumbnailImage(32, 32, null,
IntPtr.Zero);
       e.VisualItem.Layout.RightPart.Margin = new Padding(2, 0, 0, 0);
   }

   if (this.radListView1.ViewType == Telerik.WinControls.UI.ListViewType.IconsView &&
e.VisualItem.Data.DataBoundItem != null)
   {
       string albumName = ((MusicCollectionDataSet.SongsDataTableRow)
(((System.Data.DataRowView)(e.VisualItem.Data.DataBoundItem)).Row)).AlbumName;
       string artisName = ((MusicCollectionDataSet.SongsDataTableRow)
```

```
(((System.Data.DataRowView)(e.VisualItem.Data.DataBoundItem)).Row)).ArtistName;
       string songName = ((MusicCollectionDataSet.SongsDataTableRow)
(((System.Data.DataRowView)(e.VisualItem.Data.DataBoundItem)).Row)).SongName;
       e.VisualItem.Text = "<html> " + songName + "<br><span style=\"color:#999999\"> " +
artisName + "<br> " + albumName + "</span>";
   }
}
```

[VB.NET] Customize visual item

```
Private Sub radListView1_VisualItemFormatting(sender As Object, e As
Telerik.WinControls.UI.ListViewVisualItemEventArgs)
    If e.VisualItem.Data.Image IsNot Nothing Then
        e.VisualItem.Image = e.VisualItem.Data.Image.GetThumbnailImage(32, 32, Nothing,
IntPtr.Zero)
        e.VisualItem.Layout.RightPart.Margin = New Windows.Forms.Padding(2, 0, 0, 0)
    End If

    If Me.RadListView1.ViewType = Telerik.WinControls.UI.ListViewType.IconsView AndAlso
e.VisualItem.Data.DataBoundItem IsNot Nothing Then
        Dim albumName As String = DirectCast(DirectCast(e.VisualItem.Data.DataBoundItem,
System.Data.DataRowView).Row, MusicCollectionDataSet.SongsDataTableRow).AlbumName
        Dim artisName As String = DirectCast(DirectCast(e.VisualItem.Data.DataBoundItem,
System.Data.DataRowView).Row, MusicCollectionDataSet.SongsDataTableRow).ArtistName Dim
songName As String = DirectCast(DirectCast(e.VisualItem.Data.DataBoundItem,
System.Data.DataRowView).Row, MusicCollectionDataSet.SongsDataTableRow).SongName
        e.VisualItem.Text = "<html> " + songName + "<br><span style=""color:#999999""> " +
artisName + "<br> " + albumName + "</span>"
    End If
End Sub
```

6. The **CellFormatting** event is handled in order to customize the appearance of the cells, when RadListView is in **DetailsView**. Here we will set the cell image.

[C#] Set the cell image

```
void radListView1_CellFormatting(object sender, ListViewCellFormattingEventArgs e)
{
   if (e.CellElement.Image != null)
   {
       e.CellElement.Image = e.CellElement.Image.GetThumbnailImage(32, 32, null,
IntPtr.Zero);
   }
}
```

[VB.NET] Set the cell image

```
Private Sub radListView1_CellFormatting(sender As Object, e As
ListViewCellFormattingEventArgs)
    If e.CellElement.Image IsNot Nothing Then
        e.CellElement.Image = e.CellElement.Image.GetThumbnailImage(32, 32, Nothing,
IntPtr.Zero)
    End If
End Sub
```

7. The **ColumnCreating** event is fired when a column is being created. This is convenient event to hide unwanted columns. Additionally, we will use this event to set some more user friendly column headers.

[C#] Customize columns

```
void radListView1_ColumnCreating(object sender, ListViewColumnCreatingEventArgs e)
{
   if (e.Column.FieldName == "SongID" || e.Column.FieldName == "Image")
   {
       e.Column.Visible = false;
   }

   if (e.Column.FieldName == "SongName")
   {
       e.Column.HeaderText = "Song Title";
   }

   if (e.Column.FieldName == "ArtistName")
   {
       e.Column.HeaderText = "Artist";
   }

   if (e.Column.FieldName == "AlbumName")
   {
       e.Column.HeaderText = "Album";
   }
}
```

[VB.NET] Customize columns

```
Private Sub radListView1_ColumnCreating(sender As Object, e As
ListViewColumnCreatingEventArgs)
    If e.Column.FieldName = "SongID" OrElse e.Column.FieldName = "Image" Then
        e.Column.Visible = False
    End If

    If e.Column.FieldName = "SongName" Then
        e.Column.HeaderText = "Song Title"
    End If

    If e.Column.FieldName = "ArtistName" Then
        e.Column.HeaderText = "Artist"
    End If

    If e.Column.FieldName = "AlbumName" Then
        e.Column.HeaderText = "Album"
    End If
End Sub
```

8. The last event of RadListView, which we are going to handle is the **ViewTypeChanged** event - fired when the **ViewType** of the control is changed. This event is convenient to set view specific settings. To handle the event, we will create three helper methods:

- **SetupDetailsView** - here we will set the **AllowArbitraryItemHeight**, property to *true*, in order to allow the items to size themselves in height, according to their content.

- **SetupIconsView** - here we will define a custom size for the items, set some spacing between the items and again set the **AllowArbitraryItemHeight**, property to *true*
- **SetupSimpleListView** - in this method we will only set the **AllowArbitraryItemHeight**, property to *true*.

In the **ViewTypeChanged** event handler, we will simply check which is the new view and call the corresponding setup method

[C#] Handling view type changes

```
private void SetupDetailsView()
{
   this.radListView1.AllowArbitraryItemHeight = true;
}

private void SetupIconsView()
{
   this.radListView1.ItemSize = new Size(200, 64);
   this.radListView1.ItemSpacing = 5;
   this.radListView1.AllowArbitraryItemHeight = true;
}

private void SetupSimpleListView()
{
   this.radListView1.AllowArbitraryItemHeight = true;
}

void radListView1_ViewTypeChanged(object sender, EventArgs e)
{
   switch (radListView1.ViewType)
   {
       case ListViewType.ListView:
           SetupSimpleListView();
           break;
       case ListViewType.IconsView:
           SetupIconsView();
           break;
       case ListViewType.DetailsView:
           SetupDetailsView();
           break;
   }
}
```

[VB.NET] Handling view type changes

```
Private Sub SetupDetailsView()
    Me.RadListView1.AllowArbitraryItemHeight = True
End Sub

Private Sub SetupIconsView()
    Me.RadListView1.ItemSize = New Size(200, 64)
    Me.RadListView1.ItemSpacing = 5
    Me.RadListView1.AllowArbitraryItemHeight = True
End Sub

Private Sub SetupSimpleListView()
```

```
    Me.RadListView1.AllowArbitraryItemHeight = True
End Sub

Private Sub radListView1_ViewTypeChanged(sender As Object, e As EventArgs)
    Select Case RadListView1.ViewType
        Case ListViewType.ListView
            SetupSimpleListView()
            Exit Select
        Case ListViewType.IconsView
            SetupIconsView()
            Exit Select
        Case ListViewType.DetailsView
            SetupDetailsView()
            Exit Select
    End Select
End Sub
```

9. Now we only need to fill up the **RadCommandBar** elements functionality. First we are going to handle the view changing buttons. For this purpose, subscribe for the **ToggleStateChanged** and **ToggleStateChanging** events of all the **CommandBarToggleButtons** that we have added earlier. In the ToggleStateChanged event handler, check which is the clicked button, and set the rest of the buttons to *Off*. Additionally, set the **RadListViewViewType** according to the pressed button.

[C#] Handle the toggle buttons

```
private bool updatingToggleState = false;

private void ViewToggleButton_ToggleStateChanged(object sender, StateChangedEventArgs args)
{
   if (updatingToggleState)
   {
       return;
   }

   this.updatingToggleState = true;

   if (this.commandBarToggleButtonDetails != sender)
   {
       this.commandBarToggleButtonDetails.ToggleState = ToggleState.Off;
   }

   if (this.commandBarToggleButtonList != sender)
   {
       this.commandBarToggleButtonList.ToggleState = ToggleState.Off;
   }

   if (this.commandBarToggleButtonTiles != sender)
   {
       this.commandBarToggleButtonTiles.ToggleState = ToggleState.Off;
   }

   this.updatingToggleState = false;

   if (this.commandBarToggleButtonDetails.ToggleState == ToggleState.On)
   {
```

```
        this.radListView1.ViewType = ListViewType.DetailsView;
    }

    if (this.commandBarToggleButtonList.ToggleState == ToggleState.On)
    {
        this.radListView1.ViewType = ListViewType.ListView;
    }

    if (this.commandBarToggleButtonTiles.ToggleState == ToggleState.On)
    {
        this.radListView1.ViewType = ListViewType.IconsView;
    }
}

private void ViewToggleButton_ToggleStateChanging(object sender, StateChangingEventArgs
args)
{
    if (!updatingToggleState && args.OldValue == ToggleState.On)
    {
        args.Cancel = true;
    }
}
```

[VB.NET] Handle the toggle buttons

```
Private updatingToggleState As Boolean = False

Private Sub ViewToggleButton_ToggleStateChanged(sender As Object, args As
StateChangedEventArgs) Handles commandBarToggleButtonTiles.ToggleStateChanged,
commandBarToggleButtonList.ToggleStateChanged,
commandBarToggleButtonDetails.ToggleStateChanged
    If updatingToggleState Then
        Return
    End If

    Me.updatingToggleState = True

    If Me.commandBarToggleButtonDetails IsNot sender Then
        Me.commandBarToggleButtonDetails.ToggleState = ToggleState.Off
    End If

    If Me.commandBarToggleButtonList IsNot sender Then
        Me.commandBarToggleButtonList.ToggleState = ToggleState.Off
    End If

    If Me.commandBarToggleButtonTiles IsNot sender Then
        Me.commandBarToggleButtonTiles.ToggleState = ToggleState.Off
    End If

    Me.updatingToggleState = False

    If Me.commandBarToggleButtonDetails.ToggleState = ToggleState.[On] Then
        Me.RadListView1.ViewType = ListViewType.DetailsView
    End If
```

```
    If Me.commandBarToggleButtonList.ToggleState = ToggleState.[On] Then
        Me.RadListView1.ViewType = ListViewType.ListView
    End If

    If Me.commandBarToggleButtonTiles.ToggleState = ToggleState.[On] Then
        Me.RadListView1.ViewType = ListViewType.IconsView
    End If

End Sub

Private Sub ViewToggleButton_ToggleStateChanging(sender As Object, args As
StateChangingEventArgs) Handles commandBarToggleButtonTiles.ToggleStateChanging,
commandBarToggleButtonList.ToggleStateChanging,
commandBarToggleButtonDetails.ToggleStateChanging
    If Not updatingToggleState AndAlso args.OldValue = ToggleState.[On] Then
        args.Cancel = True
    End If
End Sub
```

10. Next, subscribe to the **SelectedIndexChanged** event of *commandBarDropDownSort***CommandBarDropDownList**. In the event handler, we are going to add the desired **SortDescriptors**, according to the selected item in the drop down.

[C#] Handle sorting functionality

```
private void commandBarDropDownSort_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
   this.radListView1.SortDescriptors.Clear();
   switch (this.commandBarDropDownSort.Text)
   {
       case "Song Name":
           this.radListView1.SortDescriptors.Add(new SortDescriptor("SongName",
ListSortDirection.Ascending));
           this.radListView1.EnableSorting = true;
           break;
       case "Album":
           this.radListView1.SortDescriptors.Add(new SortDescriptor("AlbumName",
ListSortDirection.Ascending));
           this.radListView1.EnableSorting = true;
           break;
       case "Artist":
           this.radListView1.SortDescriptors.Add(new SortDescriptor("ArtistName",
ListSortDirection.Ascending));
           this.radListView1.EnableSorting = true;
           break;
   }
}
```

[VB.NET] Handle sorting functionality

```
Private Sub commandBarDropDownSort_SelectedIndexChanged(sender As Object, e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles
commandBarDropDownSort.SelectedIndexChanged
    Me.RadListView1.SortDescriptors.Clear()
```

```
    Select Case Me.commandBarDropDownSort.Text
        Case "Song Name"
            Me.RadListView1.SortDescriptors.Add(New SortDescriptor("SongName",
ListSortDirection.Ascending))
            Me.RadListView1.EnableSorting = True
            Exit Select
        Case "Album"
            Me.RadListView1.SortDescriptors.Add(New SortDescriptor("AlbumName",
ListSortDirection.Ascending))
            Me.RadListView1.EnableSorting = True
            Exit Select
        Case "Artist"
            Me.RadListView1.SortDescriptors.Add(New SortDescriptor("ArtistName",
ListSortDirection.Ascending))
            Me.RadListView1.EnableSorting = True
            Exit Select
    End Select
End Sub
```

11. To add the grouping functionality, subscribe to the **SelectedIndexChanged** event of *commandBarDropDownGroup*, **CommandBarDropDownList**. Similar to the sorting functionality, add the desired **GroupDescriptors** according to the selected item.

[C#] Handle grouping functionality

```
private void commandBarDropDownGroup_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
   this.radListView1.GroupDescriptors.Clear();
   switch (this.commandBarDropDownGroup.Text)
   {
       case "None":
           this.radListView1.EnableGrouping = false;
           this.radListView1.ShowGroups = false;
           break;
       case "Album":
           this.radListView1.GroupDescriptors.Add(new GroupDescriptor(
               new SortDescriptor[] { new SortDescriptor("AlbumName",
ListSortDirection.Ascending) }));
           this.radListView1.EnableGrouping = true;
           this.radListView1.ShowGroups = true;
           break;
       case "Artist":
           this.radListView1.GroupDescriptors.Add(new GroupDescriptor(
               new SortDescriptor[] { new SortDescriptor("ArtistName",
ListSortDirection.Ascending) }));
           this.radListView1.EnableGrouping = true;
           this.radListView1.ShowGroups = true;
           break;
   }
}
```

[VB.NET] Handle grouping functionality

```
Private Sub commandBarDropDownGroup_SelectedIndexChanged(sender As Object, e As
```

```
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles
commandBarDropDownGroup.SelectedIndexChanged
    Me.RadListView1.GroupDescriptors.Clear()
    Select Case Me.commandBarDropDownGroup.Text
        Case "None"
            Me.RadListView1.EnableGrouping = False
            Me.RadListView1.ShowGroups = False
            Exit Select
        Case "Album"
            Me.RadListView1.GroupDescriptors.Add(New GroupDescriptor(New SortDescriptor()
{New SortDescriptor("AlbumName", ListSortDirection.Ascending)}))
            Me.RadListView1.EnableGrouping = True
            Me.RadListView1.ShowGroups = True
            Exit Select
        Case "Artist"
            Me.RadListView1.GroupDescriptors.Add(New GroupDescriptor(New SortDescriptor()
{New SortDescriptor("ArtistName", ListSortDirection.Ascending)}))
            Me.RadListView1.EnableGrouping = True
            Me.RadListView1.ShowGroups = True
            Exit Select
    End Select
End Sub
```

12. Lastly, lets subscribe ot the **TextChanged** event of *commandBarTextBoxFilter***CommandBarTextBox**. Here we will add **FilterDescriptor** according to the text entered in the text box:

[C#] Handle filtering functionality

```
private void commandBarTextBoxFilter_TextChanged(object sender, EventArgs e)
{
   this.radListView1.FilterDescriptors.Clear();

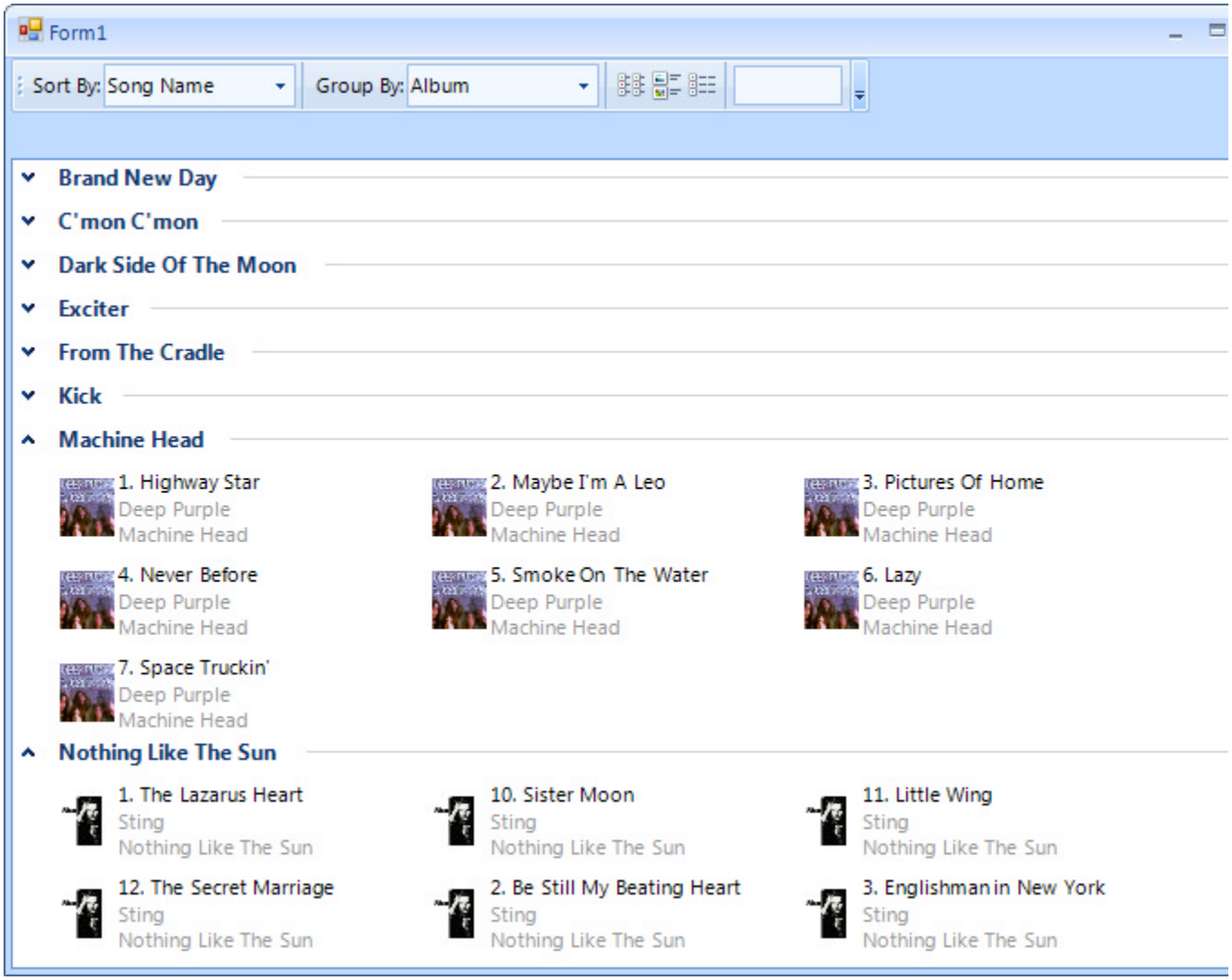
   if (String.IsNullOrEmpty(this.commandBarTextBoxFilter.Text))
   {
       this.radListView1.EnableFiltering = false;
   }
   else
   {
       this.radListView1.FilterDescriptors.LogicalOperator = FilterLogicalOperator.Or;
       this.radListView1.FilterDescriptors.Add("SongName", FilterOperator.Contains,
this.commandBarTextBoxFilter.Text);
       this.radListView1.FilterDescriptors.Add("AlbumName", FilterOperator.Contains,
this.commandBarTextBoxFilter.Text);
       this.radListView1.FilterDescriptors.Add("ArtistName", FilterOperator.Contains,
this.commandBarTextBoxFilter.Text);
       this.radListView1.EnableFiltering = true;
   }
}
```

[VB.NET] Handle filtering functionality

```
Private Sub commandBarTextBoxFilter_TextChanged(sender As Object, e As EventArgs) Handles
commandBarTextBoxFilter.TextChanged
    Me.RadListView1.FilterDescriptors.Clear()
```

```
    If [String].IsNullOrEmpty(Me.commandBarTextBoxFilter.Text) Then
        Me.RadListView1.EnableFiltering = False
    Else
        Me.RadListView1.FilterDescriptors.LogicalOperator = FilterLogicalOperator.[Or]
        Me.RadListView1.FilterDescriptors.Add("SongName", FilterOperator.Contains,
Me.commandBarTextBoxFilter.Text)
        Me.RadListView1.FilterDescriptors.Add("AlbumName", FilterOperator.Contains,
Me.commandBarTextBoxFilter.Text)
        Me.RadListView1.FilterDescriptors.Add("ArtistName", FilterOperator.Contains,
Me.commandBarTextBoxFilter.Text)
        Me.RadListView1.EnableFiltering = True
    End If
End Sub
```

13. Run the application and try the different functionalties:



10.6 Summary

In this chapter you have learnt how to use RadListView in the different ViewType modes, some basic customizations and working with the functionalities that the control provides - sorting, filtering and grouping.

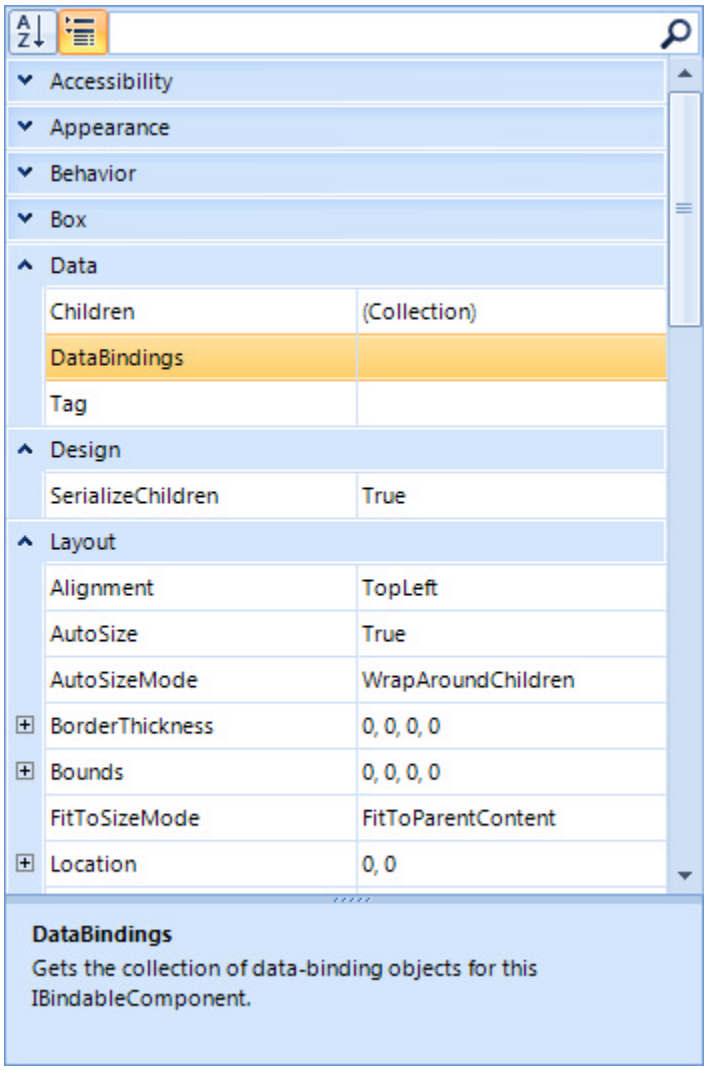
11 PropertyGrid

11.1 Objectives

- Learn how to start using RadPropertyGrid
- Learn how to setup the control.
- Explore its functionalities.
- Find our its formatting capabilities.
- Learn how to use the Validation functionalities of RadPropertyGrid.

11.2 Introduction

RadPropertyGrid for WinForms displays the properties of a given object in a user-friendly way allowing the end-user to edit these properties using our editors. Now you can concentrate all the settings in one place instead of scattering them all over your forms. RadPropertyGrid gives the end-user the ability to filter, group and sort its items thanks to our data processing engine used also by controls like RadGridView, RadListView etc. RadPropertyGrid takes full advantage of the virtualization mechanism, so even if you load an object with hundreds of properties, no problem, RadPropertyGrid will handle these cases.



RadPropertyGrid comes with a built-in tool bar which filters properties as you type. You can customize its behavior by defining the part of the property by which the filter is applied. For example, you can filter by property name, property label, property category or property description. You can also change the operator by which the filter is applied - Starts, Contains, etc.

RadPropertyGrid also includes a help bar at the bottom where the name of the selected property is displayed along with the description from the DescriptionAttribute of the property. Both bars can be easily shown or hidden by setting a single property for each bar.

For those of you who would like to additionally customize RadPropertyGrid, the good news is that this control can be easily extended by adding elements to it, such as your custom toolbars or another type of **Telerik Presentation Framework (Section 7.2)** element. You can add elements to the RadPropertyGrid or the tool bar itself.

As to the editing capabilities, RadPropertyGrid allows you to control the process of editing all the way through - from the initialization and the type of editor used to validation and value changing events.

11.3 Getting Started

In this article, we will demonstrate how easy it is to start using RadPropertyGrid to display the properties of a certain object.

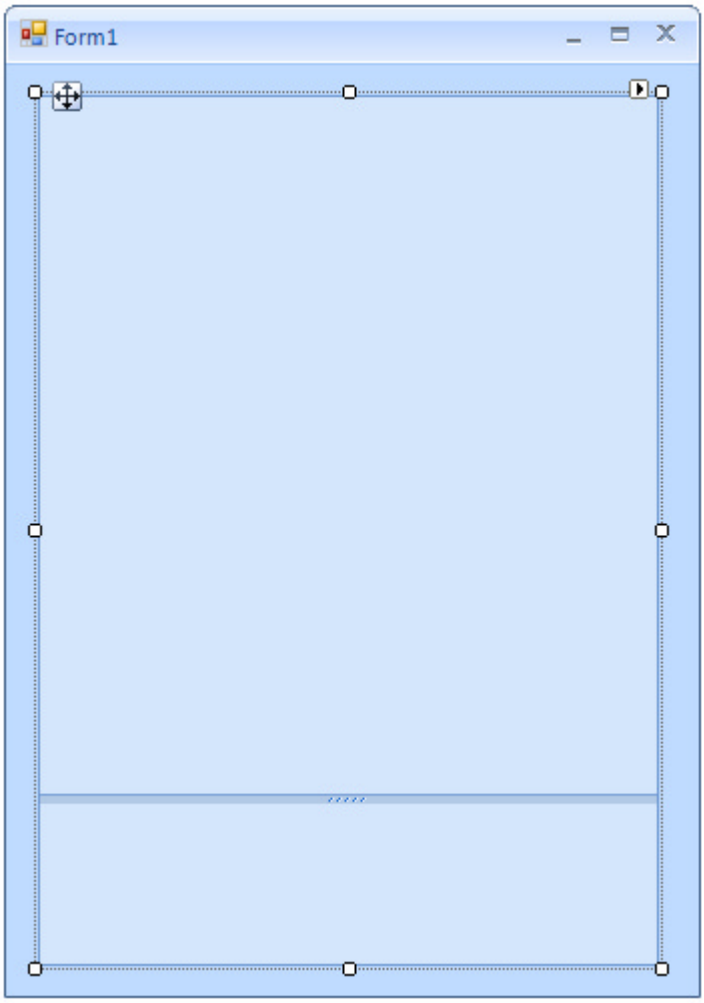
Preparing the Project



You can find the complete source for this project at:

\PropertyGrid\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. After the form is created, increase its size in height and drag RadPropertyGrid control to it.
6. Here is how it should look like in this moment



7. Now, let's create custom object with different type of public properties and one private property:
[C#] Custom object

```
public class Dog
{
public string Name { get; set; }
public int Age { get; set; }
public double Weight { get; set; }
public string Gender{ get; set; }
public bool Homeless { get; set; }
public DateTime Birthday { get; set; }
private string OwnerName { get; set; }
}
```

[VB.NET] Custom object

```
Public Class Dog
Public Property Name() As String
Get
Return m_Name
End Get
Set(ByVal value As String)
m_Name = Value
```

```
End Set
End Property
Private m_Name As String
Public Property Age() As Integer
Get
Return m_Age
End Get
Set(ByVal value As Integer)
m_Age = Value
End Set
End Property
Private m_Age As Integer
Public Property Weight() As Double
Get
Return m_Weight
End Get
Set(ByVal value As Double)
m_Weight = Value
End Set
End Property
Private m_Weight As Double
Public Property Gender() As String
Get
Return m_Gender
End Get
Set(ByVal value As String)
m_Gender = Value
End Set
End Property
Private m_Gender As String
Public Property Homeless() As Boolean
Get
Return m_Homeless
End Get
Set(ByVal value As Boolean)
m_Homeless = Value
End Set
End Property
Private m_Homeless As Boolean
Public Property Birthday() As DateTime
Get
Return m_Birthday
End Get
Set(ByVal value As DateTime)
m_Birthday = Value
End Set
End Property
Private m_Birthday As DateTime
Private Property OwnerName() As String
Get
Return m_OwnerName
End Get
Set(ByVal value As String)
m_OwnerName = Value
End Set
```

```
End Property
Private m_OwnerName As String
End Class
```

8. After the object is created, all you have to do in order to expose its **public** properties in RadPropertyGrid is to set the **SelectedObject** property of the control.

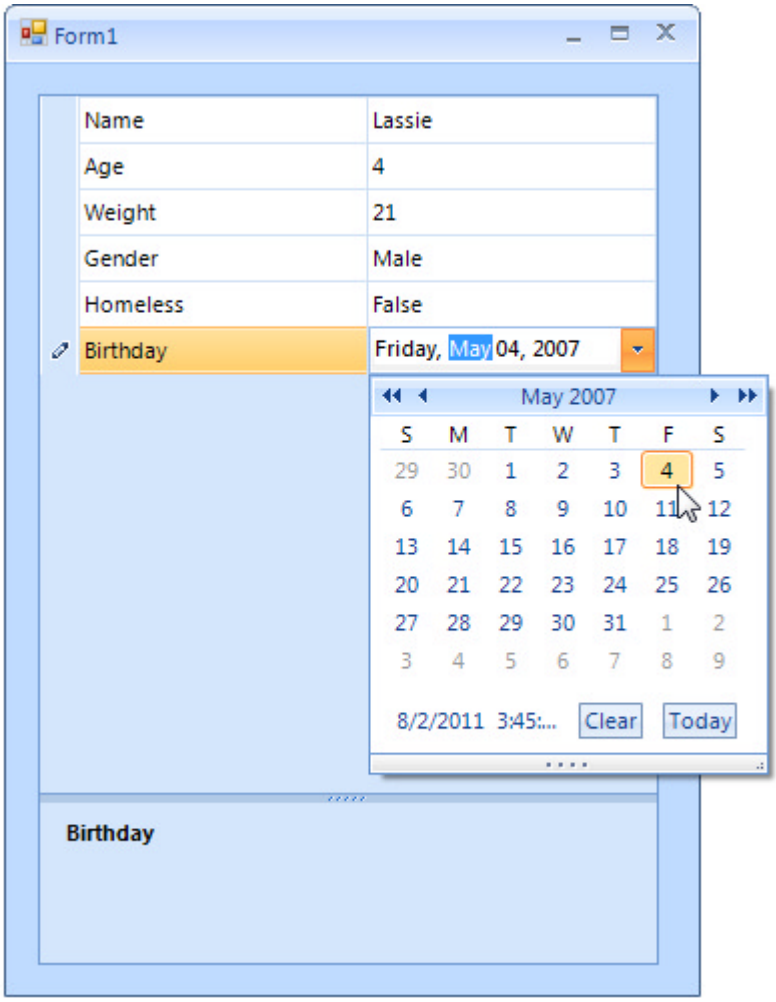
[C#] Assign object to RadPropertyGrid

```
Dog myDog = new Dog();
myDog.Name = "Lassie";
myDog.Age = 4;
myDog.Birthday = new DateTime(2007, 5, 4);
myDog.Gender = "Male";
myDog.Homeless = false;
myDog.Weight = 21;
radPropertyGrid1.SelectedObject = myDog;
```

[VB.NET] Assign object to RadPropertyGrid

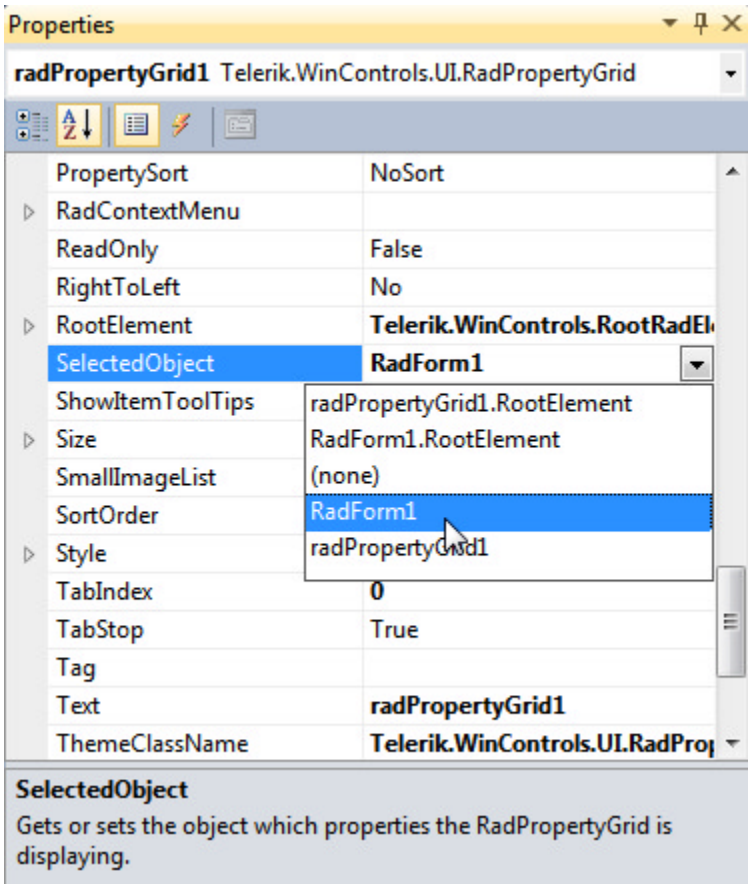
```
Dim myDog As New Dog()
myDog.Name = "Lassie"
myDog.Age = 4
myDog.Birthday = New DateTime(2007, 5, 4)
myDog.Gender = "Male"
myDog.Homeless = False
myDog.Weight = 21
RadPropertyGrid1.SelectedObject = myDog
```

9. Here is the result:

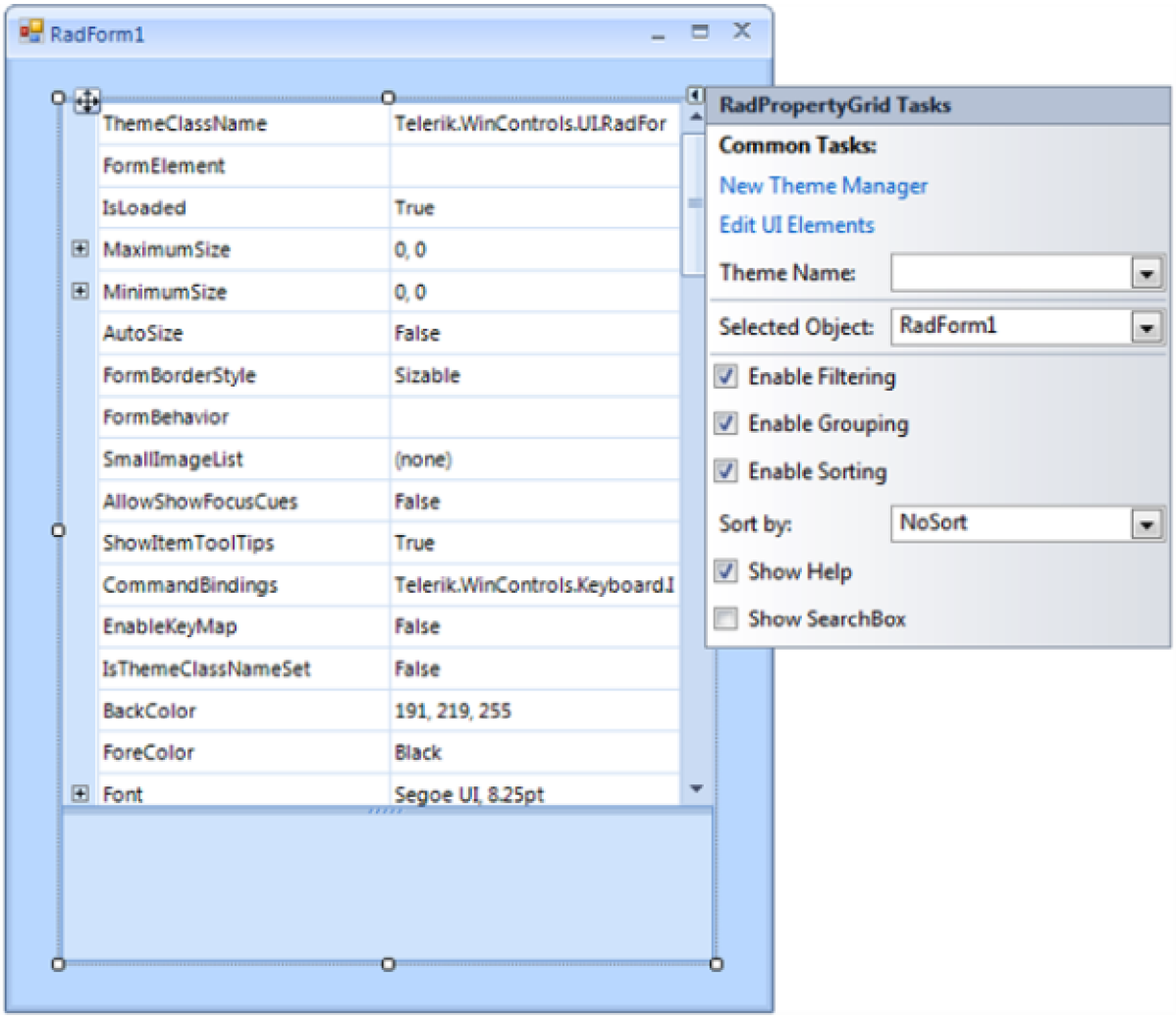


11.4 Using the Design Time Interface

The design time support of RadPropertyGrid allows you in a few click to setup a the control. All that you have to do after the control is on the form is to set the SelectedObject property of RadPropertyGrid to an object which properties you want to be displayed.



From the **Smart Tag** of the control, you can enable the most commonly used functionalities in RadPropertyGrid - .Filtering, Sorting, Grouping - and also, you can show or hide the help bar and the tool bar.



11.5 Programming

In the following tutorial, you will get familiar with the control, its basic functionalities and its basic customization abilities.



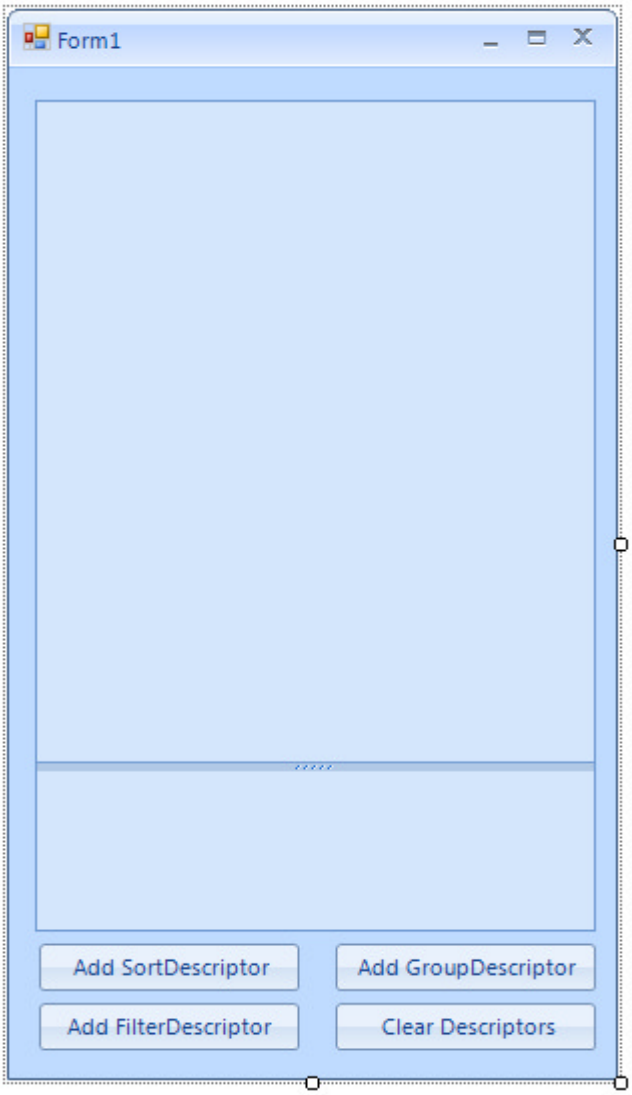
You can find the complete source for this project at:

\PropertyGrid\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. After the form is created, increase its size in height and drag RadPropertyGrid control to it.
6. Now underneath the form, add four RadButtons with the following properties and create their click event handlers
    - Name: sortButton, Text: Add SortDescriptor
    - Name: filterButton, Text: Add FilterDescriptor

- Name: groupButton, Text: Add GroupDescriptor
- Name: clearButton, Text: Clear tDescriptor

7. Here is how it should look like in this moment



8. Let's start with the code. First we are going to set the SelectedObject to PropertyGridElement:

[C#] Set SelectedObject

```
radPropertyGrid1.SelectedObject = new PropertyGridElement();
```

[VB.NET] Set SelectedObject

```
RadPropertyGrid1.SelectedObject = New PropertyGridElement()
```

9. Next, lets show the built-in toolbar, which provides the user with grouping, sorting and filtering out of the box:

[C#] Show the toolbar

```
radPropertyGrid1.ToolbarVisible = true;
```

[VB.NET] Show the toolbar

```
RadPropertyGrid1.ToolbarVisible = True
```

10. We can customize the toolbar a bit, by adding our own button, which will clear the filter on click:

[C#] Add custom button to toolbar

```
RadButtonElement clearFiltering = new RadButtonElement();
clearFiltering.Text = "Clear";
clearFiltering.MinSize = new System.Drawing.Size(25, 22);
clearFiltering.StretchHorizontally = false;
radPropertyGrid1.PropertyGridElement.ToolbarElement.Children.Add(clearFiltering);
clearFiltering.Click += new EventHandler(clearFiltering_Click);
```

[C#] Clear filter text box

```
void clearFiltering_Click(object sender, EventArgs e)
{
radPropertyGrid1.PropertyGridElement.ToolbarElement.SearchTextBoxElement.Text = "";
}
```

[VB.NET] Add custom button to toolbar

```
Dim clearFiltering As New RadButtonElement()
clearFiltering.Text = "Clear"
clearFiltering.MinSize = New System.Drawing.Size(25, 22)
clearFiltering.StretchHorizontally = False
RadPropertyGrid1.PropertyGridElement.ToolbarElement.Children.Add(clearFiltering)
AddHandler clearFiltering.Click, AddressOf clearFiltering_Click
```

[VB.NET] Clear filter text box

```
Private Sub clearFiltering_Click(ByVal sender As Object, ByVal e As EventArgs)
RadPropertyGrid1.PropertyGridElement.ToolbarElement.SearchTextBoxElement.Text = ""
End Sub
```

11. Let's set the **BedingEditMode** property to **BeginEditOnClick**, in order to start the editing process on the first mouse click. Alternative options are **BeginEditOnSecondClick** or **BeginEditProgrammatically:**

[C#] Choose BeginEditMode

```
radPropertyGrid1.BeginEditMode = RadPropertyGridBeginEditModes.BeginEditOnClick;
```

[VB.NET] Choose BeginEditMode

```
RadPropertyGrid1.BeginEditMode = RadPropertyGridBeginEditModes.BeginEditOnClick
```

12. Next, we can choose which item to be selected when the control pops up, by assigning it to the SelectedGridItem property. Let's select the "Image" propety by default:

[C#] Select item

```
radPropertyGrid1.SelectedGridItem = radPropertyGrid1.Items["Image"];
radPropertyGrid1.PropertyGridElement.PropertyTableElement.ScrollToItem
(radPropertyGrid1.Items["Image"]);
```

[VB.NET] Select item

```
RadPropertyGrid1.SelectedGridItem = RadPropertyGrid1.Items("Image")
RadPropertyGrid1.PropertyGridElement.PropertyTableElement.ScrollToItem
(RadPropertyGrid1.Items("Image"))
```

13. In order to customize the items appearance you should handle the **ItemFormatting** event of

RadPropertyGrid. The following code changes the back color of all child items to yellow and it also changes the color of the items which value is *True* to LightGreen and *False* to Red:

[C#] Customize items appearance

```
void radPropertyGrid1_ItemFormatting(object sender, PropertyGridItemFormattingEventArgs e
{
//set the back color of all child items to yellow
if (e.Item.Level > 0)
{
e.VisualElement.BackColor = Color.Yellow;
}
else
{
e.VisualElement.ResetValue(LightVisualElement.BackColorProperty,
Telerik.WinControls.ValueResetFlags.Local);
}
//set the backcolor of items with value True to LightGreen and with value False to Red
PropertyGridItem item = e.Item as PropertyGridItem;
if (item != null && item.Value != null)
{
if (item.Value.ToString() == "True")
{
e.VisualElement.BackColor = Color.LightGreen;
}
else if (item.Value.ToString() == "False")
{
e.VisualElement.BackColor = Color.Red;
}
}
else
{
e.VisualElement.ResetValue(LightVisualElement.BackColorProperty,
Telerik.WinControls.ValueResetFlags.Local);
}
}
```

[VB.NET] Customize items appearance

```
Private Sub radPropertyGrid1_ItemFormatting(ByVal sender As Object, ByVal e As
PropertyGridItemFormattingEventArgs)
'set the back color of all child items to yellow
If e.Item.Level > 0 Then
e.VisualElement.BackColor = Color.Yellow
Else
e.VisualElement.ResetValue(LightVisualElement.BackColorProperty,
Telerik.WinControls.ValueResetFlags.Local)
End If
'set the backcolor of items with value True to LightGreen and with value equal to False t
Red
Dim item As PropertyGridItem = TryCast(e.Item, PropertyGridItem)
If item IsNot Nothing AndAlso item.Value IsNot Nothing Then
If item.Value.ToString() = "True" Then
e.VisualElement.BackColor = Color.LightGreen
ElseIf item.Value.ToString() = "False" Then
e.VisualElement.BackColor = Color.Red
End If
```

```
Else
e.VisualElement.ResetValue(LightVisualElement.BackColorProperty,
Telerik.WinControls.ValueResetFlags.Local)
End If
End Sub
```

14. The control setup is done. Now lets add functionality to the buttons event handlers. In the sortButton click event handler let's add a SortDescriptor which sorts the items according to their FormattedValue, Ascending:

[C#] Sort items

```
private void sortButton_Click(object sender, EventArgs e)
{
radPropertyGrid1.EnableSorting = true;
SortDescriptor sort = new SortDescriptor("FormattedValue", ListSortDirection.Ascending);
radPropertyGrid1.SortDescriptors.Add(sort);
}
```

[VB.NET] Sort items

```
Private Sub sortButton_Click(ByVal sender As Object, ByVal e As EventArgs)
RadPropertyGrid1.EnableSorting = True
Dim sort As New SortDescriptor("FormattedValue", ListSortDirection.Ascending)
RadPropertyGrid1.SortDescriptors.Add(sort)
End Sub
```

15. In the filterButton_Click event handler we will filter the items Name if it contains the word "size"

[C#] Filter items

```
private void filterButton_Click(object sender, EventArgs e)
{
FilterDescriptor filter = new FilterDescriptor("Name", FilterOperator.Contains, "size");
radPropertyGrid1.FilterDescriptors.Add(filter);
}
```

[VB.NET] Filter items

```
Private Sub filterButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Dim filter As New FilterDescriptor("Name", FilterOperator.Contains, "size")
RadPropertyGrid1.FilterDescriptors.Add(filter)
End Sub
```

16. Next, we will add a GroupDescriptor to the control in the groupButton_Click event handler. The descriptor will group the items according to their category in ascending order

[C#] Group items

```
private void groupButton_Click(object sender, EventArgs e)
{
radPropertyGrid1.EnableGrouping = true;
GroupDescriptor group = new GroupDescriptor(new SortDescriptor[] { new SortDescriptor
("Category", ListSortDirection.Ascending) });
radPropertyGrid1.GroupDescriptors.Add(group);
}
```

[VB.NET] Group items

```
Private Sub groupButton_Click(ByVal sender As Object, ByVal e As EventArgs)
RadPropertyGrid1.EnableGrouping = True
```

```
Dim group As New GroupDescriptor(New SortDescriptor() {New SortDescriptor("Category",
ListSortDirection.Ascending)})
RadPropertyGrid1.GroupDescriptors.Add(group)
End Sub
```

17. Lastly, in the clearButton_Click we will clear all added descriptors:

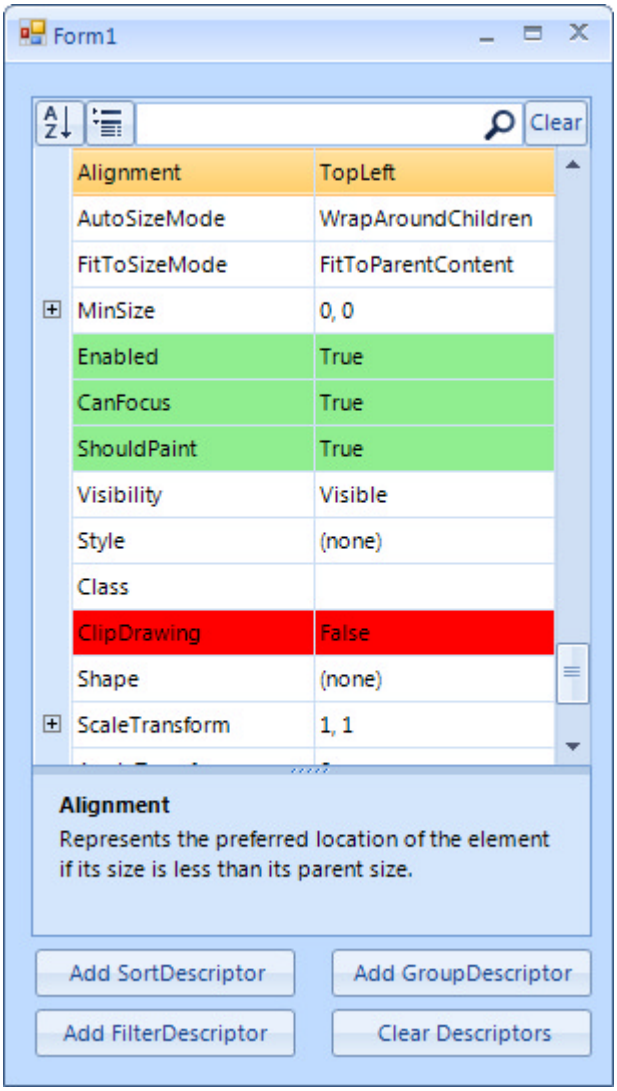
[C#] Clear descriptors

```
private void clearButton_Click(object sender, EventArgs e)
{
radPropertyGrid1.SortDescriptors.Clear();
radPropertyGrid1.FilterDescriptors.Clear();
radPropertyGrid1.GroupDescriptors.Clear();
}
```

[VB.NET] Clear descriptors

```
Private Sub clearButton_Click(ByVal sender As Object, ByVal e As EventArgs)
RadPropertyGrid1.SortDescriptors.Clear()
RadPropertyGrid1.FilterDescriptors.Clear()
RadPropertyGrid1.GroupDescriptors.Clear()
End Sub
```

18. Here is the resulted form. Feel free to investigate the created functionalities and formatting.



19. One more thing to add is validation. Let's handle the PropertyValidating event to do that. There we will forbid habind a propertiy of type string with empty string value. To test this functionality in our example, add some value to the **Text** property and then try to remove it.

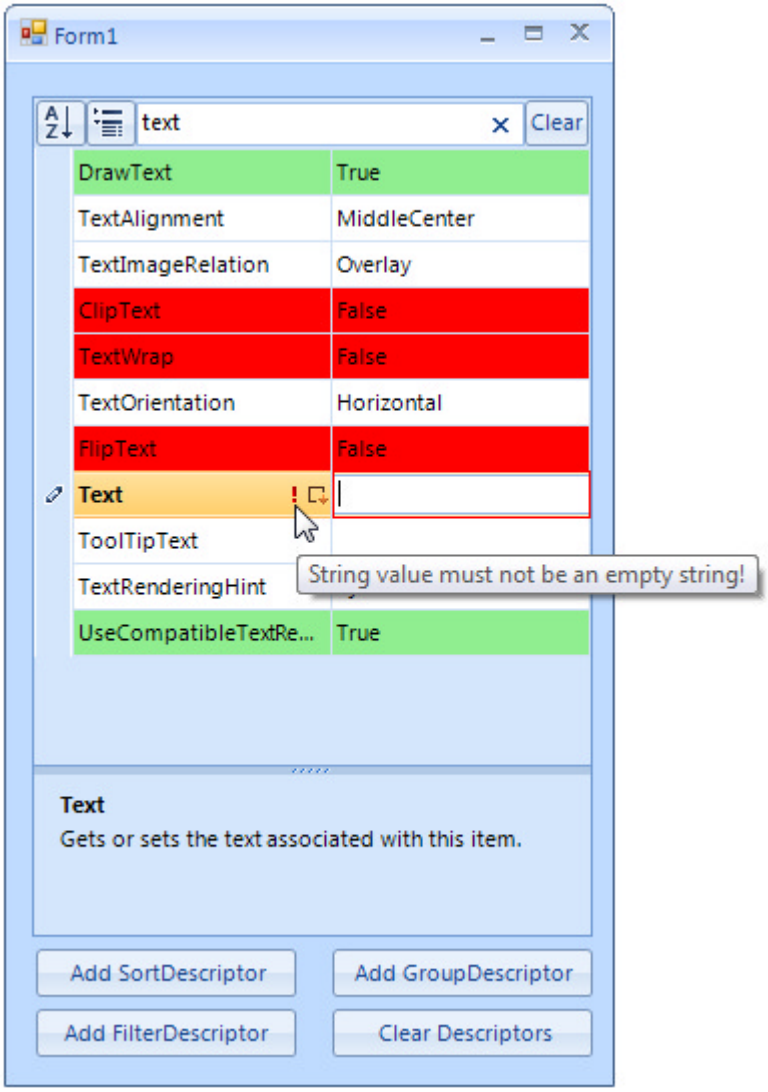
[C#] Validating

```
void radPropertyGrid1_PropertyValidating(object sender, PropertyValidatingEventArgs e)
{
PropertyGridItem item = e.Item as PropertyGridItem;
if (item.PropertyType == typeof(string))
{
if (string.IsNullOrEmpty(e.NewValue.ToString()))
{
item.ErrorMessage = "String value must not be an empty string!";
e.Cancel = true;
}
}
}
```

[VB.NET] Validating

```
Private Sub radPropertyGrid1_PropertyValidating(ByVal sender As Object, ByVal e As
```

```
PropertyValidatingEventArgs)
Dim item As PropertyGridItem = TryCast(e.Item, PropertyGridItem)
If item.PropertyType Is GetType(String) Then
If String.IsNullOrEmpty(e.NewValue.ToString()) Then
item.ErrorMessage = "String value must not be an empty string!"
e.Cancel = True
End If
End If
End Sub
```



11.6 Summary

This chapter explored how to work with RadPropertyGrid design time and how to take advantage of its functionalities like grouping, sorting and filtering. Furthermore, it showed how to customize the control appearance and also how to validate property values.

12 Data Binding

12.1 Objectives

- Learn about the controls that have DataSource properties and the bindable types that can be assigned to them.
- Become familiar with binding to simple arrays and lists of objects.
- Use BindingSource to bind to database, business objects and web services.
- Bind to simple controls using the DataBindings property.
- Bind to LINQ data sources.

12.2 Introduction

Up till now we've had to populate controls explicitly, one line of code at a time. Even the RadPageView "Custom Elements Walk Through" example where we populated a RadListControl with directory names had to be performed within a loop with each RadListDataItem being assigned properties from a column in the data. Here's how that looked:

[VB] Populating a RadListControl Iteratively

```
Dim directories As String() = Directory.GetDirectories(drive.ToString())
For Each directory As String In directories
listControl.Items.Add(New RadListDataItem(directory))
Next
```

[C#] Populating a RadListControl Iteratively

```
string[] directories = Directory.GetDirectories(drive.ToString());
foreach (string directory in directories)
{
listControl.Items.Add(new RadListDataItem(directory));
}
return listControl;
}
```

In contrast, data binding provides a consistent way of keeping a control and its data store synchronized. To data bind RadControls you assign the DataSource property. Here's how the same directory listing example above looks using data binding.

[VB] Populating a List Box by Data Binding

```
listControl.DataSource = Directory.GetDirectories("C:\")
```

[C#] Populating a List Box by Data Binding

```
listControl.DataSource = Directory.GetDirectories("C:\\");
```



If the DataSource is assigned a DataSet, you may need to assign the DataMember property to specify a data table within the DataSet.

What Data Can I Bind?

RadControls that have a DataSource property bind to any object that implements **IList** or **IListSource**. Those two interfaces are behind many familiar implementations that you can bind to: arrays, generic lists, data tables and views.

Type	Example

Array

[VB]

```
Dim myArray As String() = {"one", "two", "three"}
listControl.DataSource = myArray
```

[C#]

```
string[] myArray = {"one", "two", "three" };
listControl.DataSource = myArray;
```

List<>, BindingList<>

[VB]

```
Dim list As New List(Of String)()
listControl.DataSource = list
```

[C#]

```
List<string> list = new List<string>() {"one", "two", "three" };
listControl.DataSource = list;
```

DataTable

[VB]

```
Dim connection As New OleDbConnection(Properties.Settings.
[Default].MusicConnection)
Dim adapter As New OleDbDataAdapter("select AlbumID, AlbumName from
Albums", connection)
Dim table As New DataTable()
adapter.Fill(table)
listControl.DataSource = table
listControl.DisplayMember = "AlbumName"
listControl.ValueMember = "AlbumID"
```

[C#]

```
OleDbConnection connection = new OleDbConnection
(Properties.Settings.Default.MusicConnection);
OleDbDataAdapter adapter = new OleDbDataAdapter("select AlbumID, AlbumName
from Albums", connection);
DataTable table = new DataTable();
adapter.Fill(table);
listControl.DataSource = table;
listControl.DisplayMember = "AlbumName";
listControl.ValueMember = "AlbumID";
```

DataView

[VB]

```
Dim dataView As New DataView(table, "AlbumName LIKE 'B%'", "",
DataViewRowState.CurrentRows)
listControl.DataSource = dataView
listControl.DisplayMember = "AlbumName"
listControl.ValueMember = "AlbumID"
```

[C#]

```
DataView dataView = new DataView(table, "AlbumName LIKE 'B%'", "",
DataViewRowState.CurrentRows);
listControl.DataSource = dataView;
listControl.DisplayMember = "AlbumName";
listControl.ValueMember = "AlbumID";
```

[VB]

```
Dim table2 As New DataTable()
table2.ReadXml("c:\Albums.xml")
listControl.DataSource = table2
listControl.DisplayMember = "AlbumName"
listControl.ValueMember = "AlbumID"
```

XML Files

[C#]

```
DataTable table2 = new DataTable();
table2.ReadXml("c:\\Albums.xml");
listControl.DataSource = table2;
listControl.DisplayMember = "AlbumName";
listControl.ValueMember = "AlbumID";
```

What RadControls Can I Bind?

The following controls have a DataSource and can be bound. The properties that specify how column data is used varies by control.

- RadGridView
- RadTreeView
- RadScheduler
- RadDropDownList
- RadChart
- RadListControl
- RadCarousel

For simple binding scenarios, you can also use the DataBindings property available to all Windows Control descendants. To use DataBindings you need to describe the control property you want bound, the DataSource and the part of the DataSource you want bound to the control property. This can be done at design-time or in code. The example below shows the RadTextBox Text and Tag properties being bound to the "AlbumName" and "AlbumID" columns of the DataSource.

[VB] Using DataBindings

```
radTextBox1.DataBindings.Add("Text", table, "AlbumName")
radTextBox1.DataBindings.Add("Tag", table, "AlbumID")
```

[C#] Using DataBindings

```
radTextBox1.DataBindings.Add("Text", table, "AlbumName");
radTextBox1.DataBindings.Add("Tag", table, "AlbumID");
```

How Does the Control Know What to Display?

If there multiple columns of data, how does the control know what should be displayed? That aspect of binding is control-specific. Controls with relatively simple lists, e.g. RadListControl and RadDropDownList, have **DisplayMember** and **ValueMember** properties that correspond to the **Text** and **Value** properties of each item. The DisplayMember data is displayed in the control. ValueMember column data is stored and available programmatically. Typically ValueMember contains a record ID used to retrieve associated data. ValueMember can also hold any object you care to put in it.

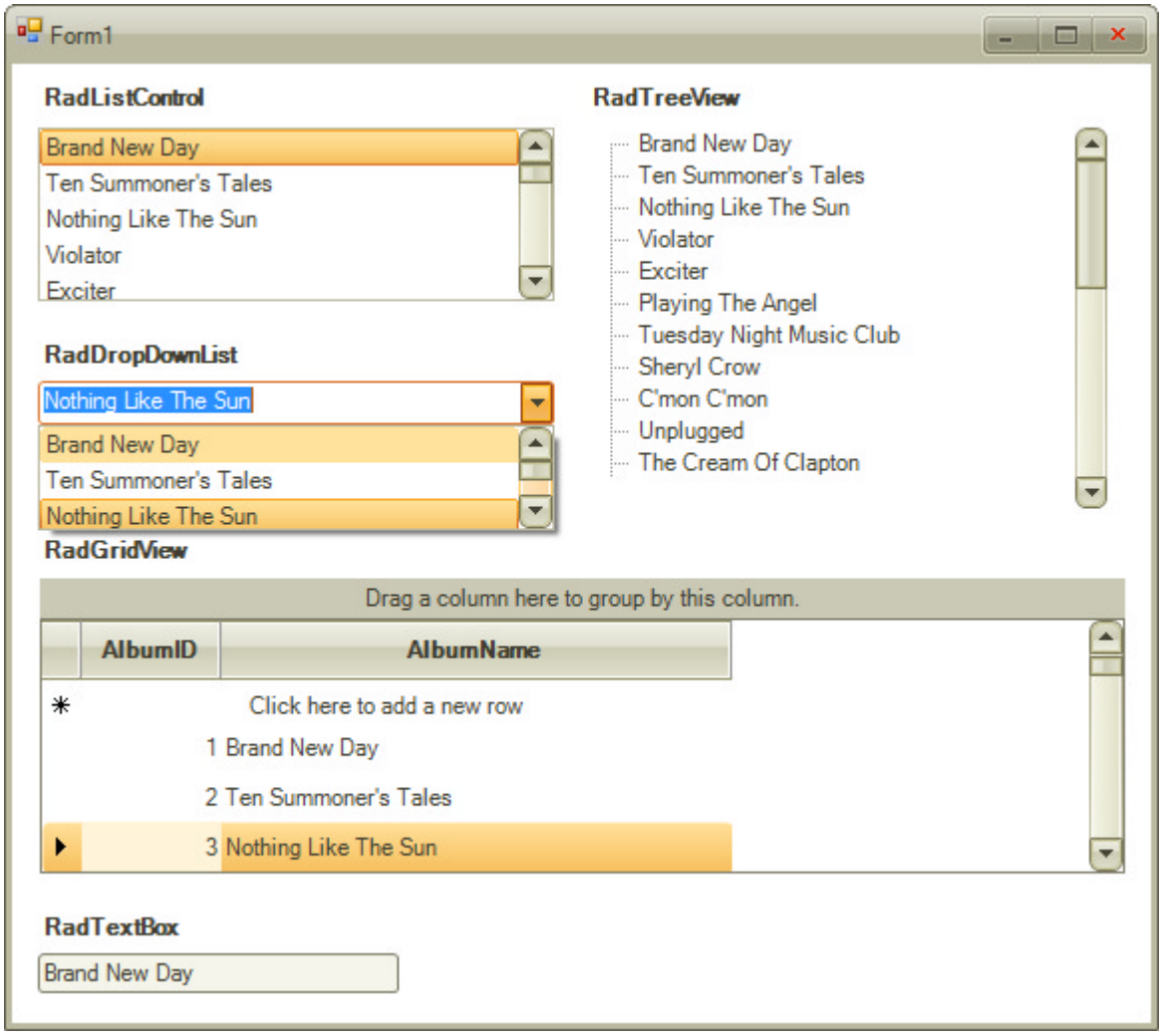
In contrast, the RadGridView is designed to display many levels of master detail tables and to display multiple columns worth of data at one time for each table. Using the most minimal route to getting data in RadGridView you still assign the DataSource/DataMember properties. You can stop right there and have all the columns in the DataSource displayed automatically. The chapter on RadGridView will detail how to customize each column and display master/detail tables.

The example below shows RadListControl, RadDropDownList, RadTreeView and RadGridView bound to the MusicCollection.mdb table that ships with RadControls for WinForms. The RadTextBox at the bottom of the form is bound using the DataBindings property. Notice that the locations within the data for each control are not synchronized.



You can find the complete source for this project at:

\DataBinding\VB|CS\Introduction



DataSource Column Assignment

You may have noticed in "What Data Can I Bind" above that DisplayMember is not always specified. If the data only has a single member, the data is displayed. Types other than string can also be displayed automatically, e.g. an array of DateTime values:



[VB] Automatically Assigning Columns

```
Dim arr As DateTime() = {DateTime.Today.AddDays(-1), DateTime.Today, DateTime.Today.AddDays
(1)}

radListControl1.DataSource = arr
```

[C#] Automatically Assigning Columns

```
DateTime[] arr = { DateTime.Today.AddDays(-1), DateTime.Today, DateTime.Today.AddDays(1) };

radListControl1.DataSource = arr;
```

Let's take one more look at a very simple "Product" object that contains an integer ID and Description string.

[VB] The Product Class

```
Public Class Product
Public Sub New(id As Integer, description As String)
 ID = id
 Description = description
End Sub
Public Property ID() As Integer
 Get
 End Get
 Set
 End Set
End Property
Public Property Description() As String
 Get
 End Get
 Set
 End Set
End Property
End Class
```

[C#] The Product Class

```
public class Product
{
  public Product(int id, string description)
  {
    ID = id;
    Description = description;
  }
  public int ID
  { get; set; }
  public string Description
  { get; set; }
}
```

If we create a list of products and bind to a RadControl without specifying a DisplayMember...

[VB] Binding the Products List

```
Dim products As New List(Of Product)()
products.Add(New Product(1, "Jute Heather Cardigan"))
products.Add(New Product(2, "Retro Cardigan"))
products.Add(New Product(3, "Cashmere Cardigan"))

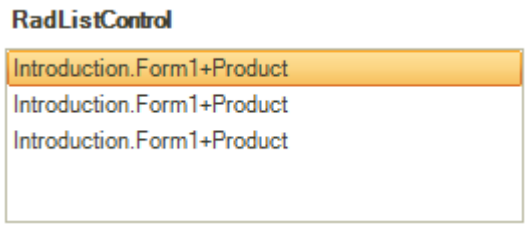
radListControl1.DataSource = products
```

[C#] Binding the Products List

```
List<Product> products = new List<Product>();
products.Add(new Product(1, "Jute Heather Cardigan"));
products.Add(new Product(2, "Retro Cardigan"));
products.Add(new Product(3, "Cashmere Cardigan"));

radListControl1.DataSource = products;
```

...the Product ***object name*** will be displayed for each record as a string:



Now with the addition of DisplayMember...

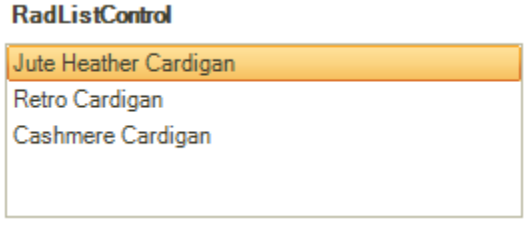
[VB] Binding the Products With DisplayMember

```
Dim products As New List(Of Product)()
products.Add(New Product(1, "Jute Heather Cardigan"))
products.Add(New Product(2, "Retro Cardigan"))
products.Add(New Product(3, "Cashmere Cardigan"))
radListControl1.DataSource = products
radListControl1.DisplayMember = "Description"
radListControl1.ValueMember = "ID"
```

[C#] Binding the Products With DisplayMember

```
List<Product> products = new List<Product>();
products.Add(new Product(1, "Jute Heather Cardigan"));
products.Add(new Product(2, "Retro Cardigan"));
products.Add(new Product(3, "Cashmere Cardigan"));
radListControl1.DataSource = products;
radListControl1.DisplayMember = "Description";
radListControl1.ValueMember = "ID";
```

...the Product "Description" column shows up in the list.



The ValueMember column is available whenever we access an item and examine the Value property. For example, if we add a SelectedIndexChange event handler to the RadListControl...

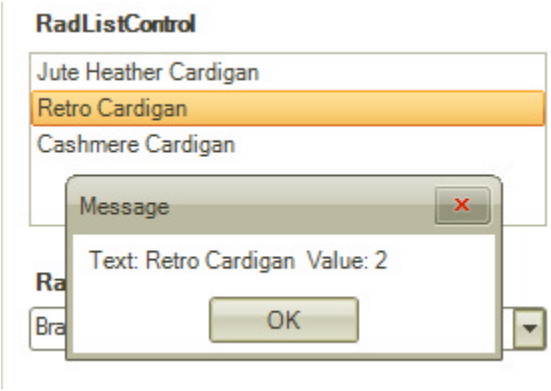
[VB] Retrieving the Value Property

```
Private Sub radListControl1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs)
Dim item As RadListDataItem = TryCast((TryCast(sender, RadListElement)).SelectedItem,
RadListDataItem)
RadMessageBox.SetThemeName("Desert")
RadMessageBox.Show(String.Format("Text: {0} Value: {1}", item.Text, item.Value))
End Sub
```

[C#] Retrieving the Value Property

```
void radListControl1_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
RadListDataItem item = (sender as RadListElement).SelectedItem as RadListDataItem;
RadMessageBox.SetThemeName("Desert");
RadMessageBox.Show(String.Format("Text: {0} Value: {1}", item.Text, item.Value));
}
```

...we can extract the Value property:



Interacting with Bound Data

When you want the ability to add, update and delete bound data, use an implementation of **IBindingList** such as BindingList<> or BindingSource.



You can find the complete source for this project at:

\DataBinding\VB|CS\BindingList

The objects that populate the IBindingList should implement the **INotifyPropertyChanged** interface. INotifyPropertyChanged supports the ListChanged event of BindingList and automatically refreshes bound controls, such as RadGridView, RadListControl, etc. To implement INotifyPropertyChanged you only need to add a PropertyChanged event.

[VB] Class with INotifyPropertyChanged

```
Class [MyClass]
 Implements INotifyPropertyChanged
 #region INotifyPropertyChanged Members
 Public Event PropertyChanged As PropertyChangedEventHandler
#End Region
End Class
```

[C#] Class with INotifyPropertyChanged

```
class MyClass : INotifyPropertyChanged
{
 #region INotifyPropertyChanged Members
 public event PropertyChangedEventHandler PropertyChanged;
 #endregion
}
```

The event is triggered from the "setter" of your properties after checking that the event handler is not null.

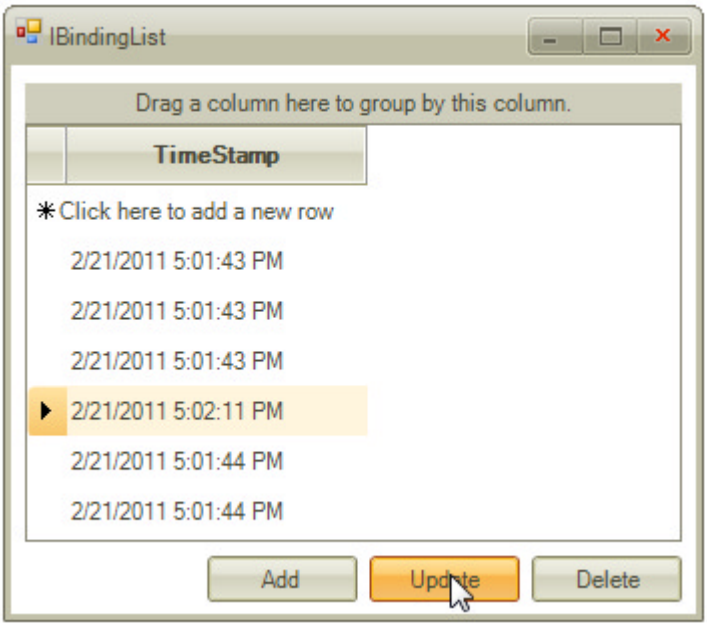
[VB] Triggering the Event

```
If PropertyChanged <> Nothing Then
 PropertyChanged(Me, New PropertyChangedEventArgs("MyPropertyName"))
End If
```

[C#] Triggering the Event

```
if (PropertyChanged != null)
{
 PropertyChanged(this, new PropertyChangedEventArgs("MyPropertyName"));
}
```

Here's a small example that shows add, update and delete entries in a BindingList of objects, and where the object implements INotifyPropertyChanged. Notice that the grid is never explicitly refreshed, yet changes to the BindingList are immediately apparent.



[VB] Using BindingList and INotifyPropertyChanged

```
Imports System
Imports System.ComponentModel
Imports System.Windows.Forms
Imports Telerik.WinControls.UI;

Namespace BindingList
 Public Partial Class Form1
  Inherits Form
  Public Sub New()
   InitializeComponent()
```

```
  End Sub
  ' BindingList to contain list of MyObject
  Private _myObjects As BindingList(Of MyObject)
  Private Sub Form1_Load(sender As Object, e As EventArgs)
   ' instantiate _myObjects and assign to gridview datasource
   _myObjects = New BindingList(Of MyObject)()
   radGridView1.DataSource = _myObjects
  End Sub
  Private Sub btnAdd_Click(sender As Object, e As EventArgs)
   ' newly added MyObject shows up automatically in the grid
   _myObjects.Add(New MyObject())
  End Sub
  Private Sub btnUpdate_Click(sender As Object, e As EventArgs)
   ' updated MyObject TimeStamp property shows up automatically in the grid
   If radGridView1.CurrentRow <> Nothing Then
    Dim currentObject As MyObject = TryCast(radGridView1.CurrentRow.DataBoundItem, MyObject)
    currentObject.TimeStamp = DateTime.Now
   End If
  End Sub
  Private Sub btnDelete_Click(sender As Object, e As EventArgs)
   ' deleted MyObject displays automatically in the grid
   If radGridView1.CurrentRow <> Nothing Then
    Dim currentObject As MyObject = TryCast(radGridView1.CurrentRow.DataBoundItem, MyObject)
    _myObjects.Remove(currentObject)
   End If
  End Sub
 End Class
 ' MyObject implements INotifyPropertyChanged.
 ' The object has a single property "TimeStamp". When
 ' TimeStamp is modified, PropertyChanged is fired
 Public Class MyObject
  Implements INotifyPropertyChanged
  Public Sub New()
   _timeStamp = DateTime.Now
  End Sub
  Private _timeStamp As DateTime
  Public Property TimeStamp() As DateTime
   Get
    Return _timeStamp
   End Get
   Set
    _timeStamp = value
    If PropertyChanged <> Nothing Then
     PropertyChanged(Me, New PropertyChangedEventArgs("TimeStamp"))
    End If
   End Set
  End Property
  Public Event PropertyChanged As PropertyChangedEventHandler
 End Class
End Namespace
```

[C#] Using BindingList and INotifyPropertyChanged

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
```

```
using Telerik.WinControls.UI;
namespace BindingList
{
 public partial class Form1 : RadForm
 {
   public Form1()
   {
     InitializeComponent();
   }
   // BindingList to contain list of MyObject
   private BindingList<MyObject> _myObjects;

   private void Form1_Load(object sender, EventArgs e)
   {
     // instantiate _myObjects and assign to gridview datasource
     _myObjects = new BindingList<MyObject>();
     radGridView1.DataSource = _myObjects;
   }
   private void btnAdd_Click(object sender, EventArgs e)
   {
     // newly added MyObject shows up automatically in the grid
     _myObjects.Add(new MyObject());
   }
   private void btnUpdate_Click(object sender, EventArgs e)
   {
     // updated MyObject TimeStamp property shows up automatically in the grid
     if (radGridView1.CurrentRow != null)
     {
       MyObject currentObject = radGridView1.CurrentRow.DataBoundItem as MyObject;
       currentObject.TimeStamp = DateTime.Now;
     }
   }
   private void btnDelete_Click(object sender, EventArgs e)
   {
     // deleted MyObject displays automatically in the grid
     if (radGridView1.CurrentRow != null)
     {
       MyObject currentObject = radGridView1.CurrentRow.DataBoundItem as MyObject;
       _myObjects.Remove(currentObject);
     }
   }
 }
 // MyObject implements INotifyPropertyChanged.
 // The object has a single property "TimeStamp". When
 // TimeStamp is modified, PropertyChanged is fired
 public class MyObject : INotifyPropertyChanged
 {
   public MyObject()
   {
     _timeStamp = DateTime.Now;
   }
   private DateTime _timeStamp;
   public DateTime TimeStamp
   {
     get
```

```
    {
      return _timeStamp;
    }
    set
    {
      _timeStamp = value;
      if (PropertyChanged != null)
      {
        PropertyChanged(this, new PropertyChangedEventArgs("TimeStamp"));
      }
    }
  }
  public event PropertyChangedEventHandler PropertyChanged;
 }
}
```

12.3 Getting Started

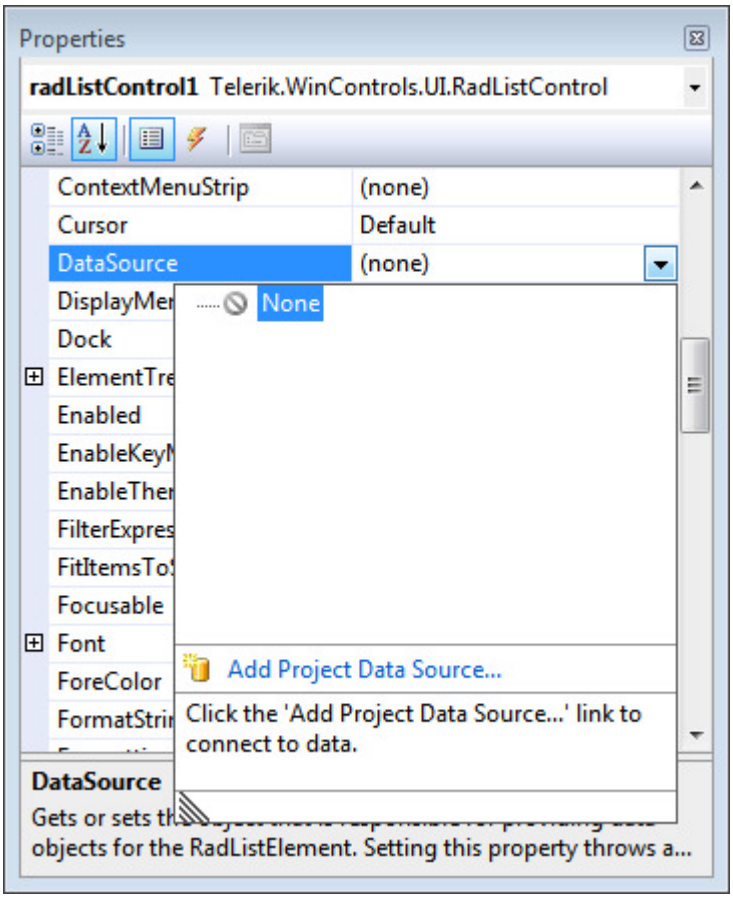
This project demonstrates binding a RadListControl to database data in the Albums table of the MusicCollection.mdb file that ships with RadControls for WinForms. You can follow these basic steps whenever you connect to database data at design time.



You can find the complete source for this project at:

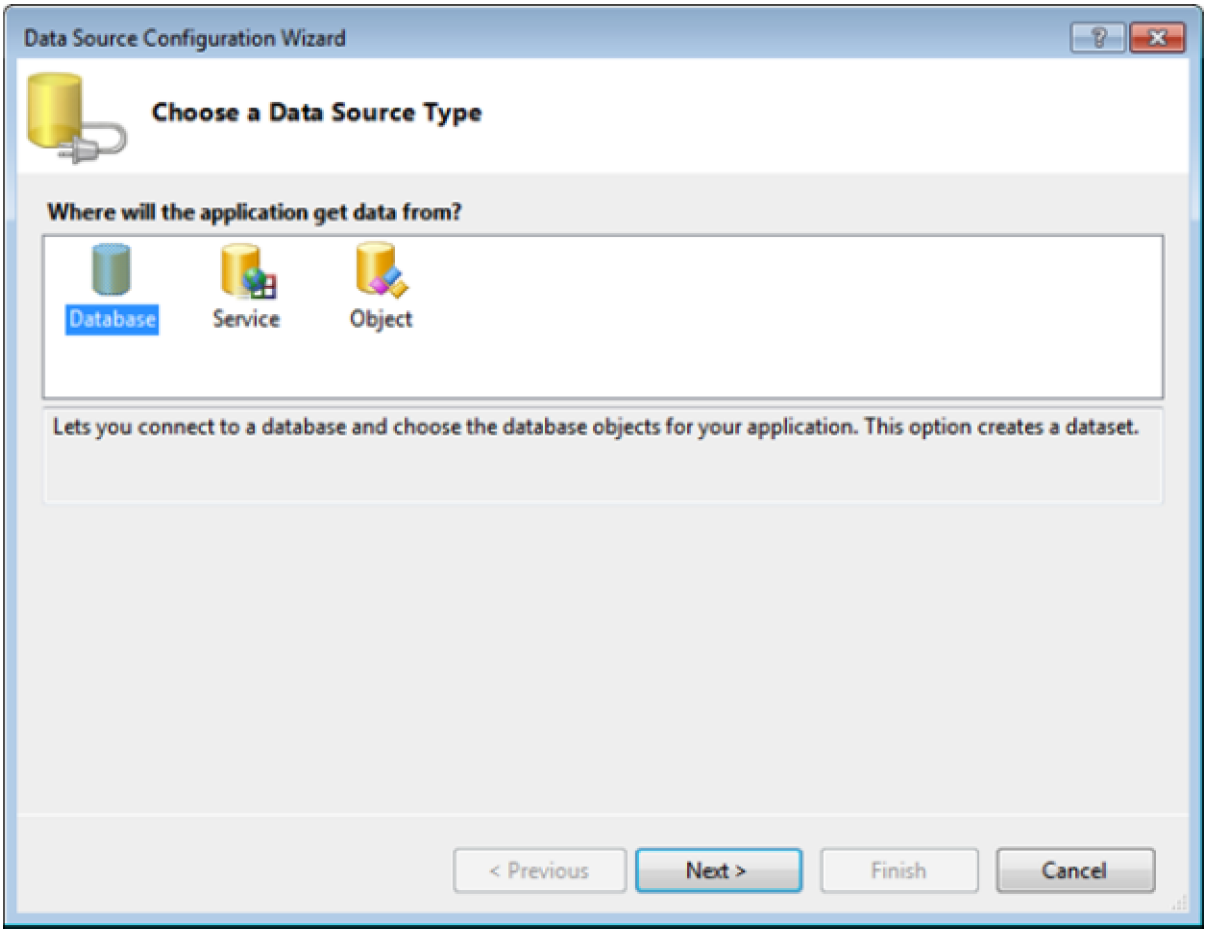
\Databinding\VB|CS\GettingStarted

1. In a new Windows Forms Application, drop a RadListControl on the default form.
2. Drop down the Properties window editor for the DataSource property and click the **Add Project Data Source...** link.



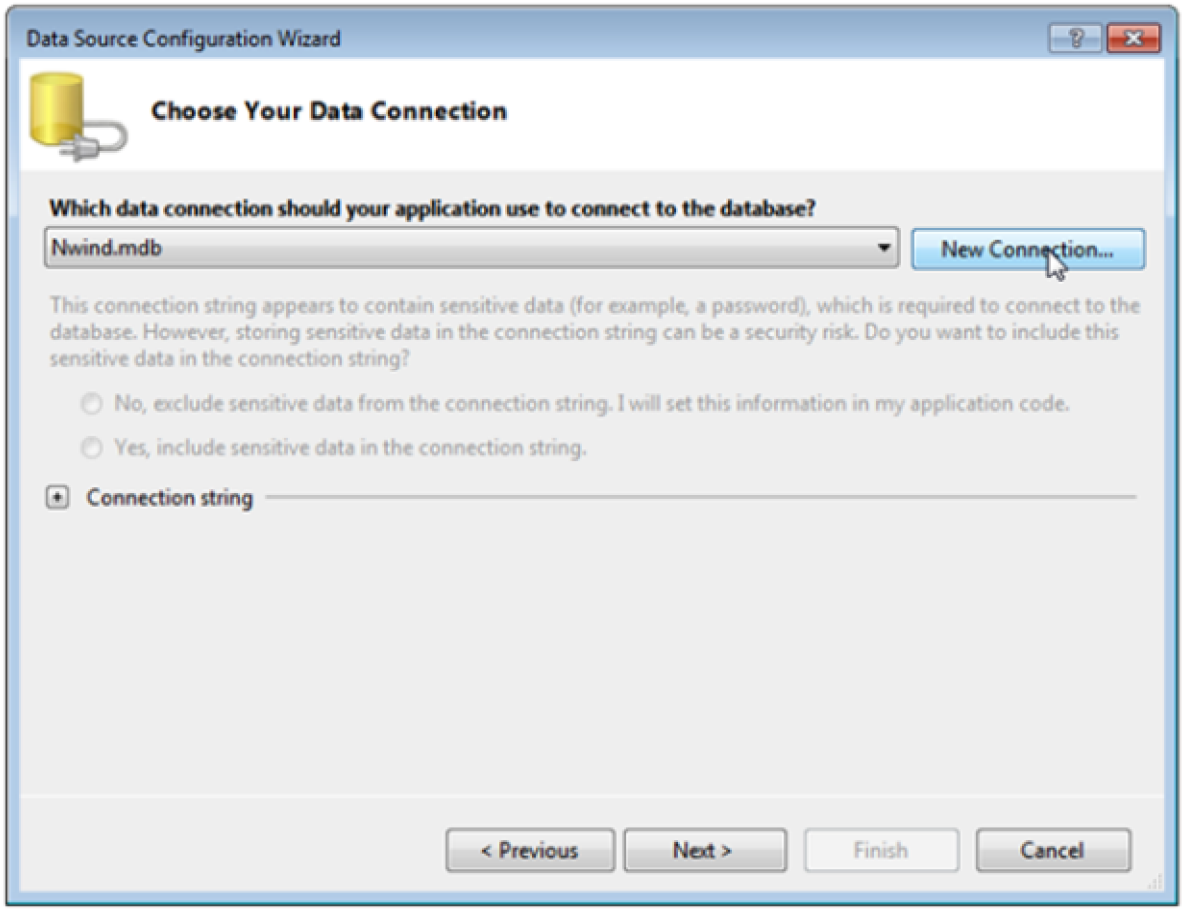
This step will display the Data Source Configuration Wizard where you can connect to database data, business objects or services.

3. In the Data Source Configuration Wizard "Choose a Data Source Type" page, select the Database icon and click **Next** to continue.

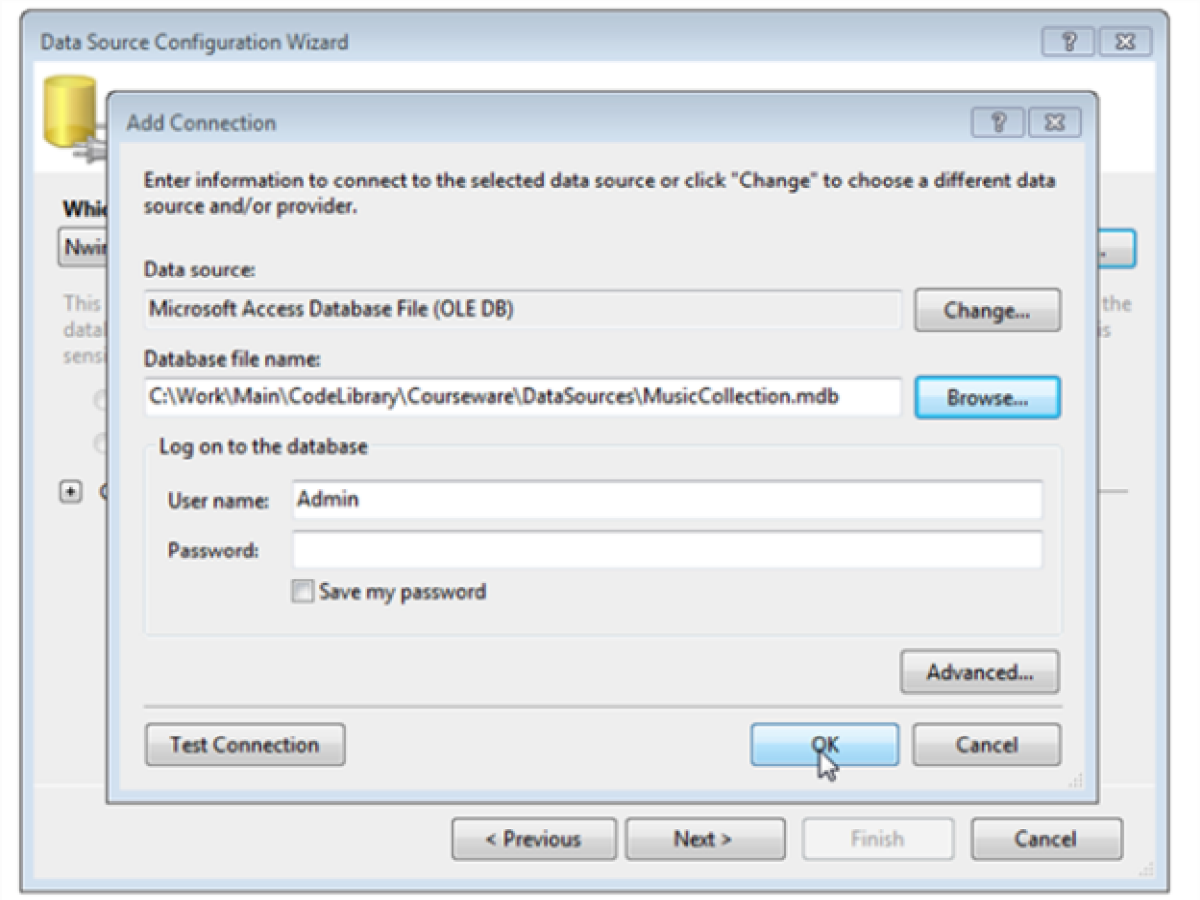


4. In the "Choose Your Data Connection" page, click the **New Connection** button.

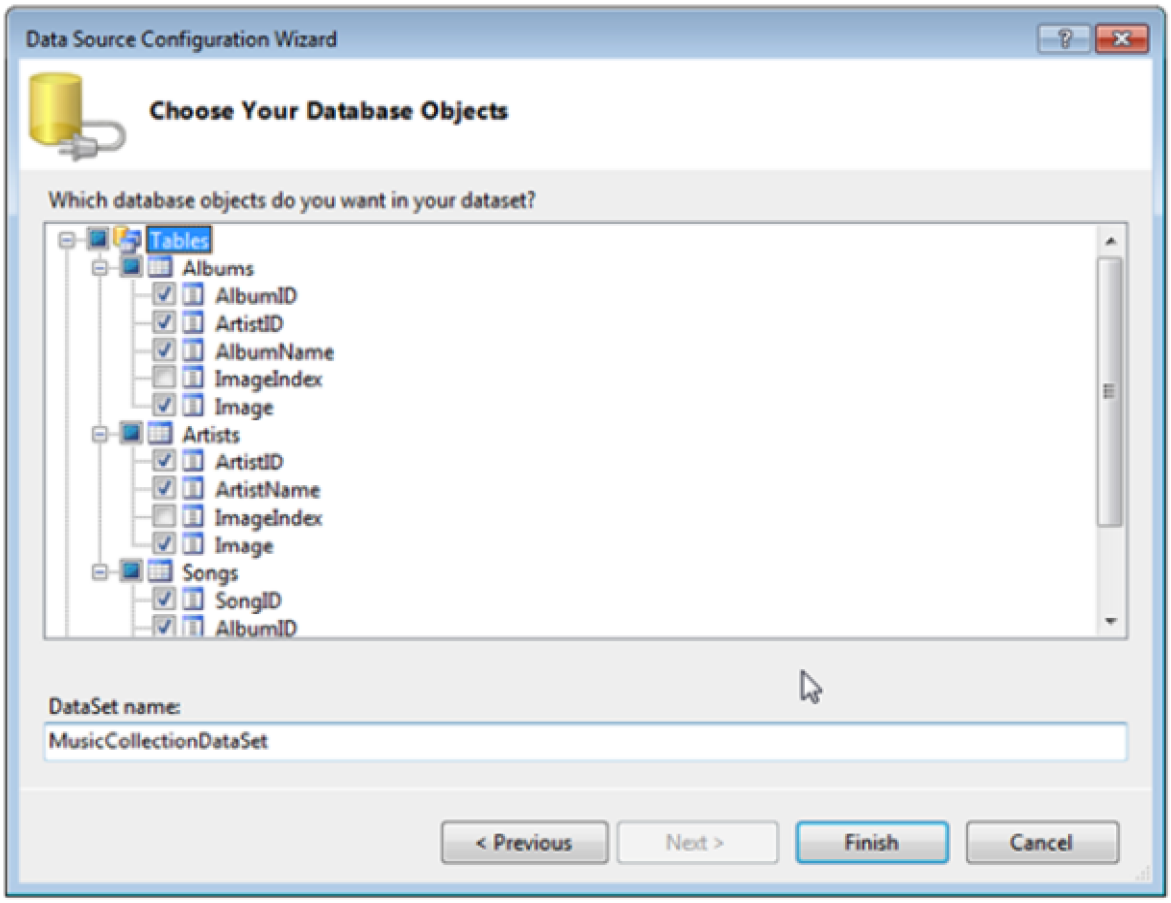
   This will display the Add Connection dialog. Note that you can also select an existing connection if you have already been through the configuration process and have a configuration that matches your needs. The Add Connection dialog will help us describe the kind of data (SQL Server, Access, etc.) and the location of the data. In the RadControls for WinForms installation directory you can find a series of Access (.mdb) files in the /Examples/Datasources subdirectory. You can just as easily connect to SQL Server or other enterprise level data.*



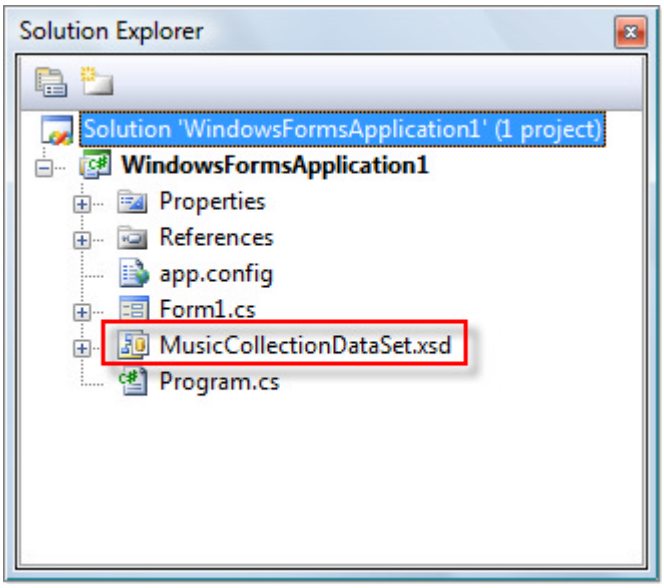
5. In the Add Connection dialog:
   - If the Data Source doesn't list "Microsoft Access Database File", then click the **Change** button, select it from the list in the Change Data Source dialog and click **OK**.
   - Click the **Browse...** button and navigate to the RadControls for WinForms installation directory, locate the \Examples\DataSources directory and select the MusicCollection.mdb file.
   - Click **OK** to close the Add Connection dialog.



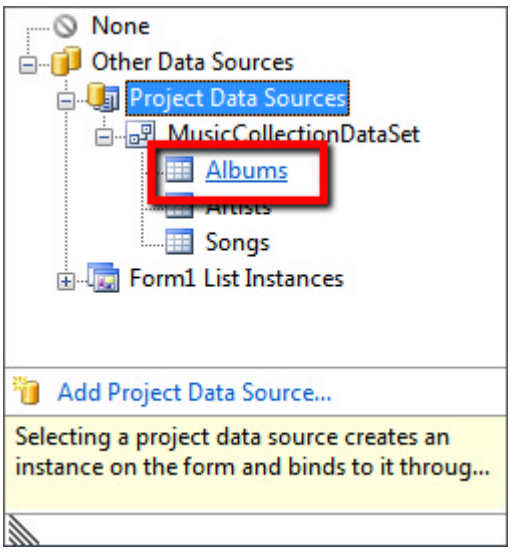
6. In the Data Source Configuration Wizard "Choose Your Data Connection" page, click **Next** to continue.
7. A dialog will display asking if you want to copy this file locally. Select **No** to reference the database file in its original location.
8. In the "Choose Database Objects" page of the wizard, open up the treeview and select the tables and fields you want to include in your data source.



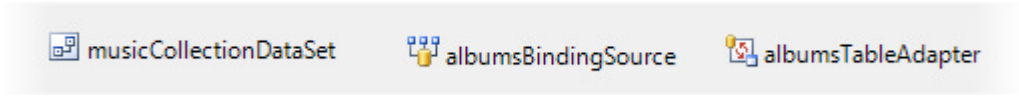
9. Click **Finish** to close the wizard. A DataSet object that encapsulates selected tables and fields is created automatically.



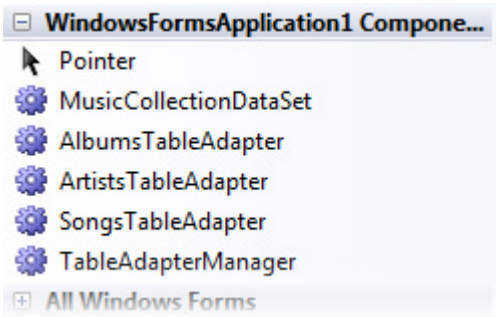
10. Back in the Properties window, locate the DataSource property for your control, drop down the list and select one of the tables as your data source.



This step will automatically create a series of components and drop them in your component tray: a DataSet component specific to the data you chose, a BindingSource and a TableAdapter to handle the lower level jobs of querying and updating your data.



C:\Users\stefanov\Desktop\databinding-getting-started006.png
This step also adds several components to the Toolbox:



A line of code is added to the form load event to populate the table with data:

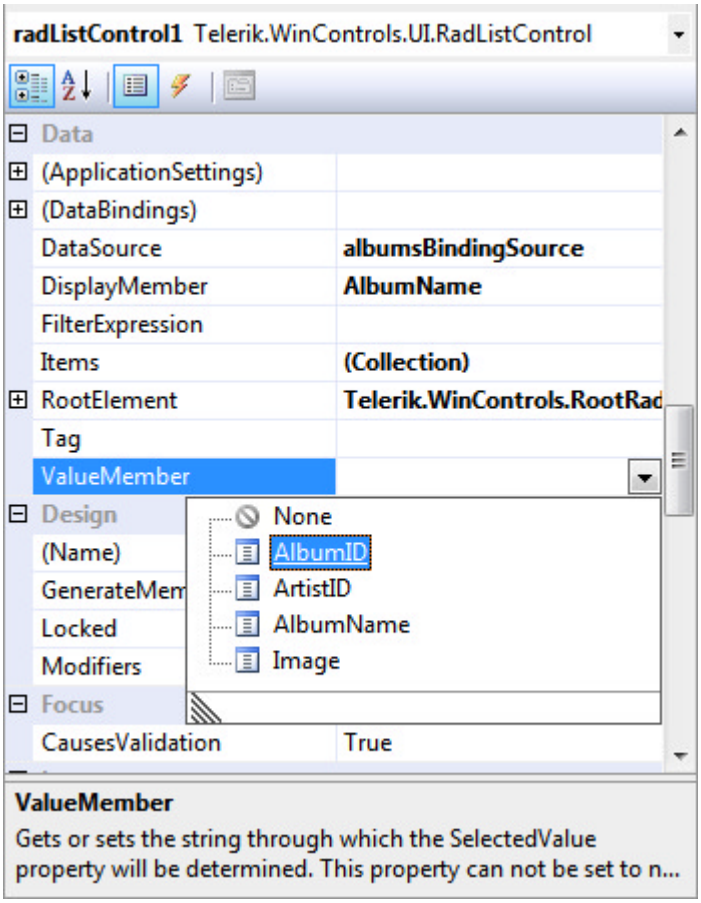
[VB] Automatically Generated Code

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 ' TODO: This line of code loads data into the 'musicCollectionDataSet.Albums' table. You
can move, or remove it, as needed.
 Me.albumsTableAdapter.Fill(Me.musicCollectionDataSet.Albums)
End Sub
```

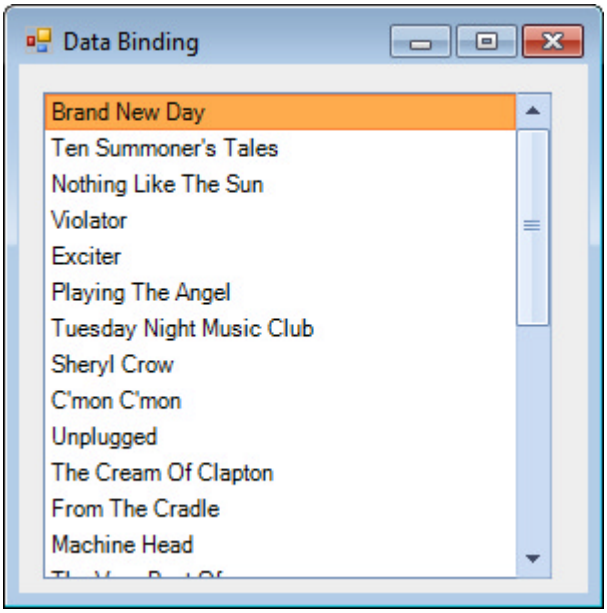
[C#] Automatically Generated Code

```
private void Form1_Load(object sender, EventArgs e)
{
 // TODO: This line of code loads data into the 'musicCollectionDataSet.Albums' table. Yo
can move, or remove it, as needed.
 this.albumsTableAdapter.Fill(this.musicCollectionDataSet.Albums);
}
```

11. Tell the bound control what columns you want to see. This step is more control-specific. For this example, we have a RadListControl and so you can click the DisplayMember property and drop down a list of elligible columns to display. DisplayMember populates the text for each item. The ValueMember property corresponds to the Value property of each item. Select "AlbumName" for the DisplayMember and "AlbumID" for the ValueMember.



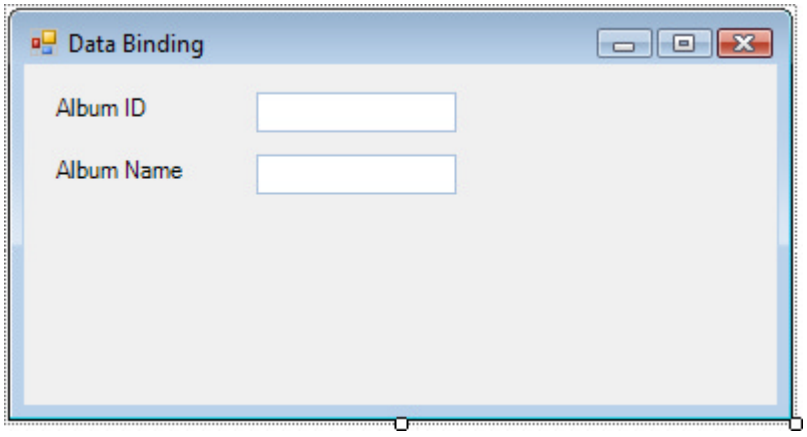
12. Run the application to view your data.



12.4 Binding to Simple Controls Using DataBindings

You can bind to any Control descendant using the DataBindings property. To make it work, add a binding with the control property to modify, the data source and the data element that will be bound to the control property. With simple data binding you can bind to any property of the object being used as a data source.

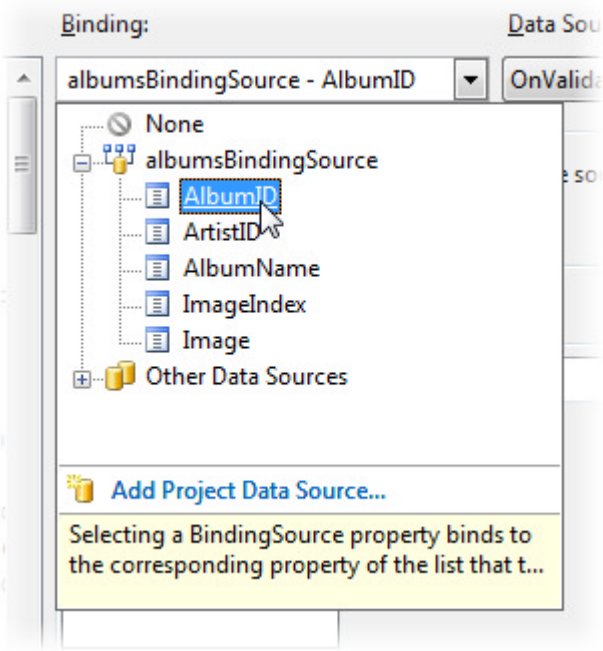
1. In a new Windows Forms Application, add controls to the default form with the following properties and roughly the arragement shown in the screenshot:
   - RadLabel: **Text** = "Album ID"
   - RadTextBox: **Name** = "tbAlbumID", **Text** = ""
   - RadLabel: **Text** = "Album Name"
   - RadTextBox: **Name** = "tbAlbumName", **Text** = "



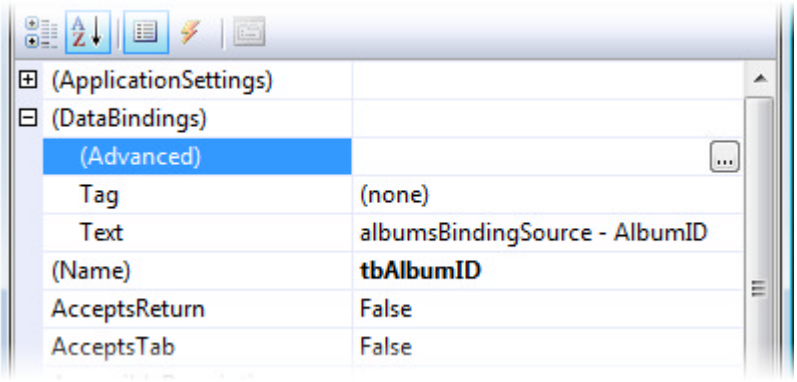
2. Bind the RadTextBox Text property:
   - Click the tbAlbumID RadTextBox, navigate to the Properties window and in the "(Databindings)" property (aways at the top of the list), click the ellipses for the Advanced property.

This displays the Advanced Formatting and Binding dialog that allows you to bind any property to a data source and to format the output.

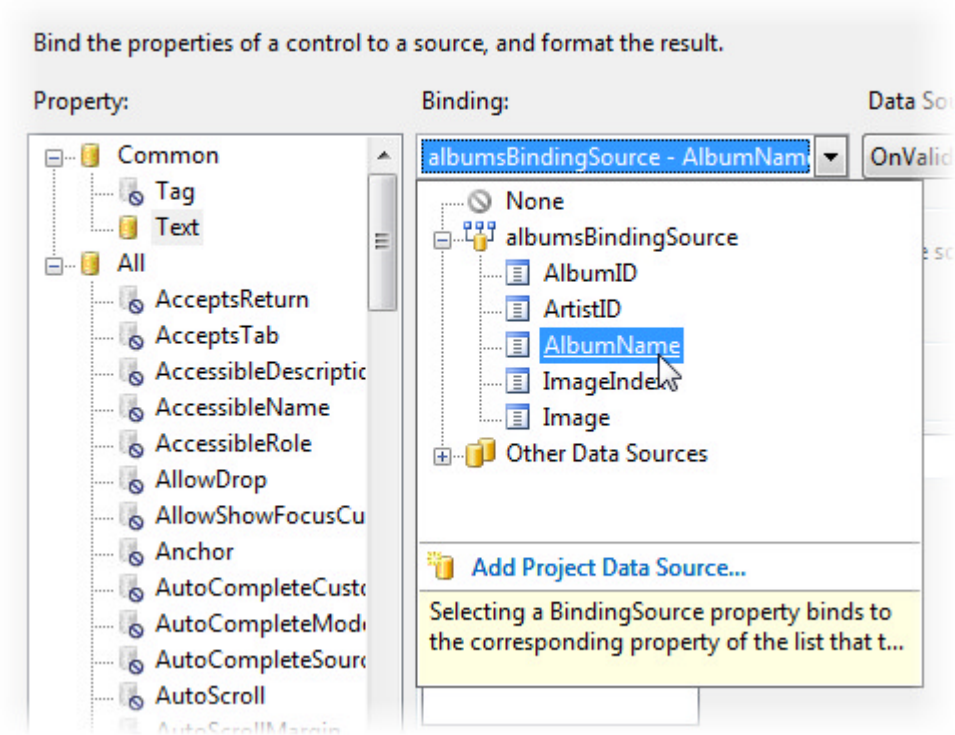
- In the Property list on the left of the dialog select the Text property. In the drop down under the label "Binding:", click the **Add Project DataSource...** link. This will bring up the Data Source Configuration wizard. Follow the steps in the "Getting Started" tutorial and configure the data source for all the columns in the Albums table. When you close the Data Source Configuration wizard, select the "Album" column of the "albumsBindingSource" as the data source.



- Click the **OK** button to close the Advanced Formatting and Binding dialog. Notice that the (DataBindings) property has the Text property assigned "albumsBindingSource - AlbumID". Scroll down and also take a look at the **Text** Property. It will be marked with a small database icon with a tool tip that describes the binding.



3. Use the DataBindings property to bind the tbAlbumName property to the "AlbumName" column of the database.



4. Press **Ctl-F5** to run the application. The data for the first record shows up in the bound text boxes.

12.5 Using BindingSource

BindingSource sits between your data source and UI controls. BindingSource provides a number of services:

- Centralized control for binding operations. Unlike CurrencyManager, BindingSource works for complex binding scenarios.

> BindingSource encapsulates and is a replacement for CurrrencyManager. CurrencyManager can still be used directly, but BindingSource is easier to use and more flexible.

- "Up-converts" non IBindingList data sources by copying elements into an internal IBindingList. This allows you to use data sources like SqlDataRead automatically.
- The IBindingList AddNew method can be extended for custom implementations, for example so that factory objects can create new items in the list.
- Supports type based binding. You can assign a type instead of an instance and BindingSource will handle instance creation automatically.
- Simplifies binding to web services.
- Supports editing operations and events, e.g. RemoveCurrent, EndEdit, CancelEdit, Add and the CurrentItemChanged event.
- Can be visualized and managed with the BindingNavigator (VCR like UI that allows navigation and editing).

In the earlier example "Introduction", only the RadDropDownList and RadGridView are synchronized. This is because all the other controls use a DataSet or DataTable directly, but RadDropDownList and RadGridView both use "albumsBindingSource", a BindingSource instance. If we hook up all the controls on the page with the BindingSource, all controls are completely synchronized automatically.

The example below takes the "Introduction" project, adds a BindingNavigator and points the DataSource property of all controls to "albumsBindingSource". The addition of the BindingNavigator just adds a UI control so

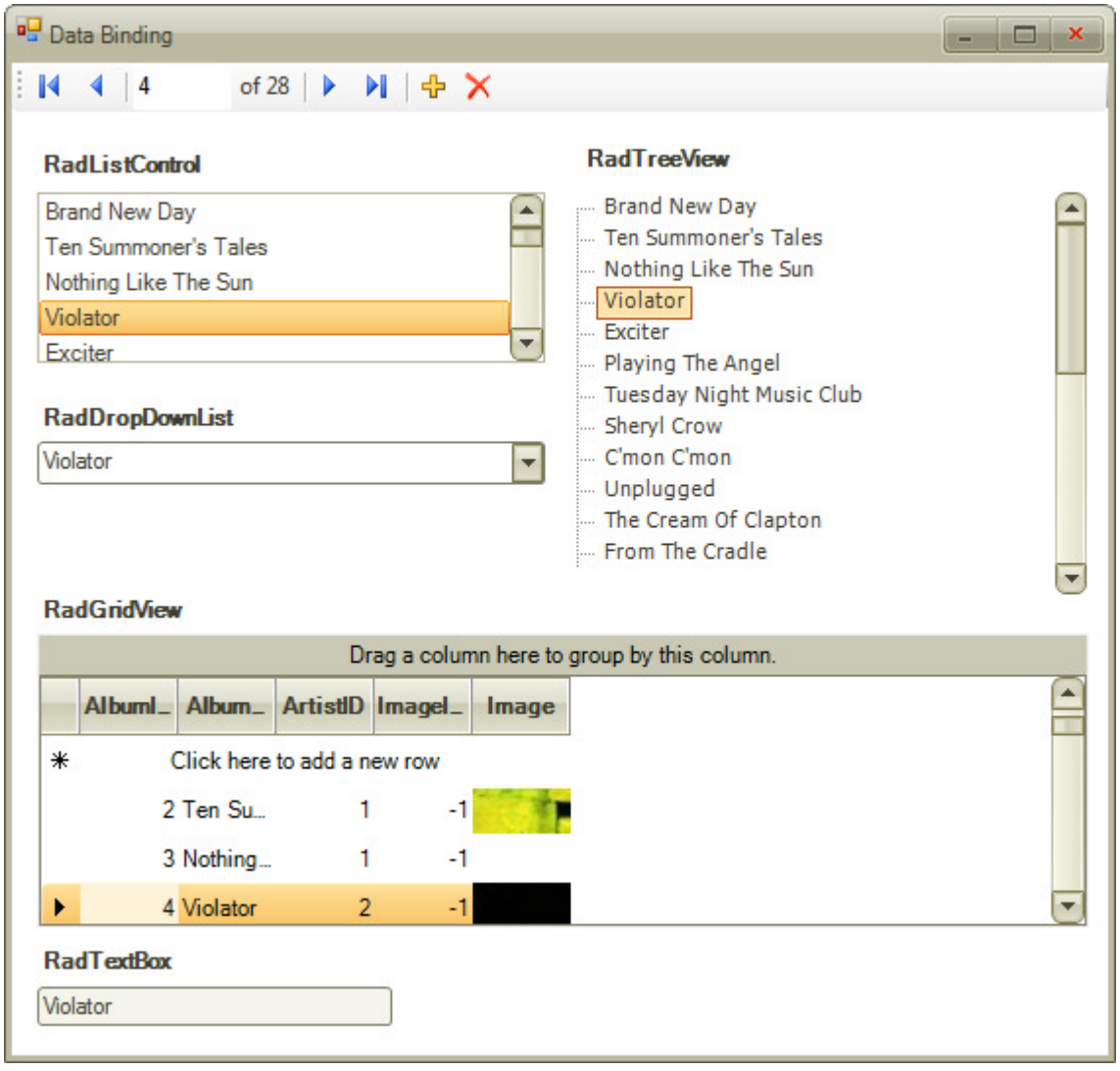
the user can navigate, add, edit and delete records.



You can find the complete source for this project at:

\DataBinding\VB|CS\BindingSource

Now when you run the project and select a record using any of the controls, including the BindingNavigator, all the controls respond as one. Notice in the screenshot below how "Tuesday Night Music Club" is selected and all the controls, including the RadTextBox, are at the same record.



12.6 Binding to Business Objects

You can use the BindingSource to bind collections of your own custom objects. For example, let's say we have a very simple Product business object with ID and Description properties. BindingSource has "type support" which means that it can bind to a type as a data source and it will create the infrastructure automatically in the background:

[VB] Assigning a Type to the DataSource

```
productBindingSource.DataSource = GetType(Product)
```

[C#] Assigning a Type to the DataSource

```
productBindingSource.DataSource = typeof(Product);
```

Then you can handle the BindingSource event **AddNew** to initialize new objects. Use the BindingSource **ResetBindings()** method to update the UI.

Binding to Custom Object Walk Through



You can find the complete source for this project at:

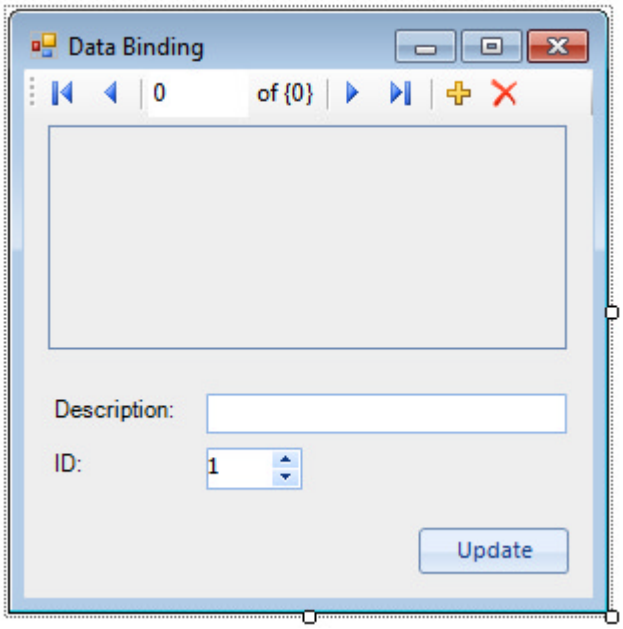
\DataBinding\VB|CS\BusinessObject

1. In a new Windows Forms Application, add a **BindingSource** component to the default form.



In this example we will set all the BindingSource properties in code, but you can also use the Data Source Configuration Wizard (initiated from the DataSource property editor) to point at a custom object and create the BindingSource for you.

2. Add the following controls to the default form arranged like the screenshot below and set properties:
   - BindingNavigator: **BindingSource** = <the binding source component you just added>
   - RadListControl: **Name** = "lbProducts"
   - RadLabel: **Text** = "Description"
   - RadTextBox: **Name** = "tbDescription"
   - RadLabel: **Text** = "ID"
   - RadSpinEditor: **Name** = "seID"
   - RadButton: **Name** = "btnUpdate"



3. Add a new class "Product.cs" to the project and define the Product class.

[VB] Defining the Product Class

```
Public Class Product
 Implements INotifyPropertyChanged
 Public Sub New()
 End Sub
 Public Sub New(id As Integer, description As String)
  Me.id = id
  Me.description = description
 End Sub
 Private id As Integer
 Public Property ID() As Integer
  Get
   Return Me.id
  End Get
  Set
   Me.id = value
   NotifyPropertyChanged("ID")
  End Set
 End Property
 Private description As String
 Public Property Description() As String
  Get
   Return Me.description
  End Get
  Set
   Me.description = value
   NotifyPropertyChanged("Description")
  End Set
 End Property
 #region INotifyPropertyChanged Members
 Public Event PropertyChanged As PropertyChangedEventHandler
 Private Sub NotifyPropertyChanged(info As [String])
  OnPropertyChanged(New PropertyChangedEventArgs(info))
 End Sub
 Protected Overridable Sub OnPropertyChanged(e As PropertyChangedEventArgs)
  If PropertyChanged <> Nothing Then
   PropertyChanged(Me, New PropertyChangedEventArgs(e.PropertyName))
  End If
 End Sub
#End Region
End Class
```

[C#] Defining the Product Class

```
public class Product : INotifyPropertyChanged
{
 public Product()
 {
 }
 public Product(int id, string description)
 {
   this.id = id;
   this.description = description;
 }
 private int id;
```

```
 public int ID
 {
   get { return this.id; }
   set
   {
     this.id = value;
     NotifyPropertyChanged("ID");
   }
 }
 private string description;
 public string Description
 {
   get { return this.description; }
   set
   {
     this.description = value;
     NotifyPropertyChanged("Description");
   }
 }
 #region INotifyPropertyChanged Members
 public event PropertyChangedEventHandler PropertyChanged;
 private void NotifyPropertyChanged(String info)
 {
   OnPropertyChanged(new PropertyChangedEventArgs(info));
 }
 protected virtual void OnPropertyChanged(PropertyChangedEventArgs e)
 {
   if (PropertyChanged != null)
   {
     PropertyChanged(this, new PropertyChangedEventArgs(e.PropertyName));
   }
 }
 #endregion
}
```

4. Double-click the form to create a Load event handler. Add the code below.

Notice how the code below assigns a Type and not an instance to the BindingSource DataSource. The BindingSource will take care of providing the underlying IBindingList used to store the actual objects.

[VB] Handling the Form Load Event

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 ' assign the datasource the Product type
 bindingSource1.DataSource = GetType(Product)
 ' respond to new and changed items
 bindingSource1.AddingNew += New AddingNewEventHandler(bindingSource1_AddingNew)
 ' bind the list control
 lbProducts.DisplayMember = "Description"
 lbProducts.ValueMember = "ID"
 lbProducts.DataSource = bindingSource1
 ' bind text box and spin editor
 tbDescription.DataBindings.Add("Text", bindingSource1, "Description")
 seID.DataBindings.Add("Value", bindingSource1, "ID")
End Sub
```

[C#] Handling the Form Load Event

```
private void Form1_Load(object sender, EventArgs e)
{
 // assign the datasource the Product type
 bindingSource1.DataSource = typeof(Product);
 // respond to new and changed items
 bindingSource1.AddingNew += new AddingNewEventHandler(bindingSource1_AddingNew);
 // bind the list control
 lbProducts.DisplayMember = "Description";
 lbProducts.ValueMember = "ID";
 lbProducts.DataSource = bindingSource1;
 // bind text box and spin editor
 tbDescription.DataBindings.Add("Text", bindingSource1, "Description");
 seID.DataBindings.Add("Value", bindingSource1, "ID");
}
```

5. Add the event handler to create instances of new objects.

   In the AddingNew event handler you create and assign the NewObject property of the AddingNewEventArgs parameter. In this particular handler we're iterating the existing products and getting the maximum ID value to use as a starting point.

   [VB] Handling the AddingNew Event

```
Sub bindingSource1_AddingNew(sender As Object, e As AddingNewEventArgs)
 ' look through existing products and get the maximum ID.
 Dim maxID As Integer = 0
 For Each product As Product In (TryCast(bindingSource1.List, BindingList(Of Product)))
  maxID = Math.Max(maxID, product.ID)
 Next
 ' create and assign new product with the maxID + 1 and a unique description
 e.NewObject = New Product(System.Threading.Interlocked.Increment(maxID), "New Product "
maxID.ToString())
End Sub
```

   [C#] Handling the AddingNew Event

```
void bindingSource1_AddingNew(object sender, AddingNewEventArgs e)
{
 // look through existing products and get the maximum ID.
 int maxID = 0;
 foreach (Product product in (bindingSource1.List as BindingList<Product>))
 {
   maxID = Math.Max(maxID, product.ID);
 }
 // create and assign new product with the maxID + 1 and a unique description
 e.NewObject = new Product(++maxID, "New Product " + maxID.ToString());
}
```

6. Back in design mode, double-click the RadButton to create a Click event handler that will also update the user interface on demand.

   [VB] Handling the Click Event

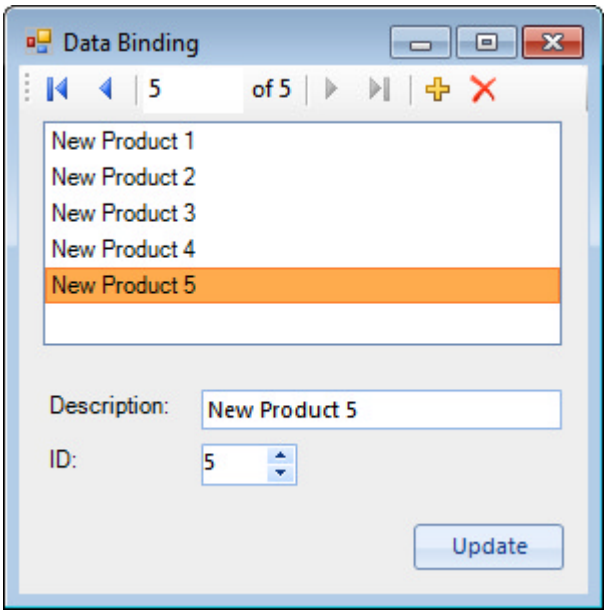
```
Private Sub btnUpdate_Click(sender As Object, e As EventArgs)
 bindingSource1.ResetBindings(False)
End Sub
```

   [C#] Handling the Click Event

```
private void btnUpdate_Click(object sender, EventArgs e)
```

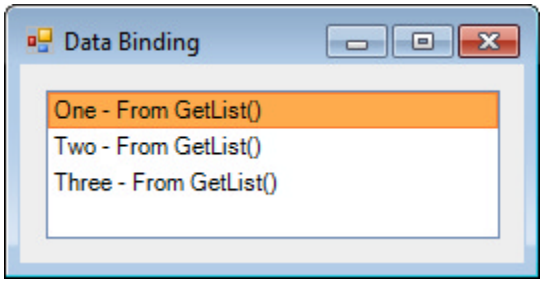
```
{
  bindingSource1.ResetBindings(false);
}
```

7. Press **Ctl-F5** to run the application. Add several new records using the navigator bar. Navigate between records, edit the Description and ID and try using the Update button. Also try editing and moving to other records without clicking the Update button.



12.7 Binding to Services

Windows Communication Foundation (WCF) is a Microsoft programming model used to handle communications in distributed application environments. This walk-through example shows you how to build a basic WCF service and retrieve a list of objects that are then bound to a RadListControl.



The example shown here will be vastly simplified from a production level application in that the service will exist on the same machine as the WinForms application and in fact will be in the same Visual Studio solution and application domain. The example does not delve into the intricacies of WCF. Please refer to MSDN and other online sources for more about WCF.

Also know that the paradigm used here (create a service, reference a service, use a service client to call service methods and return data) will all work similarly if instead you substitute the older standard web service instead of a WCF service.

You can find the complete source for this project at:



\DataBinding\VB|CS\BindingToServices

1. Create a new Windows Forms Application.
2. In the Solution Explorer, right-click the solution and select **Add | New Project** from the context menu. Select the WCF Service Library project type, name the project "BindingToServices" and click **OK** to create the new service. This step will create a IService1.cs file that will contain the IService1 interface "contract" and Service.cs that will contain the implementation for the service methods.
3. In the code for IService1.cs, add a method called "GetList()" that will supply the data.

   Notice the "CompositeType" class defined further down in the IService1.cs file. GetList() will return a generic list of CompositeType.

   [VB] Adding the GetList() Method

```
<OperationContract> _
Function GetList() As List(Of CompositeType)
End Function
```

   [C#] Adding the GetList() Method

```
[OperationContract]
List<CompositeType> GetList();
```

4. In the code for Service1.cs, add a GetList() method implementation to the Service1 class.

   The method creates several "CompositeType" instances, adds them to the list, then returns the list.

   WCF will automatically convert the GetList() generic list to an array of CompositeType. List<> is specific to .NET and using a simple array makes the method results accesible by a wider audience of clients.

   [VB] Adding the GetList() Implementation

```
Public Function GetList() As List(Of CompositeType)
 Dim list As New List(Of CompositeType)()
 Dim composite1 As New CompositeType()
 composite1.StringValue = "One - From GetList()"
 list.Add(composite1)
 Dim composite2 As New CompositeType()
 composite2.StringValue = "Two - From GetList()"
 list.Add(composite2)
 Dim composite3 As New CompositeType()
 composite3.StringValue = "Three - From GetList()"
 list.Add(composite3)
 Return list
End Function
```

   [C#] Adding the GetList() Implementation

```
public List<CompositeType> GetList()
{
 List<CompositeType> list = new List<CompositeType>();
 CompositeType composite1 = new CompositeType();
 composite1.StringValue = "One - From GetList()";
 list.Add(composite1);
 CompositeType composite2 = new CompositeType();
 composite2.StringValue = "Two - From GetList()";
```

```
 list.Add(composite2);
 CompositeType composite3 = new CompositeType();
 composite3.StringValue = "Three - From GetList()";
 list.Add(composite3);
 return list;
}
```

5. In the Solution Explorer, locate the Windows Forms application and open the References node. Right-click and select **Add Service Reference...** from the context menu.

   This step will automatically create a client wrapper class that lets you easily access the types and methods of the service.

6. Add a RadListControl to the default form for the Windows Forms application.
7. Double-click the form to create a Load event handler.
8. Add a reference "BindingToServices.ServiceReference1" to the "Imports" (VB) or "uses" (C#) section of code.

   The first part of the name you reference here will be the solution name, i.e. the default namespace.

9. Add code to the form's Load event handler to create an instance of the service client.

   [VB] Getting Data From the Service Client

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 Dim client As New Service1Client()

  radListControl1.DataSource = client.GetList()
  radListControl1.DisplayMember = "StringValue"
End Sub
```

   [C#] Getting Data From the Service Client

```
private void Form1_Load(object sender, EventArgs e)
{
 Service1Client client = new Service1Client();

 radListControl1.DataSource = client.GetList();
 radListControl1.DisplayMember = "StringValue";
}
```

10. Press **F5** to run the application. The WCF service will be hosted automatically.

12.8 Binding to LINQ Data Sources

Language-Integrated Query (LINQ) is a set of extensions to the .NET Framework that allow you to use query expressions as an integrated feature of your primary programming language. This allows you work against all kinds of data using a single set of mechanisms; .NET native objects, SQL relational data and XML data all can be traversed, filtered and minipulated with LINQ expressions.

Here we will show how BindingSource can consume output from LINQ statements. Let's start with our earlier example of a list of products and assign to a RadListBox, without any LINQ operations. The Product object will be defined with an integer "ID", string "Description" and double "Price".



LINQ is a wide and deep subject. The purpose here is to show how LINQ results can be used as data sources for RadControls. Please read the Microsoft Developers Network (MSDN) pages for further background on LINQ.

[VB] Starting Point Without LINQ

```
Dim products As New BindingList(Of Product)()
products.Add(New Product(1, "Coffee", 1.4))
products.Add(New Product(2, "Latte", 2.4))
products.Add(New Product(3, "Mocha", 2.7))
products.Add(New Product(4, "Espresso", 2.5))
products.Add(New Product(5, "Tea", 1.3))
products.Add(New Product(6, "Chai", 1.5))
products.Add(New Product(7, "Cafe au Lait", 1.5))
radListControl1.DataSource = products
radListControl1.DisplayMember = "Description"
radListControl1.ValueMember = "ID"
```

[C#] Starting Point Without LINQ

```
BindingList<Product> products = new BindingList<Product>();
products.Add(new Product(1, "Coffee", 1.40));
products.Add(new Product(2, "Latte", 2.40));
products.Add(new Product(3, "Mocha", 2.70));
products.Add(new Product(4, "Espresso", 2.50));
products.Add(new Product(5, "Tea", 1.30));
products.Add(new Product(6, "Chai", 1.50));
products.Add(new Product(7, "Cafe au Lait", 1.50));
radListControl1.DataSource = products;
radListControl1.DisplayMember = "Description";
radListControl1.ValueMember = "ID";
```

Now we can add a LINQ statement to select and sort products using any number of criteria, for example:

[VB] Using a LINQ Statement Against Products List

```
Dim productQuery As IEnumerable(Of Product) = From product In products _
 Where product.Description.StartsWith("C") _
 Where product.Price > 1.49 _
 Order By product.Description _
 Select product
```

[C#] Using a LINQ Statement Against Products List

```
IEnumerable<Product> productQuery = from product in products
                                    where product.Description.StartsWith("C")
                                    where product.Price > 1.49
                                    orderby product.Description
                                    select product;
```

The returned productQuery can be iterated to show the "Chai" and "Cafe au Lait" products that survived the filtering process.

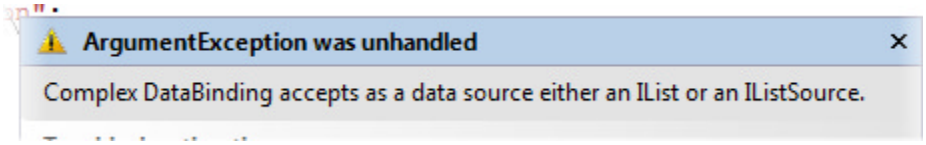
[VB] Iterating the LINQ Statement Results

```
For Each product As Product In productQuery
 RadMessageBox.Show(product.Description)
Next
```

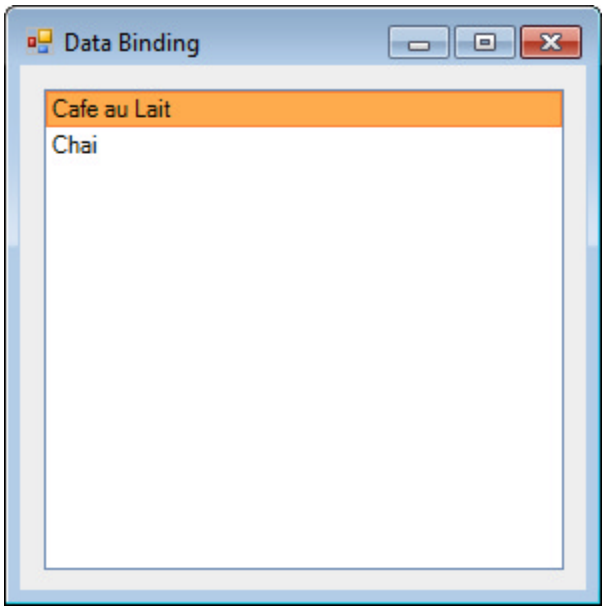
[C#] Iterating the LINQ Statement Results

```
foreach (Product product in productQuery)
 RadMessageBox.Show(product.Description);
```

But can I assign productQuery as a DataSource? No, not directly. The results are a little too low-level to use without help. You would get an error if you attempted to assign productQuery to the RadListBox DataSource:



This is where BindingSource comes to the rescue again. It creates an IList behind-the-scenes and loads the IEnumerable based results into the list.



Now the code has the intermediate step of creating the BindingSource. In the constructor, pass the LINQ results. The second constructor argument is the data member and can be left blank.

[VB] Using BindingSource With LINQ Results

```
Dim products As New BindingList(Of Product)()
products.Add(New Product(1, "Coffee", 1.4))
products.Add(New Product(2, "Latte", 2.4))
products.Add(New Product(3, "Mocha", 2.7))
products.Add(New Product(4, "Espresso", 2.5))
products.Add(New Product(5, "Tea", 1.3))
products.Add(New Product(6, "Chai", 1.5))
products.Add(New Product(7, "Cafe au Lait", 1.5))
Dim productQuery As IEnumerable(Of Product) = From product In products _
 Where product.Description.StartsWith("C") _
 Where product.Price > 1.49 _
 Order By product.Description _
 Select product
radListControl1.DataSource = New BindingSource(productQuery, "")
radListControl1.DisplayMember = "Description"
radListControl1.ValueMember = "ID"
```

[C#] Using BindingSource With LINQ Results

```
BindingList<Product> products = new BindingList<Product>();
products.Add(new Product(1, "Coffee", 1.40));
products.Add(new Product(2, "Latte", 2.40));
products.Add(new Product(3, "Mocha", 2.70));
products.Add(new Product(4, "Espresso", 2.50));
products.Add(new Product(5, "Tea", 1.30));
```

```
products.Add(new Product(6, "Chai", 1.50));
products.Add(new Product(7, "Cafe au Lait", 1.50));
IEnumerable<Product> productQuery = from product in products
                                    where product.Description.StartsWith("C")
                                    where product.Price > 1.49
                                    orderby product.Description
                                    select product;
radListControl1.DataSource = new BindingSource(productQuery, "");
radListControl1.DisplayMember = "Description";
radListControl1.ValueMember = "ID";
```



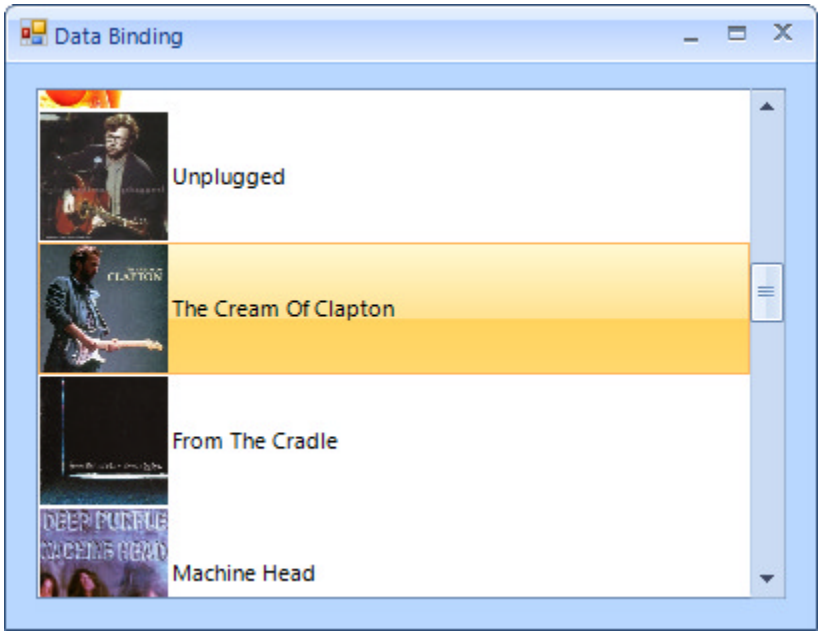
You can find the complete source for this project at:

\DataBinding\VB|CS\BindingToLINQ

12.9 Binding Image Data

An image in a database is typically stored as an array of bytes, so the first step is to write these bytes to a MemoryStream and then use the stream to populate a Bitmap or Image object. There's a special case problem with some data. The image data may have OLE DB header bytes that make the data unusable as-is. What we need is a method that will return an image from a byte array and that will ignore the OLE DB header if it is present. We just happen to have a method that does just that.

This walk-through builds on the "Getting Started" project and will display the album cover graphics in a RadListBox.





You can find the complete source for this project at:

\DataBinding\VB|CS\BindingImages

1. Start with the "Getting Started" project or a copy.
2. In the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu, then select **Telerik RadForm** and click the **Add** button.

3. In the designer, copy the MusicCollectionDataSet, AlbumsBindingSource and AlbumsTableAdapter from the original Form1 and paste all three to the new RadForm.
4. Set the RadForm **Anchor** property to "Top, Bottom, Left, Right".
5. Copy the **RadListControl** from Form1 to the new RadForm. The DataSource and DisplayMember properties should still be populated to "albumsBindingSource" and "AlbumName", respectively.
6. Set the list control **AutoSizeItems** property to true.
7. Double-click the form to create a new Load event handler. Add the following code to populate the Albums table:

[VB] Handling the Load Event

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 Me.albumsTableAdapter.Fill(Me.musicCollectionDataSet.Albums)
End Sub
```

[C#] Handling the Load Event

```
private void RadForm1_Load(object sender, EventArgs e)
{
 this.albumsTableAdapter.Fill(this.musicCollectionDataSet.Albums);
}
```

8. Add "Telerik.WinControls.UI" and "System.IO" namespace references to the "Imports" (VB) or "uses" (C#) section of code.
9. Add two helper methods that will convert an array of bytes to an Image object.

The HasOleContainerHeader() method detects if the byte array contains the OLE DB header. The GetImageFromData() method converts the byte array to an image and uses HasOleContainer() to determine which bytes to copy - the whole array or just the array without the header bytes.

[VB] Adding Methods to Detect OLE DB Headers and to Convert Bytes to an Image

```
' return true if the byte array has an OLE DB header
Private Function HasOleContainerHeader(imageByteArray As Byte()) As Boolean
 Const OleByte0 As Byte = 21
 Const OleByte1 As Byte = 28
 Return (imageByteArray(0) = OleByte0) AndAlso (imageByteArray(1) = OleByte1)
End Function
' convert the image byte array to a memory stream, then a bitmap
Private Function GetImageFromData(imageData As Byte()) As Image
 Const OleHeaderLength As Integer = 78
 Dim memoryStream As New MemoryStream()
 ' if the byte array has a OLE DB header, leave the header out of the copy
 If HasOleContainerHeader(imageData) Then
  memoryStream.Write(imageData, OleHeaderLength, imageData.Length - OleHeaderLength)
 Else
  memoryStream.Write(imageData, 0, imageData.Length)
 End If
 ' convert to Bitmap/Image
 Return New Bitmap(memoryStream)
End Function
```

[C#] Adding Methods to Detect OLE DB Headers and to Convert Bytes to an Image

```
// return true if the byte array has an OLE DB header
private bool HasOleContainerHeader(byte[] imageByteArray)
{
 const byte OleByte0 = 21;
```

```
 const byte OleByte1 = 28;
 return (imageByteArray[0] == OleByte0) && (imageByteArray[1] == OleByte1);
}
// convert the image byte array to a memory stream, then a bitmap
private Image GetImageFromData(byte[] imageData)
{
 const int OleHeaderLength = 78;
 MemoryStream memoryStream = new MemoryStream();
 // if the byte array has a OLE DB header, leave the header out of the copy
 if (HasOleContainerHeader(imageData))
 {
   memoryStream.Write(imageData, OleHeaderLength, imageData.Length - OleHeaderLength);
 }
 else
 {
   memoryStream.Write(imageData, 0, imageData.Length);
 }
 // convert to Bitmap/Image
 return new Bitmap(memoryStream);
}
```

10. In the Property window, navigate to the Events tab and locate the RadListControl ItemDataBound event and double-click to create an event handler. Add the code below.rad

    The ItemDataBound event fires as each item is bound and passes a ListItemDataBoundEventArgs that contain a reference to the item being bound. The DataBoundItem of the NewItem is cast to a DataRowView and it's Row property cast to AlbumsRow. Now that we have a typed row, we can directly access its columns by name, i.e. row.Image. Later in the event handler, call GetImageFromData() to return an Image. Notice that we call the Image object's GetThumbnailImage() method to scale the image down. Here we set the dimensions to be 64 x 64 pixels. The last two parameters of GetThumbnailImage() are used for callbacks and not really necessary for our purposes.

    [VB] Handling the ItemDataBound Event

```
Private Sub radListControl1_ItemDataBound(ByVal sender As System.Object, ByVal args As
Telerik.WinControls.UI.ListItemDataBoundEventArgs) Handles radListControl1.ItemDataBound

' retrieve the data row that was bound

Dim row As MusicCollectionDataSet.AlbumsRow = (TryCast((TryCast(args.NewItem.DataBoundIte
DataRowView)).Row, MusicCollectionDataSet.AlbumsRow))

' retrieve the item that was bound to and set properties

Dim item As RadListDataItem = TryCast(args.NewItem, RadListDataItem)

item.TextImageRelation = TextImageRelation.ImageBeforeText

' assign the image as a thumbnail

item.Image = GetImageFromData(row.Image).GetThumbnailImage(64, 64, Nothing, New IntPtr(),

End Sub
```

    [C#] Handling the ItemDataBound Event

```
private void radListControl1_ItemDataBound(object sender, ListItemDataBoundEventArgs args
{
// retrieve the data row that was bound
MusicCollectionDataSet.AlbumsRow row = ((args.NewItem.DataBoundItem as DataRowView).Row a
MusicCollectionDataSet.AlbumsRow);
// retrieve the item that was bound to and set properties
```

```
RadListDataItem item = args.NewItem as RadListDataItem;
//item.VisualItem.Padding = new Padding(5);
item.TextImageRelation = TextImageRelation.ImageBeforeText;
// assign the image as a thumbnail
item.Image = GetImageFromData(row.Image).GetThumbnailImage(64, 64, null, new IntPtr());
}
```

11. Locate Program.cs in the Solution Explorer and open it. Change the call to Application.Run() statement to run RadForm1 instead of Form1.
12. Press **Ctrl-F5** to run the application.

12.10 Binding RadGridView

RadGridView is such a control that is used in mops business applications that we should take a walkthrough of binding the grid to database data. This example will demonstrate binding to a single database table. See the upcoming chapter "Grid" for more information on binding to hierarchical data and binding to Dynamic LINQ data.

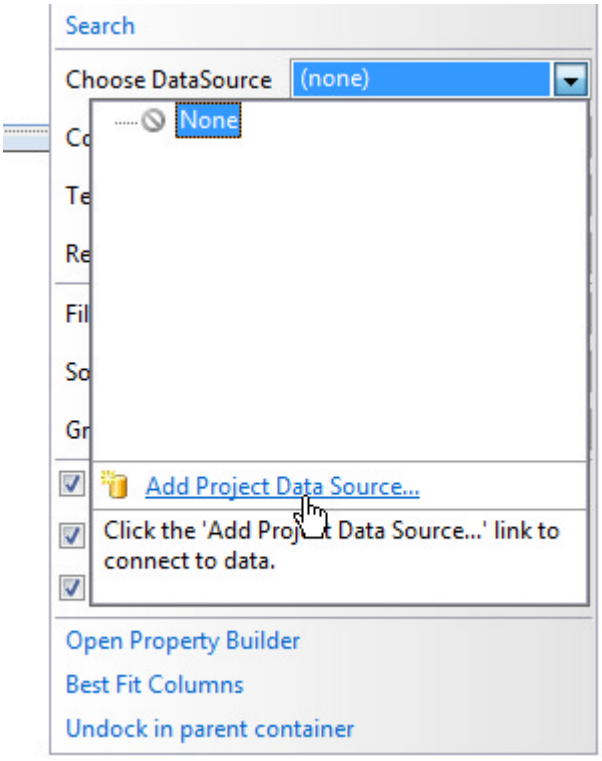


You can find the complete source for this project at:

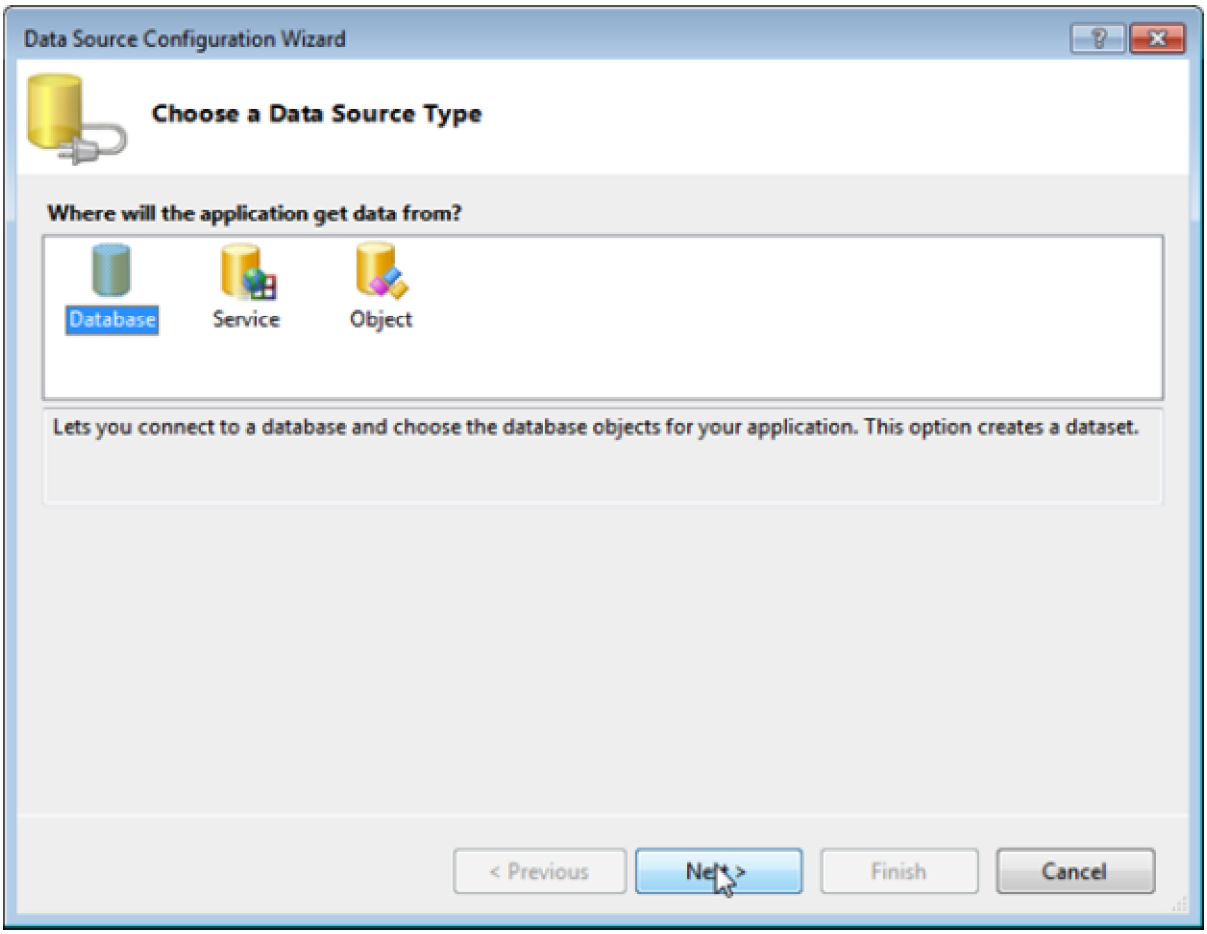
\Databinding\VB|CS\GridView

Please note that this example assumes that you have installed Northwind database on your local sql server.

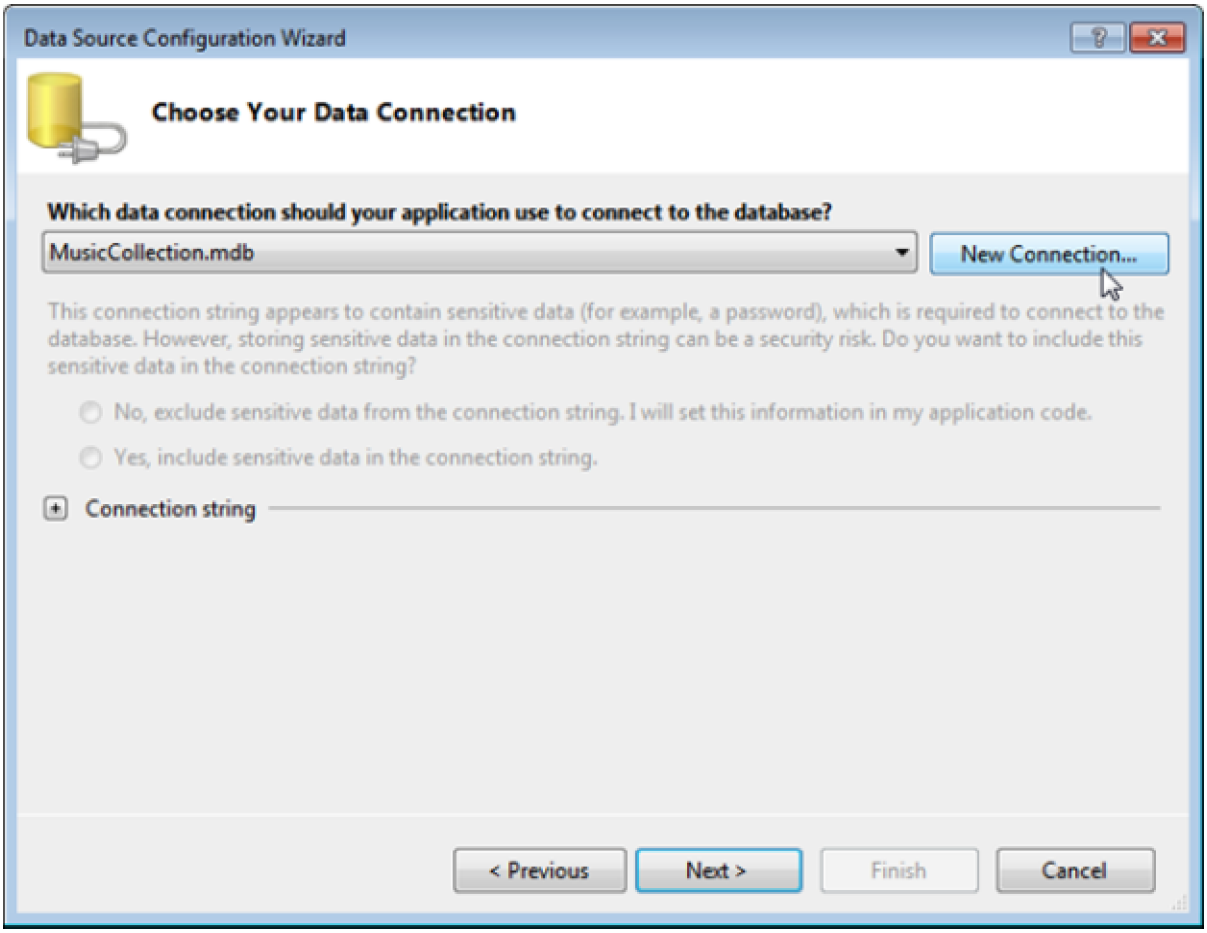
1. In a new Windows Forms Application, drop a **RadGridView** on the default form.
2. From the grid Smart Tag, drop down the **Choose DataSource** list and select **Add Project Data Source...**



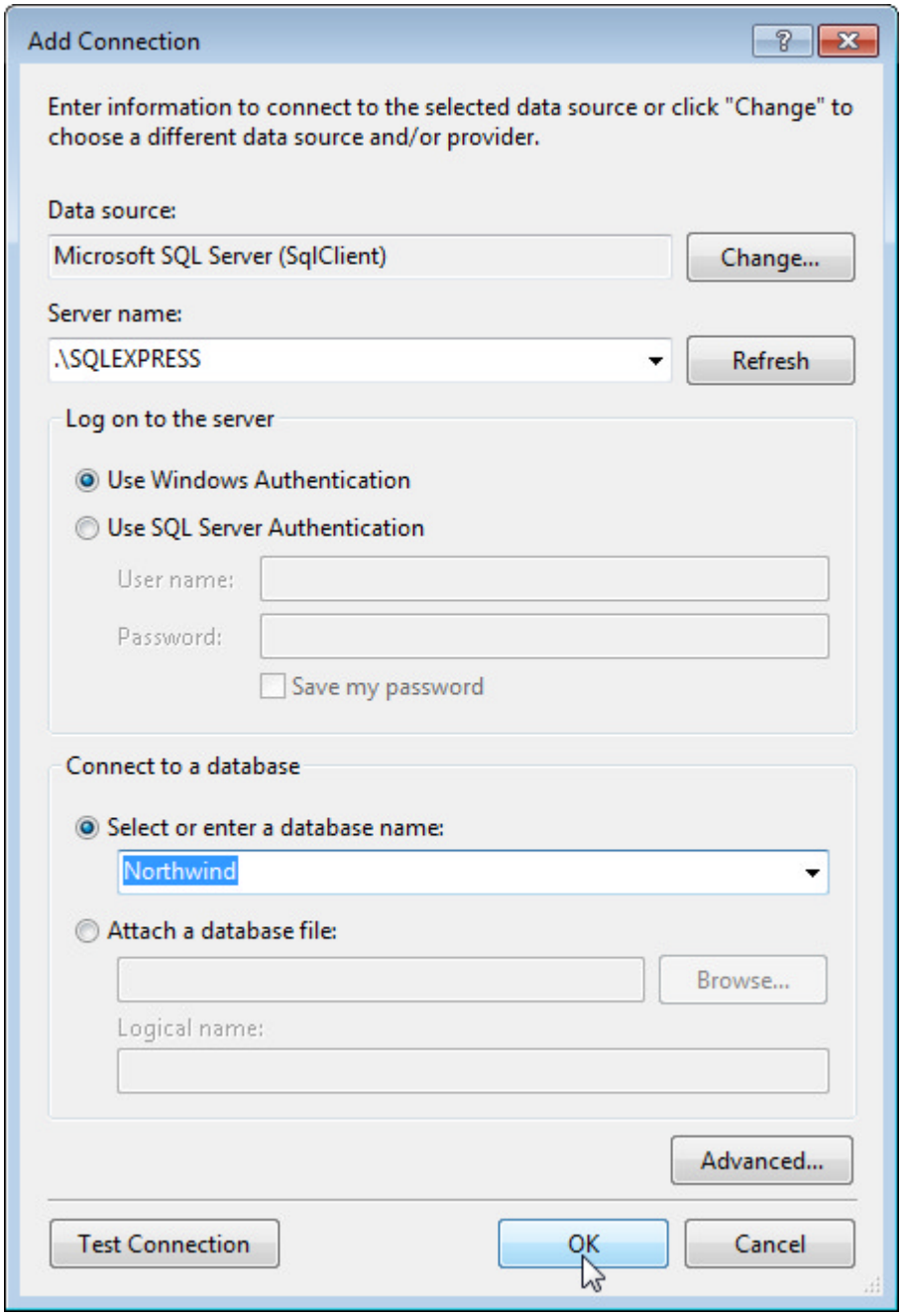
3. In the Data Source Configuration Wizard "Choose a Data Source Type" page, select the Database icon and click **Next** to continue.



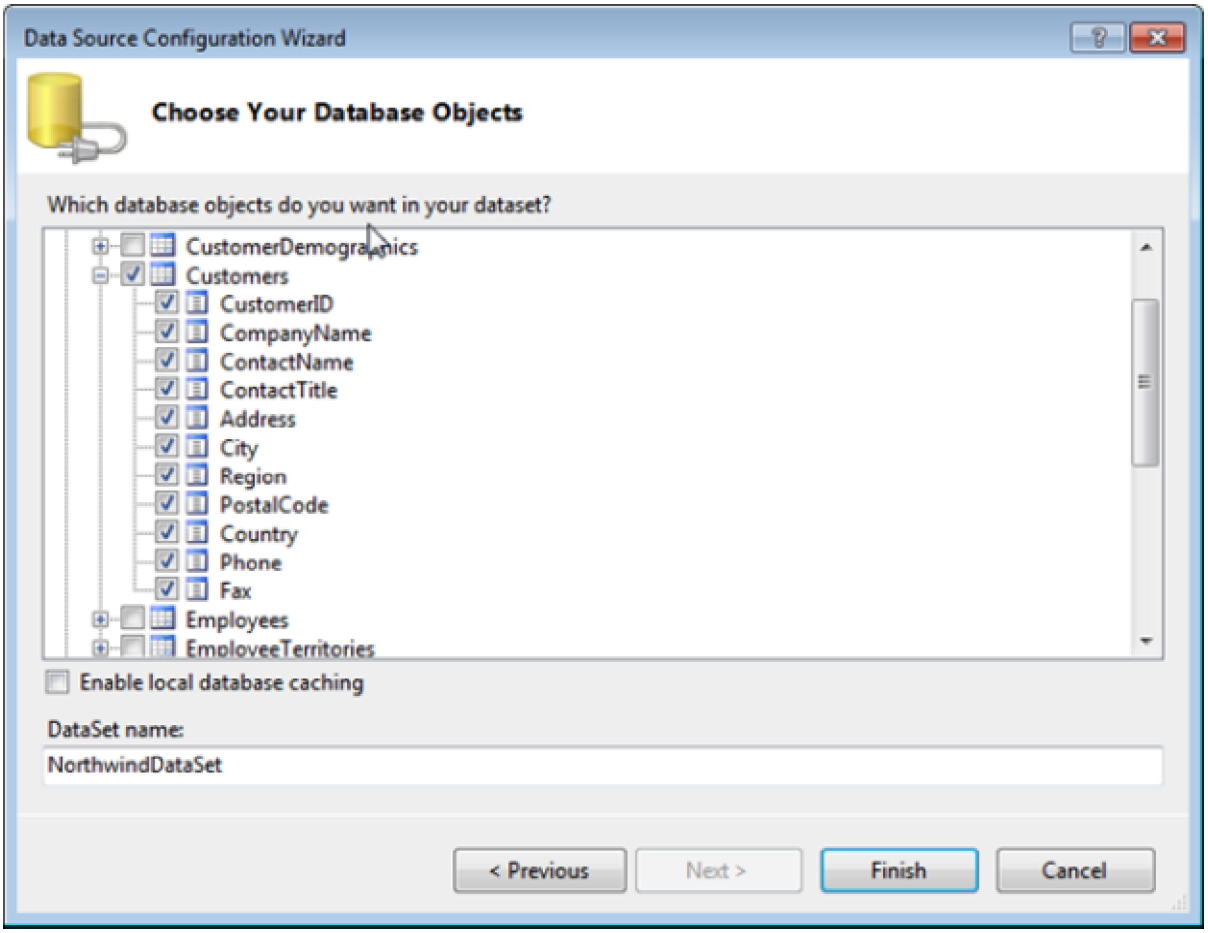
4. In the "Choose Your Data Connection" page, click the **New Connection** button.



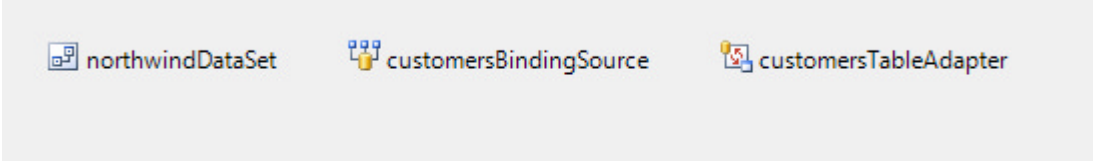
5. In the Add Connection dialog:
   - Click the **Change...** button to display the Change Data Source dialog, select the **MS SQL Server** datasource type and click **OK** to close the dialog.
   - Enter ".\SQLEXPRESS" as the Server Name and select "AdventureWorks" as the database name.
   - Click **OK** to close the Add Connection dialog.



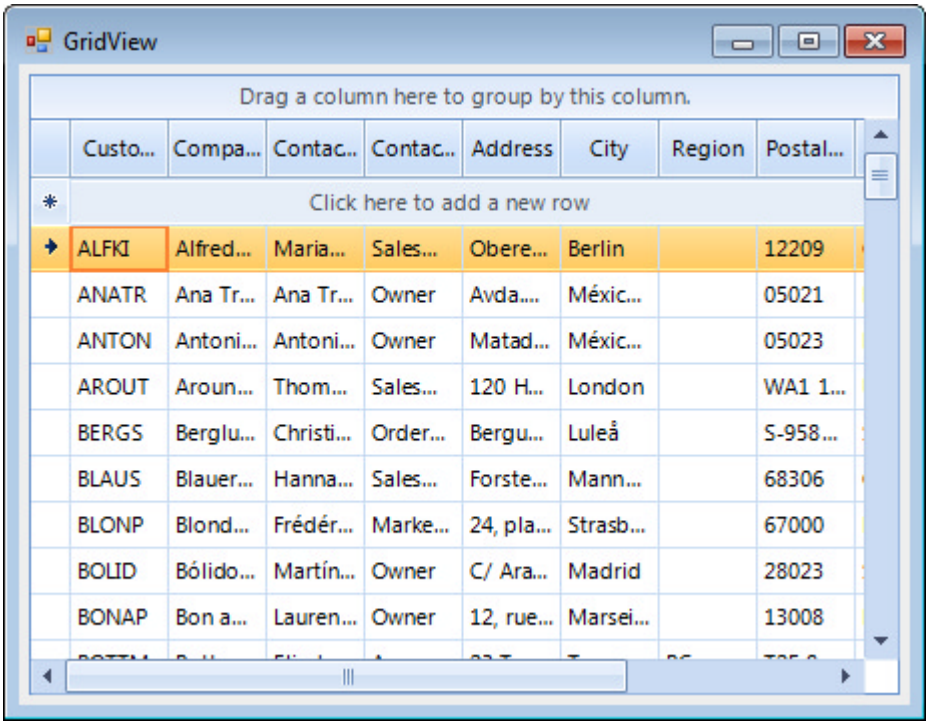
6. In the Data Source Configuration Wizard "Choose Your Data Connection" page, click **Next** to continue.
7. In the Data Source Configuration Wizard "Save the Connection String to the Application Configuration File" page, click **Next** to continue.
8. In the "Choose Database Objects" page of the wizard, open up the treeview and select the tables and fields you want to include in your data source. For the sake of this example, choose all the fields of the "SalesPerson" table.



9. Click **Finish** to close the wizard. A DataSet object that encapsulates selected tables and fields is created automatically. Also, a line of code is placed in the Form's Load event handler that fills the SalesPerson table in the dataset.



10. For a minimal implementation, this is all that's needed to bind the grid to a database file. Press **Ctrl-F5** to run the applicaiton and view the data. **Note**: The example below adds a "BreezeTheme" component to the form and sets the RadGridView Theme property to "Breeze".



In the upcoming "Grid" chapter you will learn how to extend this example to manipulate rows and columns, bind hierarchical data and much more.

12.11 Summary

In this chapter you learned which controls have DataSource properties and the bindable types that can be assigned to them. The chapter started out by showing examples of binding to simple arrays and lists of objects, then went on to use BindingSource to bind to database, business objects and web services. The chapter also showed how to bind to simple controls using the DataBindings property. Finally you learned how to bind to LINQ data sources.

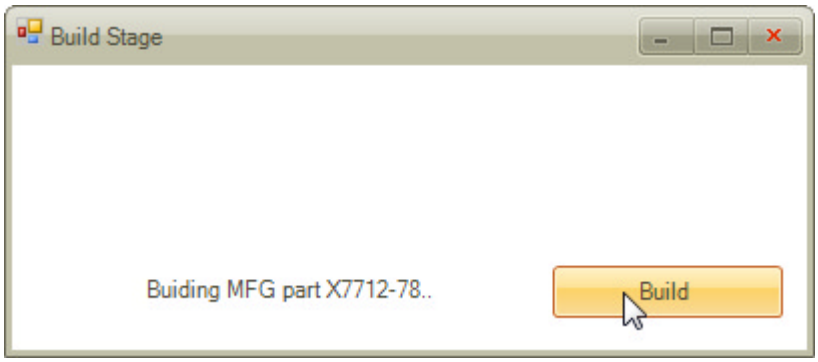
13 User Feedback

13.1 Objectives

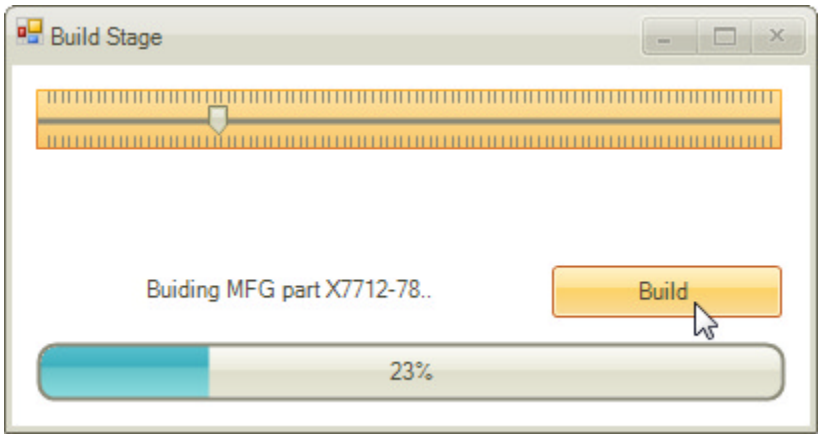
- Learn how to provide user feedback using RadStatusStrip, RadTrackBar, RadWaitingBar, RadProgressBar and RadLabel controls.
- Learn usage and basic characteristics for each type of control.
- Learn how to add items to the RadStatusStrip Items collection.
- Programmatically add items and hosted items to the status strip.
- Learn how to coordinate long running background processes with the waiting bar and progress bar.

13.2 Introduction

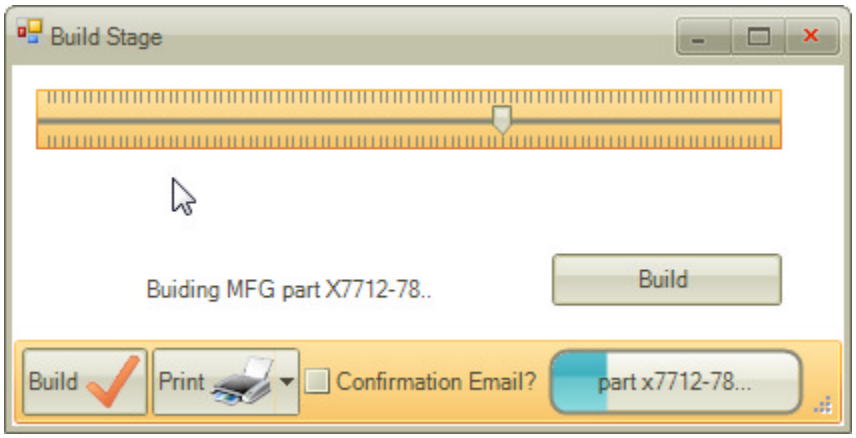
If half the UI battle is to retrieve information from the user, the other half is to provide feedback. The user needs to know what just happened, what's going on now and what's coming up next. Applications with strong feedback keep the user involved and oriented within the overall process. Over time a number of UI conventions for feedback have evolved that are encapsulated within controls. The most basic is the general purpose label that can be updated to notify the user of changes. RadLabel is a themable label that coordinates nicely with other themed controls like this example where the RadForm, RadButton and RadLabel are all set to the "Desert" theme.



RadProgressBar and RadWaitingBar are used to provide feedback for long running operations. RadTrackBar can be used to collect input and to visually represent a numeric value within a range.

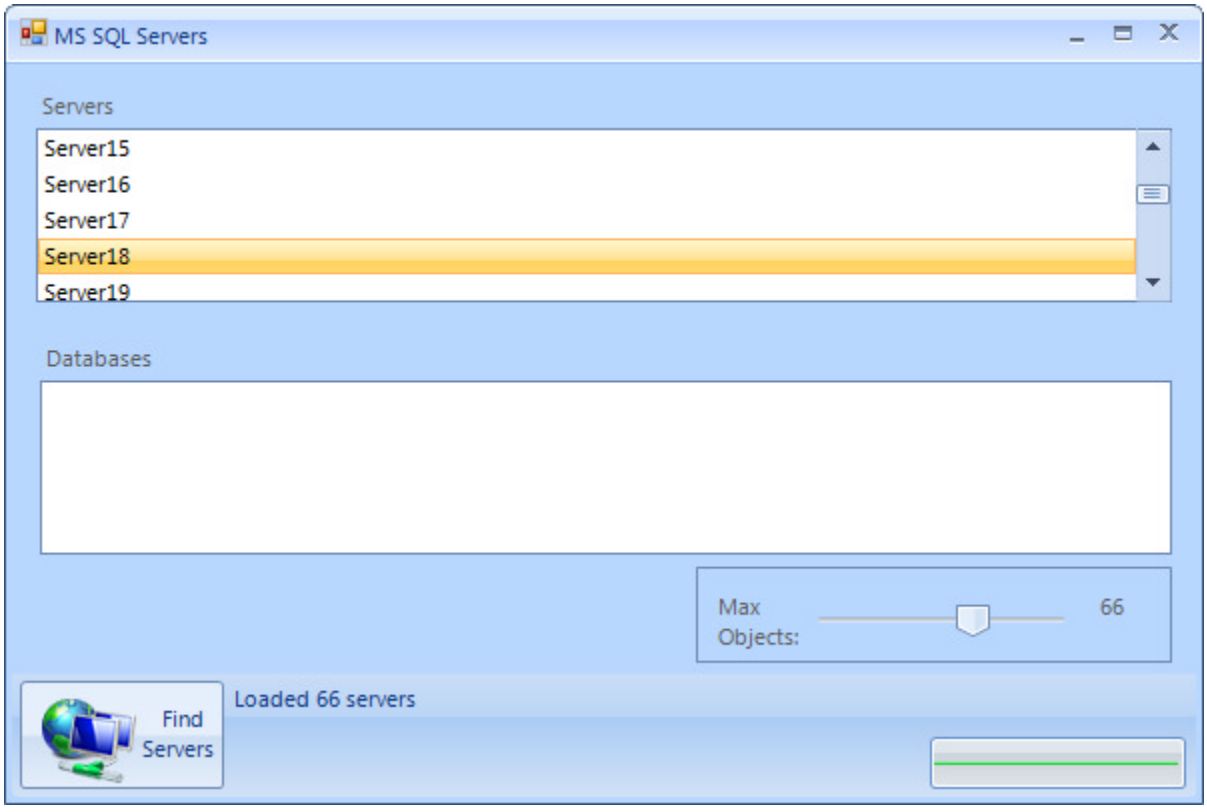


RadStatusStrip can be used both as a feedback mechanism and a menu. The status strip can contain feedback elements, i.e. label, progress bar, waiting bar and even track bars as well as various button types and can also host any Control descendant.



13.3 Getting Started

In this project we will use the RadTrackBar, RadStatusStrip and several elements of the status strip. When the user clicks the "Find Servers" button, the servers list box will be populated with a number of dummy server names; the number to be set by the "Max Objects" track bar. In later sections of this chapter we will expand this example to list actual MS SQL servers and databases for each server.



Preparing the Project

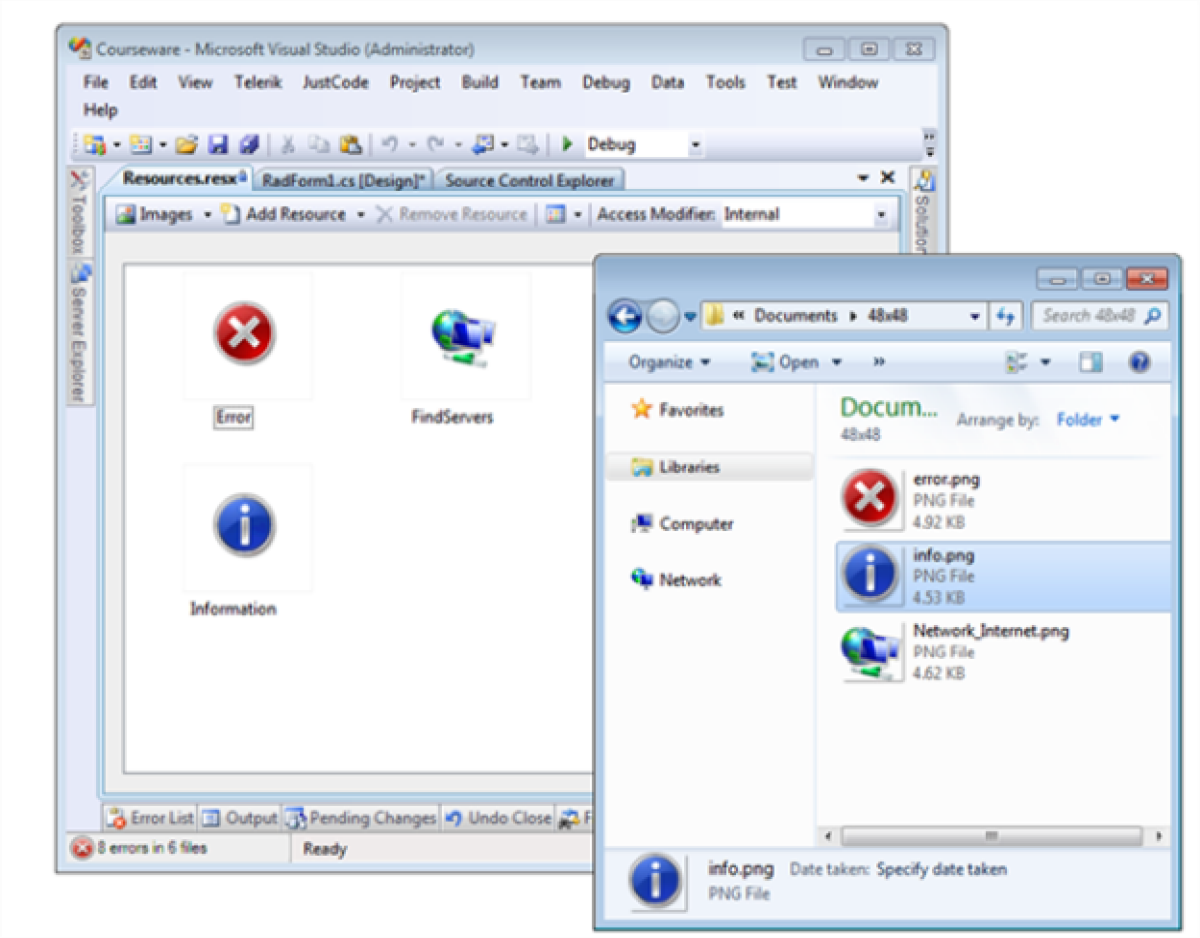
You can find the complete source for this project at:

\User Feedback\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.

2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. In the Properties window, set the form's **Size** property to "600, 400" and the **Text** property to "MS SQL Servers".
6. In the Solution Explorer, double click the Properties/Resources.resx node. Drag image files for user interface tasks and name them as listed below.

- Error
- Information
- FindServers
- FindServersAnimated (animated gif)



After you've dragged the images into the Resources.resx editor, right-click and Rename to the spelling and case shown in the list above. In later sections of this chapter you will want to reference these images using the syntax "Properties.Resources.Error" -- the spellings of each image resource object in Properties.Resources is case sensitive.

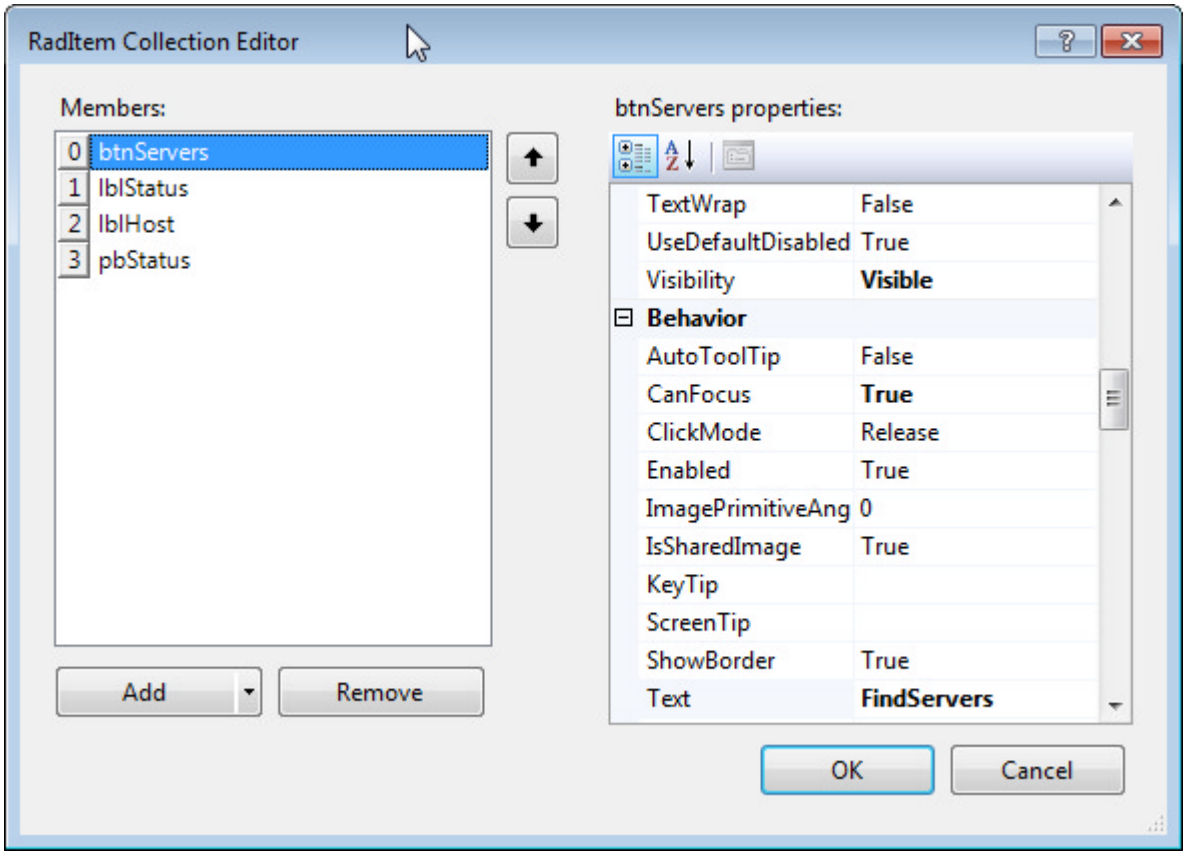
7. Change the project to start with RadForm1 instead of Form1.

Preparing the Form

1. Add a **RadStatusStrip** to the form.

   The status strip will automatically dock with the bottom of the form.

2. In the Properties Window, click the Items property to initiate the RadItemCollection editor. Use the Add button drop down list to include four items with the following properties:
   - RadButtonElement: **Name** = "btnServers", **Text** = "Find Servers", **TextImageRelationship** = "ImageBeforeText.
   - RadLabelElement: **Name** = "lblStatus", **Text** = "", **Spring** = "True", **TextImageRelationship** = "ImageBeforeText.
   - RadLabelElement: **Name** = "lblHost", **Text** = "", **AutoSize** = "False", **Size** = "54, 54".
   - RadProgressBarElement: **Name** = "pbStatus", **Alignment** = "BottomCenter", **MaxSize** = 0,23, **Spring** = True.



   The status strip should look now look something like...

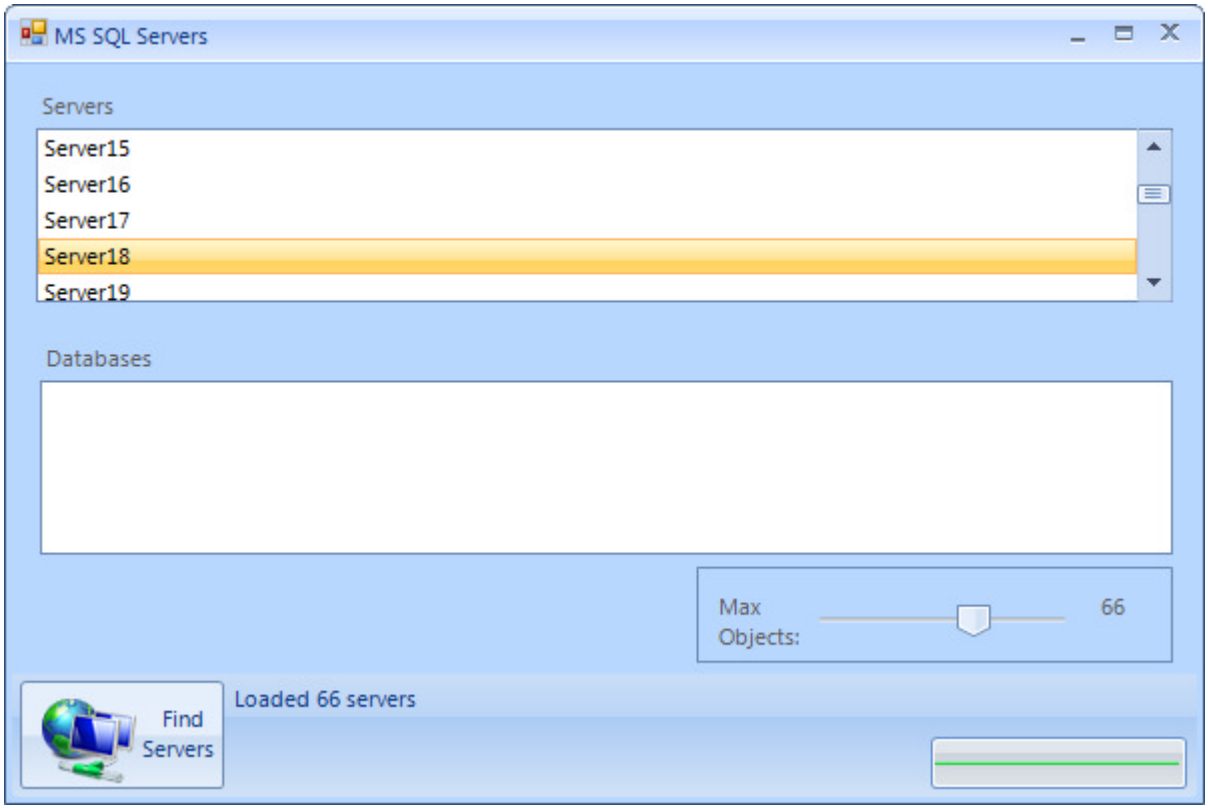


3. In the area above the status strip add the following controls and set properties:
   - RadLabel: **Text** = "Servers".

- RadListControl: **Name** = "lcServers", **Anchor** = "Top, Left, Right".
- RadLabel: **Text** = "Databases".
- RadListControl: **Name** = "lcDatabases", **Anchor** = "Top, Left, Right".
- RadPanel: **Name** = "pnlMaxObjects", **Anchor** = "Top, Right"

On the RadPanel, add the following:

- RadLabel: **Text** = "Max Objects".
- RadTrackBar: **Name** = "tbMaxObjects", **Maximum** = "100", **BackColor** = "Transparent", **ShowTicks** = "False", **SliderAreaColor1** = "PaleTurquoise", **SliderAreaColor2** = "SteelBlue", **ThemeName** = "Aqua", **TicksColor** = "Transparent".
- RadLabel: **Name** = "lblCount", **BackColor** = "Transparent", **Text** = "0".



Add Event Handlers

1. Double-click the "Find Servers" button to create a click event handler and replace the handler code.

   This code just spins in a simple loop, adding items to the list control each iteration and updating the progress bar.

> ⚠ *In this example we're calling the Refresh() method for the RadStatusStrip. This is used here so we can quickly show some of the basic properties for the progress bar. You will want to use a threading mechanism and synchronize the user interface with the background thread. Please see the upcoming section on threading.*

[VB] Handling the Click Event

```
Private Sub btnServers_Click(sender As Object, e As System.EventArgs)
 lcServers.Items.Clear()
 pbStatus.Maximum = tbMaxObjects.Value
 Dim i As Integer = 0
 While i < tbMaxObjects.Value
  Dim item As New RadListDataItem("Server" + i.ToString())
  lcServers.Items.Add(item)
  pbStatus.Value1 = i
  ssMain.Refresh()
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
 lblStatus.Text = "Loaded " + tbMaxObjects.Value.ToString() + " servers"
End Sub
```

[C#] Handling the Click Event

```
private void btnServers_Click(object sender, System.EventArgs e)
{
 lcServers.Items.Clear();
 pbStatus.Maximum = tbMaxObjects.Value;
 for (int i = 0; i < tbMaxObjects.Value; i++)
 {

RadListDataItem item = new RadListDataItem("Server" + i.ToString());
   lcServers.Items.Add(item);
   pbStatus.Value1 = i;
   ssMain.Refresh();
 }
 lblStatus.Text = "Loaded " + tbMaxObjects.Value.ToString() + " servers";
}
```

2. Add a reference for Telerik.WinControls.UI to the "Imports" (VB) or "uses" (C#) section of code.

[VB] Handling the ValueChanged Event

```
Private Sub tbMaxObjects_ValueChanged(sender As Object, e As System.EventArgs)
 lblCount.Text = tbMaxObjects.Value.ToString()
End Sub
```

[C#] Handling the ValueChanged Event

```
private void tbMaxObjects_ValueChanged(object sender, System.EventArgs e)
{
 lblCount.Text = tbMaxObjects.Value.ToString();
}
```

3. Double-click the form to create a form Load event handler and add code to set the track bar Maximum and label:

[VB] Handling the Load Event

```
Private Sub RadForm1_Load(sender As Object, e As System.EventArgs)
 tbMaxObjects.Value = tbMaxObjects.Maximum
 lblCount.Text = tbMaxObjects.Value.ToString()
End Sub
```

[C#] Handling the Load Event

```
private void RadForm1_Load(object sender, System.EventArgs e)
{
 tbMaxObjects.Value = tbMaxObjects.Maximum;
```

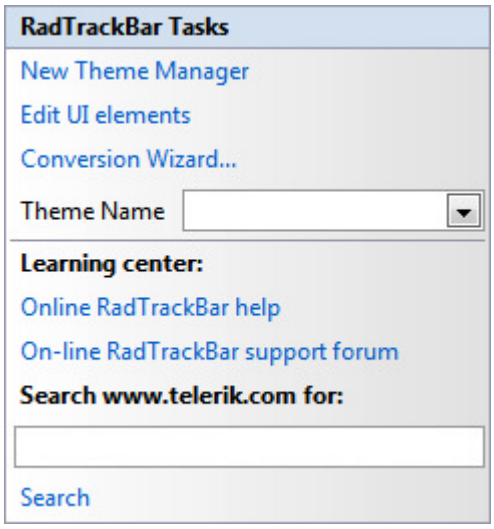
```
    lblCount.Text = tbMaxObjects.Value.ToString();
}
```

4. Press **Ctl-F5** to run the application.

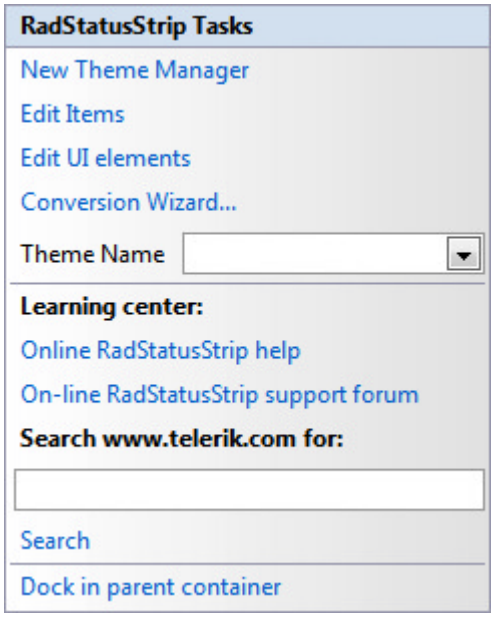
13.4 Using the Design Time Interface

Smart Tag

The Smart Tag for RadLabel, RadWaitingBar, RadProgressBar and RadTrackBar have all the minimal RadControls Smart Tag items.



The RadStatusStrip Smart Tag has an additional "Edit Items" link that invokes the RadItems collection editor dialog. The Smart Tag also has a "Dock in parent container" link.



Tasks

From the Smart Tag Tasks menu you can select New Theme Manager to add a Theme Manager component to the component tray, Edit UI elements to browse all of the elements of the button and to change properties for any element. The Theme Name drop down lets you pick an existing theme to style your control.

Learning center

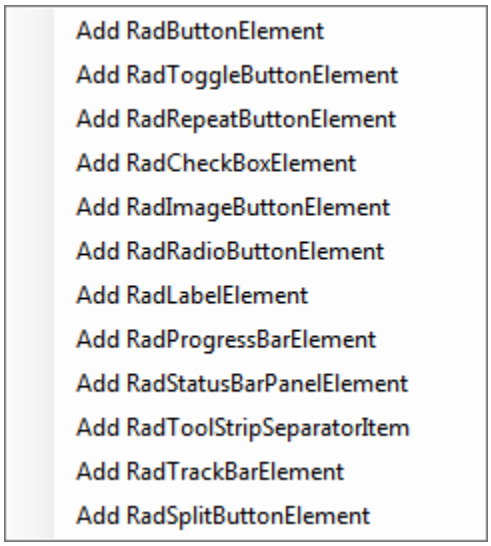
The Learning center lets you navigate to a web browser with online help for the currently selected button control or to the online support forums.

Search

Enter search criteria in the edit space provided and click the Search link to navigate directly to search on the Telerik web site.

Status Strip Designer

You can edit status strip items right in the designer. Enter text into the "Type here" box and then hit Enter to create a RadButtonElement in the status strip...

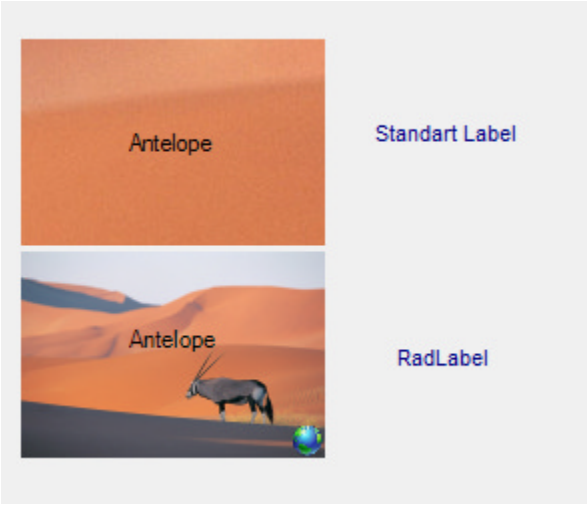


...Or click the drop down button arrow to select from one of the element types. The types listed in the drop down will be the same item types available in the RadItem Collection editor dialog. To get to the dialog click the Smart Tag "Edit Items" link or click the Items property ellipses in the Property window.

Properties

RadLabel

RadLabel works as a themable alternative to a standard Label control. The example below shows a standard label with the Image property set to a picture of an Antelope and the Text aligned MiddleCenter. The RadLabel has the **BackgroundImage** set to the same picture and the Image property set to the picture of a 16 x 16 globe. The **ImageAlignment** property is BottomRight, **TextImageRelation** set to TextAboveImage and the ThemeName is "Desert".



RadWaitingBar

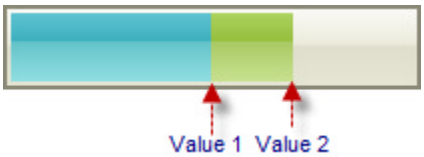
Use RadWaitingBar to let your user know something is happening, but you don't know how long the process will take. Connectivity to network or database resources can fall into this category. As your network API method casts around looking for servers or IP addresses, the API may find 10 or 100 available. There is no way to know ahead of time. RadWaitingBar uses the **StartWaiting()** method to animate the waiting bar and **EndWaiting()** to cease animation.



- **Orientation** can be Vertical or Horizontal. Using Vertical orientation, the thumb moves from top to bottom down the bar.
- **WaitingIndicatorWidth** is the width of the thumb.
- **WaitingSpeed** sets the speed of the animation. Higher numbers move the thumb more quickly across the bar.
- **WaitingStep** is the number of pixels the thumb moves in each step. The default is "1" and provides a smooth animation. Larger number may appear "jumpy".

RadProgressBar

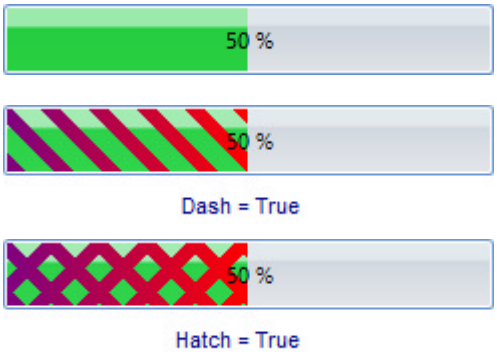
Unlike RadWaitingBar, RadProgress bar lets the user know where they are within a process and to get a good feel for about how long the process will take to complete. RadProgressBar lets you set and display two different progress indicators independently, **Value1** and **Value2**. The Theme will determine how the two values are displayed, or even if Value2 will be visible. This screenshot shows a RadProgressBar with the "Desert" theme, Value1 = "50" and Value2 = "70".



- **Dash** and **Hatch** properties control the drawing style of the active portion of the progress bar. If Dash is set to False (whatever the Hatch setting), the active portion of the progress bar will be drawn in a solid color set by the BackColor property. If Dash is set to True and Hatch is set to False, the active portion of the progress bar will be draw in a striped pattern. The stripes will fade from the **Separator1** color to the **Separator2** color. If Dash is set to True and Hatch is set to True, the active portion of the progress bar will be drawn in a cross-hatched pattern. The cross hatches will fade from the Separator1 color to the

Separator2 color.

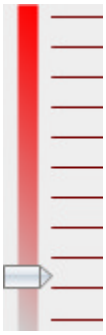
This screenshot shows combinations of Dash and Hatch settings. SeparatorColor1 is Purple, SeparatorColor2 is Red, SweepAngle is 135 and SeparatorWidth is 10. If **SeparatorWidth** was set to a larger number, more of the green would be blocked out by the Red and Purple.



- **SeparatorWidth** controls the width of the stripes or cross hatches in the active portion of the progress bar. **StepWidth** controls the spacing between the stripes or cross hatches in the active portion of the progress bar. **SweepAngle** sets the angle of the stripes or cross hatches in the active portion of the progress bar.
- **BackColor** sets the color of the control background where the progress bar isn't being painted. Depending on the theme, you may or may not be able to see this color.
- **ForeColor** is the color of the text that reports the completion percentage or the Text.
- **Minimum and Maximum** properties set the allowed range of values for the Value1 and Value2 properties.
- **ProgressOrientation** represents the direction that the progress bar moves. You can set this property to **Top**, **Left**, **Right**, or **Bottom** to display progress starting at the top, left, right, or bottom of the control.
- **ShowProgressIndicators** when true replaces the Text with the percentage of completion.

RadTrackBar

RadTrackBar can be used both to input numeric values and to indicate the current status of a numeric value within a range. When working programmatically with RadTrackBar, use the **ValueChanged** event to detect that the user has moved the slider. The **Value** property indicates the current position of the slider.



- **ShowSlideArea** controls whether the line down the middle of the control where the slider rides is visible.
- **ShowTicks** controls whether tick marks are drawn. **TicksColor** sets the color of the tick marks, **TickFrequency** sets the spacing of the tick marks and **TickStyle** controls whether the tick marks are drawn on one or both sides of the control.
- **SlideAreaWidth** property sets the width of the slide area in pixels.
- **SlideAreaColor1**, **SlideAreaColor2**, and **SlideAreaGradientAngle** properties control the way that the slide area is shaded.

- **ThumbWidth** sets the width of the slider thumb
- **LargeChange** is the change in value that one click of the mouse outside of the slider makes.
- **Minimum** and **Maximum** limit the Value property.
- **Orientation** can be set to Horizontal or Vertical.

In the example above Orientation is Vertical, Maximum is "100" and the Value is "80". SliderAreaColor1 is "LightGray", SliderAreaColor2 is "Red" and SliderAreaGradientAngle is "320". SliderAreaWidth is "10", making it wide enough to see the color gradient. TickFrequency is "10", spacing out the ticks to be easily visible. TickColor is "Maroon" and TickStyle is "BottomRight". the theme is Office2007Silver and finally from the smart tag select the EditUI elements, navigate to TrackBarThumb and set its AngleTransform to 0.

Now click on the track bar smart tag and select Edit UI elements. Select the TrackBarPrimitive and set its BackColor to Control, BackColor2 to ControlDark and GradientStyle to Linear. Finally select the TrackBarThumb element and change its AngleTransform property to 0.

13.5 Programming the User Feedback Controls

Background Tasks and the User Interface

If you are processing within a tight loop, it's likely the progress bar or waiting bar will freeze as computing cycles are consumed. A good pattern for working with UI and background processing is to use Thread objects to perform work and to synchronize with the main thread when the user interface is being updated. If you want a well defined interface that handles threading details and has events for displaying progress and completion, consider the BackgroundWorker component found in the System.ComponentModel namespace.



You can find the complete source for this project at:

\User Feedback\<VB|CS>\Threading

Threading and RadWaitingBar

1. Start with the "Getting Started" project or a copy.
2. Add a RadWaitingBar to the form.
3. Add System.ComponentModel to the "Imports" (VB) or "uses" (C#) section of code.
4. Replace the Click event handling code.

   This starts the waiting bar animation. Then a BackgroundWorker object assigns DoWork and RunWorkerCompleted events. Finally the RunWorkerAsync() method is called to trigger the DoWork event.

   [VB] Handling the Click Event

```
Private Sub btnServers_Click(sender As Object, e As System.EventArgs)

RadWaitingBar1.StartWaiting()
lblStatus.Text = "Finding"

Dim worker As New BackgroundWorker()
AddHandler worker.RunWorkerCompleted, AddressOf worker_RunWorkerCompleted
AddHandler worker.DoWork, AddressOf worker_DoWork
worker.RunWorkerAsync()
End Sub
```

   [C#] Handling the Click Event

```
private void btnServers_Click(object sender, System.EventArgs e)
```

```
{
 radWaitingBar1.StartWaiting();
 lblStatus.Text = "Finding";

 BackgroundWorker worker = new BackgroundWorker();
 worker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(worker_RunWorkerComplete
 worker.DoWork += new DoWorkEventHandler(worker_DoWork);
 worker.RunWorkerAsync();
}
```

5. Create the DoWork and RunWorkerCompleted event handlers.

   DoWork simply puts the background thread to sleep for three seconds to represent processing. RunWorkerCompleted fires when DoWork is done. The status label is updated and the waiting bar EndWaiting() method halts animation.

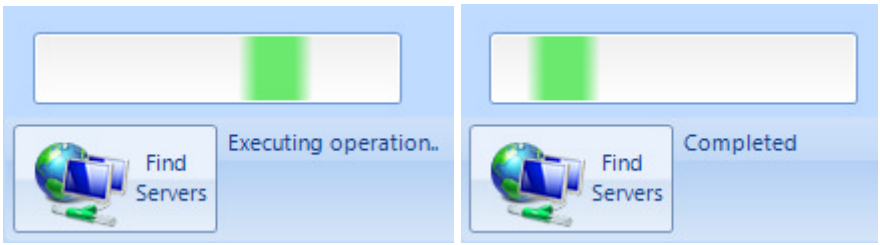
   [VB] Handling the DoWork and RunWorkerCompleted Events

```
Sub worker_DoWork(sender As Object, e As DoWorkEventArgs)
 Thread.Sleep(3000)
End Sub
Sub worker_RunWorkerCompleted(sender As Object, e As RunWorkerCompletedEventArgs)
 lblStatus.Text = "Completed"
 radWaitingBar1.EndWaiting()
End Sub
```

   [C#] Handling the DoWork and RunWorkerCompleted Events

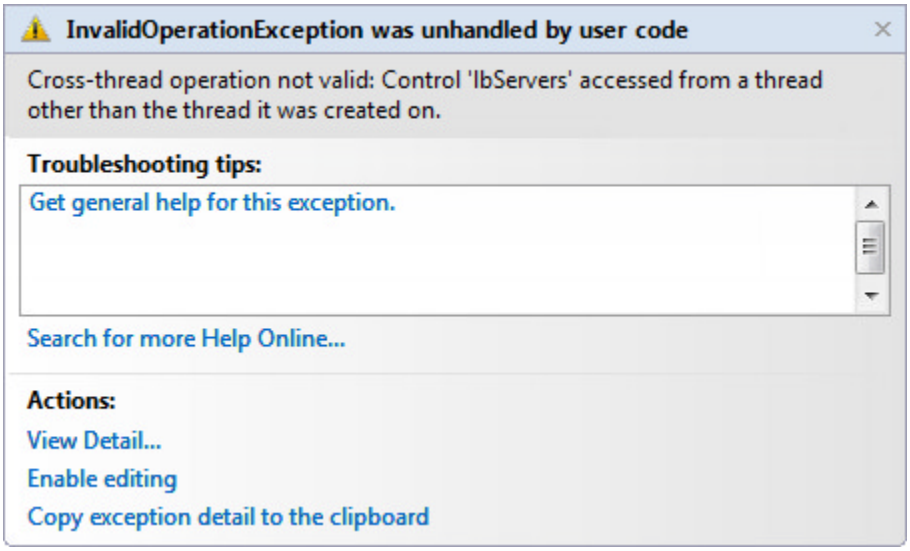
```
void worker_DoWork(object sender, DoWorkEventArgs e)
{
 Thread.Sleep(3000);
}
void worker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
 lblStatus.Text = "Completed";
 radWaitingBar1.EndWaiting();
}
```

6. Press **Ctrl-F5** to run the application.



Updating Progress in the User Interface

Be careful not to put code in the DoWork() handler that needs to talk to the main thread. Referencing any of the controls in the UI during the DoWork() handler is a sure route to an exception like this:



So how do we update a progress bar as work processes and avoid threading clashes? There are multiple methods available to synchronize threads that can be appropriate. The BackgroundWorker component has a **ProgressChanged** event that is ideal for this situation. You will need to set the BackgroundWorker **WorkerReportsProgress** property to true and call the **ReportProgress()** method. Notice in particular that the ProgressChanged event has been hooked up to a handler and the WorkerReportsProgress property is set to true.

[VB] Adding The ProgressChanged Event

```
Private Sub btnServers_Click(sender As Object, e As System.EventArgs)
 lblStatus.Text = "Finding..."
 Dim worker As New BackgroundWorker()
 worker.RunWorkerCompleted += New RunWorkerCompletedEventHandler(worker_RunWorkerCompleted)
 worker.DoWork += New DoWorkEventHandler(worker_DoWork)
 worker.WorkerReportsProgress = True
 worker.ProgressChanged += New ProgressChangedEventHandler(worker_ProgressChanged)
 worker.RunWorkerAsync()
End Sub
```

[C#] Adding The ProgressChanged Event

```
private void btnServers_Click(object sender, System.EventArgs e)
{
 lblStatus.Text = "Finding...";
 BackgroundWorker worker = new BackgroundWorker();
 worker.RunWorkerCompleted += new RunWorkerCompletedEventHandler(worker_RunWorkerCompleted);
 worker.DoWork += new DoWorkEventHandler(worker_DoWork);
 worker.WorkerReportsProgress = true;
 worker.ProgressChanged += new ProgressChangedEventHandler(worker_ProgressChanged);
 worker.RunWorkerAsync();
}
```



Gotcha! Attempting to use the ProgressChanged event without setting WorkerReportsProgress to true will generate an exception.

Inside the DoWork event handler call ReportProgress() every time you want to update the user interface. Pass an integer representing the percentage of progress achieved and a **UserState** object that can contain any arbitrary data. Notice that there is no UI related code inside the DoWork event handler -- that is for the ProgressChanged event to take care of. At the conclusion of DoWork assign the **Result** argument property. This value can be retrieved later in the RunWorkerCompleted event.

Inside of the ProgressChanged event handler, add to the server list box, update the status label and set the progress bar value.

[VB] Handling the DoWork and ProgressChanged Events

```
Private Sub worker_DoWork(ByVal sender As Object, ByVal e As DoWorkEventArgs)
Dim i As Integer = 0
While i < tbMaxObjects.Value
TryCast(sender, BackgroundWorker).ReportProgress(i, i)
Thread.Sleep(5)
System.Math.Max(System.Threading.Interlocked.Increment(i), i - 1)
End While
e.Result = tbMaxObjects.Value
End Sub
Private Sub worker_ProgressChanged(ByVal sender As Object, ByVal e As
ProgressChangedEventArgs)
Dim item As New RadListDataItem("Server" & e.UserState.ToString())
lcServers.Items.Add(item)
pbStatus.Value1 = e.ProgressPercentage
End Sub
```

[C#] Handling the DoWork and ProgressChanged Events

```
void worker_DoWork(object sender, DoWorkEventArgs e)
{
for (int i = 0; i < tbMaxObjects.Value; i++)
{
(sender as BackgroundWorker).ReportProgress(i, i);
Thread.Sleep(5);
}
e.Result = tbMaxObjects.Value;
}
void worker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
RadListDataItem item = new RadListDataItem("Server" + e.UserState.ToString());
lcServers.Items.Add(item);
pbStatus.Value1 = e.ProgressPercentage;
}
```

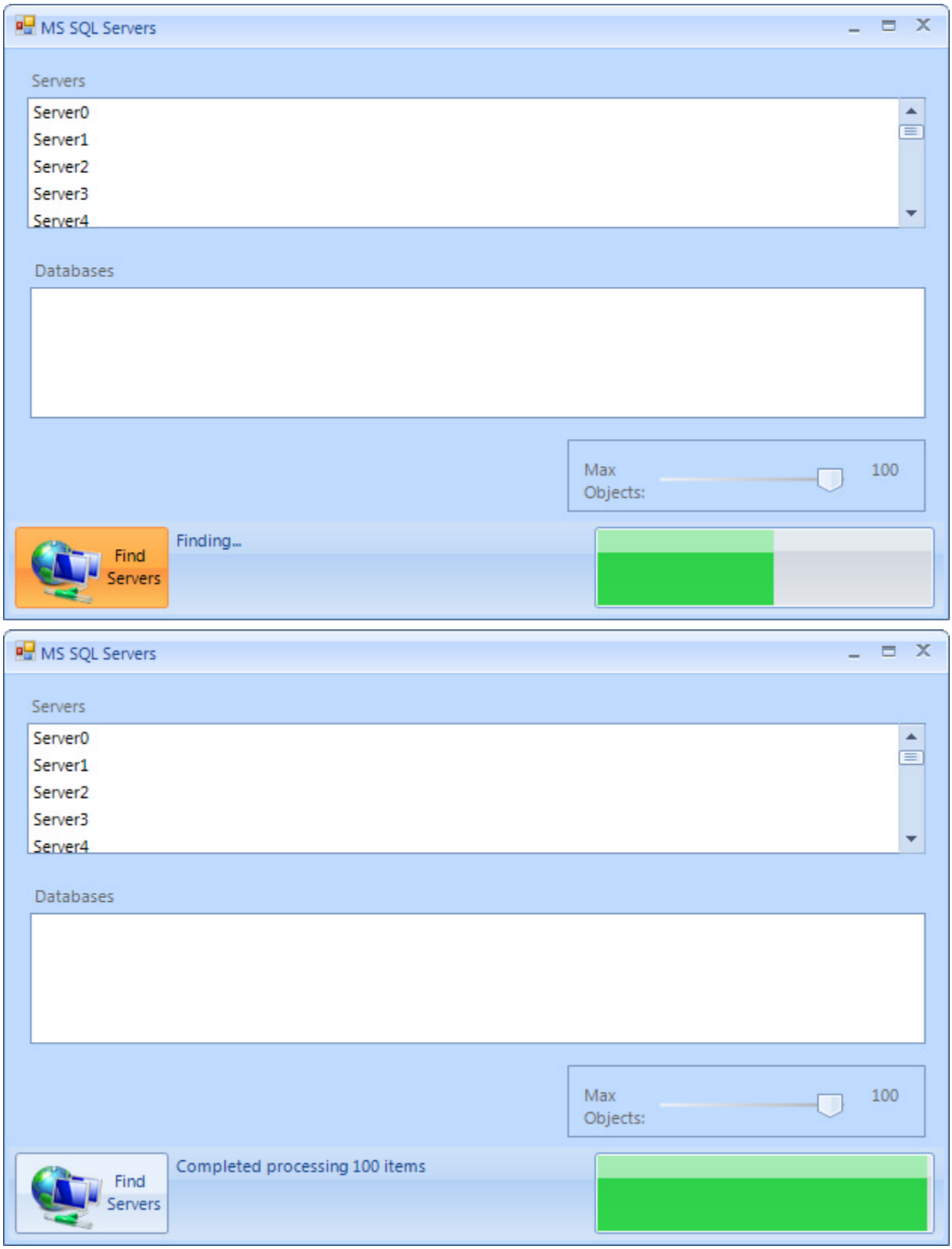
When processing completes in the DoWork event handler, the RunWorkerCompleted event fires, giving you a chance to show the final status of the process. The event takes a RunWorkerCompletedEventArgs that contains the result populated earlier by the DoWork event handler.

[VB] Handling the RunWorkerCompleted Event

```
Sub worker_RunWorkerCompleted(sender As Object, e As RunWorkerCompletedEventArgs)
 lblStatus.Text = "Completed processing " + e.Result.ToString() + " items"
End Sub
```

[C#] Handling the RunWorkerCompleted Event

```
void worker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
 lblStatus.Text = "Completed processing " + e.Result.ToString() + " items";
}
```



Background Tasks Walk-Through

Lets use these techniques to perform real work listing MS SQL servers and databases within those servers. We will also filter databases shown in the second list based on the number of objects in the database and the track bar Value. We will use the waiting bar during the server retrieval because we don't know how long that's going to take, but the number of databases is available before we process so we can use the progress bar there. The waiting bar will be removed from the form and added to the status strip; this will give you an opportunity to see a control dynamically added and removed from the status bar.

This project will use the MS SQL Server Management Objects (SMO) API. The API lets you enumerate all available MS SQL servers, databases and other information about each database. The Visual Studio install typically includes SQL Express, so you should have at least that server on your local machine to work with.



You can find the complete source for this project at:

\User Feedback\<VB|CS>\ThreadingWalkthrough

Preparing the Project

1. Start with the previous "Threading" project or a copy.
2. Add reference to the **Microsoft.SqlServer.Smo** and **Microsoft.SqlServer.ConnectionInfo** assemblies.



This example is using the 9.0 version of these assemblies typically found at \Program Files\Microsoft SQL Server\90\SDK\Assemblies. These assemblies work with the SQL Express 2005 version.

Adding Supporting Code

1. Add class files DatabaseInfo.cs and DBWorkerResult.cs to the project. Populate the code for these three classes:

[VB] The DatabaseInfo Class

```
Public Class DatabaseInfo
 Public Sub New(serverName As String, databaseName As String, objectCount As Integer)
  Me.ServerName = serverName
  Me.DatabaseName = databaseName
  Me.ObjectCount = objectCount
 End Sub
 Public Property ServerName() As String
  Get
  End Get
  Set
  End Set
 End Property
 Public Property DatabaseName() As String
  Get
  End Get
  Set
  End Set
 End Property
 Public Property ObjectCount() As Integer
  Get
  End Get
  Set
  End Set
 End Property
```

```
End Class
```

[C#] The DatabaseInfo Class

```
public class DatabaseInfo
{
 public DatabaseInfo(string serverName, string databaseName, int objectCount)
 {
   this.ServerName = serverName;
   this.DatabaseName = databaseName;
   this.ObjectCount = objectCount;
 }
 public string ServerName { get; set; }
 public string DatabaseName { get; set; }
 public int ObjectCount { get; set; }
}
```

[VB] The DBWorkerResult Class

```
Public Class DBWorkerResult
 Public Sub New(serverName As String, maxItems As Integer, minItems As Integer)
  Me.ServerName = serverName
  Me.MaxItems = maxItems
  Me.MinItems = minItems
 End Sub
 Public Property ServerName() As String
  Get
  End Get
  Set
  End Set
 End Property
 Public Property MaxItems() As Integer
  Get
  End Get
  Set
  End Set
 End Property
 Public Property MinItems() As Integer
  Get
  End Get
  Set
  End Set
 End Property
End Class
```

[C#] The DBWorkerResult Class

```
public class DBWorkerResult
{
 public DBWorkerResult(string serverName, int maxItems, int minItems)
 {
   this.ServerName = serverName;
   this.MaxItems = maxItems;
   this.MinItems = minItems;
 }
 public string ServerName { get; set; }
 public int MaxItems { get; set; }
 public int MinItems { get; set; }
}
```

2. In the code for the form replace the current "Imports" (VB) or "uses" (C#) clause with the following namespaces:

[VB] Adding References

```
Imports System
Imports System.Collections.Generic
Imports System.ComponentModel
Imports System.Data
Imports System.Drawing
Imports System.Windows.Forms
Imports Microsoft.SqlServer.Management.Smo
Imports Telerik.WinControls
Imports Telerik.WinControls.Primitives
Imports Telerik.WinControls.UI
```

[C#] Adding References

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using Microsoft.SqlServer.Management.Smo;
using Telerik.WinControls;
using Telerik.WinControls.Primitives;
using Telerik.WinControls.UI;
```

3. Add an enumeration that will be used later when displaying status information:

[VB] The StatusTypes Enumeration

```
Enum StatusTypes
 Info
 [Error]
 Find
End Enum
```

[C#] The StatusTypes Enumeration

```
enum StatusTypes { Info, Error, Find };
```

4. Add a private member that will store a list of DatabaseInfo objects and a RadLabelElement that will host our waiting bar control.

[VB] Generic List of DatabaseInfo

```
Private _dbInfoList As List(Of DatabaseInfo)
Private _waitingBarHostLabel As RadLabelElement
```

[C#] Generic List of DatabaseInfo

```
private List<DatabaseInfo> _dbInfoList;
private RadLabelElement _waitingBarHostLabel;
```

5. Add the helper methods below. The helper methods will:
   - Get an icon representing a given StatusTypes member.
   - Display status messages and icons.
   - Show/Hide both the progress and waiting bars.
   - Toggle the UI Enabled properties to prevent the user from initiating new processes before existing

work is completed.

[VB] Helper Methods

```
' retrieve an image for a given status
Private Function GetStatusImage(ByVal statusType As StatusTypes) As Image
Select Case statusType
Case StatusTypes.Error
Return My.Resources.Error1
Case StatusTypes.Info
Return My.Resources.Information
Case StatusTypes.Find
Return My.Resources.FindServersAnimated
Case Else
Return My.Resources.Information
End Select
End Function
' display a text and image status message
Private Sub DisplayStatus(ByVal message As String, ByVal statusType As StatusTypes)
lblStatus.Text = message
lblStatus.Image = GetStatusImage(statusType)
End Sub
' create a RadLabel\HostedItem\RadWaitingBar and insert into
' the status strip.
Private Function GetHostedWaitingBar(ByVal name As String) As RadLabelElement
' make height and width the same so ellipse will be a circle
Dim size As System.Drawing.Size = New System.Drawing.Size(btnServers.Size.Height,
btnServers.Size.Height)
' use hosted waiting bar control instead of element for access to
' start/stop waiting methods.
Dim waitingBar As RadWaitingBar = New RadWaitingBar()
' waiting bar animation sweeps top to bottom
waitingBar.Orientation = Orientation.Vertical
waitingBar.WaitingBarElement.BackColor = Color.SteelBlue
waitingBar.WaitingBarElement.Shape = New EllipseShape()
Dim waitingBarHostLabel As RadLabelElement = New RadLabelElement()
' status label to the left will respect DefaultSize and
' leave room for waiting bar
waitingBarHostLabel.DefaultSize = size
waitingBarHostLabel.Name = name
Dim host As RadHostItem = New RadHostItem(waitingBar)
waitingBarHostLabel.Children.Insert(0, host)
' set transparent after being inserted to host children
waitingBar.BackColor = Color.Transparent
Return waitingBarHostLabel
End Function
' Loads a list control with generic List of DatabaseInfo;
' Name to the Text and number of objects to the description
Private Sub LoadDatabaseList(ByVal listControl As RadListControl, ByVal maxObjects As
Integer)
listControl.Items.Clear()
For Each info As DatabaseInfo In _dbInfoList
' limit databases to those with number of objects within trackbar limits
If info.ObjectCountProp <= maxObjects Then
Dim item As RadListDataItem = New RadListDataItem(info.DatabaseNameProp, info)
listControl.Items.Add(item)
```

```
End If
Next info
End Sub
' reset progress bar value and add to status strip
Private Sub ShowProgressBar()
pbStatus.Value1 = 0
ssMain.Items.Add(pbStatus)
End Sub
' remove progress bar from status strip
Private Sub HideProgressBar()
ssMain.Items.Remove(pbStatus)
End Sub
' add waiting bar element to status strip,
' get host item, drill down to waitiing bar and StartWaiting()
Private Sub ShowWaitingBar()
ssMain.Items.Add(_waitingBarHostLabel)
Dim hostItem As RadHostItem = TryCast(_waitingBarHostLabel.Children(0), RadHostItem)
TryCast(hostItem.HostedControl, RadWaitingBar).StartWaiting()
End Sub
' get host item, drill down to waitiing bar, StopWaiting() and
' remove element from status strip.
Private Sub HideWaitingBar()
Dim hostItem As RadHostItem = TryCast(_waitingBarHostLabel.Children(0), RadHostItem)
TryCast(hostItem.HostedControl, RadWaitingBar).StopWaiting()
ssMain.Items.Remove(_waitingBarHostLabel)
End Sub
' disable controls during processing
Private Sub SetUIEnabled(ByVal enabled As Boolean)
btnServers.Enabled = enabled
lbServers.Enabled = enabled
lbDatabases.Enabled = enabled
' only enable when there are objects to filter
If (Not enabled) Then
tbMaxObjects.Enabled = False
End If
End Sub
```

[C#] Helper Methods

```
// retrieve an image for a given status
private Image GetStatusImage(StatusTypes statusType)
{
switch (statusType)
{
case StatusTypes.Error:
return Properties.Resources.Error;
case StatusTypes.Info:
return Properties.Resources.Information;
case StatusTypes.Find:
return Properties.Resources.FindServersAnimated;
default:
return Properties.Resources.Information;
}
}
// display a text and image status message
private void DisplayStatus(string message, StatusTypes statusType)
```

```
{
lblStatus.Text = message;
lblStatus.Image = GetStatusImage(statusType);
}
// create a RadLabel\HostedItem\RadWaitingBar and insert into
// the status strip.
private RadLabelElement GetHostedWaitingBar(string name)
{
// make height and width the same so ellipse will be a circle
Size size = new Size(btnServers.Size.Height, btnServers.Size.Height);
// use hosted waiting bar control instead of element for access to
// start/stop waiting methods.
RadWaitingBar waitingBar = new RadWaitingBar();
// waiting bar animation sweeps top to bottom
waitingBar.Orientation = Orientation.Vertical;
waitingBar.WaitingBarElement.BackColor = Color.SteelBlue;
waitingBar.WaitingBarElement.Shape = new EllipseShape();
RadLabelElement waitingBarHostLabel = new RadLabelElement();
// status label to the left will respect DefaultSize and
// leave room for waiting bar
waitingBarHostLabel.DefaultSize = size;
waitingBarHostLabel.Name = name;
RadHostItem host = new RadHostItem(waitingBar);
waitingBarHostLabel.Children.Insert(0, host);
// set transparent after being inserted to host children
waitingBar.BackColor = Color.Transparent;
return waitingBarHostLabel;
}
// Loads a list control with generic List of DatabaseInfo;
// Name to the Text and number of objects to the description
private void LoadDatabaseList(RadListControl listControl, int maxObjects)
{
listControl.Items.Clear();
foreach (DatabaseInfo info in _dbInfoList)
{
// limit databases to those with number of objects within trackbar limits
if (info.ObjectCount <= maxObjects)
{
RadListDataItem item = new RadListDataItem(info.DatabaseName, info);
listControl.Items.Add(item);
}
}
}
// reset progress bar value and add to status strip
private void ShowProgressBar()
{
pbStatus.Value1 = 0;
ssMain.Items.Add(pbStatus);
}
// remove progress bar from status strip
private void HideProgressBar()
{
ssMain.Items.Remove(pbStatus);
}
// add waiting bar element to status strip,
```

```
// get host item, drill down to waitiing bar and StartWaiting()
private void ShowWaitingBar()
{
ssMain.Items.Add(_waitingBarHostLabel);
RadHostItem hostItem =
_waitingBarHostLabel.Children[0] as RadHostItem;
(hostItem.HostedControl as RadWaitingBar).StartWaiting();
}
// get host item, drill down to waitiing bar, StopWaiting() and
// remove element from status strip.
private void HideWaitingBar()
{
RadHostItem hostItem = _waitingBarHostLabel.Children[0] as RadHostItem;
(hostItem.HostedControl as RadWaitingBar).StopWaiting();
ssMain.Items.Remove(_waitingBarHostLabel);
}
// disable controls during processing
private void SetUIEnabled(bool enabled)
{
btnServers.Enabled = enabled;
lbServers.Enabled = enabled;
lbDatabases.Enabled = enabled;
// only enable when there are objects to filter
if (!enabled)
tbMaxObjects.Enabled = false;
}
```



The relationship of objects created in GetHostedWaitingBar() looks something like the image below. Why so many levels? In our database listing example we need to call the StartWaiting() method of the RadWaitingBar; this method does not exist in a RadWaitingBarElement, so we need to contain a RadWaitingBar control using RadHostItem.

Why use RadLabelElement as the outermost element inside the status strip? We could have used a RadPanelElement for example, but RadLabelElement has the fewest visual artifacts that might be in the way, e.g. borders, that would need to be removed. Use RadLabelElement with the Text set to "" to see just the hosted RadWaitingBar control.



6. Replace the code in the Form's Load event handler with the code below. The first line of code here will simply give the "Max Objects" panel rounded corners. Then create a RadLabelElement that hosts the waiting bar. Finally, hide the progress bar until needed.

[VB] Handling the Form Load Event

```
Private Sub RadForm1_Load(sender As Object, e As System.EventArgs)
 tbMaxObjects.Enabled = False
 ' panel should have rounded shape
 (TryCast(pnlMaxObjects.RootElement.Children(0).Children(1), BorderPrimitive)).Shape = Ne
RoundRectShape()
 _waitingBarHostLabel = GetHostedWaitingBar("WaitingBarHostLabel")
 HideProgressBar()
End Sub
```

[C#] Handling the Form Load Event

```
private void RadForm1_Load(object sender, System.EventArgs e)
{
 tbMaxObjects.Enabled = false;
 // panel should have rounded shape
 (pnlMaxObjects.RootElement.Children[0].Children[1] as BorderPrimitive).Shape = new
   RoundRectShape();
 _waitingBarHostLabel = GetHostedWaitingBar("WaitingBarHostLabel");
 HideProgressBar();
}
```

List Servers

1. To handle tasks related to the BackgroundWorker for the list of servers, add the code below.
   - RunServerWorker() creates the BackgroundWorker object instance, hooks up the events and calls RunWorkerAsync() to begin processing.
   - The DoWork() event handler calls the SmoApplication static EnumAvailableSqlServers() method. The task of retrieving any available MS SQL servers takes an unknown amount of time so the waiting bar is the most appropriate control to display feedback.
   - The RunWorkerCompleted() event handler gets the result containing a DataTable with the list of databases and displays a completion method, or displays an error message if an exception occurs. Exceptions thrown in DoWork() are passed to RunWorkerCompleted in the argument Error property.

   [VB] Handle the Server BackgroundWorker

```
' create and configure the BackgroundWorker object
Private Sub RunServerWorker()
 Dim serverWorker As New BackgroundWorker()
 serverWorker.DoWork += New DoWorkEventHandler(serverWorker_DoWork)
 serverWorker.RunWorkerCompleted += New RunWorkerCompletedEventHandler
(serverWorker_RunWorkerCompleted)
 serverWorker.RunWorkerAsync()
End Sub
' retrieve MS SQL servers, pass back DataTable result
Sub serverWorker_DoWork(sender As Object, e As DoWorkEventArgs)
 e.Result = SmoApplication.EnumAvailableSqlServers()
End Sub
' update the UI using the Result property of the args, or
' the Error property if the operation fails.
Sub serverWorker_RunWorkerCompleted(sender As Object, e As RunWorkerCompletedEventArg
 If e.[Error] <> Nothing Then
  DisplayStatus(e.[Error].Message, StatusTypes.[Error])
 Else
  ' populate list control from data table of servers passed in as Result
  Dim dt As DataTable = TryCast(e.Result, DataTable)
  For Each row As DataRow In dt.Rows
```

```
    lbServers.Items.Add(New RadListDataItem(row("Name").ToString()))
   Next
   DisplayStatus("Found " + dt.Rows.Count.ToString() + " servers", StatusTypes.Info)
  End If
  HideWaitingBar()
  SetUIEnabled(True)
End Sub
```

[C#] Handle the Server BackgroundWorker

```
// create and configure the BackgroundWorker object
private void RunServerWorker()
{
 BackgroundWorker serverWorker = new BackgroundWorker();
 serverWorker.DoWork += new DoWorkEventHandler(serverWorker_DoWork);
 serverWorker.RunWorkerCompleted += new RunWorkerCompletedEventHandler
(serverWorker_RunWorkerCompleted);
 serverWorker.RunWorkerAsync();
}
// retrieve MS SQL servers, pass back DataTable result
void serverWorker_DoWork(object sender, DoWorkEventArgs e)
{
 e.Result = SmoApplication.EnumAvailableSqlServers();
}
// update the UI using the Result property of the args, or
// the Error property if the operation fails.
void serverWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
 if (e.Error != null)
 {
   DisplayStatus(e.Error.Message, StatusTypes.Error);
 }
 else
 {
   // populate list control from data table of servers passed in as Result
   DataTable dt = e.Result as DataTable;
   foreach (DataRow row in dt.Rows)
   {
     lbServers.Items.Add(new RadListDataItem(row["Name"].ToString()));
   }
   DisplayStatus("Found " + dt.Rows.Count.ToString() + " servers",
     StatusTypes.Info);
 }
 HideWaitingBar();
 SetUIEnabled(true);
}
```

2. Replace the existing Click event handling code for "btnServers" with the code below:

[VB] Handling the Click Event

```
Private Sub btnServers_Click(sender As Object, e As System.EventArgs)
 SetUIEnabled(False)
 lbServers.Items.Clear()
 lbDatabases.Items.Clear()
 lblDatabases.Text = "Databases"
 ShowWaitingBar()
```

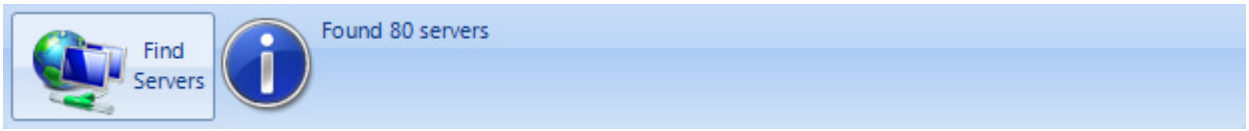
```
 DisplayStatus("Searching for MS SQL Servers", StatusTypes.Find)
 RunServerWorker()
End Sub
```

[C#] Handling the Click Event

```
private void btnServers_Click(object sender, System.EventArgs e)
{
 SetUIEnabled(false);
 lbServers.Items.Clear();
 lbDatabases.Items.Clear();
 lblDatabases.Text = "Databases";
 ShowWaitingBar();
 DisplayStatus("Searching for MS SQL Servers", StatusTypes.Find);
 RunServerWorker();
}
```

3. Press **Ctrl-F5** to run the project. Pressing the "Find Servers" button displays the animated gif, status message and waiting bar (shown at the far right of the status bar). When the process completes, the servers are listed and the status bar displays a completion message and information icon.





Try testing the exception handling portion of code by deliberately throwing an Application exception during the DoWork event handler. The RunWorkerCompleted event will fire and e.Error will contain the Exception object.

[VB] Triggering an Exception

```
Sub serverWorker_DoWork(sender As Object, e As DoWorkEventArgs)
 Throw New ApplicationException("Something bad happened")
 e.Result = SmoApplication.EnumAvailableSqlServers()
End Sub
```

[C#] Triggering an Exception

```
void serverWorker_DoWork(object sender, DoWorkEventArgs e)
{
 throw new ApplicationException("Something bad happened");
 e.Result = SmoApplication.EnumAvailableSqlServers();
}
```

If you are debugging, Visual Studio will break at the line that causes the exception. Press Ctrl-F5 to continue to the RunWorkerCompleted event handler. If you're not debugging you should see the exception message displayed in the status strip immediately.



List Databases

1. To handle tasks related to the BackgroundWorker that lists databases for a server, add the code below.
   - RunDBWorker() creates the BackgroundWorker object instance, hooks up events and calls RunWorkerAsync() to begin processing. Notice that the WorkerReportsProgress property is set to true and the ProgressChanged event is assigned a handler. The server name is passed to the RunWorkerAsync() method.
   - The DoWork() event handler retrieves and processes the list of databases on the server. During processing the event handler tracks the minimum and maximum number of objects for all databases. As each database is processed, a DatabaseInfo object is created, saved in a list and sent through to the ReportProgress() method. When all processing is complete, a DBWorkerResult object is created, populated and sent back in the arguments Result property.
   - The RunWorkerCompleted() event handler gets the result containing the DBWorkerResult and the information is used to update the UI.

   [VB] Handle the Server BackgroundWorker

```
' create and configure BackgroundWorker. Populate and pass server name as argument.
' enable progress reporting
Private Sub RunDbWorker(serverName As String, maxObjects As Integer)
 Dim dbWorker As New BackgroundWorker()
 dbWorker.WorkerReportsProgress = True
 dbWorker.DoWork += New DoWorkEventHandler(dbWorker_DoWork)
 dbWorker.ProgressChanged += New ProgressChangedEventHandler(dbWorker_ProgressChanged
 dbWorker.RunWorkerCompleted += New RunWorkerCompletedEventHandler
(dbWorker_RunWorkerCompleted)
 dbWorker.RunWorkerAsync(serverName)
End Sub

' retrieve the list of databases for the server:
' -get the number of objects for each database,
' -keep track of the min and max number of objects.
' -calculate progress,
' -create and populate a DatabaseInfo object and pass to
' ProgressChanged event handler.
' Create and populate a DBWorkerResult object and assign to the arguments
' Result property.
Sub dbWorker_DoWork(sender As Object, e As DoWorkEventArgs)
 Dim server As New Server(serverName)
 Dim count As Integer = 0
 Dim maxObjectCount As Integer = 0
 Dim minObjectCount As Integer = 0
 For Each database As Database In server.Databases
  System.Math.Max(System.Threading.Interlocked.Increment(count),count - 1)
  ' ignore empty databases
  If database.IsAccessible Then
   ' get all objects in the database
   Dim dtObjects As DataTable = database.EnumObjects()
   ' calculate progress
   Dim progress As Integer = DirectCast(((DirectCast(count, Double) / DirectCast
(server.Databases.Count, Double)) * 100), Integer)
   ' assign the first non-zero value
   If (minObjectCount = 0) AndAlso (dtObjects.Rows.Count > 0) Then
    minObjectCount = dtObjects.Rows.Count
   End If
```

```
    ' get min and max number of objects for all databases in server
    maxObjectCount = Math.Max(maxObjectCount, dtObjects.Rows.Count)
    minObjectCount = Math.Min(maxObjectCount, dtObjects.Rows.Count)
    ' Call ReportProgress to trigger ProgressChanged event.
    ' DatabaseInfo passed as the UserState property
    Dim databaseInfo As New DatabaseInfo(server.Name, database.Name, dtObjects.Rows.Cc
    (TryCast(sender, BackgroundWorker)).ReportProgress(progress, databaseInfo)
   End If
  Next
  e.Result = New DBWorkerResult(server.Name, maxObjectCount, minObjectCount)
End Sub
' respond to ReportProgress() method calls, set progress bar value,
' extract the DatabaseInfo object from the argument's UserState property:
' add to the generic list of DatabaseInfo objects for use on completion,
' and display a status message with the current database name.
Sub dbWorker_ProgressChanged(sender As Object, e As ProgressChangedEventArgs)
 pbStatus.Value1 = e.ProgressPercentage
 Dim info As DatabaseInfo = TryCast(e.UserState, DatabaseInfo)
 _dbInfoList.Add(info)
 DisplayStatus(info.DatabaseName, StatusTypes.Info)
End Sub
' update the UI using the Result property of the args, or
' the Error property if the operation fails.
Sub dbWorker_RunWorkerCompleted(sender As Object, e As RunWorkerCompletedEventArgs)
 If e.[Error] <> Nothing Then
  ' reset the "Databases" label to default value
  lblDatabases.Text = "Databases"
  ' display the exception
  DisplayStatus(e.[Error].Message, StatusTypes.[Error])
 Else
  Dim workerResult As DBWorkerResult = TryCast(e.Result, DBWorkerResult)
  tbMaxObjects.Maximum = workerResult.MaxItems
  ' if there are no items, then minimum is zero, otherwise,
  ' set to the fewest number of items (will always display at least one)
  tbMaxObjects.Minimum = If(workerResult.MinItems = 0, 0, workerResult.MinItems - 1)
  Dim range As Integer = workerResult.MaxItems - workerResult.MinItems
  tbMaxObjects.TickFrequency = range / 10
  tbMaxObjects.Value = tbMaxObjects.Maximum
  ' only enable when there are items
  tbMaxObjects.Enabled = workerResult.MaxItems > 0
  lblCount.Text = tbMaxObjects.Value.ToString()
  LoadDatabaseList(lbDatabases, tbMaxObjects.Value)
  lblDatabases.Text = "Databases for " + workerResult.ServerName
  Dim message As String = "Found " + lbDatabases.Items.Count + " databases for " +
workerResult.ServerName
  DisplayStatus(message, StatusTypes.Info)
 End If
 HideProgressBar()
 SetUIEnabled(True)
End Sub
```

[C#] Handle the Server BackgroundWorker

```
// create and configure BackgroundWorker. Populate and pass server name as argument.
// enable progress reporting
private void RunDbWorker(string serverName, int maxObjects)
```

```
{
 BackgroundWorker dbWorker = new BackgroundWorker();
 dbWorker.WorkerReportsProgress = true;
 dbWorker.DoWork += new DoWorkEventHandler(dbWorker_DoWork);
 dbWorker.ProgressChanged += new ProgressChangedEventHandler(dbWorker_ProgressChanged
 dbWorker.RunWorkerCompleted += new RunWorkerCompletedEventHandler
(dbWorker_RunWorkerCompleted);
 dbWorker.RunWorkerAsync(serverName);
}

// retrieve the list of databases for the server:
// -get the number of objects for each database,
// -keep track of the min and max number of objects.
// -calculate progress,
// -create and populate a DatabaseInfo object and pass to
// ProgressChanged event handler.
// Create and populate a DBWorkerResult object and assign to the arguments
// Result property.
void dbWorker_DoWork(object sender, DoWorkEventArgs e)
{
 Server server = new Server(workerParams.ServerName);
 int count = 0;
 int maxObjectCount = 0;
 int minObjectCount = 0;
 foreach (Database database in server.Databases)
 {
   count++;
   // ignore empty databases
   if (database.IsAccessible)
   {
     // get all objects in the database
     DataTable dtObjects = database.EnumObjects();
     // calculate progress
     int progress = (int)(((double)count / (double)server.Databases.Count) * 100);
     // assign the first non-zero value
     if ((minObjectCount == 0) && (dtObjects.Rows.Count > 0))
       minObjectCount = dtObjects.Rows.Count;
     // get min and max number of objects for all databases in server
     maxObjectCount = Math.Max(maxObjectCount, dtObjects.Rows.Count);
     minObjectCount = Math.Min(maxObjectCount, dtObjects.Rows.Count);
     // Call ReportProgress to trigger ProgressChanged event.
     // DatabaseInfo passed as the UserState property
     DatabaseInfo databaseInfo = new DatabaseInfo(server.Name, database.Name,
dtObjects.Rows.Count);
     (sender as BackgroundWorker).ReportProgress(progress, databaseInfo);
   }
 }
 e.Result = new DBWorkerResult(server.Name, maxObjectCount, minObjectCount);
}
// respond to ReportProgress() method calls, set progress bar value,
// extract the DatabaseInfo object from the argument's UserState property:
// add to the generic list of DatabaseInfo objects for use on completion,
// and display a status message with the current database name.
void dbWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
```

```
 pbStatus.Value1 = e.ProgressPercentage;
 DatabaseInfo info = e.UserState as DatabaseInfo;
 _dbInfoList.Add(info);
 DisplayStatus(info.DatabaseName, StatusTypes.Info);
}
// update the UI using the Result property of the args, or
// the Error property if the operation fails.
void dbWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs e)
{
 if (e.Error != null)
 {
   // reset the "Databases" label to default value
   lblDatabases.Text = "Databases";
   // display the exception
   DisplayStatus(e.Error.Message, StatusTypes.Error);
 }
 else
 {
   DBWorkerResult workerResult = e.Result as DBWorkerResult;
   tbMaxObjects.Maximum = workerResult.MaxItems;
   // if there are no items, then minimum is zero, otherwise,
   // set to the fewest number of items (will always display at least one)
   tbMaxObjects.Minimum = workerResult.MinItems == 0 ? 0 : workerResult.MinItems - 1;
   int range = workerResult.MaxItems - workerResult.MinItems;
   tbMaxObjects.TickFrequency = range / 10;
   tbMaxObjects.Value = tbMaxObjects.Maximum;
   // only enable when there are items
   tbMaxObjects.Enabled = workerResult.MaxItems > 0;
   lblCount.Text = tbMaxObjects.Value.ToString();
   LoadDatabaseList(lbDatabases, tbMaxObjects.Value);
   lblDatabases.Text = "Databases for " + workerResult.ServerName;
   string message = "Found " + lbDatabases.Items.Count + " databases for " +
workerResult.ServerName;
   DisplayStatus(message, StatusTypes.Info);
 }
 HideProgressBar();
 SetUIEnabled(true);
}
```



Gotcha! In the DoWork event handler at the line that references IsAccessible you may receive a SqlError if you have connections setup that no longer point to valid files, as in the case of an Access *.mdb file where the file has been removed from the path. You can either remove these connections with a tool like SQL Server Management Studio, or use try..catch block around the failing code.

2. Using the Property window Events tab for "lbServers" list control, create a SelectedIndexChanged event handler with the code below. The event handler will repopulate the list of databases.

[VB] Handling the SelectedIndexChanged Event

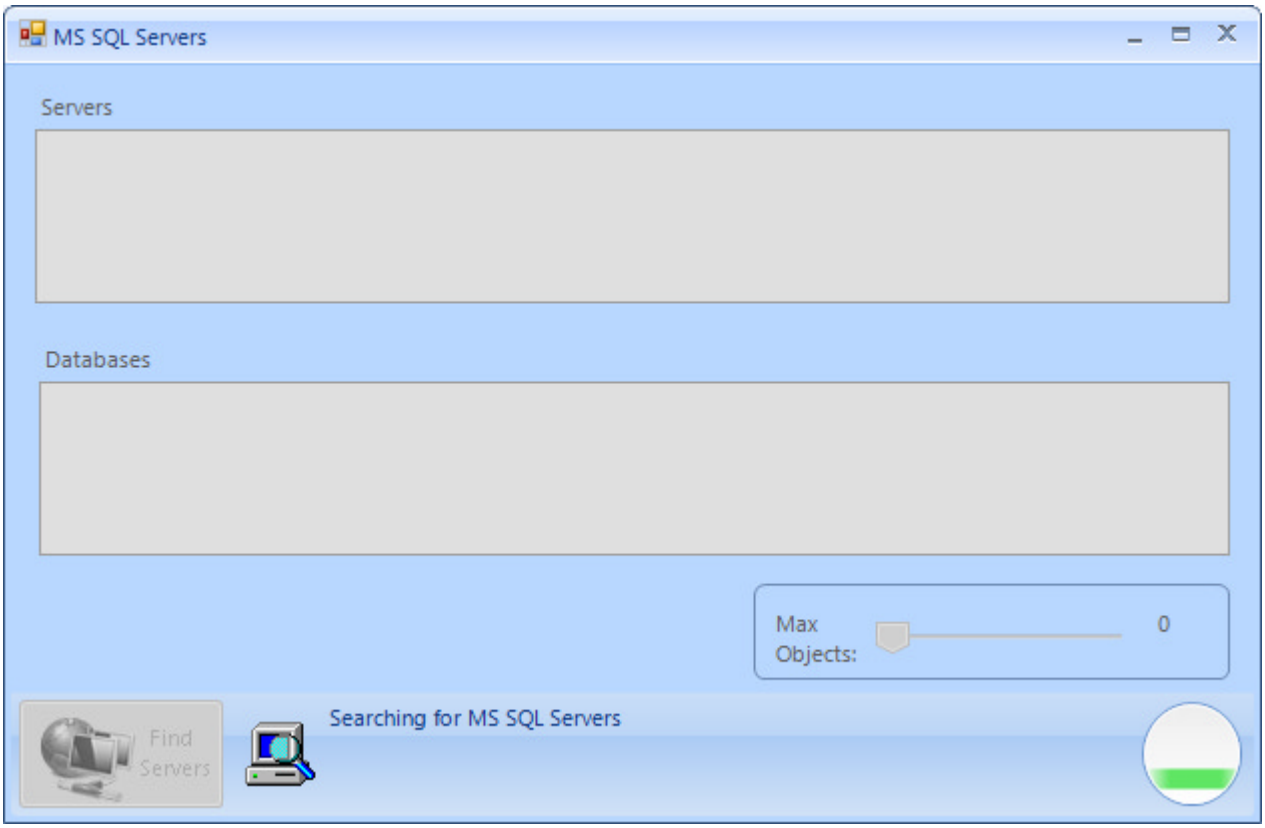
```
Private Sub lbServers_SelectedIndexChanged(ByVal sender As Object, ByVal e As EventArgs)
Handles lbServers.SelectedIndexChanged
Dim listControl As RadListElement = TryCast(sender, RadListElement )
If Not listControl.SelectedItem Is Nothing Then
SetUIEnabled(False)
lbDatabases.Items.Clear()
```

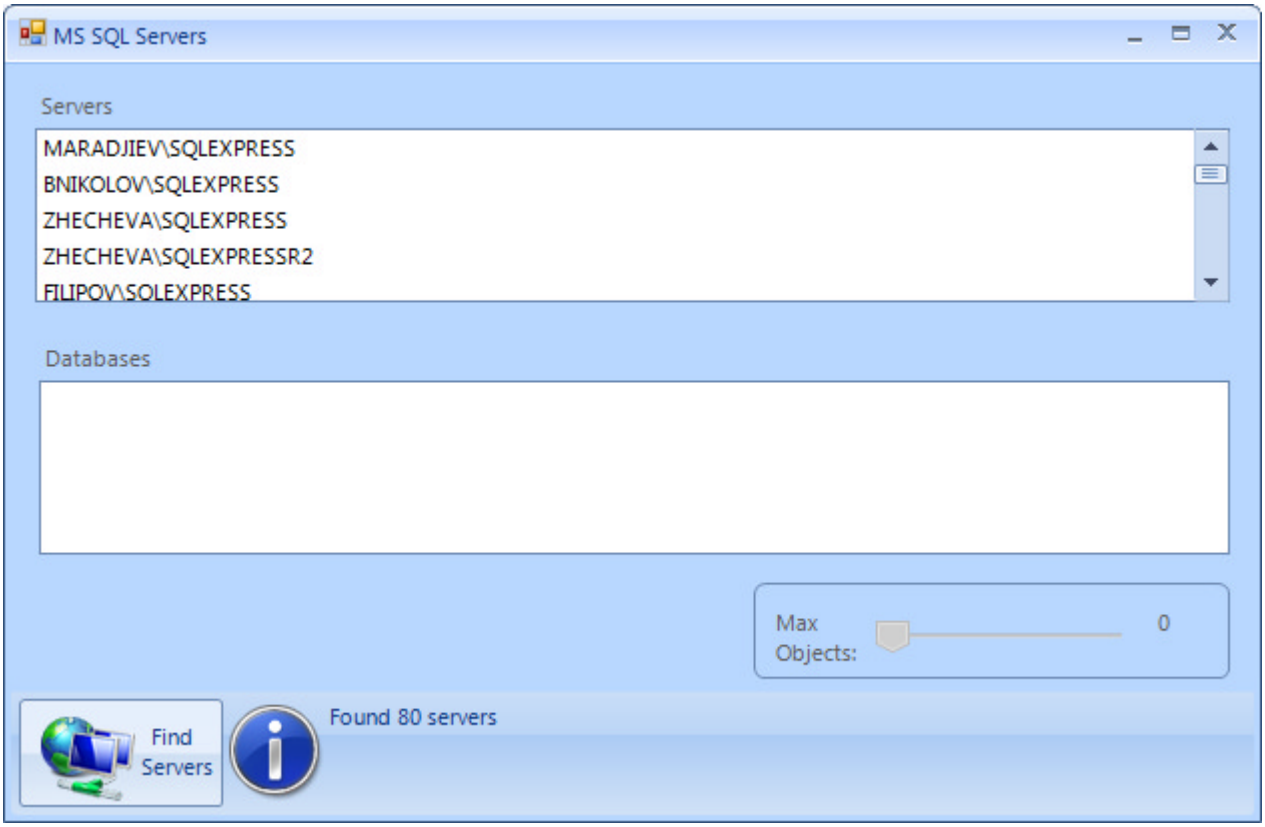
```
_dbInfoList = New List(Of DatabaseInfo)()
Dim serverName As String = (TryCast(listControl.SelectedItem, RadListDataItem)).Text
ShowProgressBar()
DisplayStatus("Finding databases for " & serverName, StatusTypes.Find)
RunDbWorker(serverName, tbMaxObjects.Value)
End If
End Sub
```

[C#] Handling the SelectedIndexChanged Event

```
private void lbServers_SelectedIndexChanged(object sender, EventArgs e)
{
RadListElement listControl = sender as RadListElement;
if (listControl.SelectedItem != null)
{
SetUIEnabled(false);
lbDatabases.Items.Clear();
_dbInfoList = new List<DatabaseInfo>();
string serverName = (listControl.SelectedItem as RadListDataItem).Text;
ShowProgressBar();
DisplayStatus("Finding databases for " + serverName, StatusTypes.Find);
RunDbWorker(serverName, tbMaxObjects.Value);
}
}
```

3. Press **Ctrl-F5** to run the project. Again press the "Find Servers" button. Then click on the list control to choose a server name. That will populate the second list with database names.





Respond to Events

Two more events still need to be handled, the SelectedIndexChanged for the database list control and the ValueChanged event handler needs to be replaced with new code.

1. Create a SelectedIndexChanged event handler for "lbDatabases" and add the code below. This code retrieves the DatabaseInfo object that's stored in the Value property of the selected item. The server name, database name and number of objects are displayed in the status strip. Notice that the RadLabelElement in the status strip displays text on multiple lines.

[VB] Handling the SelectedIndexChanged Event for the Database List Control

```
Private Sub lbDatabases_SelectedIndexChanged(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles
lbDatabases.SelectedIndexChanged
If Not (TryCast(sender, RadListElement)).SelectedValue Is Nothing Then
Dim info As DatabaseInfo = TryCast((TryCast(sender, RadListControl)).SelectedValue,
DatabaseInfo)
Dim message As String = String.Format("{0}\{1}{2}{3} objects", info.ServerNameProp,
info.DatabaseNameProp, Environment.NewLine, info.ObjectCountProp)
DisplayStatus(message, StatusTypes.Info)
End If
End Sub
```

[C#] Handling the SelectedIndexChanged Event for the Database List Control

```
private void lbDatabases_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
if ((sender as RadListElement).SelectedValue != null)
{
```

```
DatabaseInfo info = (sender as RadListControl).SelectedValue as DatabaseInfo;
string message = String.Format("{0}\\{1}{2}{3} objects",
info.ServerName, info.DatabaseName, Environment.NewLine, info.ObjectCount);
DisplayStatus(message, StatusTypes.Info);
}
}
```

2. Replace the ValueChanged event handler for the "Max Items" track bar. The event handler now calls LoadDatabaseList() and passes the new track bar value.

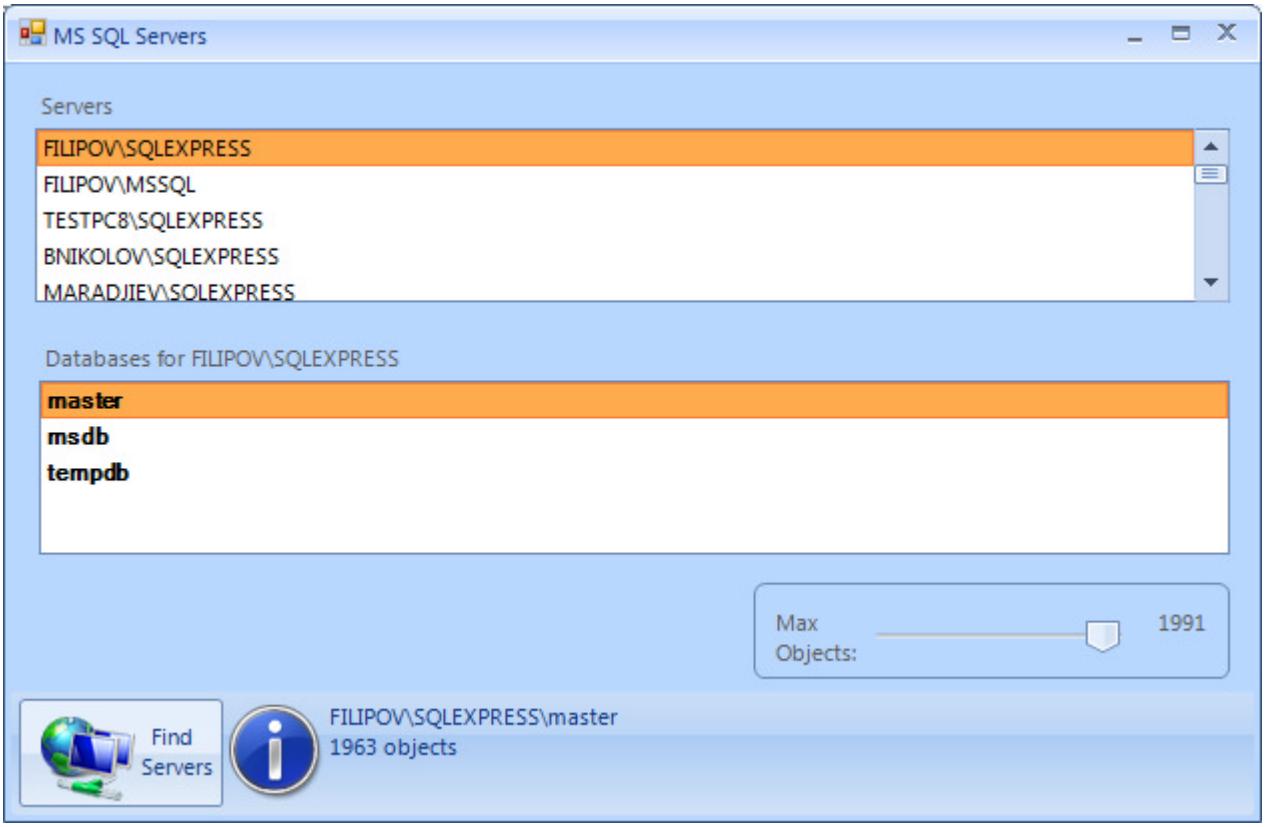
[VB] Handling the ValueChanged Event

```
Private Sub tbMaxObjects_ValueChanged(sender As Object, e As System.EventArgs)
 Dim trackBar As RadTrackBar = TryCast(sender, RadTrackBar)
 LoadDatabaseList(lbDatabases, trackBar.Value)
 lblCount.Text = trackBar.Value.ToString()
End Sub
```

[C#] Handling the ValueChanged Event

```
private void tbMaxObjects_ValueChanged(object sender, System.EventArgs e)
{
 RadTrackBar trackBar = sender as RadTrackBar;
 LoadDatabaseList(lbDatabases, trackBar.Value);
 lblCount.Text = trackBar.Value.ToString();
}
```

3. Press **Ctrl-F5** to run the application.



13.6 Summary

In this chapter you learned how to provide user feedback using RadStatusStrip, RadTrackBar, RadWaitingBar, RadProgressBar and RadLabel controls. You learned the appropriate circumstances to use a RadWaitingBar vs a RadProgressBar. You learned how to use the RadStatusStrip at design time and programmatically. You hosted a control inside the status strip. Finally, you learned how to coordinate long running background processes with the user interface.

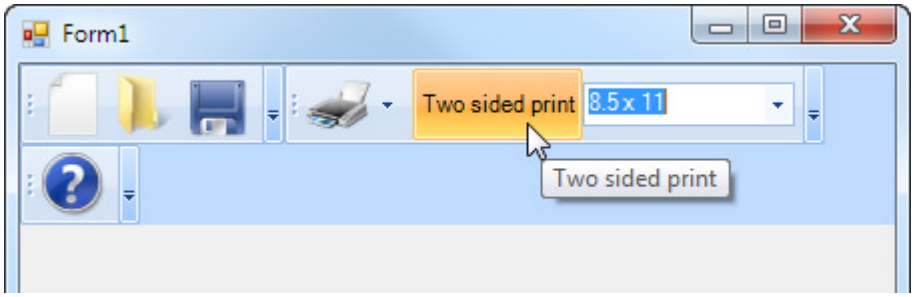
14 CommandBar

14.1 Objectives

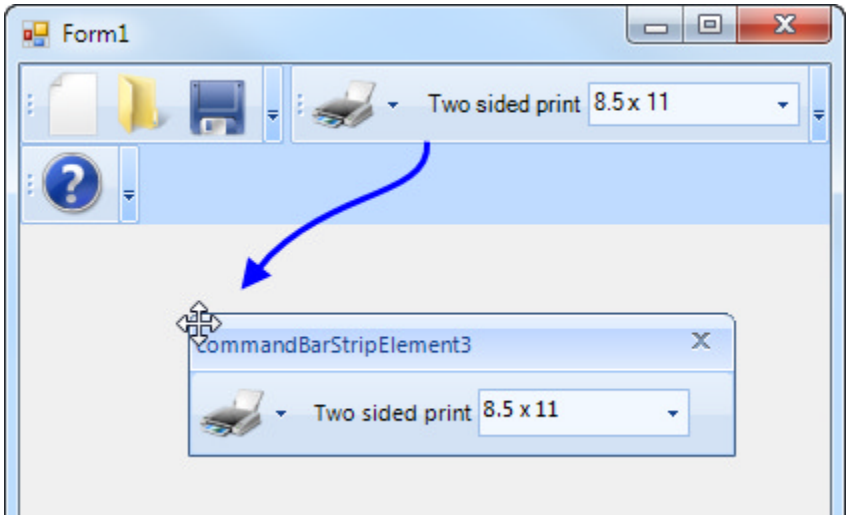
- Learn how to create rows and strip with elements in RadCommandBar
- Learn how to add items to the strip.
- Programmatically create and add rows, strips and items.
- Learn how to customize the appearance of the command bar items.

14.2 Introduction

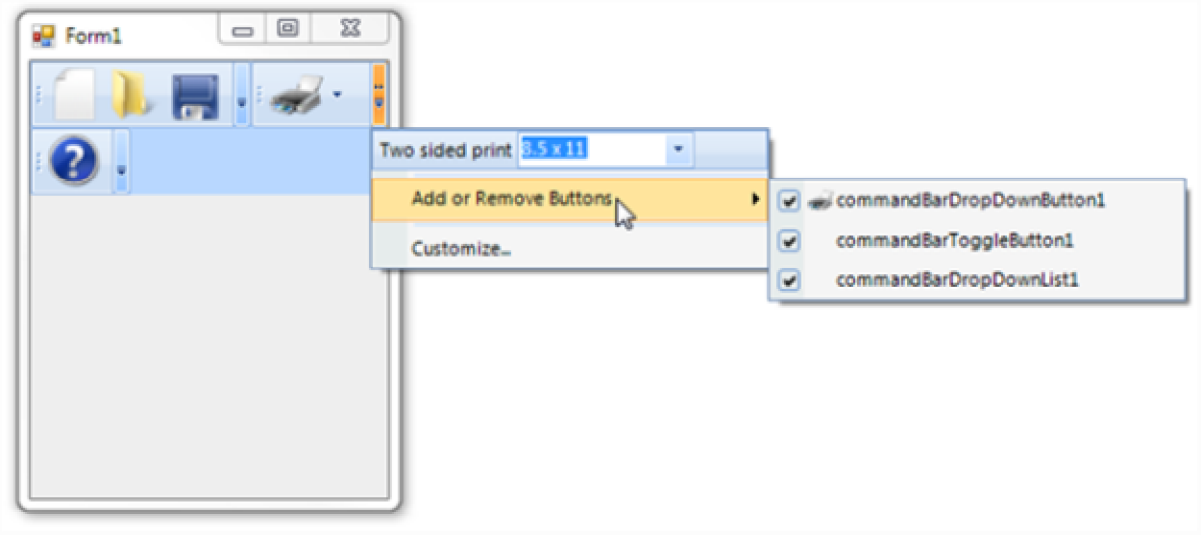
RadCommandBar is a fully theme-able tool strip that provides unprecedented flexibility. More than just a collection of buttons, **RadCommandBar** hosts any **RadControl**, including combo boxes, text boxes, split buttons, drop-down buttons, toggle buttons and more. CommandBar can be moved, rearranged and resized at run time for easy end-user customization. **RadCommandBar** can be styled to match any user interface using a predefined theme or a theme you design yourself using the Visual Style Builder.



RadCommandBar supports horizontal and vertical orientation. **CommandBarStrip** areas can be dragged within the **CommandBarRows** or it can be floated off the form.



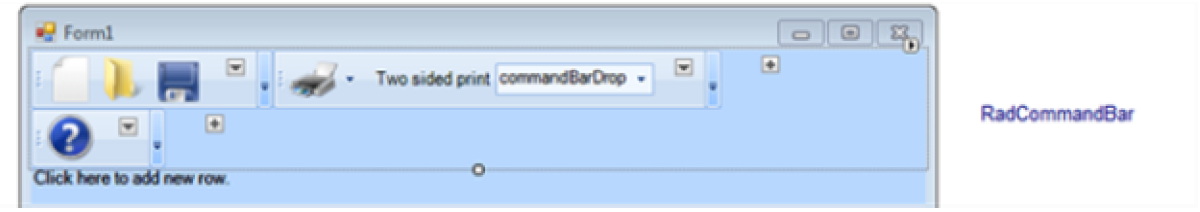
The overflow button automatically displays items that don't have the real estate to display by default. The end user can also customize the strips by adding and removing buttons.



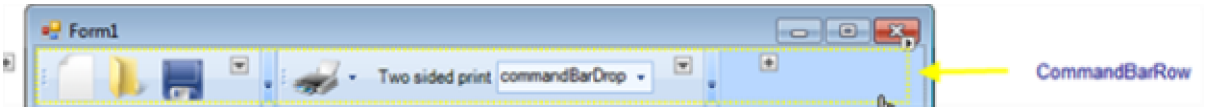
Structure

RadCommandBar is built up of four levels of nested components:

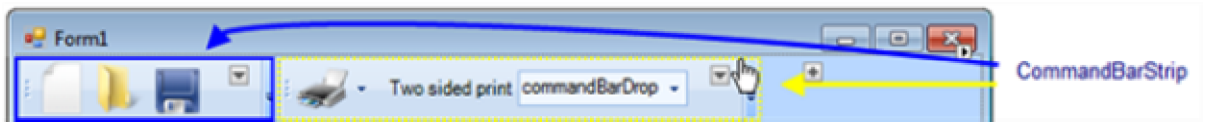
- The **RadCommandBar** control contains one or more elements.



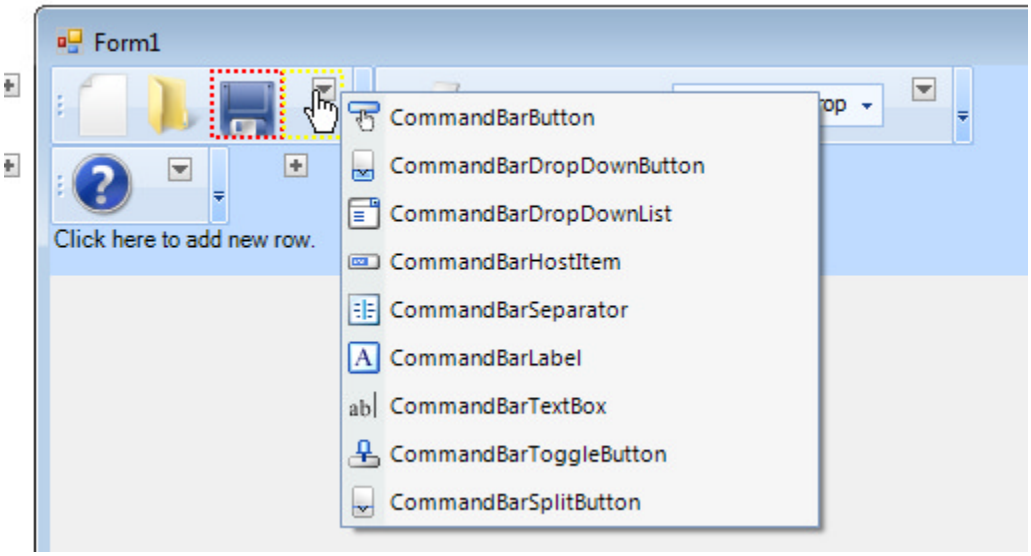
- The **CommandBarRowElement** hosts CommandBarStrip elements.



- The **CommandBarStripElement** represents a horizontal band within the RadCommandBar where command bar items are displayed. CommandBarStripElement is associated with a particular CommandBarRowElement but it may be relocated at design or at run time.

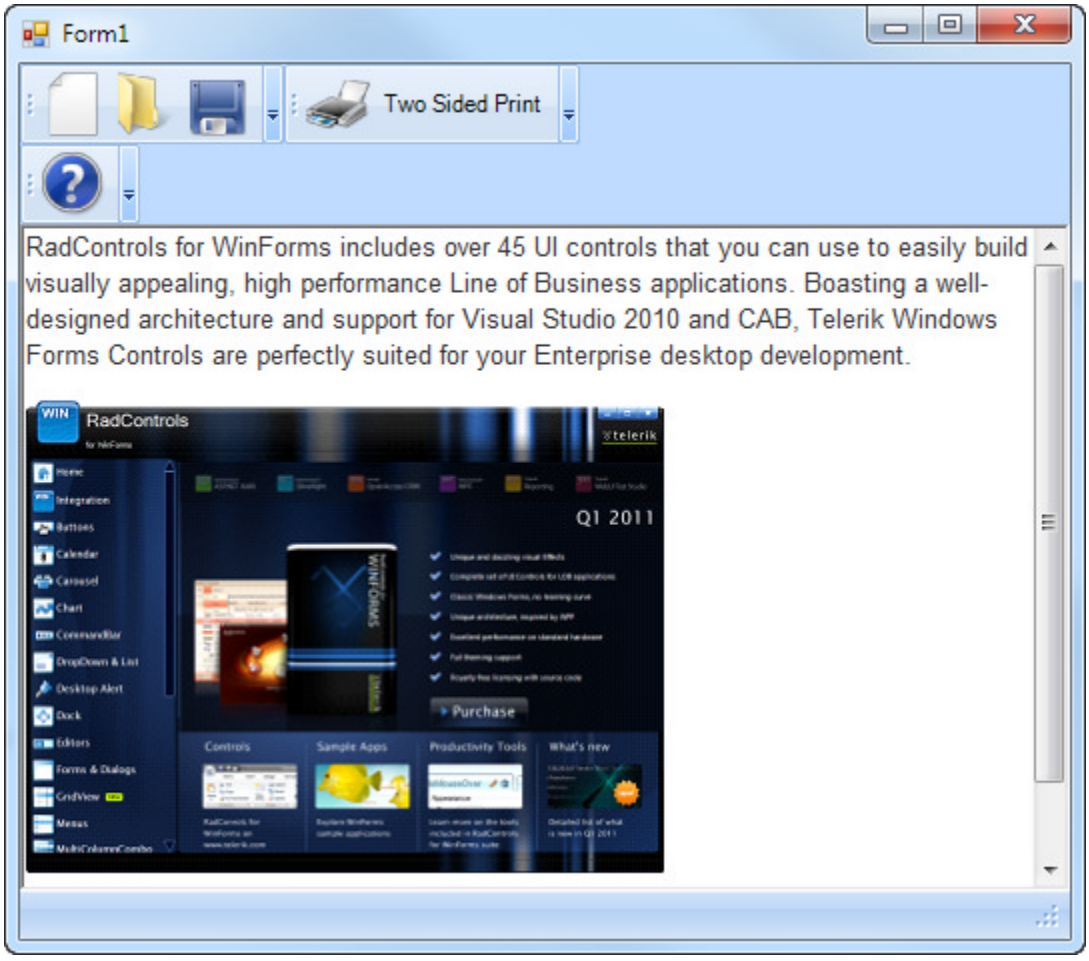


- The **CommandBarItems** represents a items within a CommandBarStrips. The individual items are shown in the drop down list in the screenshot below. Of course, custom controls can be hosted too.



14.3 Getting Started

This tutorial demonstrates creating a strips with multiple rows and multiple types of items and elements within those strips. The project takes the form of a rich text editor with limited New, Open and Save functionality.



Preparing the Project



You can find the complete source for this project at:

\CommandBar\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. Add a standard ImageList component to the form and name it "illcons". Use the ImageList Smart Tag to set the Image Size to "32, 32", and the Image Bit Depth to "Depth32Bit". Click the ImageList Smart Tag "Choose Images" option and add images to represent the ideas below.
    - New
    - Open
    - Save
    - Print
    - Help



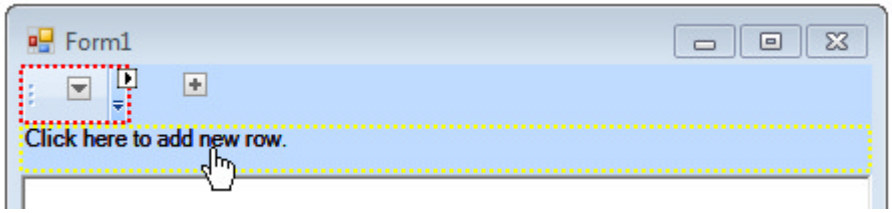
You can typically find images in the Visual Studio Common7\VS2008ImageLibrary\1033 directory in "VS2008ImageLibrary.zip".

Form Setup

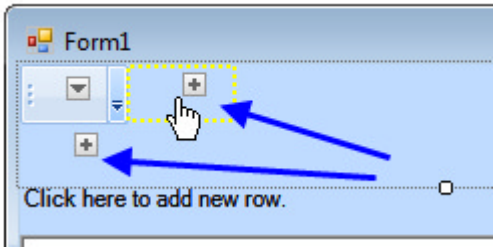
1. From the Toolbox, add a **RadCommandBar** to the form. Point the **ImageList** property at the ImageList component that you added earlier.
2. Add a **RadStatusStrip** to the form. It will automatically dock to the bottom. Point the **ImageList** property at the ImageList component that you added earlier.
3. Drop a standard Windows **RichTextBox** control below the RadCommandBar. Set the **Dock** property to "Fill".

CommandBar Configuration

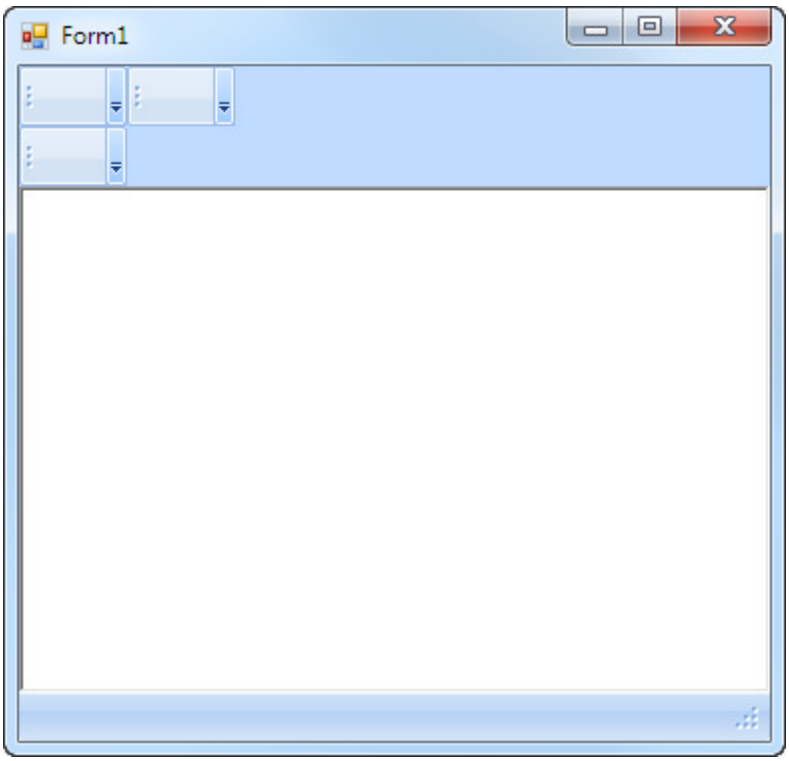
1. When RadCommandBar was created it automatically populates itself with one CommandBarRowElement and one CommandBarStripElement.
2. In design-time click the "*Click here to add new row*." to create second row in RadCommandBar.



3. Click the plus image next to the first strip on the first row to create second CommandBarStrip on the first row. Also click the plus sign on the second row to create strip element there too.

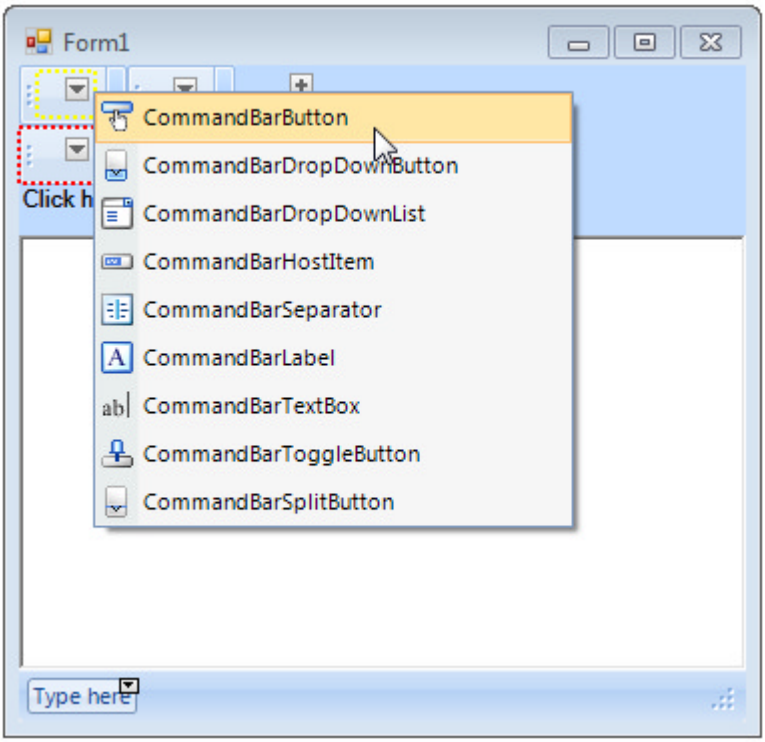


4. The project should look something like the example below.



Add Items

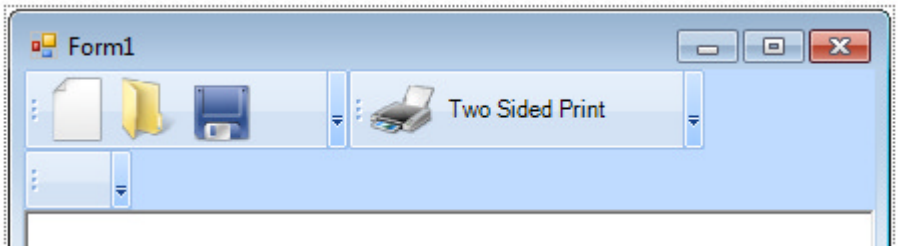
1. Click the first CommandBarStripElement and from the drop down select **CommandBarButton**. Set the following **CommandBarButton** properties:



- **Name** = "btnNew"
- **ToolTipText** = "File New"
- **ImageIndex** = <ImageList image that represents "New">

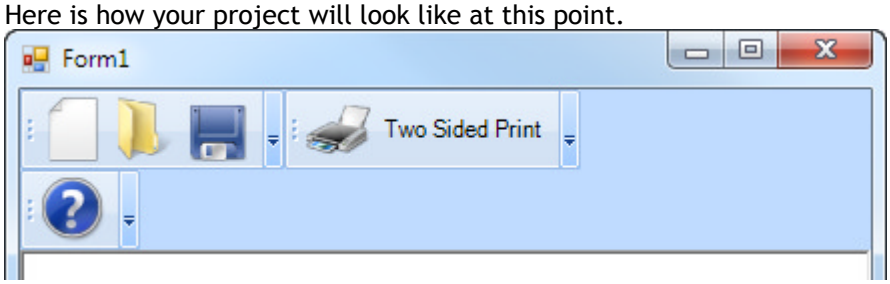
2. Create another **CommandBarButton** from the dropdown and set the following properties:
   - **Name** = "btnOpen"
   - **ToolTipText** = "File Open"
   - **ImageIndex** = <ImageList image that represents "Open">
3. Create another **CommandBarButton** from the dropdown menu and set following properties:
   - **Name** = "btnSave"
   - **ToolTipText** = "File Save"
   - **ImageIndex** = <ImageList image that represents "Save">
4. Click the second strip element amd create a **CommandBarButton** with the following properties:
   - **Name** = "btnPrint"
   - **ToolTipText** = "Print"
   - **ImageIndex** = <ImageList image that represents "Print">
5. Next to the print button add **CommandBarToggleButton**. Set the following properties:
   - **Name** = "tbTwoSidedPrint"
   - **ToolTipText** = "Two Sided Print"
   - **Text** = "Two Sided Print"
   - **DrawText** = true
   - **Image** = ""

The project at this point should look like this:



6. On the second row of add a new **CommandBarButton** with the following properties
   - **Name** = "btnHelp"
   - **ToolTipText** = "Help"
   - **ImageIndex** = <ImageList image that represents "Help">

   Here is how your project will look like at this point.



Add Code for New, Open and Save Functionality

1. Double-click the New, Save and Open buttons to create event handlers for each button. Add the code below to enable corresponding functionality for each button.

   The New button simply clears the RichTextBox. Both Save and Open use a private method GetStreamType () that infers a file type based on file extension. The Save button Click handler uses a standard Windows SaveFileDialog component, sets the filter to use either text or rich text file formats, shows the dialog and if the user clicks OK the file is saved to disk. The Open button follows the same pattern as Save but adds a check for file length before calling the OpenFileDialog LoadFile() method.

   [VB] Button Click Event Handlers

```
Private Sub btnNew_Click(ByVal sender As System.Object, ByVal e As System.EventArgs) Hand
btnNew.Click
RichTextBox1.Clear()
End Sub
Private Sub btnOpen_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnOpen.Click
Dim dialog As New OpenFileDialog()
dialog.Filter = "*.txt,*.rtf|*.txt;*.rtf"
If dialog.ShowDialog() = DialogResult.OK Then
Dim fileName As String = dialog.FileName
If fileName.Length > 0 Then
```

```
RichTextBox1.LoadFile(fileName, GetStreamType(fileName))
End If
End If
End Sub
Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnSave.Click
Dim dialog As New SaveFileDialog()
dialog.Filter = "*.txt,*.rtf|*.txt;*.rtf"
If dialog.ShowDialog() = DialogResult.OK Then
Dim fileName As String = dialog.FileName
richTextBox1.SaveFile(fileName, GetStreamType(dialog.FileName))
End If
End Sub
Private Function GetStreamType(ByVal fileName As String) As RichTextBoxStreamType
Return If(fileName.EndsWith("rtf"), RichTextBoxStreamType.RichText,
RichTextBoxStreamType.PlainText)
End Function
```

[C#] Button Click Event Handlers

```
private void btnNew_Click(object sender, EventArgs e)
{
  richTextBox1.Clear();
}
private void btnOpen_Click(object sender, EventArgs e)
{
  OpenFileDialog dialog = new OpenFileDialog();
  dialog.Filter = "*.txt,*.rtf|*.txt;*.rtf";
  if (dialog.ShowDialog() == DialogResult.OK)
  {
    string fileName = dialog.FileName;
    if (fileName.Length > 0)
    {
      richTextBox1.LoadFile(fileName, GetStreamType(fileName));
    }
  }
}
private void btnSave_Click(object sender, EventArgs e)
{
  SaveFileDialog dialog = new SaveFileDialog();
  dialog.Filter = "*.txt,*.rtf|*.txt;*.rtf";
  if (dialog.ShowDialog() == DialogResult.OK)
  {
    string fileName = dialog.FileName;
    richTextBox1.SaveFile(fileName, GetStreamType(dialog.FileName));
  }
}
private RichTextBoxStreamType GetStreamType(string fileName)
```

```
{
  return fileName.EndsWith("rtf") ?
      RichTextBoxStreamType.RichText : RichTextBoxStreamType.PlainText;
}
```

Test the Application

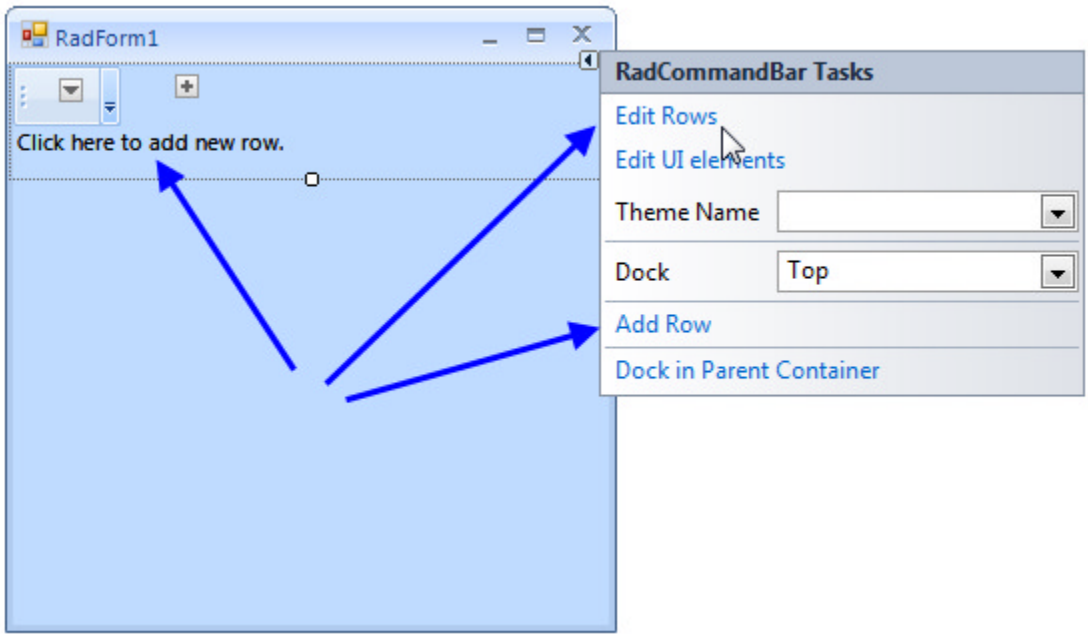
1. Press **Ctrl-F5** to run the application.
2. Click the "Open" button and use the Open dialog to open either a .txt or an .rtf file.
3. Make changes to the text.
4. Click the "Save" button and use the Save As dialog to save the file as either text or rich text.
5. Click the "New" button to clear the text.

14.4 Using the Design Time Interface

There are two routes to designing the nested set of collections that make up the RadCommandBar. You can use tje design surface and with simple clicks design the desired control or use Smart Tag to build each collection. Or you can use the Properties window and start with the CommandBarItems collection and drill down from there.

Adding rows

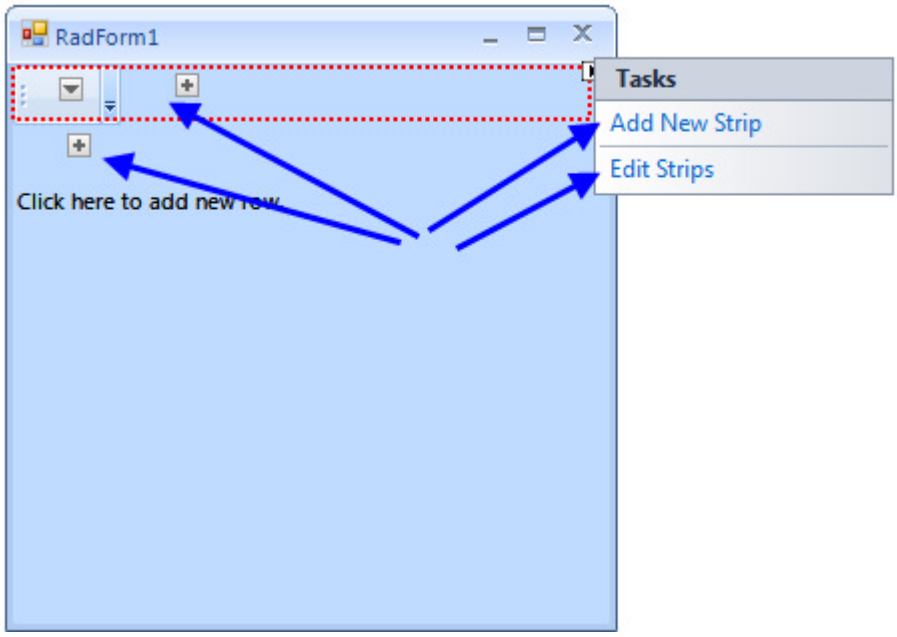
You can add CommandBarRowElements by clicking on the "Click here to add new row" on the design time surface, by selecting "Add Row" from RadCommandBar Smart Tag or by editing the CommandBarRowElementCollection again from the Smart Tag.



Adding strips

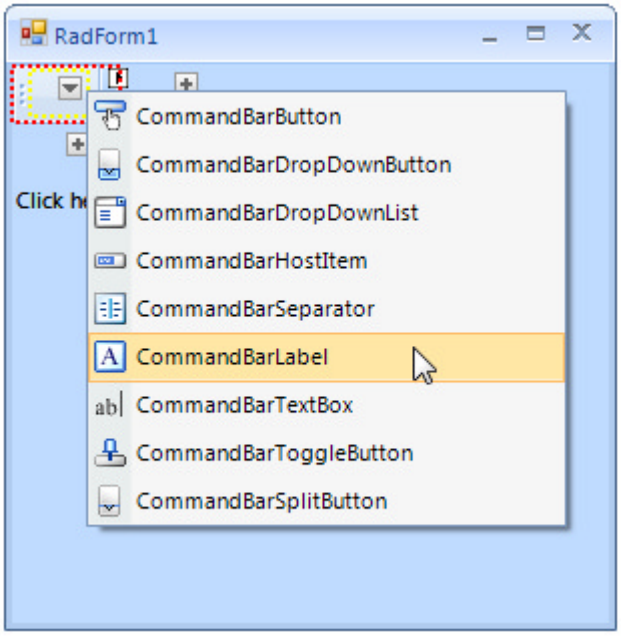
Strips are added by clicking the plus sing on the design time surface, by selecting "Add New Strip" from the CommandBarRowElement Smart Tag, or by editing the CommandBarStripElementCollection from the Smart Tag

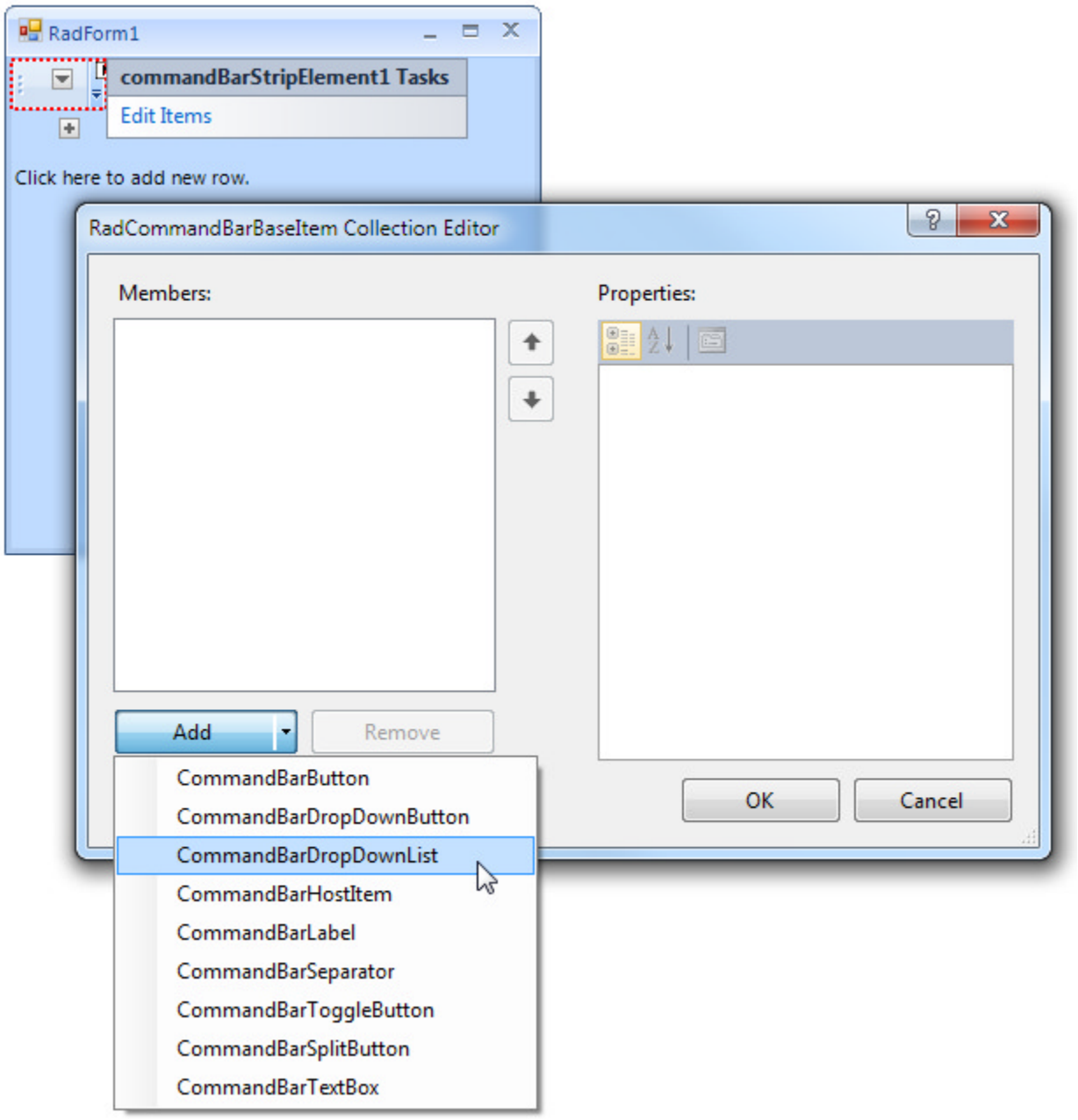
or the row element.



Adding items

Visually designing the RadCommandBar begins at the design time surface where you can click the dropdown arrow button to bring up the drop down with available CommandBarItems or use the Smart tag of the strip element and click the "Edit Items" to bring up the RadCommandBarBaseItemCollecition editor.

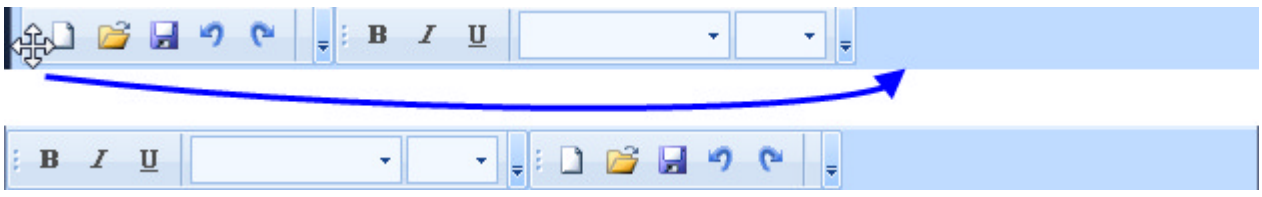




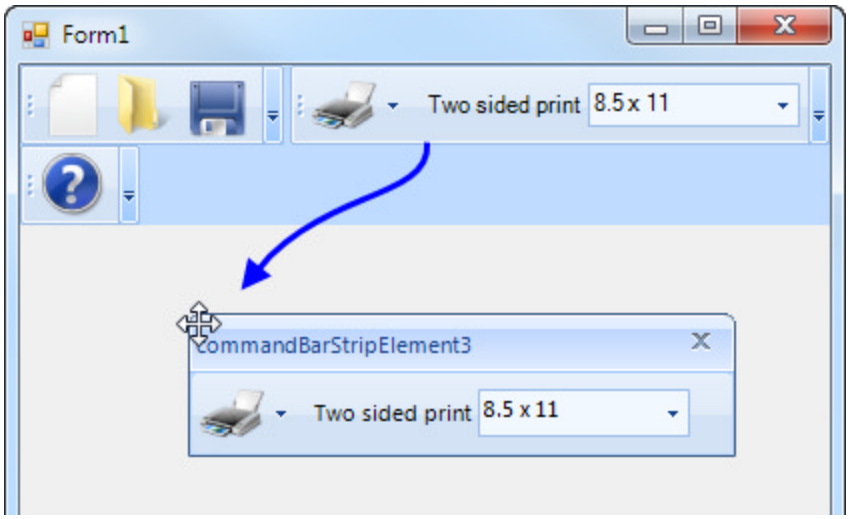
In either case, all the elements and items are available for editing in the Properties window.

RadCommandBar Properties

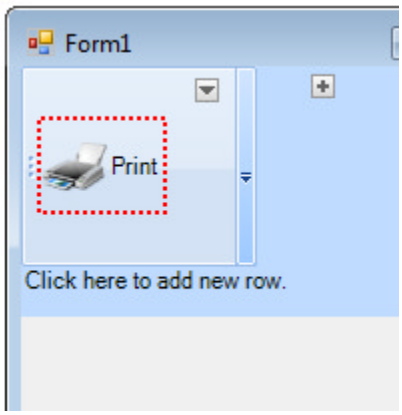
EnableDragging property allows the user to move the CommandBarStripElement within the strip using the element's drag handles.



The EnableDragging setting alone doesn't allow the user to drag the element off the strip. For that you need to set the **EnableFloating** property to true.

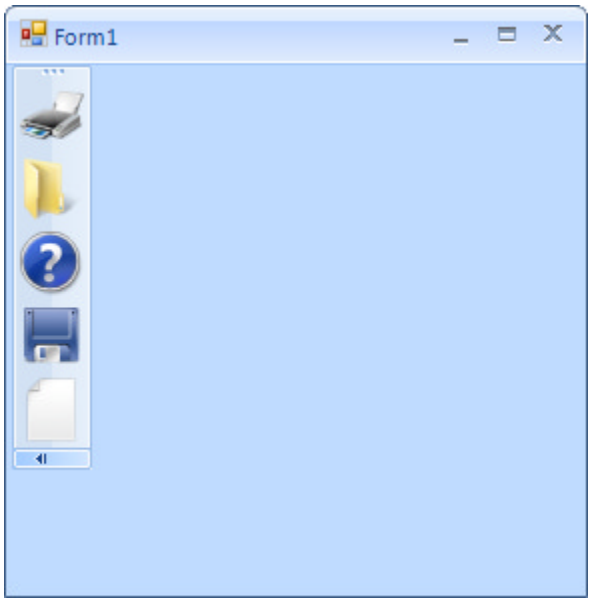


To make the command bar items sit in larger strip, use the **MinSize** property. The screenshot below shows a CommandBarButton with a MinSize of 50, 50, the CommandBarStripElement that the button is sitting on has MinSize 100, 100.



Modifing the visibility of the **OverflowButton** is set from the Properties window of the desired strip. Expand the OverflowButton and set the Visibility property to Collapsed to hide it.

The orientation of RadCommandBar it set automatically, when the **Dock** property is changed.



14.5 Programming

Building RadCommandBar Items Programmatically

Building strips in code follows the same pattern as working in the designer. The hierarchy is:

RadCommandBar
  CommandBarRowElement[]
    CommandBarStripElement[]
      CommandBarItem[]

This next demo shows how to add everything in RadCommandBar. The empty command bar will be docked to the top of the form. A number of icons have been loaded into the Solution Explorer Properties | Resources and will be used to populate items.



You can find the complete source for this project at:

\CommandBar\<VB|CS>\Programming

In the form load, create an instance of **RadCommandBar** and add it to the controls collection of the form. Create a **CommandBarRowElement**, set its **Dock** property to "Top" and add it to the Rows collection of RadCommandBar. Then create an instace of **CommandBarStripElement** and add it to the **Strips** collection of the created row element. Finally call the LoadItems method, which we will implement next.

[VB] Building the CommandBar

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handles
MyBase.Load
Dim radCommandBar1 As New RadCommandBar()
radCommandBar1.Dock = DockStyle.Top
radCommandBar1.Name = "radCommandBar1"
Me.Controls.Add(radCommandBar1)
Dim commandBarRowElement1 As New CommandBarRowElement()
commandBarRowElement1.Name = "commandBarRowElement1"
radCommandBar1.Rows.Add(commandBarRowElement1)
```

```
Dim commandBarStripElement1 As New CommandBarStripElement()
commandBarStripElement1.Name = "commandBarStripElement1"
commandBarRowElement1.Strips.Add(commandBarStripElement1)
End Sub
```

[C#] Building the CommandBar

```
private void Form1_Load(object sender, EventArgs e)
{
RadCommandBar radCommandBar1 = new RadCommandBar();
radCommandBar1.Dock = DockStyle.Top;
radCommandBar1.Name = "radCommandBar1";
this.Controls.Add(radCommandBar1);
CommandBarRowElement commandBarRowElement1 = new CommandBarRowElement();
commandBarRowElement1.Name = "commandBarRowElement1";
radCommandBar1.Rows.Add(commandBarRowElement1);
CommandBarStripElement commandBarStripElement1 = new CommandBarStripElement();
commandBarStripElement1.Name = "commandBarStripElement1";
commandBarRowElement1.Strips.Add(commandBarStripElement1);
}
```

Add five images as resources to the project. Name the images *New*, *Open*, *Save*, *Print* and *Help*.

The LoadItems () enumerates resources in the current project. The image name is set as ToolTipText for the corresponding item and the image itself it set as Image for the item. Additionally, we are adding Click event handlder for each item.

[VB] Loading items to CommandBarStripItem

```
Private Sub LoadItems(ByVal stripElement As CommandBarStripElement)
' list the resources in Properties.Resources
Dim resourceSet As ResourceSet = Programming.My.Resources.ResourceManager.GetResourceSet
(CultureInfo.CurrentCulture, True, True)
Dim enumerator As IDictionaryEnumerator = resourceSet.GetEnumerator()
' for each image resources, retrieve the image name and object,
' create and populate the list items
While enumerator.MoveNext()
Dim item As New CommandBarButton()
item.Image = DirectCast(enumerator.Value, Bitmap)
'the name have to be set since the save and load layout mechanism is using it
item.Name = enumerator.Key.ToString().Replace("_", " ")
item.ToolTipText = item.Name
'sets the text that will be displayed in the overflow button
item.DisplayName = item.Name
item.TextImageRelation = TextImageRelation.ImageAboveText
AddHandler item.Click, AddressOf item_Click
stripElement.Items.Add(item)
End While
End Sub
```

[C#] Loading items to CommandBarStripItem

```
private void LoadItems(CommandBarStripElement stripElement)
{
// list the resources in Properties.Resources
ResourceSet resourceSet = Properties.Resources.ResourceManager.GetResourceSet(
CultureInfo.CurrentCulture, true, true);
IDictionaryEnumerator enumerator = resourceSet.GetEnumerator();
// for each image resources, retrieve the image name and object,
// create and populate the list items
```

```
while (enumerator.MoveNext())
{
CommandBarButton item = new CommandBarButton();
item.Image = (Bitmap)enumerator.Value;
//the name have to be set since the save and load layout mechanism is using it
item.Name = enumerator.Key.ToString().Replace('_', ' ');
item.ToolTipText = item.Name;
//sets the text that will be displayed in the overflow button
item.DisplayName = item.Name;
item.TextImageRelation = TextImageRelation.ImageAboveText;
item.Click+=new EventHandler(item_Click);
stripElement.Items.Add(item);
}
}
```

The Click event handler demonstrates a few useful techniques. First, you can get the parent control for any element through the ElementTree.Control property. Second, when you use the RadMessageBox Show() method (we'll talk more about RadMessageBox in the Forms chapter). And lastly, you can call an overload of the Show() method that lets you display a bitmap; in this case a bitmap of the CommandBarButton Image.

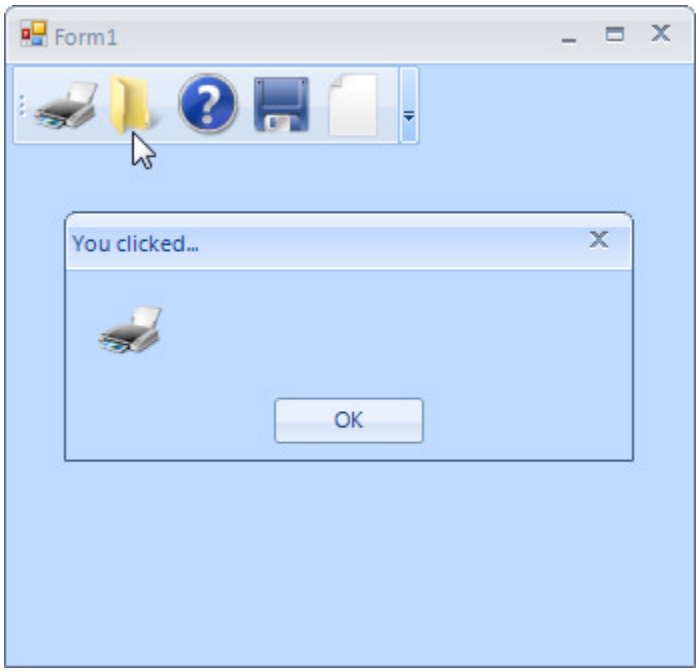
[VB] Handling the Click Event

```
Private Sub item_Click(ByVal sender As Object, ByVal e As EventArgs)
Dim buttonElement As CommandBarButton = TryCast(sender, CommandBarButton)
Dim commandBar As RadCommandBar = TryCast(buttonElement.ElementTree.Control, RadCommandBar)
Dim bitmap As New Bitmap(buttonElement.Image)
RadMessageBox.Show(Me, buttonElement.Text, "You clicked...", MessageBoxButtons.OK, bitmap)
End Sub
```

[C#] Handling the Click Event

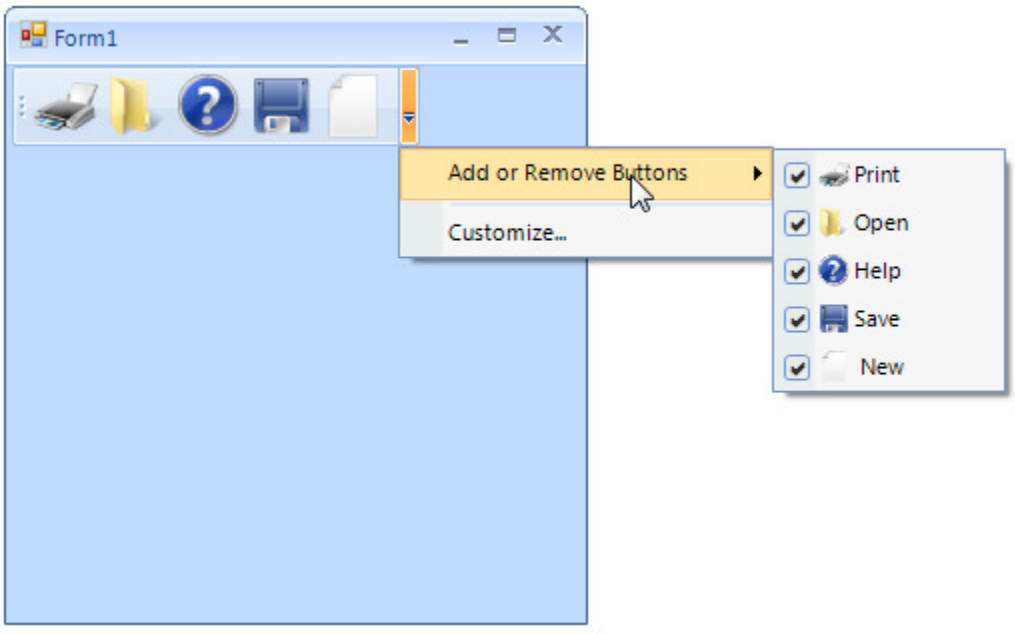
```
void item_Click(object sender, EventArgs e)
{
CommandBarButton buttonElement = sender as CommandBarButton;
RadCommandBar commandBar = buttonElement.ElementTree.Control as RadCommandBar;
Bitmap bitmap = new Bitmap(buttonElement.Image);
RadMessageBox.Show(this, buttonElement.Text, "You clicked...", MessageBoxButtons.OK,
bitmap);
}
```

When the example application runs, series of CommandBarButtons are loaded to the CommandBarStripElement.



Saving and Loading RadCommandBar Layout

Your users can customize the strip using the drop down list and adding or removing buttons from the list. Its natural that they would want their customizations to be preserved when they restart the application.



The example below shows the general approach for saving and loading the tool strip layout. You will probably want to make changes in the storage medium used to contain the XML (e.g. database, isolated storage, etc.). You will also need additional safety code to prevent or handle partial serialization resulting in corrupt XML.

Lets continue with the previous project. Move the instance of the RadCommandBar outside the Load event handler in order to be able to access it from other places. Add two RadButtons to the form and set their text to "Save Layout" and "Load Layout" and their names to btnSave and btnLoad respectively. Click both of them in

design time to create event handlers. Here is the code that you need to add to save the layout of RadCommandBar

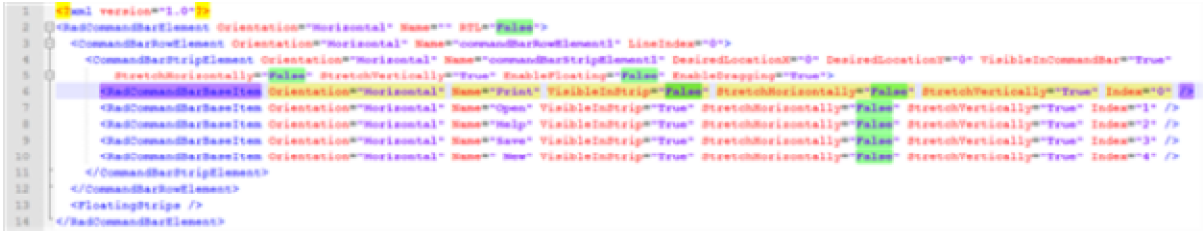
[VB] Save RadCommandBar Layout

```
Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnLoad.Click
Dim s As String = "default.xml"
Dim dialog As New OpenFileDialog()
dialog.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*"
dialog.Title = "Select a xml file"
If dialog.ShowDialog() = DialogResult.OK Then
s = dialog.FileName
End If
Me.radCommandBar1.CommandBarElement.LoadLayout(s)
End Sub
```

[C#] Save RadCommandBar Layout

```
private void btnSave_Click(object sender, EventArgs e)
{
string s = "default.xml";
SaveFileDialog dialog = new SaveFileDialog();
dialog.Filter =
"xml files (*.xml)|*.xml|All files (*.*)|*.*";
dialog.Title = "Select a xml file";
if (dialog.ShowDialog() == DialogResult.OK)
{
s = dialog.FileName;
}
this.radCommandBar1.CommandBarElement.SaveLayout(s);
}
```

The resulting XML contains the entire control definition, i.e. properties, items collections, the whole enchilada. One of the button elements is hidden by user customization and in the XML the element's VisibleInStrip property shows as "False".



To recreate the RadCommandBar state at the time it was saved, pass the XML file to the LoadLayout method of the control.

[VB] Restoring the Layout

```
Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnLoad.Click
Dim s As String = "default.xml"
Dim dialog As New OpenFileDialog()
dialog.Filter = "xml files (*.xml)|*.xml|All files (*.*)|*.*"
dialog.Title = "Select a xml file"
If dialog.ShowDialog() = DialogResult.OK Then
s = dialog.FileName
End If
```

```
Me.radCommandBar1.CommandBarElement.LoadLayout(s)
End Sub
```

[C#] Restoring the Layout

```
private void btnLoad_Click(object sender, EventArgs e)
{
string s = "default.xml";
OpenFileDialog dialog = new OpenFileDialog();
dialog.Filter =
"xml files (*.xml)|*.xml|All files (*.*)|*.*";
dialog.Title = "Select a xml file";
if (dialog.ShowDialog() == DialogResult.OK)
{
s = dialog.FileName;
}
this.radCommandBar1.CommandBarElement.LoadLayout(s);
}
```

LoadLayout and SaveLayout can be called in the Form Load and FormClosed event handlers so that state is stored automatically.

14.6 Summary

In this chapter you learned how to build a command bar strips at design time and programmatically at run time. You explored floating and overflow behavior. You learned how to configure strip dimensioning, orientation and layout. Finally, you learned how to save and load the RadCommandBar layout.

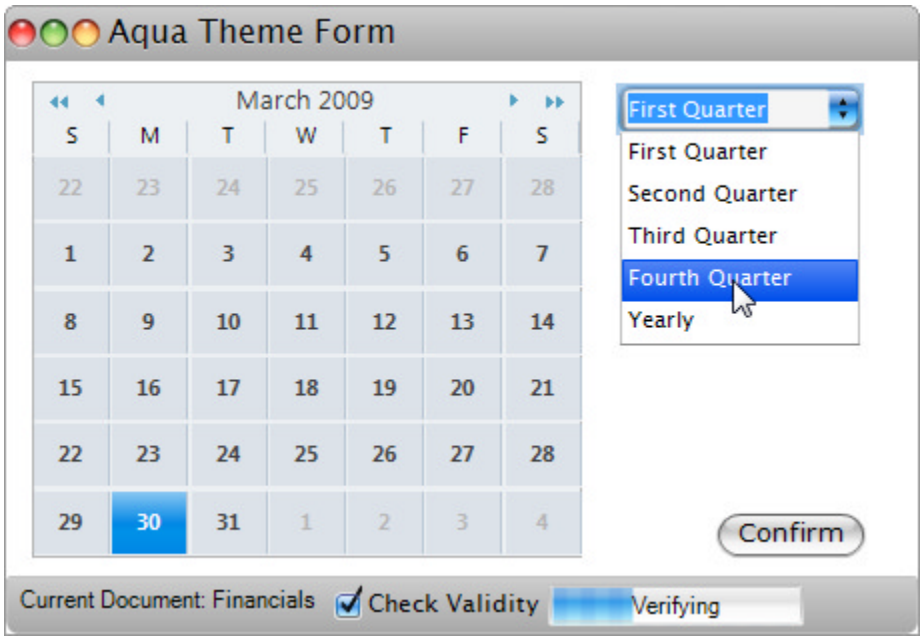
15 Forms

15.1 Objectives

- Learn how the special RadForm, ShapedForm and RadRibbonForm classes can be inherited from to create themable, custom shaped forms.
- Learn how Visual Studio project item templates are used to create instances of each form class.
- See how RadTitleBar is used with RadForm as a replacement for the built-in Windows form title bar.
- Learn how to use the RadMessageBox as a themable replacement for the standard MessageBox.

15.2 Introduction

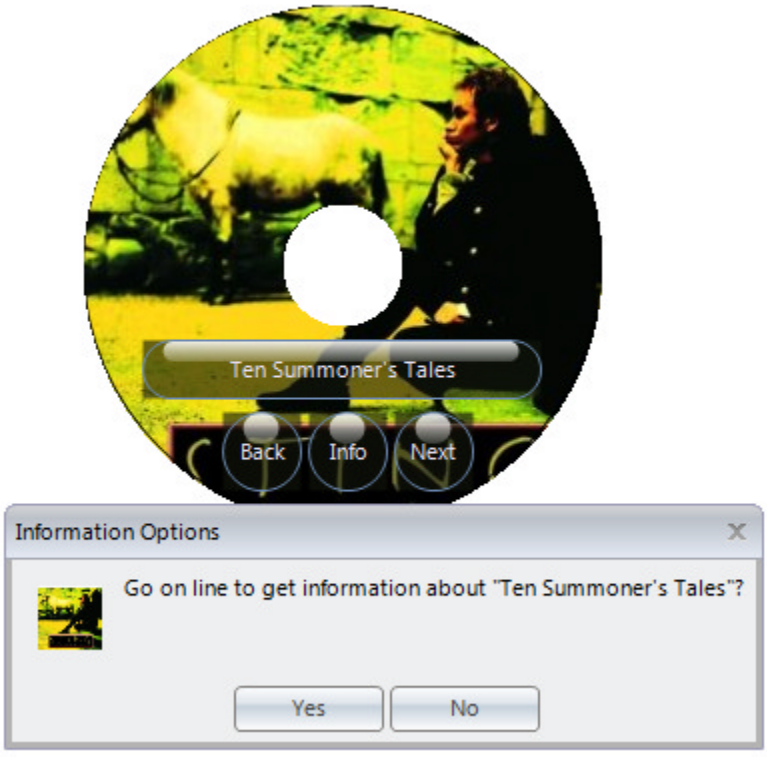
The RadForm control allows you to design a styled Windows form with rounded corners. Use the pre-defined themes to quickly build a user interface that has a sharp, unified look-and-feel. The control comes with integrated TitleBar, support for Multiple Document Interface (MDI) applications and languages that require right-to-left layout. The RadControls for WinForms installation includes templates for RadForm and ShapedForm and specially configured RadForm types RadRibbonBarForm and RadAboutBox for greater productivity.



Use the ShapedForm to design and display a Windows form with any conceivable shape.

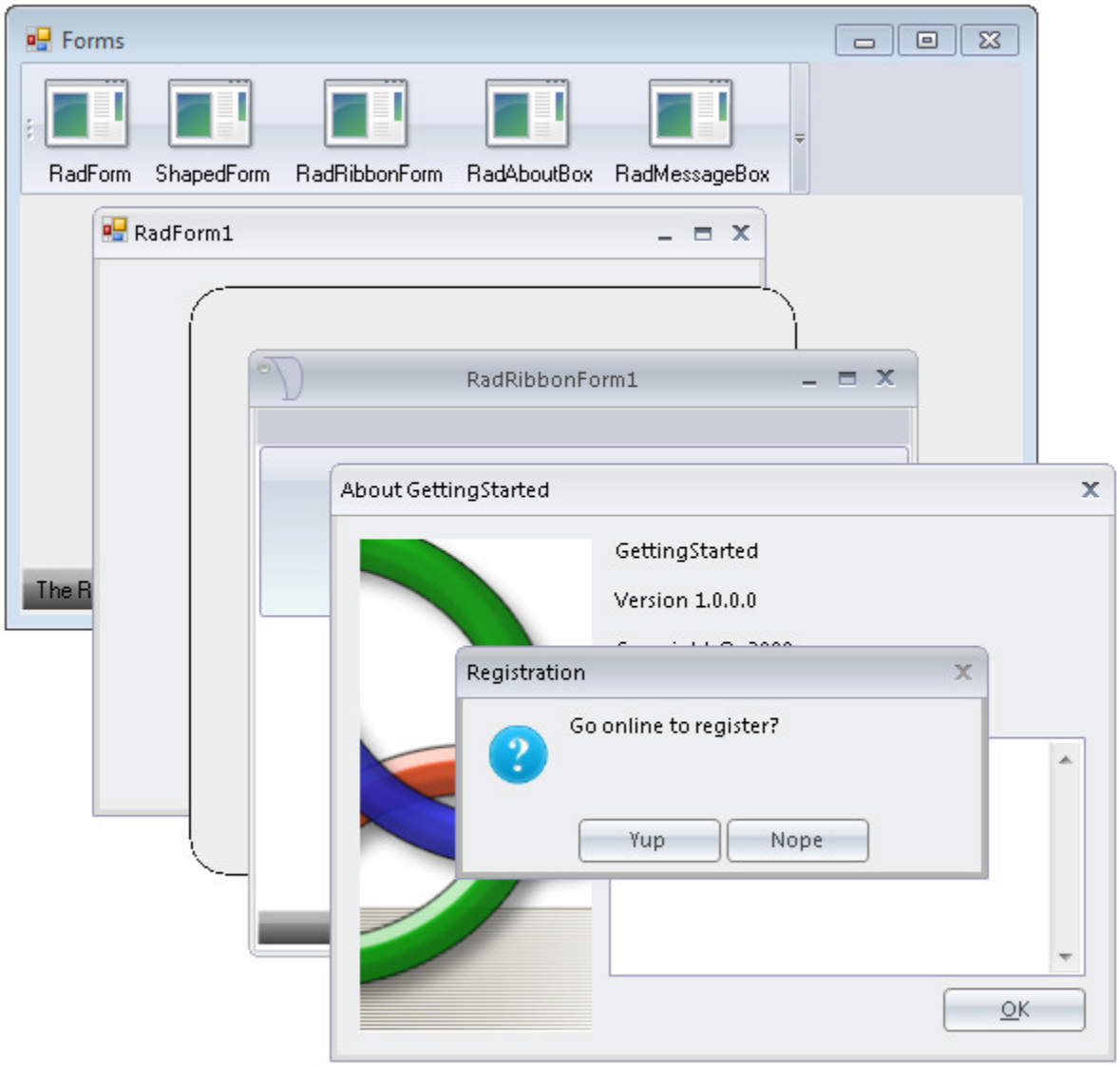


Use the handy RadMessageBox class to display modal dialog boxes that show status information or a requests for confirmation. RadMessageBox dialogs can be configured to display with captions, message text, icons and button sets.



15.3 Getting Started

This tutorial demonstrates creating the available Telerik form types on the fly from a main form.



Preparing the Project



You can find the complete source for this project at:

\Forms\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. Drag Office2007SilverTheme from the toolbox to the form. **Note**: this will be the main form.
3. From the Toolbox, add a **RadCommandBar** to the form. Add four elements:
   - **CommandBarButton**: **Name** = "btnRadForm", **TextImageRelation** = "ImageAboveText", **Text** = "RadForm", **Image** = <import an image that represents a form>.
   - **CommandBarButton**: **Name** = "btnShapedForm", **TextImageRelation** = "ImageAboveText", **Text** = "ShapedForm", **Image** = <import an image that represents a form>.
   - **CommandBarButton**: **Name** = "btnRadRibbonBar", **TextImageRelation** = "ImageAboveText", **Text** = "RadRibbonBar", **Image** = <import an image that represents a form>.

- **CommandBarButton**: **Name** = "btnRadAboutBox", **TextImageRelation** = "ImageAboveText", **Text** = "RadAboutBox", **Image** = <import an image that represents a form>.
- **CommandBarButton**: **Name** = "btnRadMessageBox", **TextImageRelation** = "ImageAboveText", **Text** = "RadMessageBox", **Image** = <import an image that represents a form>.

4. Add a **RadStatusStrip** to the form. It will automatically dock to the bottom of the form. Add a single **RadLabelElement** to the status strip and set **Name** = "lblStatus", **Spring** = "True".

Adding New Forms to the Project

1. Add a **RadForm** to the project:
   - In the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
   - Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
2. Add a **ShapedForm** to the project:
   - In the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
   - Select the "Telerik ShapedForm" template and click the **Add** button to close the dialog.
3. Add a **RadRibbonBarForm** to the project:
   - In the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
   - Select the "Telerik RadRibbonBar" template and click the **Add** button to close the dialog.
4. Add a **RadAboutBox** to the project:
   - In the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
   - Select the "Telerik RadAboutBox" template and click the **Add** button to close the dialog.

Adding Code to the Main Form

1. Double-click the "RadForm" button and add code to the Click event handler.

   Create an instance of the RadForm you added to the project earlier, hook up the Activated event to a handler you will write later and call the Show() method. This is the basic pattern we will follow for the Click event handlers for each form type.

   [VB] Handling the "RadForm" Button Click Event

```
Private Sub btnRadForm_Click(sender As Object, e As EventArgs)
 Dim radForm1 As New RadForm1()
 radForm1.Activated += New EventHandler(formActivated)
 radForm1.Show()
End Sub
```

   [C#] Handling the "RadForm" Button Click Event

```
private void btnRadForm_Click(object sender, EventArgs e)
{
 RadForm1 radForm1 = new RadForm1();
 radForm1.Activated += new EventHandler(formActivated);
 radForm1.Show();
}
```

2. Create button Click event handlers for each of the remaining form types and add the code below:

   [VB] Handling the Remaining Button Click Events

```
Private Sub btnShapedForm_Click(sender As Object, e As EventArgs)
```

```
 Dim shapedForm1 As New ShapedForm1()
 shapedForm1.Activated += New EventHandler(formActivated)
 shapedForm1.Show()
End Sub
Private Sub btnRadRibbonForm_Click(sender As Object, e As EventArgs)
 Dim radRibbonForm1 As New RadRibbonForm1()
 radRibbonForm1.Activated += formActivated
 radRibbonForm1.Show()
End Sub
Private Sub btnRadAboutBox_Click(sender As Object, e As EventArgs)
 Dim radAboutBox1 As New RadAboutBox1()
 radAboutBox1.Activated += New EventHandler(formActivated)
 radAboutBox1.Show()
End Sub
Private Sub btnRadMessageBox_Click(ByVal sender As Object, ByVal e As EventArgs) Handles
btnMessageBox.Click
Dim result As DialogResult = RadMessageBox.Show("Go online to register?", "Registration",
MessageBoxButtons.YesNo, RadMessageIcon.Question, MessageBoxDefaultButton.Button1)
If result = DialogResult.Yes Then
' do something...
End If
End Sub
```

[C#] Handling the Remaining Button Click Events

```
private void btnShapedForm_Click(object sender, EventArgs e)
{
 ShapedForm1 shapedForm1 = new ShapedForm1();
 shapedForm1.Activated += new EventHandler(formActivated);
 shapedForm1.Show();
}
private void btnRadRibbonForm_Click(object sender, EventArgs e)
{
 RadRibbonForm1 radRibbonForm1 = new RadRibbonForm1();
 radRibbonForm1.Activated += formActivated;
 radRibbonForm1.Show();
}
private void btnRadAboutBox_Click(object sender, EventArgs e)
{
 RadAboutBox1 radAboutBox1 = new RadAboutBox1();
 radAboutBox1.Activated += new EventHandler(formActivated);
 radAboutBox1.Show();
}
private void btnRadMessageBox_Click(object sender, EventArgs e)
{
DialogResult result = RadMessageBox.Show(
"Go online to register?",
"Registration",
MessageBoxButtons.YesNo,
RadMessageIcon.Question,
MessageBoxDefaultButton.Button1);
if (result == DialogResult.Yes)
{
// do something...
}
}
```

3. Add the common form **Activated** event handler. The handler will show the name of the currently active form in the status strip.

   [VB] Handling the FormActivated Event

```
Sub formActivated(sender As Object, e As EventArgs)
 lblStatus.Text = "The " + (TryCast(sender, Control)).Name + " form is active"
End Sub
```

   [C#] Handling the FormActivated Event

```
void formActivated(object sender, EventArgs e)
{
 lblStatus.Text = "The " + (sender as Control).Name + " form is active";
}
```

4. Add a form Load event handler to set the theme for the entire application.

   [VB] Handling the FormLoad Event

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 ThemeResolutionService.ApplicationThemeName = "Office2007Silver"
End Sub
```

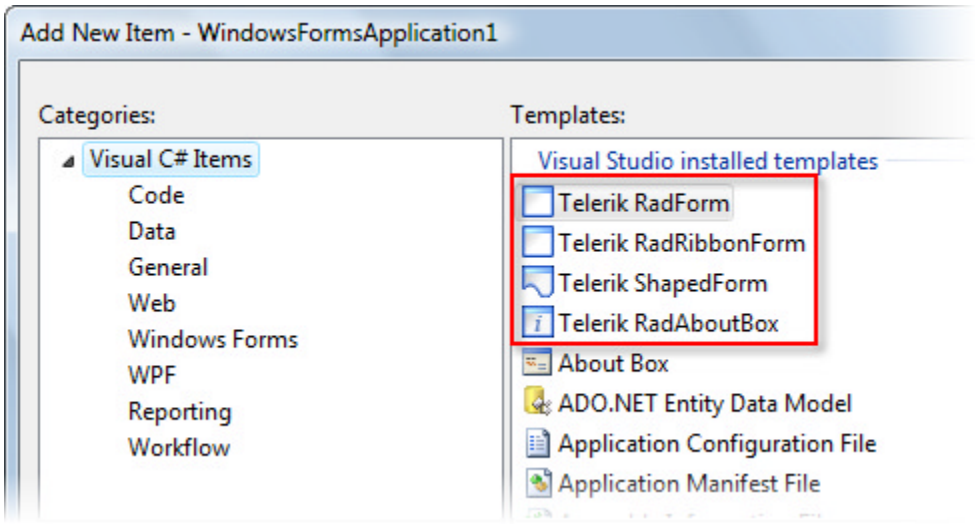
   [C#] Handling the FormLoad Event

```
private void Form1_Load(object sender, EventArgs e)
{
 ThemeResolutionService.ApplicationThemeName = "Office2007Silver";
}
```

5. Press **Ctrl-F5** to run the application. Show each of the types of forms. Notice how the status bar responds as each form is activated. Also notice that the theme is applied to all forms and controls within the application.

15.4 Using the Design Time Interface

The Visual Studio templates that install with RadControls for WinForms let you add **RadForm** and **ShapedForm** to your application without any coding steps. There are also two other specialized forms, **RadRibbonForm** that contains a built-in RadRibbonBar and **RadAboutBox** that is an enhanced, themeable version of the regular AboutBox.



15.5 Programming Rad Form and Message Box

Form Types Available in Visual Studio

RadForm

We have been using RadForm right along throughout this courseware to display a themed form with rounded edges. The shape of RadForm is predefined, and unlike the ShapedForm it cannot be changed. RadForm descends from RadFormBase and includes properties for **ThemeName** and **FormElement**. You can access the title bar and its buttons using the **FormElement.TitleBar** member. To customize the minimize, maximize and close buttons, use the **RadTitleBar.TitleBarElement MinimizeButton**, **MaximizeButton** and **CloseButton** members. Each button is a RadButtonElement type that includes properties to control text, image, and layout.

[VB] Using RadForm Properties

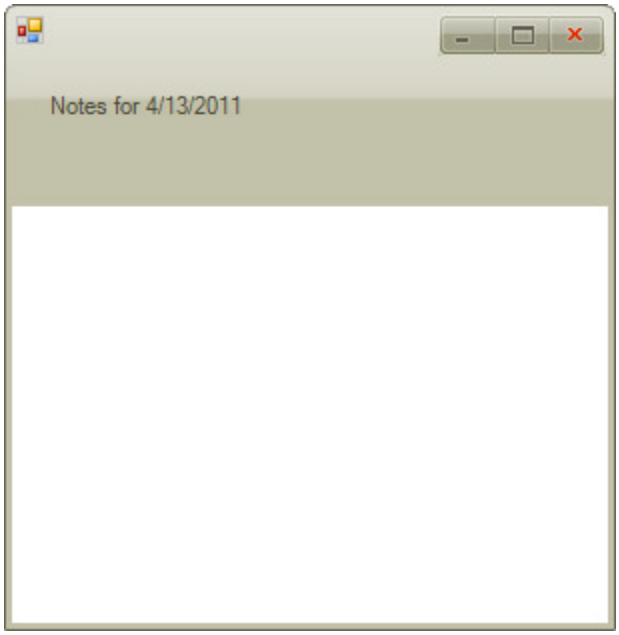
```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
Me.ThemeName = "Desert"
Me.FormElement.TitleBar.MinSize = new Size(0, 100)
Me.Text = "Notes for " + DateTime.Today.ToShortDateString()
Me.FormElement.TitleBar.MaximizeButton.Enabled = False
Me.FormElement.TitleBar.MinimizeButton.Enabled = False
End Sub
```

[C#] Using RadForm Properties

```
private void RadForm1_Load(object sender, EventArgs e)
{

this.ThemeName = "Desert";
this.FormElement.TitleBar.MinSize = new Size(0, 100);
this.Text = "Notes for " + DateTime.Today.ToShortDateString();
this.FormElement.TitleBar.MaximizeButton.Enabled = false;
this.FormElement.TitleBar.MinimizeButton.Enabled = false;
}
```

The code snippet above results in the form looking something like this screenshot:



Aero support is not currently available for RadForm.

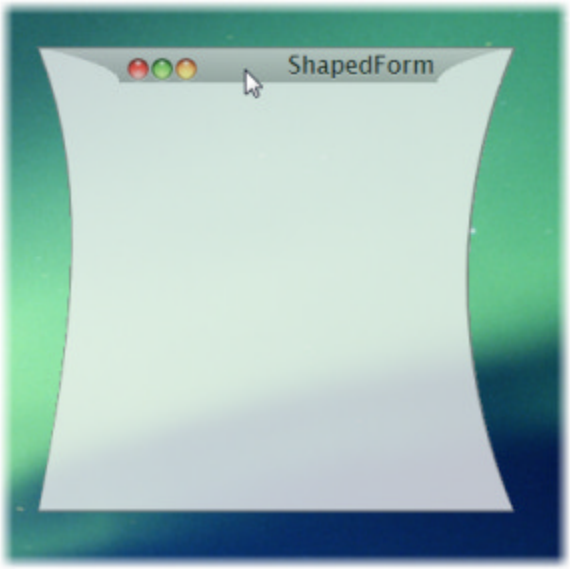
ShapedForm

Unlike RadForm, ShapedForm actually descends from a standard windows Form. Shaped form comes with a **RadTitleBar** so the user can move, close, minimize and maximize the form. Both the RadTitleBar and the ShapedForm by default each have their **Shape** property set to RoundRectShape components. You can also use one of the predefined shapes, i.e. **DonutShape**, **MediaShape**, **EllipseShape**, **QAShape**, **RoundRectShape**, **OfficeShape**, **TablEShape**, **TabOffice12Shape**, **TabVsShape**, **TrackBarDThumbShape**, **TrackBarUThumbShape**, **TrackBarLThumbShape**, and **TrackBarRThumbShape**. Or you can create your own custom shape using the **Shape Editor** tool.

In order to hide the default titlebar, the ShapedForm class has the FormBorderStyle property set to None while the RadForm FormBorderStyle property by default is set to Sizable. This may impact form behavior in some cases.

RadTitleBar is also available as a separate RadControl in the ToolBox that can be dropped on any standard Form, ShapedForm or RadForm.

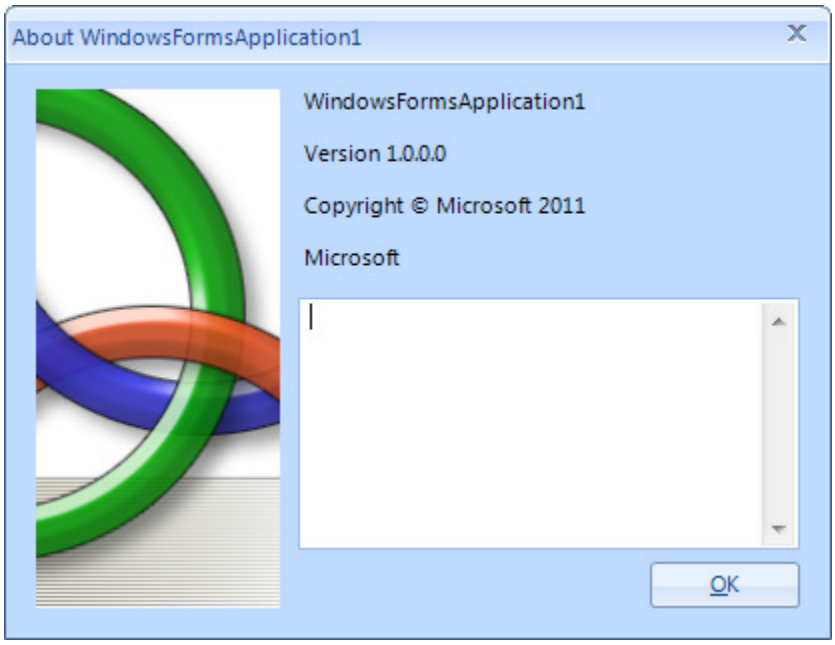
The example screenshot below shows a ShapedForm and RadTitle, both with their own custom shapes.



ShapedForm treats the whole space as its client area. When you dock a control in the form, it can cover the border of the window and disable form resizing. You can change the form Padding to make sure this area stays clear, or anchor a panel that will contain all the controls for the form.

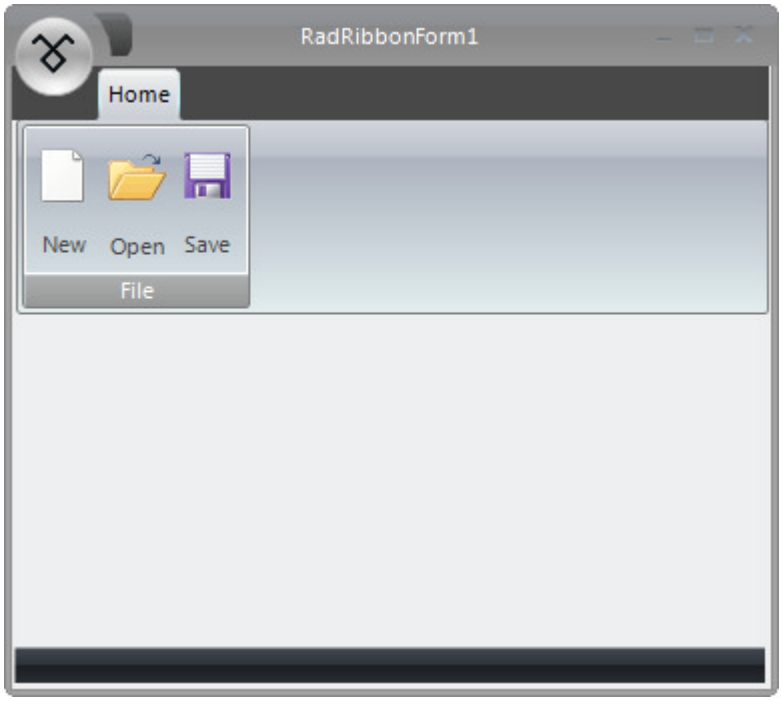
RadAboutBox

The RadAboutBox has a little extra code to access the assembly you're running and will display all that information automatically in the about box along with an icon.



RadRibbonForm

RadRibbonForm comes pre-configured with a RadRibbonBar and a RadStatusStrip so that you don't have to spend time setting up this first layer of the UI.



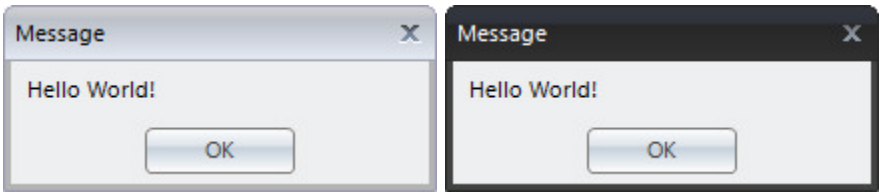
RadMessageBox

RadMessageBox is a flexible, themeable, and not-to-mention quite handy replacement for a standard Windows MessageBox. Some of the advantages to using RadMessageBox over MessageBox:

- Auto-sizing
- Various button layouts

- Localization
- Theming
- RightToLeft layout
- Copy to clipboard with Ctrl+C command

Like MessageBox, there's no design-time component involved. Use RadMessageBox Static methods **SetTheme()** to theme the dialog so it will blend with your application and call one of the many **Show()** method overloads to display the dialog. The minimal overload usage of Show() simply takes some message text:



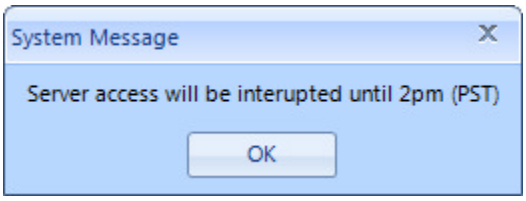
[VB] Calling SetTheme()

```
Private Sub btnRadMessageBox_Click(sender As Object, e As EventArgs)
 RadMessageBox.SetThemeName("Office2007Silver")
'or RadMessageBox.SetThemeName("Office2007Black")
 RadMessageBox.Show("Hello World!")
End Sub
```

[C#] Calling SetTheme()

```
private void btnRadMessageBox_Click(object sender, EventArgs e)
{
 RadMessageBox.SetThemeName("Office2007Silver");
//or RadMessageBox.SetThemeName("Office2007Black");
 RadMessageBox.Show("Hello World!");
}
```

You can add a caption that displays in the title bar of the dialog:



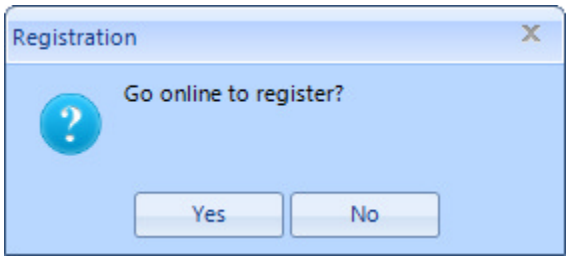
[VB] Show() With Caption

```
Private Sub btnRadMessageBox_Click(sender As Object, e As EventArgs)
 RadMessageBox.Show("Server access will be interupted until 2pm (PST)", "System Message")
End Sub
```

[C#] Show() With Caption

```
private void btnRadMessageBox_Click(object sender, EventArgs e)
{
 RadMessageBox.Show("Server access will be interupted until 2pm (PST)",
   "System Message");
}
```

As with MessageBox, evaluate the DialogResult return value from Show() to respond to user button selections. Here's an example that uses a more complex overload of the Show() method to include a predefined set of buttons, a system icon and the button that will be default if the user hits Enter instead of clicking with the mouse.



[VB] Getting User Feedback

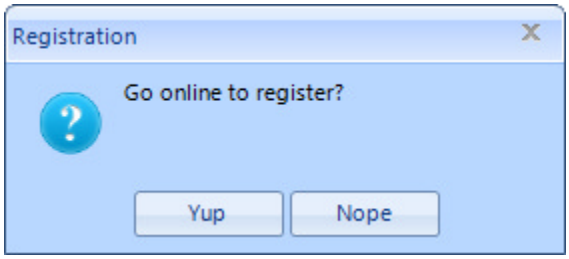
```
Private Sub btnRadMessageBox_Click(sender As Object, e As EventArgs)
 Dim result As DialogResult = RadMessageBox.Show("Go online to register?", "Registration",
MessageBoxButtons.YesNo, RadMessageIcon.Question, MessageBoxDefaultButton.Button1)
    ' do something...
 If result = DialogResult.Yes Then
 End If
End Sub
```

[C#] Getting User Feedback

```
private void btnRadMessageBox_Click(object sender, EventArgs e)
{
 DialogResult result = RadMessageBox.Show(
   "Go online to register?",
   "Registration",
   MessageBoxButtons.YesNo,
   RadMessageIcon.Question,
   MessageBoxDefaultButton.Button1);
 if (result == DialogResult.Yes)
 {
   // do something...
 }
}
```

Localization

If you need to localize or otherwise customize the text for a RadMessageBox, you can create your own **RadMessageLocalizationProvider** and assign it to be the current provider.



[VB] Custom Localization Provider Class and Assignment

```
Public Class MyMessageLocalizationProvider
 Inherits RadMessageLocalizationProvider
 Public Overloads Overrides Function GetLocalizedString(id As String) As String
  Select Case id
   Case RadMessageStringID.AbortButton
    Return "Hold it!"
   Case RadMessageStringID.CancelButton
    Return "Lets not"
```

```
   Case RadMessageStringID.IgnoreButton
    Return "Forget it"
   Case RadMessageStringID.NoButton
    Return "Nope"
   Case RadMessageStringID.OKButton
    Return "Allright"
   Case RadMessageStringID.RetryButton
    Return "Again Please"
   Case RadMessageStringID.YesButton
    Return "Yup"
   Case Else
    Return MyBase.GetLocalizedString(id)
  End Select
 End Function
End Class
'...
Private Sub btnRadMessageBox_Click(sender As Object, e As EventArgs)
 RadMessageLocalizationProvider.CurrentProvider = New MyMessageLocalizationProvider()
 Dim result As DialogResult = RadMessageBox.Show("Go online to register?", "Registration",
MessageBoxButtons.YesNo, RadMessageIcon.Question, MessageBoxDefaultButton.Button1)
   ' do something...
 If result = DialogResult.Yes Then
 End If
End Sub
```

[C#] Custom Localization Provider Class and Assignment

```
public class MyMessageLocalizationProvider : RadMessageLocalizationProvider
{
 public override string GetLocalizedString(string id)
 {
   switch (id)
   {
     case RadMessageStringID.AbortButton:
       return "Hold it!";
     case RadMessageStringID.CancelButton:
       return "Lets not";
     case RadMessageStringID.IgnoreButton:
       return "Forget it";
     case RadMessageStringID.NoButton:
       return "Nope";
     case RadMessageStringID.OKButton:
       return "Allright";
     case RadMessageStringID.RetryButton:
       return "Again Please";
     case RadMessageStringID.YesButton:
       return "Yup";
     default:
       return base.GetLocalizedString(id);
   }
 }
}
//...
private void btnRadMessageBox_Click(object sender, EventArgs e)
{
 RadMessageLocalizationProvider.CurrentProvider = new MyMessageLocalizationProvider();
```

```
 DialogResult result = RadMessageBox.Show(
   "Go online to register?",
   "Registration",
   MessageBoxButtons.YesNo,
   RadMessageIcon.Question,
   MessageBoxDefaultButton.Button1);
 if (result == DialogResult.Yes)
 {
   // do something...
 }
}
```

ShapedForm and RadMessageBox Walk Through

This project walks you through creating a CD browser in a ShapedForm. The project covers a diverse set of techniques:

- Assigning shapes to a ShapedForm, RadPanel and RadButton.
- Using BindingSource methods to navigate a dataset and to bind both image and text properties.
- Working with gradient styles of a fill primitive to achieve a glassy, translucent button and where the button color overtones match the image background.
- Using RadMessageBox to display a message and a specific bitmap thumbnail.



Preparing the Project



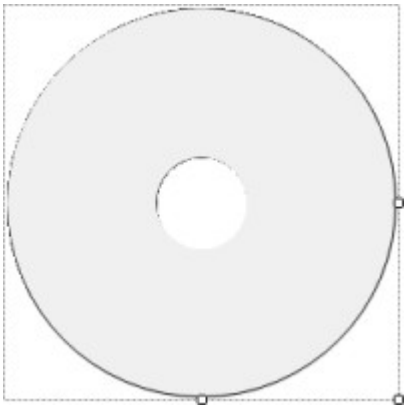
You can find the complete source for this project at:

\Toolstrip\<VB|CS>\ShapedFormWalkthrough

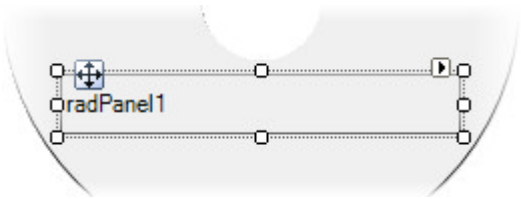
1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik ShapedForm" template and click the **Add** button to close the dialog.
5. Make the shaped form the default form to run on startup.

Designing the Form

1. In the **Shape** property of the ShapedForm select the "Donut" shape. Expand the properties of the shape and set the Thickness sub-property to "100".
2. Set the **Size** property of the ShapedForm to "260, 260".
3. Delete the **RadTitleBar** from the ShapedForm.
4. The shaped form in the designer should look something like this screenshot:



5. Add a RadPanel centered on the lower part of the donut shape. Set the **Name** property of the panel to "pnlTitle". Set the **Size** property to "200, 30".

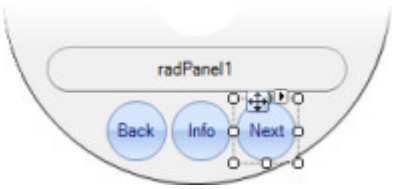


6. Using the Smart Tag, select the Edit UI Elements link to bring up the Element Hierarchy Editor:
   - Select the **RadPanelElement** from the Control element structure treeview.
   - Set the **Shape** property to "RoundRectShape"
   - Open up the Shape property and set the **Radius** sub-property to "20".
   - Also within the Control element structure tree, locate the TextPrimitive (it should currently read "Text: radPanel1") and set the **Alignment** property to "MiddleCenter".
   - Close the Element Hierarchy editor dialog.

     The layout on the form should look something like this screenshot:

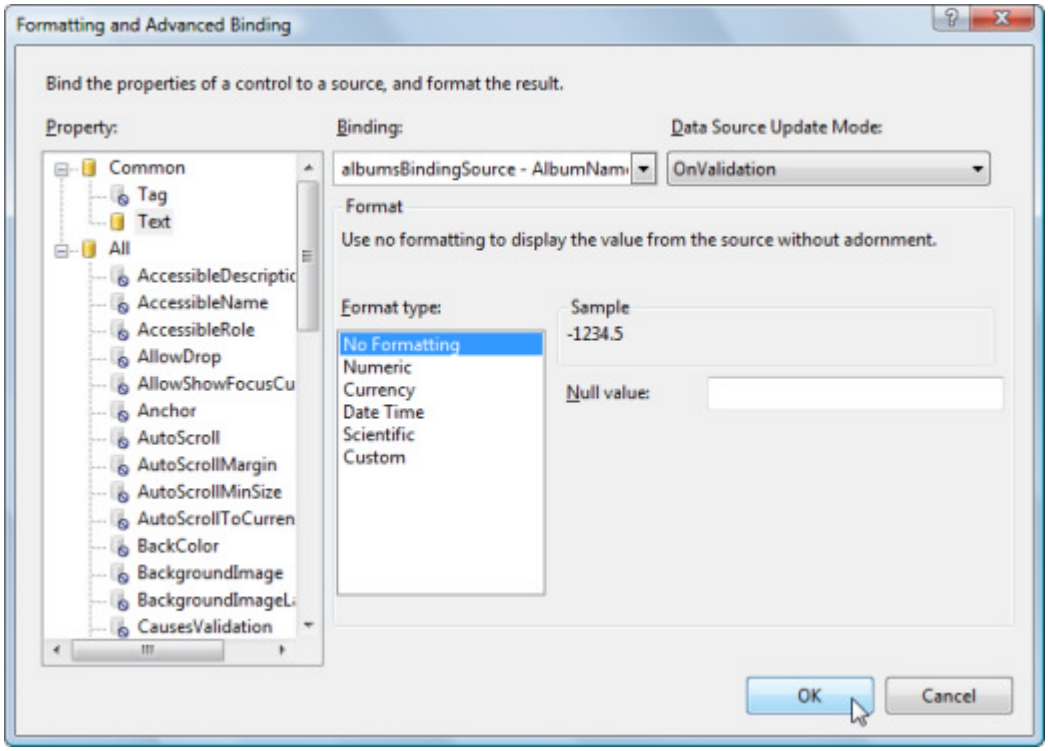


7. Add three RadButtons below the panel in a horizontal line and set properties:
   - **Name** = "btnBack", **Text** = "Back", **Size** = "40, 40"
   - **Name** = "btnInfo", **Text** = "Info", **Size** = "40, 40"
   - **Name** = "btnNext", **Text** = "Next", **Size** = "40, 40"
8. For each of the three buttons, open the Smart Tag and select the Edit UI Elements link to bring up the Element Hierarchy Editor. Set the **RadButtonElement Shape** property to "EllipseShape". The form designer should look something like this screenshot:

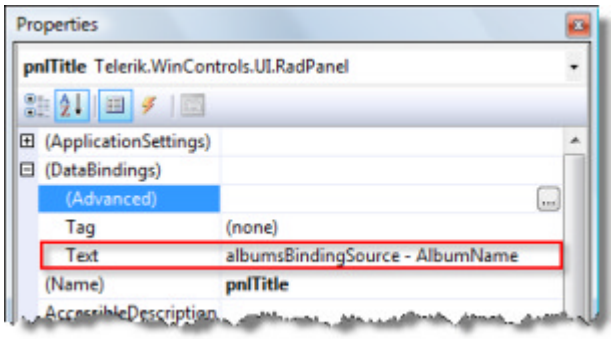


9. Select "pnlTitle" in the designer and in the Properties window, open the Databindings property and click the ellipses for the Advanced property. This will bring up the Formatting and Advanced Binding dialog.
10. Configure the bindings for "pnlTitle".
    - In the tree on the left select the **Text** property.
    - In the Binding drop down list select Add Project Data Source. Configure the RadPanel to use the MusicCollection Albums table, "AlbumName" column.

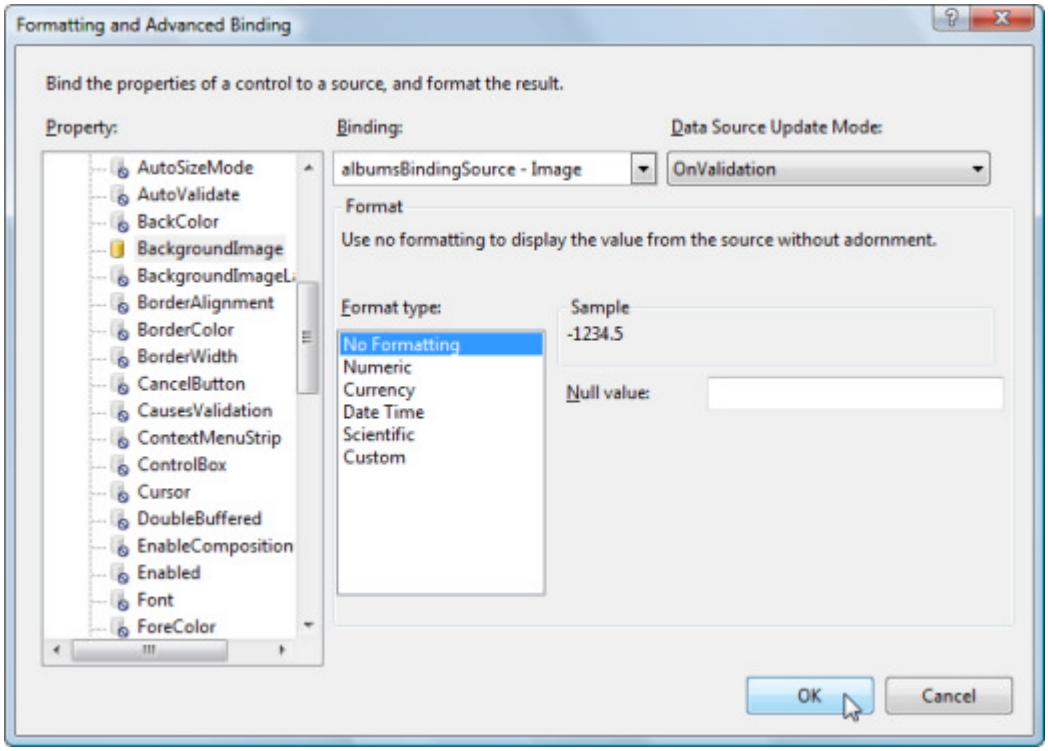
      For more detail, revisit the Data Binding chapter section "Binding to Simple Controls Using DataBindings". The Formatting and Advanced Binding dialog will look something like the screenshot below.



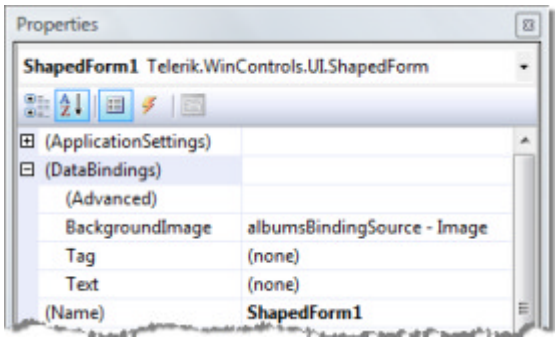
- Click **OK** to close the dialog. The RadPanel DataBindings Text property should look like the screenshot below:



11. Select the form in the designer and again configure the DataBindings.
    - This time set the **BackgroundImage** property to the "Image" column of the Albums table. The Formatting and Advanced Binding dialog will look something like the screenshot below.



- Click **OK** to close the dialog. The ShapedForm DataBindings BackgroundImage property should look like the screenshot below:



12. In the designer, double-click the "Back" button and handle the Click event.

    *Earlier when you bound the RadPanel to the Albums table you automatically created an "albumsBindingSource" component that was placed in the component tray below the form designer. Call the BindingSource **MovePrevious()** method.*

    [VB] Handling the "Back" Button Click Event

```
Private Sub btnBack_Click(sender As Object, e As EventArgs)
 albumsBindingSource.MovePrevious()
End Sub
```

    [C#] Handling the "Back" Button Click Event

```
private void btnBack_Click(object sender, EventArgs e)
{
 albumsBindingSource.MovePrevious();
}
```

13. Create another Click event handler for the "Next" button and add the code below.

    *This event handler simply calls the BindingSource **MoveNext()** method.*

    [VB] Handling the "Next" Button Click Event

```
Private Sub btnNext_Click(sender As Object, e As EventArgs)
 albumsBindingSource.MoveNext()
End Sub
```

    [C#] Handling the "Next" Button Click Event

```
private void btnNext_Click(object sender, EventArgs e)
{
 albumsBindingSource.MoveNext();
}
```

14. Press **Ctrl-F5** to run the application. Test the "Back" and "Next" buttons. Verify that the images show up on the ShapedForm background and that the Album Title changes as you move next and back through the data.



    That all works pretty well except that the panel and buttons look clunky in this context. We can make these transparent and subtly tinted to fit the color scheme of each image.

Handle Background Color Changes

1. Be sure that references to **Telerik.WinControls** and **Telerik.WinControls.Primitives** exist in your "Imports" (VB) or "uses" (C#) section of code.
2. Add two helper methods. One to calculate a contrasting color and the second to configure a FillPrimitive to display in a "Gel" gradient style in two colors.

    ***Note**: The GetContrastingColor() calculation will not work perfectly for all background color situations. To solve this more permanently you would need a more complex color calculation or a workaround, e.g. shadow the text in a second color.*

    [VB] Adding Helper Methods

```
Private Function GetContrastingColor(color As Color) As Color
 ' "exclusive OR" the color passed in to leave only the opposite color
 Return Color.FromArgb(color.ToArgb() Xor &Hffffff)
End Function
' configure the FillPrimitive to display in a "Gel" gradient style and set color
Private Sub SetFillPrimitiveColor(element As RadElement, color1 As Color)
 Dim fillPrimitive As FillPrimitive = TryCast(element, FillPrimitive)
 fillPrimitive.NumberOfColors = 1
```

```
 fillPrimitive.GradientStyle = Telerik.WinControls.GradientStyles.Gel
 fillPrimitive.BackColor = color1
End Sub
```

[C#] Adding Helper Methods

```
private Color GetContrastingColor(Color color)
{
 // "exclusive OR" the color passed in to leave only the opposite color
 return Color.FromArgb(color.ToArgb() ^ 0xffffff);
}
// configure the FillPrimitive to display in a "Gel" gradient style and set color
private void SetFillPrimitiveColor(RadElement element, Color color1)
{
 FillPrimitive fillPrimitive = element as FillPrimitive;
 fillPrimitive.NumberOfColors = 1;
 fillPrimitive.GradientStyle = Telerik.WinControls.GradientStyles.Gel;
 fillPrimitive.BackColor = color1;
}
```

3. In the designer, select the "albumsBindingSource" in the component tray and in the Properties window Events tab create an event handler for the BindingComplete event. Add the code below:

***Note**: you could have handled the CurrentChanged event but that will not run when the form first displays, whereas BindingComplete will.*

To make the panel and buttons react to the new background image colors, convert the Image to a Bitmap. Use the Bitmap GetPixel() method to snag a pixel color from the center of the image. Get a contrasting color to be used in the button and panel text. Also calculate a translucent version of the color using the Color.FromArgb and passing it an alpha value of 150 (255 is completely opaque, 0 is completely transparent). Call SetFillPrimitiveColor() for the panel and button elements to set the gradient style of each to "Gel", the number of colors to "1" and the color to the translucent color.

[VB] Handle the BindingComplete Event

```
Private Sub albumsBindingSource_BindingComplete(sender As Object, e As
BindingCompleteEventArgs)
 ' background image will be null when the form first loads
 If Me.BackgroundImage <> Nothing Then
  ' convert Image to Bitmap for further manipulation
  Dim bitmap As New Bitmap(Me.BackgroundImage)

  ' get a color from the very center of the background image
  Dim color1 As Color = bitmap.GetPixel(bitmap.Width / 2, bitmap.Height / 2)
  ' get a color that contrasts with the center of the image
  Dim contrastingColor As Color = GetContrastingColor(color1)
  ' make a transluscent version of this color
  Dim semiTransparentColor As Color = Color.FromArgb(150, color1)

  ' set the buttons and panel with gel style using the translucent color
  SetFillPrimitiveColor(pnlTitle.PanelElement.Children(0), semiTransparentColor)
  SetFillPrimitiveColor(btnBack.ButtonElement.Children(0), semiTransparentColor)
  SetFillPrimitiveColor(btnNext.ButtonElement.Children(0), semiTransparentColor)
  SetFillPrimitiveColor(btnInfo.ButtonElement.Children(0), semiTransparentColor)

  ' set the color for the text to the contrasting color
  (TryCast(btnBack.ButtonElement.Children(1).Children(1), TextPrimitive)).ForeColor =
contrastingColor
  (TryCast(btnNext.ButtonElement.Children(1).Children(1), TextPrimitive)).ForeColor =
```

```
contrastingColor
  (TryCast(btnInfo.ButtonElement.Children(1).Children(1), TextPrimitive)).ForeColor =
contrastingColor
  pnlTitle.ForeColor = contrastingColor
 End If
End Sub
```

[C#] Handle the BindingComplete Event

```
private void albumsBindingSource_BindingComplete(object sender, BindingCompleteEventArgs
{
 // background image will be null when the form first loads
 if (this.BackgroundImage != null)
 {
   // convert Image to Bitmap for further manipulation
   Bitmap bitmap = new Bitmap(this.BackgroundImage);
   // get a color from the very center of the background image
   Color color1 = bitmap.GetPixel(bitmap.Width / 2, bitmap.Height / 2);
   // get a color that contrasts with the center of the image
   Color contrastingColor = GetContrastingColor(color1);
   // make a transluscent version of this color
   Color semiTransparentColor = Color.FromArgb(150, color1);
   // set the buttons and panel with gel style using the translucent color
   SetFillPrimitiveColor(pnlTitle.PanelElement.Children[0], semiTransparentColor);
   SetFillPrimitiveColor(btnBack.ButtonElement.Children[0], semiTransparentColor);
   SetFillPrimitiveColor(btnNext.ButtonElement.Children[0], semiTransparentColor);
   SetFillPrimitiveColor(btnInfo.ButtonElement.Children[0], semiTransparentColor);
   // set the color for the text to the contrasting color
   (btnBack.ButtonElement.Children[1].Children[1] as TextPrimitive).ForeColor =
contrastingColor;
   (btnNext.ButtonElement.Children[1].Children[1] as TextPrimitive).ForeColor =
contrastingColor;
   (btnInfo.ButtonElement.Children[1].Children[1] as TextPrimitive).ForeColor =
contrastingColor;
   pnlTitle.ForeColor = contrastingColor;
 }
}
```

4. Press **Ctrl-F5** to run the application. The panel and buttons should co-exist nicely with the background.



Add a Information Dialog

The final piece is to add logic to the "Info" button to display a RadMessageBox that displays a thumbnail of the background image and the name of the album. Add a Click event handler to the "Info" button and add the code below.

Optionally, you can add a theme to the application and call the static RadMessageBox SetTheme() method before calling Show(). Call the overload of the Show() method that takes a Bitmap object and displays that instead of a system icon. This example reuses the text stored in the RadPanel to display in the RadMessageBox. Note: You could also use the BindingSource Current property to get a DataRowView; that would let you access any row of the current record even if it was not displayed by another control.

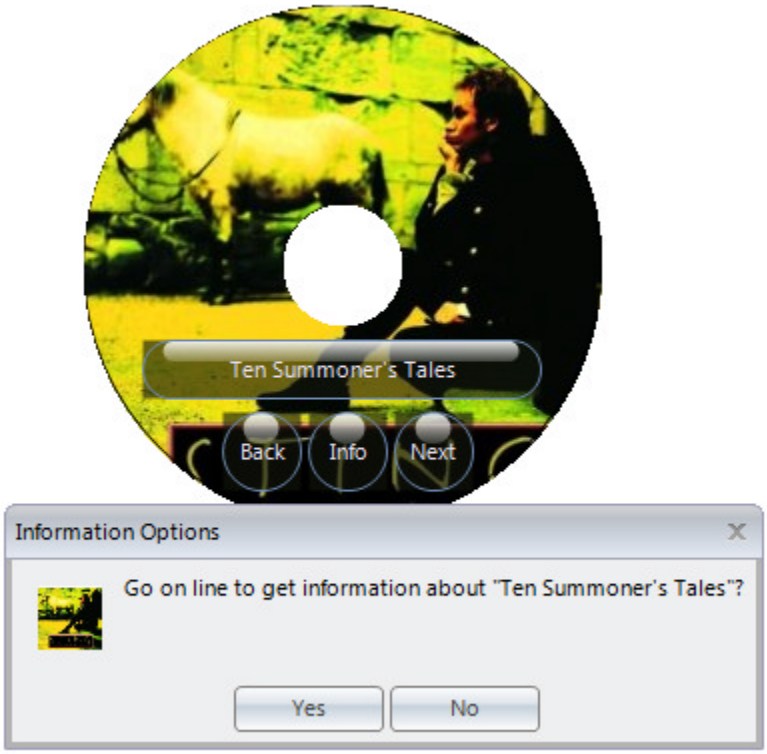
[VB] Handle the "Info" Button Click Event

```
Private Sub btnInfo_Click(sender As Object, e As EventArgs)
 RadMessageBox.SetThemeName("Office2007Silver")
 Dim message As String = "Go on line to get information about """ + pnlTitle.Text + """?"
 Dim bitmap As New Bitmap(Me.BackgroundImage)
 RadMessageBox.Show(Me, message, "Information Options", MessageBoxButtons.YesNo, bitmap)
End Sub
```

[C#] Handle the "Info" Button Click Event

```
private void btnInfo_Click(object sender, EventArgs e)
{
 RadMessageBox.SetThemeName("Office2007Silver");
 string message = "Go on line to get information about \"" + pnlTitle.Text + "\"?";
 Bitmap bitmap = new Bitmap(this.BackgroundImage);
 RadMessageBox.Show(this, message, "Information Options", MessageBoxButtons.YesNo, bitmap);
}
```

Run the application and test the "Info" button. It should display the image thumbnail and the name of the album.



Multiple Document Interface

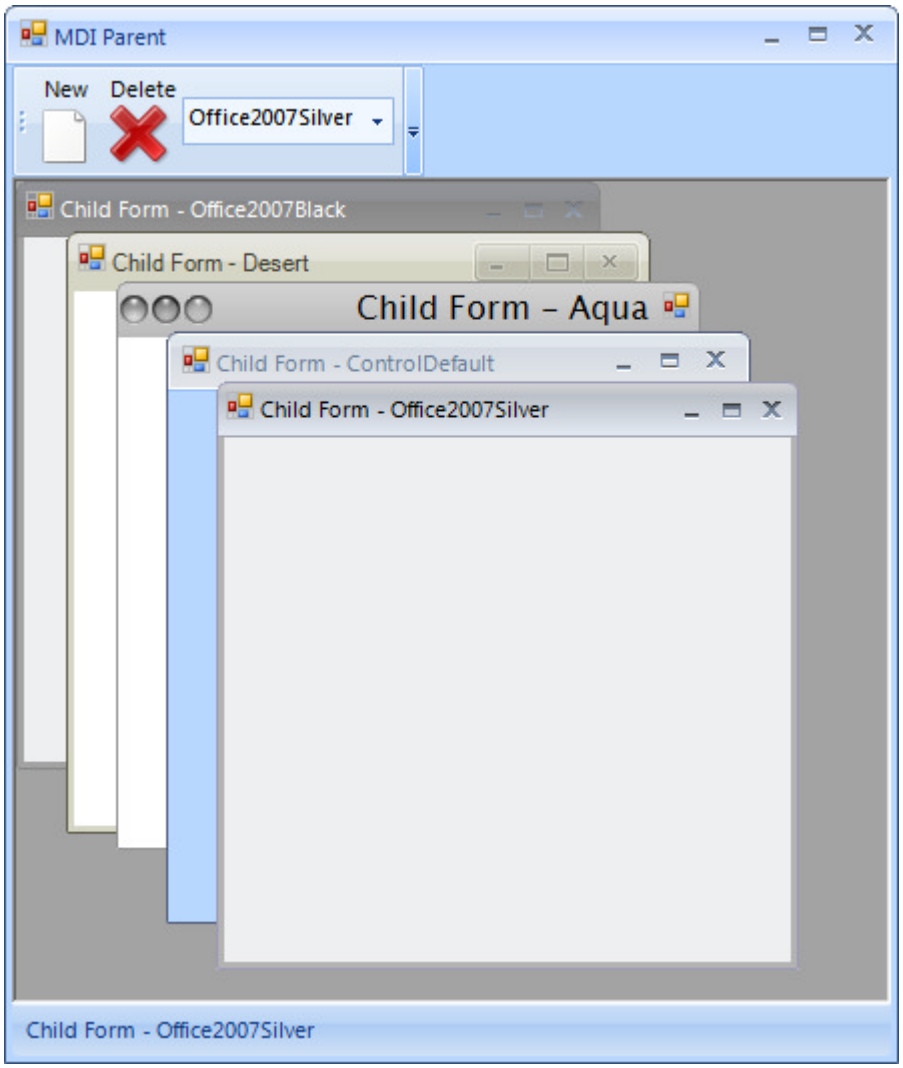
With Multiple Document Interface (MDI), a single menu bar or toolbar is shared between all child windows, reducing clutter and increasing the efficient use of screen space. RadForm works nicely in MDI scenarios by automatically providing minimize, restore, maximize, and inactive states.

Each MDI child form and the parent container control can have a different theme. The example below allows the user to open and close multiple MDI child windows, each having a theme specified in a drop down list. The command bar and status bar controls are located on the parent form, but actions can be taken on the child forms by referencing the parent form standard ActiveMdiChild property.



You can find the complete source for this project at:

\Forms\<VB|CS>\MDI



The first step is to set the parent form's **IsMdiContainer** property to True. Failing to do this will generate an error when you attempt to set the MdiParent of a child form later. The drop down list gets filled with the themes available for a RadForm. Also, we hook up a **MdiChildActivate** event handler that will sense when a child form is clicked on.

[VB] Handling the Form Load Event

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 Me.IsMdiContainer = True
 ' disable the delete button - there are no child forms
 Me.btnDelete.Enabled = False
 ' load drop down list with theme names
 Dim themes As ThemeList = ThemeResolutionService.GetAvailableThemes(Me)
 For Each theme As Theme In themes

commandBarDropDownList1.Items.Add(New RadListDataItem(theme.ThemeName))
 Next
 commandBarDropDownList1.SelectedIndex = 0
 Me.MdiChildActivate += New EventHandler(RadForm1_MdiChildActivate)
End Sub
```

[C#] Handling the Form Load Event

```
private void RadForm1_Load(object sender, EventArgs e)
{
 this.IsMdiContainer = true;
 // disable the delete button - there are no child forms
 this.btnDelete.Enabled = false;
 // load drop down with theme names
 ThemeList themes =
    ThemeResolutionService.GetAvailableThemes(this);
 foreach (Theme theme in themes)
 {
   commandBarDropDownList1.Items.Add(new RadListDataItem(theme.ThemeName));
 }
 commandBarDropDownList1.SelectedIndex = 0;
 this.MdiChildActivate += new EventHandler(RadForm1_MdiChildActivate);
}
```

When the MdiChildActivate event fires we can get a reference to the **ActiveMdiChild,** assign the title bar text to a label on the status bar. Be sure to check ActiveMdiChild for null because that occurs when the last child is deleted.

[VB] Handling the MdiChildActivate Event

```
Sub RadForm1_MdiChildActivate(sender As Object, e As EventArgs)
 If Me.ActiveMdiChild <> Nothing Then
  Me.lblStatus.Text = (TryCast(Me.ActiveMdiChild, RadForm)).Text
 End If
End Sub
```

[C#] Handling the MdiChildActivate Event

```
void RadForm1_MdiChildActivate(object sender, EventArgs e)
{
 if (this.ActiveMdiChild != null)
 {
   this.lblStatus.Text = (this.ActiveMdiChild as RadForm).Text;
 }
}
```

In the "New" button Click event handler you create a child form, which can be a standard Windows form, a RadForm, a ShapedForm or a descendent of any of these types.

[VB] Handling the "New" Button Click Event

```
Private Sub btnNew_Click(sender As Object, e As EventArgs)
 ' Create a new mdi child form, using the current theme
 Dim child As New RadForm()
 child.MdiParent = Me
 child.Text = "Child Form - " + cbThemes.SelectedText
 child.ThemeName = commandBarDropDownList1.SelectedText
 child.FormClosed += New FormClosedEventHandler(child_FormClosed)
 child.Show()
 ' enable the delete button
 Me.btnDelete.Enabled = True
End Sub
```

[C#] Handling the "New" Button Click Event

```
private void btnNew_Click(object sender, EventArgs e)
{
 // Create a new mdi child form, using the current theme
 RadForm child = new RadForm();
```

```
 child.MdiParent = this;
 child.Text = "Child Form - " + commandBarDropDownList1.SelectedText;
 child.ThemeName = commandBarDropDownList1.SelectedText;
 child.FormClosed += new FormClosedEventHandler(child_FormClosed);
 child.Show();
 // enable the delete button
 this.btnDelete.Enabled = true;
}
```

When the "Delete" button is clicked, call the Close() method of the active Mdi child.

[VB] Handling the "Delete" Button Click Event

```
Private Sub btnDelete_Click(sender As Object, e As EventArgs)
 ' close the active mdi child form
 If Me.ActiveMdiChild <> Nothing Then
  Me.ActiveMdiChild.Close()
 End If
End Sub
```

[C#] Handling the "Delete" Button Click Event

```
private void btnDelete_Click(object sender, EventArgs e)
{
 // close the active mdi child form
 if (this.ActiveMdiChild != null)
 {
   this.ActiveMdiChild.Close();
 }
}
```

As each child form is closed, you can check that there are still more child forms to delete:

[VB] Handling the FormClosed Event

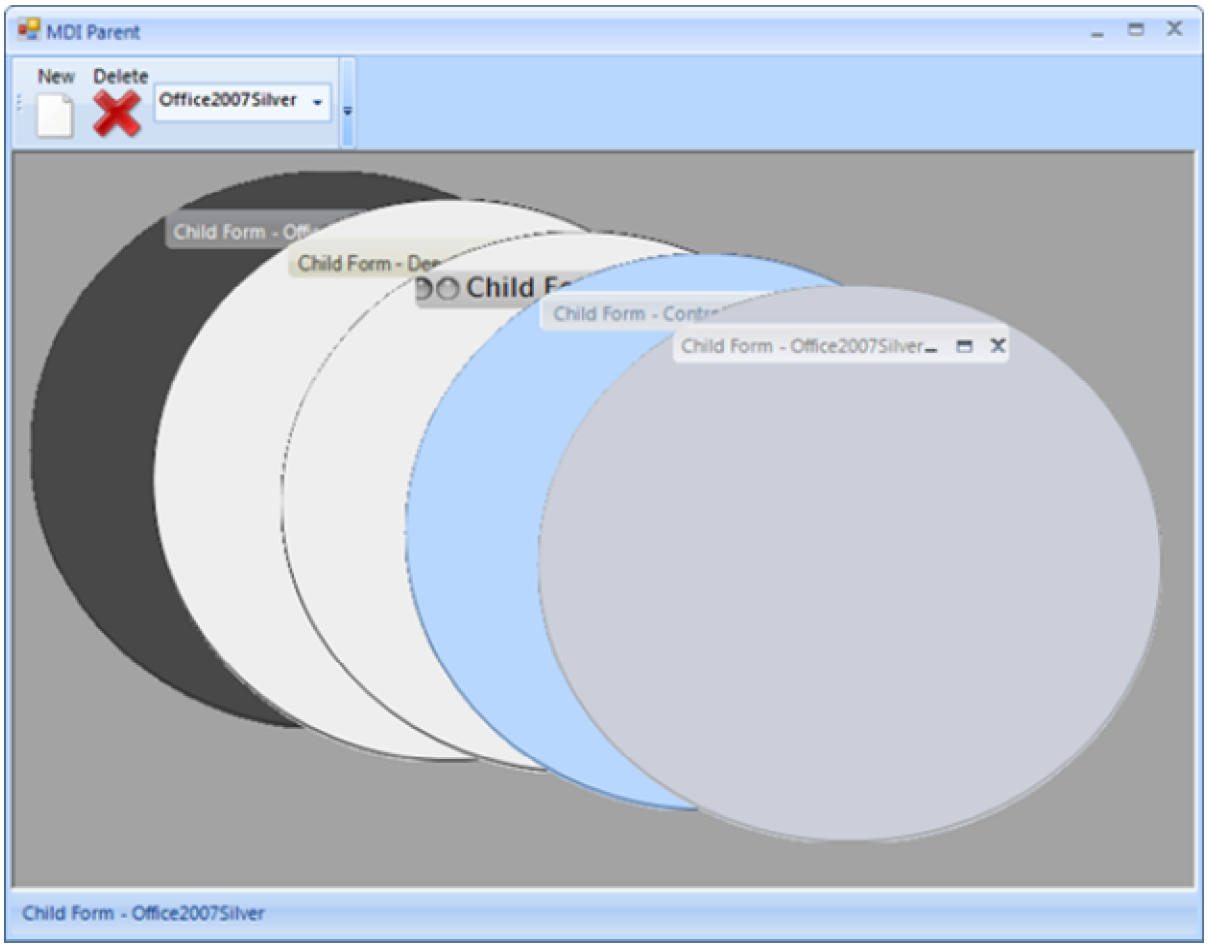
```
Sub child_FormClosed(sender As Object, e As FormClosedEventArgs)
 ' if there are child forms, enable the delete button
 ' add one to the count - the child form being closed still exists at this point
 Me.btnDelete.Enabled = Me.MdiChildren.Length > 1
End Sub
```

[C#] Handling the FormClosed Event

```
void child_FormClosed(object sender, FormClosedEventArgs e)
{
 // if there are child forms, enable the delete button
 // add one to the count - the child form being closed still exists at this point
 this.btnDelete.Enabled = this.MdiChildren.Length > 1;
}
```

Other MDI Child Form Types

Because ShapedForm is a descendant of the standard Form object you can also use ShapedForm as a child form.



The logic above will work for the most part except that you will need to create a ShapedForm in the "New" button Click event handler and you will need to change references to the title bar, for example:

[VB] Accessing the TitleBar

```
' instead of...
child.FormElement.TitleBar
' use...
TryCast(child.Controls("radTitleBar1"), RadTitleBar)
```

[C#] Accessing the TitleBar

```
// instead of...
child.FormElement.TitleBar;
// use...
child.Controls["radTitleBar1"] as RadTitleBar;
```

15.6 Summary

In this chapter you learned how the special RadForm, ShapedForm and RadRibbonForm classes can be inherited from to create themeable, custom shaped forms to complete and polish the look-and-feel of an application. You saw how the new project item templates are used to create instances of each form class. The chapter demonstrated how RadTitleBar is used as a replacement for the built-in Windows form title bar. Finally, the chapter introduced the RadMessage box as a themable replacement for the standard MessageBox.

16 Screen "Real Estate"

16.1 Objectives

- Learn different strategies for organizing form space using RadControls
- Use RadDock to handle dockable windows and tabbed documents
- Use RadCarousel to animate a series of clickable images
- Use RadRotator to display constantly changing personalized content.

16.2 Introduction

For interactive, unique navigation and form "real estate" management, RadDock, RadRotator and RadCarousel let you catch your users attention and make wise use of form space at the same time.

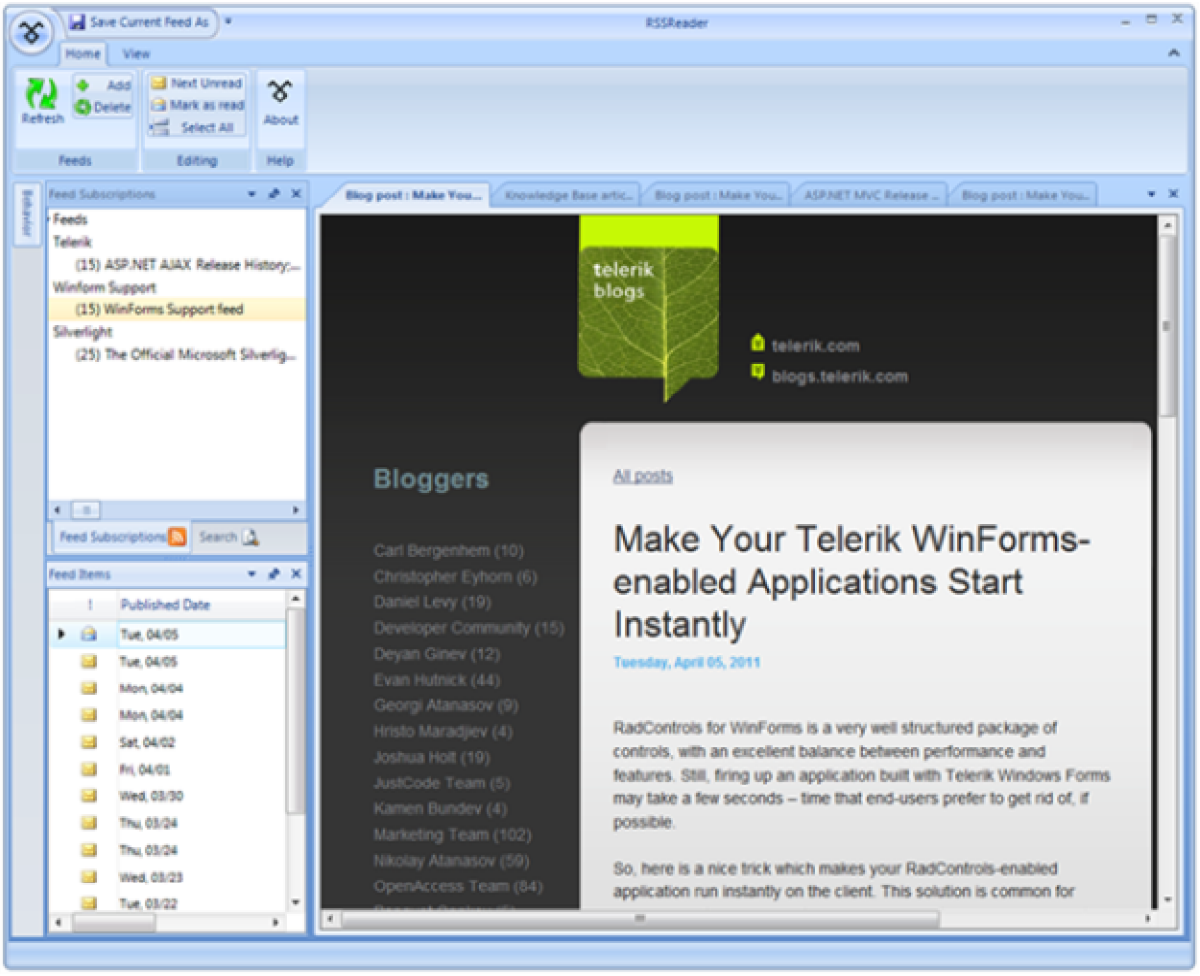
RadDock

RadDock helps you manage multiple windows in your application with a docking system similar to Microsoft Visual Studio 2005. RadDock can contain both tool and tabbed document style windows. RadDock also includes fully interactive design-time layout management.

- The RadDock control provides a container that holds dockable windows. This container can fill the entire client area of a Windows Form, or can be limited to any rectangular area you choose to manage.
- Tabbed documents can be switched between, resized, dragged to arrange in various configurations and closed. Tabbed documents are supported by the DocumentWindow control. You can place other controls (either Telerik RadControls or standard Windows Forms controls) within a DocumentWnidow.
- Tool windows can be dragged outside their containers ("floating"), dragged to other containers, collapsed against the side of the RadDock and transformed to tabbed documents. Tool windows are supported by the ToolWindow control. You can place other controls (either Telerik RadControls or standard Windows Forms controls) within a ToolWindow.
- The RadDock Advanced Layout Designer gives you fully interactive design-time control over the number, position, and properties of DockWindows within a RadDock.
- RadDock collects and uses the information about the state of each DockWindow - FloatingSize, FloatingLocation, AutoHideSize, Previous position, AutoHidePostion, etc. dynamically.
- RadDock offers an access to the active window.
- There are CloseAction and AutoDispose properties which allow control over the window’s Close behavior, as well as over memory management of the SplitPanel and DockWindow objects.
- Different sizing modes (Auto, Relative, Absolute, Fill) virtually any layout scenario can be easily achieved. The user has complete control over sizing and layout behavior.
- RadSplitContainer composes a RadSplitContainerLayoutStrategy, which handles any layout request for this container. If the available size modes are not enough to fit into a layout scenario, then the entire layout strategy may be easily replaced/extended with a custom one.
- RadDock stores all sizing information per panel in a separate object allows for easy transition from one state to another and vice-versa without loosing the sizing information for each state.
- “Load Layout” allows for restoring previous scene at 100% - a layout persistency operation should now only serialize/de-serialize the sizing info of each panel in order to completely store/restore its position on a split container.
- Redock support (transition to previous docked or floating state of Dock Windows) has been completely revisited for the new RadDock implementation. A service-based semantic is introduced, which saves a window’s state just before any dock operation is about to occur, and this state can easily be restored later

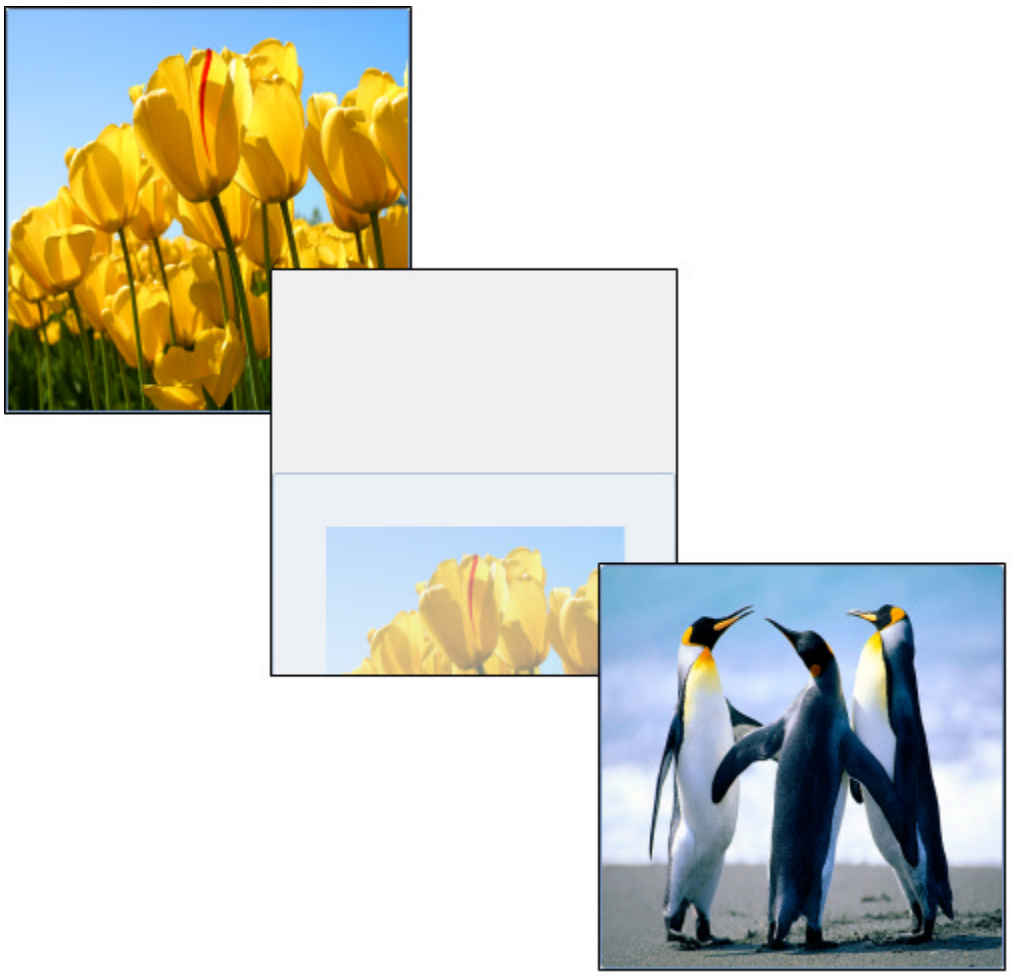
on, upon user request (such as double-clicking a window's caption area). This allows for exact transition to previous state - floating to docked and vice-versa. Even more, if an application requires completely custom redock semantic, it may be easily replaced by registering a custom service, which handles internal Save/Restore requests, made by the framework.

This screenshot shows an RSS reader demo using RadDock together with several other RadControls.



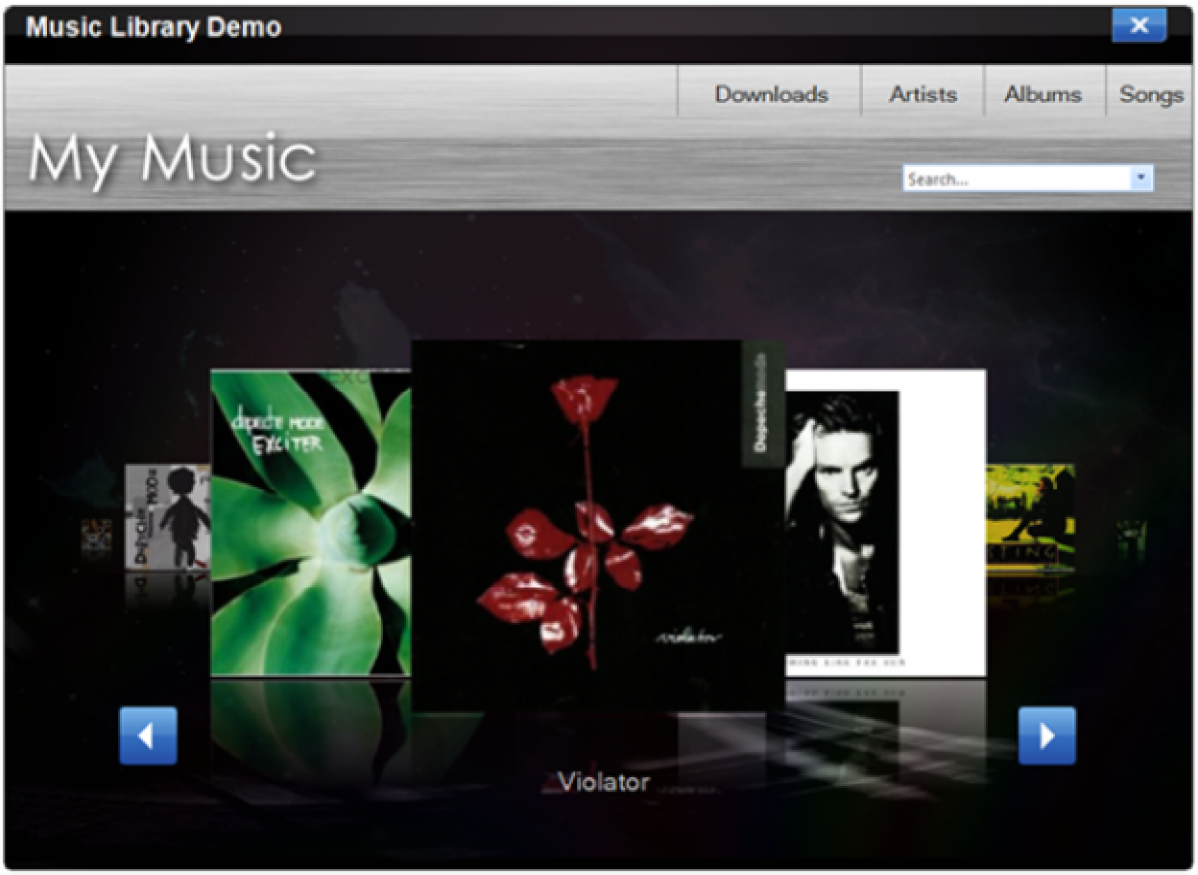
RadRotator

RadRotator is a multi-purpose component for content rotation and personalization. Highly customizable, it delivers high interactivity and user involvement. You can display a series of images, web URLs or any collection of rad elements. For example, if you use RadButtonElement or RadButtonImageElement, the user will be able to click an item before it rotates out of view. Animation between frames can be opaque or transparent and the movement can be in any direction. You can adjust the level of granularity and interval between frames.



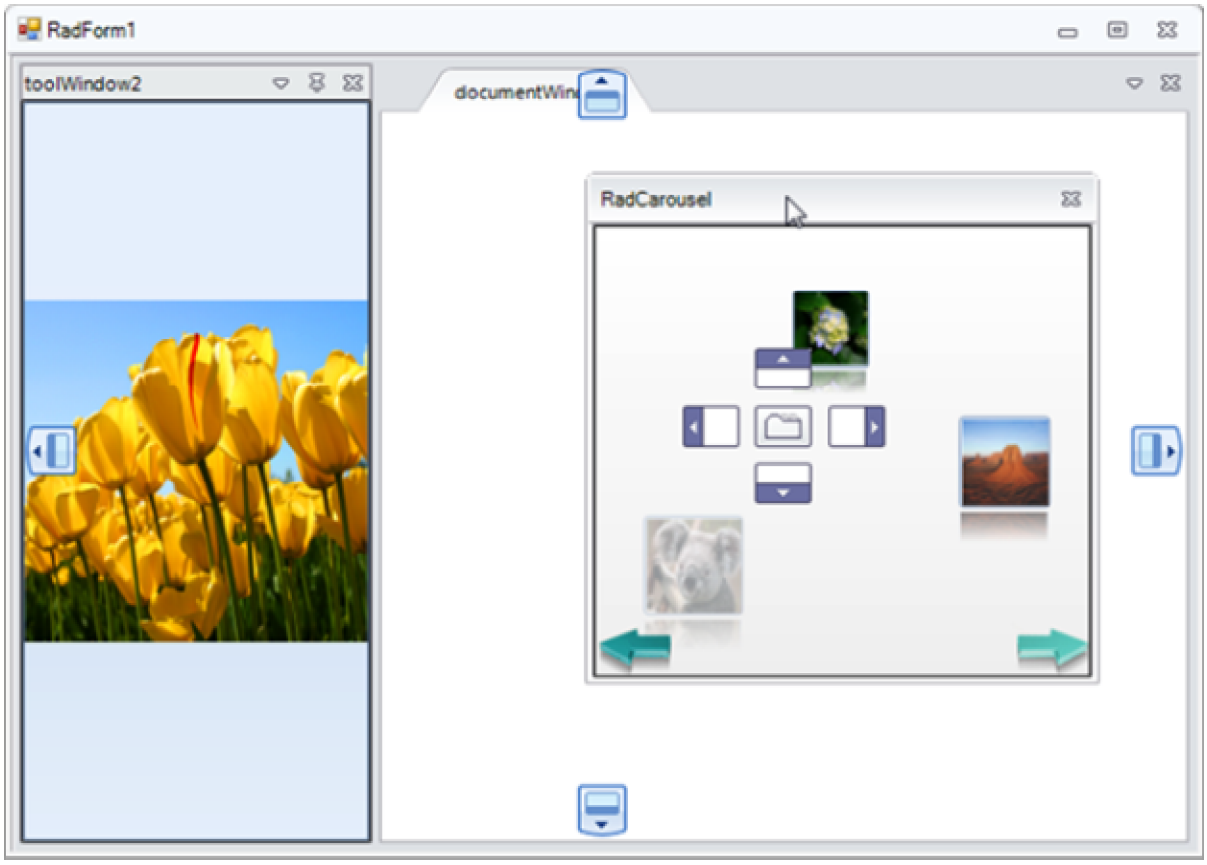
RadCarousel

RadCarousel animates a series of elements either by the user clicking a particular element or by clicking the forward and back arrows. The carousel elements appear to animate through a "path", i.e. a route in three dimensions. The path can be elliptical or along a Bezier curve. RadCarousel supports databinding, smooth animations and transitions, automatic generation of image reflections and dynamic addition and removal of items.



16.3 Getting Started

Using RadDock you can pack a lot of user interface into a single form. You'll get splitter-bars, auto-hidden and floating panels automatically. With RadRotator and RadCarousel you can make a few of these items really stand out. In this project you will configure RadDock with several windows and place a RadCarousel and RadRotator within two of those windows. You will also configure a sample set of items to display in the carousel and rotator.



Preparing the Project

You can find the complete source for this project at:

\RealEstate\<VB|CS>\GettingStarted

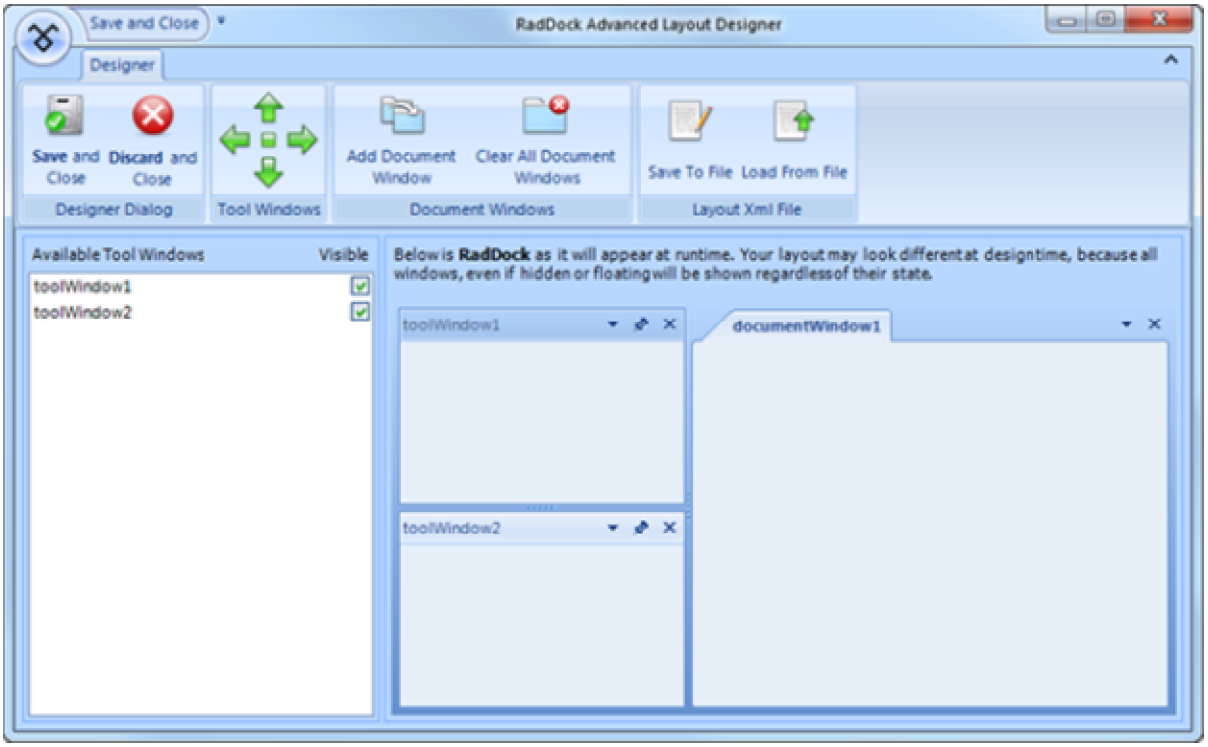
1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. Add the **Office2010Silver** theme rom the ToolBox to the form.
6. In the Properties window, set the form **Size** property to "700, 500" and **ThemeName** to **Office2010Silver**.
7. Change the new RadForm1 to be the startup form.

RadDock Setup

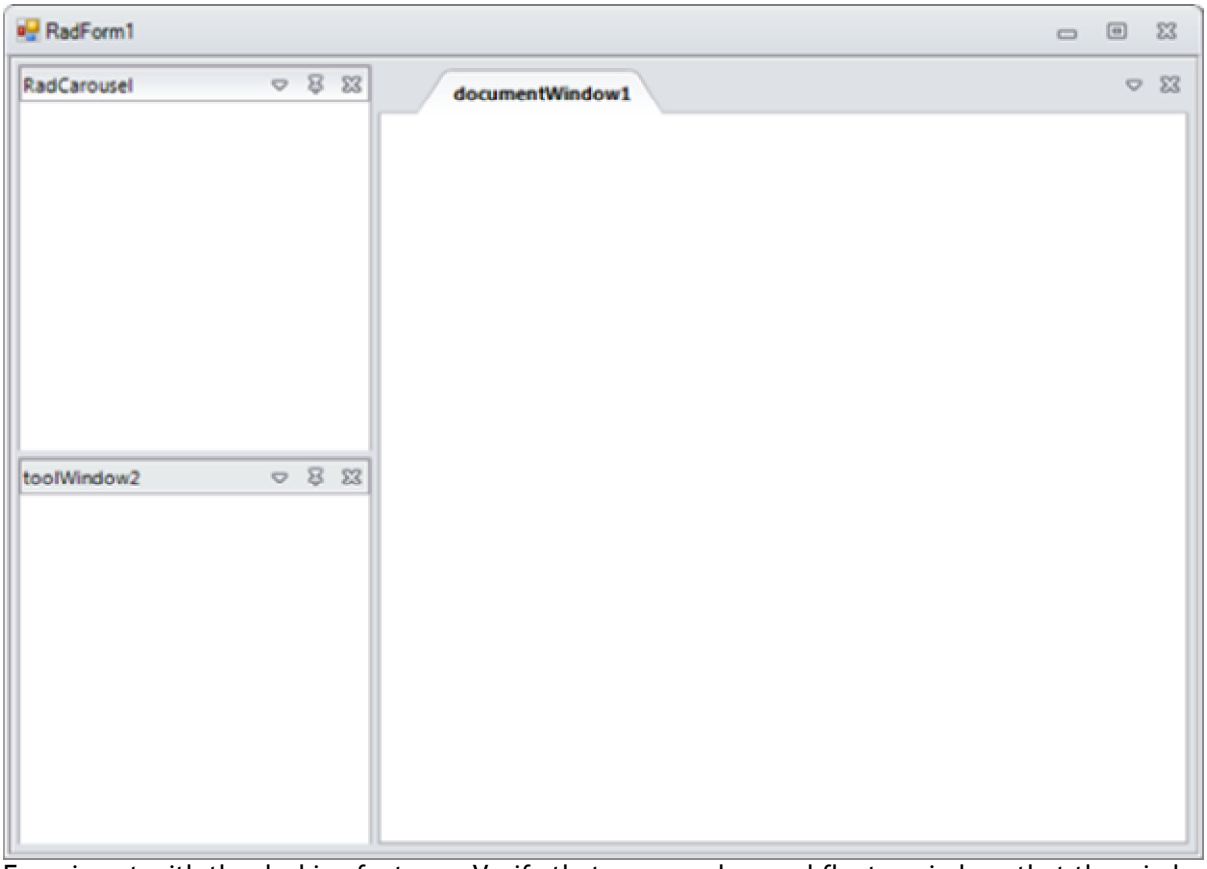
1. From the Toolbox, add a **RadDock** to the form. Set the **Dock** property to "Fill"and **ThemeName** to **Office2007Silver**.
2. In the RadDock Smart Tag click the **Show Advanced Layout Designer...** link. Using the options in the ribbon, configure the layout:
    - Click the **ToolWindow | Left** menu item.
    - Click the **ToolWindow | Bottom** menu item.

- Click the **Add Document Window** menu item.
- Drag toolWindow2 under toolWindow1.

When the designer looks like this screenshot below, click the **Save and Close** button.



3. Click the client area (not the title bar) for "toolWindow1" to select it. In the Properties window change the **Text** property to "RadCarousel".
4. Click the client area for "toolWindow2" to select it. In the Properties window change the **Text** property to "RadRotator".
5. Increase the size of the split container so the toolwindows to take approximately half of the form size.
6. Press **Ctrl-F5** to run the application so far.



Experiment with the docking features. Verify that you can drag and float a window, that the window will re-dock either as a panel or in the tabbed documents area. Check that the auto-hidden (un-pinned) panel can be expanded and pinned. Use the splitter bars to resize the relative areas.

7. Add a **RadRotator** to the "RadRotator" panel and set its **Dock** property to "Fill".
8. Click the ellipses for the **Items** property to display the RadItem Collection Editor. Using the **Add** button drop down list, add 3 **RadButtonElements.**
9. Drag a **RadCarousel** control to the dock panel and set its **Dock** property to "Fill".
10. Click the ellipses for the **Items** property to display the RadItem Collection Editor. Using the **Add** button drop down list, add 3 **RadButtonElements .**
11. Add six images as project resourses, which we will use for the button elements.
12. In the form Load event, set the Image, DisplayStyle and ImageAlignment properties for all six button elemens. Then set the EnableAutoLoop property of RadCarousel to true and the Running property of RadRotator to true.

[VB] Configure images and start rotating and looping for RadRotator and RadCarousel

```
Private Sub RadForm1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
Me.RadButtonElement1.Image = New Bitmap(My.Resources.Lighthouse, New Size(200, 200))
Me.RadButtonElement1.DisplayStyle = Telerik.WinControls.DisplayStyle.Image
Me.RadButtonElement1.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter
Me.RadButtonElement2.Image = New Bitmap(My.Resources.Penguins, New Size(200, 200))
Me.RadButtonElement2.DisplayStyle = Telerik.WinControls.DisplayStyle.Image
Me.RadButtonElement2.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter
Me.RadButtonElement3.Image = New Bitmap(My.Resources.Tulips, New Size(200, 200))
Me.RadButtonElement3.DisplayStyle = Telerik.WinControls.DisplayStyle.Image
Me.RadButtonElement3.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter
```

```
Me.RadButtonElement4.Image = New Bitmap(My.Resources.Desert, New Size(50, 50))
Me.RadButtonElement4.DisplayStyle = Telerik.WinControls.DisplayStyle.Image
Me.RadButtonElement4.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter
Me.RadButtonElement5.Image = New Bitmap(My.Resources.Hydrangeas, New Size(50, 50))
Me.RadButtonElement5.DisplayStyle = Telerik.WinControls.DisplayStyle.Image
Me.RadButtonElement5.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter
Me.RadButtonElement6.Image = New Bitmap(My.Resources.Koala, New Size(50, 50))
Me.RadButtonElement6.DisplayStyle = Telerik.WinControls.DisplayStyle.Image
Me.RadButtonElement6.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter
RadCarousel1.EnableAutoLoop = True
RadRotator1.Running = True
End Sub
```

[C#] Configure images and start rotating and looping for RadRotator and RadCarousel

```
private void RadForm1_Load(object sender, EventArgs e)
{
this.radButtonElement1.Image = new Bitmap(Properties.Resources.Lighthouse, new Size(200,
200));
this.radButtonElement1.DisplayStyle = Telerik.WinControls.DisplayStyle.Image;
this.radButtonElement1.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter;
this.radButtonElement2.Image = new Bitmap(Properties.Resources.Penguins, new Size(200,
200));
this.radButtonElement2.DisplayStyle = Telerik.WinControls.DisplayStyle.Image;
this.radButtonElement2.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter;
this.radButtonElement3.Image = new Bitmap(Properties.Resources.Tulips, new Size(200, 200
this.radButtonElement3.DisplayStyle = Telerik.WinControls.DisplayStyle.Image;
this.radButtonElement3.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter;
this.radButtonElement4.Image = new Bitmap(Properties.Resources.Desert, new Size(50, 50));
this.radButtonElement4.DisplayStyle = Telerik.WinControls.DisplayStyle.Image;
this.radButtonElement4.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter;
this.radButtonElement5.Image = new Bitmap(Properties.Resources.Hydrangeas, new Size(50,
50));
this.radButtonElement5.DisplayStyle = Telerik.WinControls.DisplayStyle.Image;
this.radButtonElement5.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter;
this.radButtonElement6.Image = new Bitmap(Properties.Resources.Koala, new Size(50, 50));
this.radButtonElement6.DisplayStyle = Telerik.WinControls.DisplayStyle.Image;
this.radButtonElement6.ImageAlignment = System.Drawing.ContentAlignment.MiddleCenter;
radCarousel1.EnableAutoLoop = true;
radRotator1.Running = true;
}
```

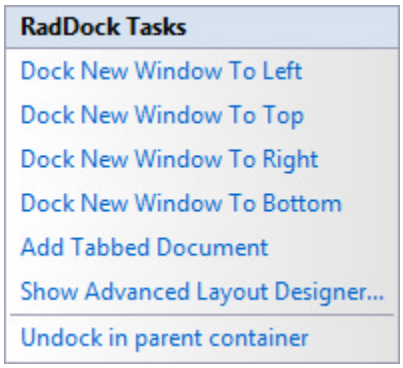
13. Press **Ctrl-F5** to run the application.

16.4 Using the Design Time Interface

RadDock

Smart Tag

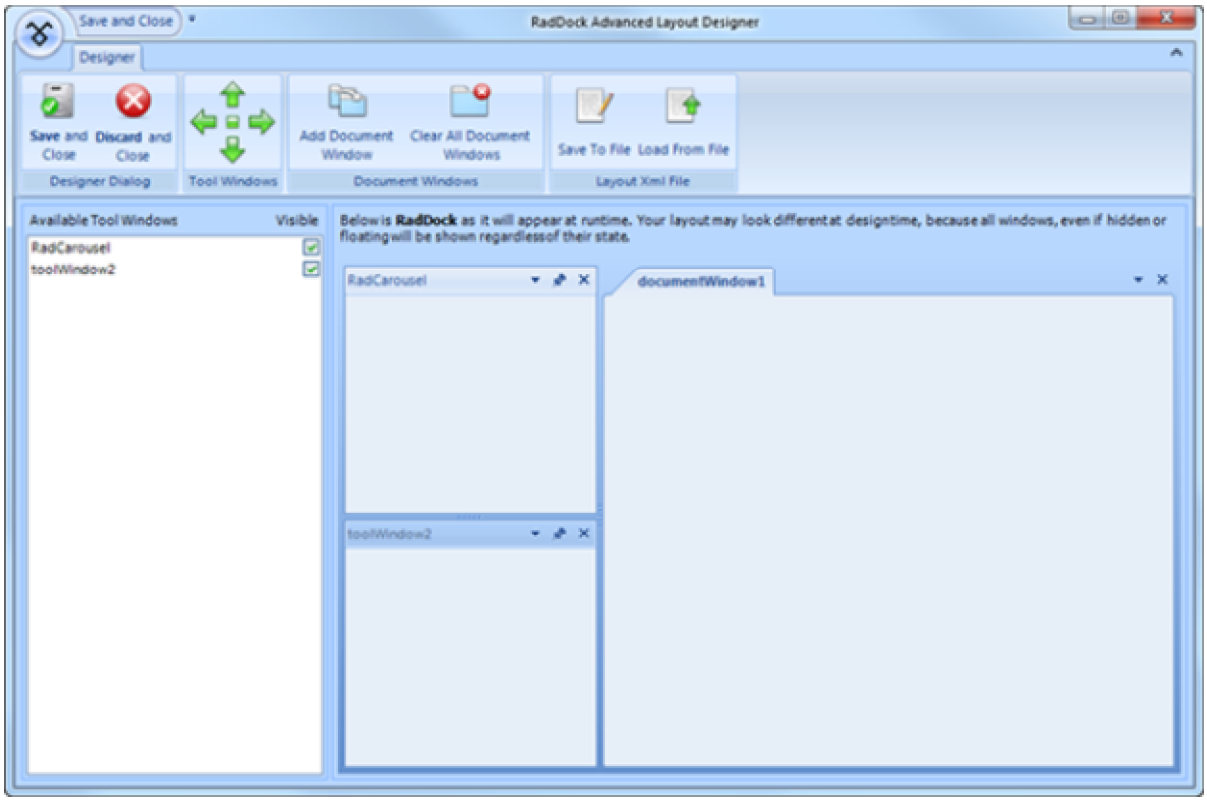
The Smart Tag for the RadDock lets you quickly access common tasks involved with docking new windows, tabbed documents and layout. The unique items are:



- **Dock New Window To Left/Top/Right/Bottom**: Adds tool windows to their respective positions within RadDock.
- **Add Tabbed Document**: Adds DocumentWindows to RadDock.
- **Show Advanced Layout Designer**: This designer lets you work with tool windows and documents windows with drag-and-drop ease similar to the runtime behavior, then save the results back to the design environment.

Advanced Layout Designer

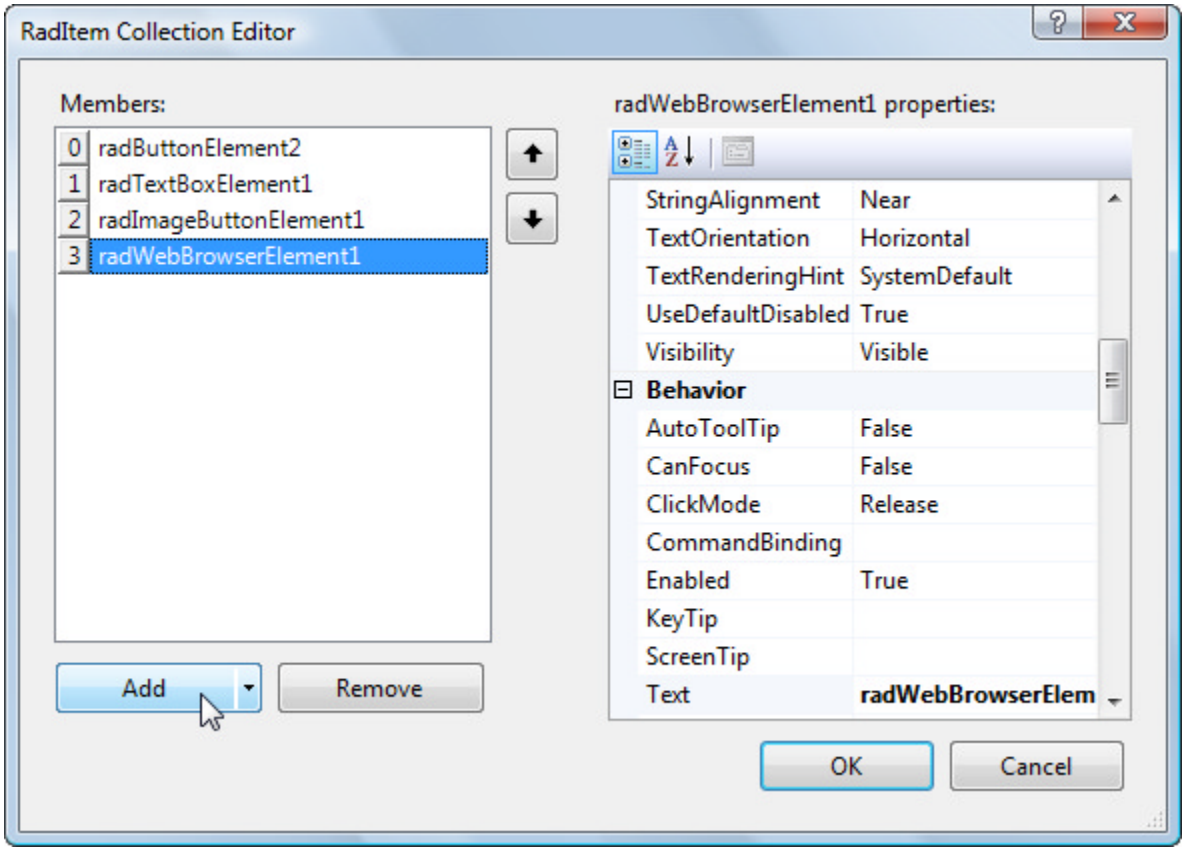
Use the Advanced Layout Designer to quickly set up the initial RadDock layout.



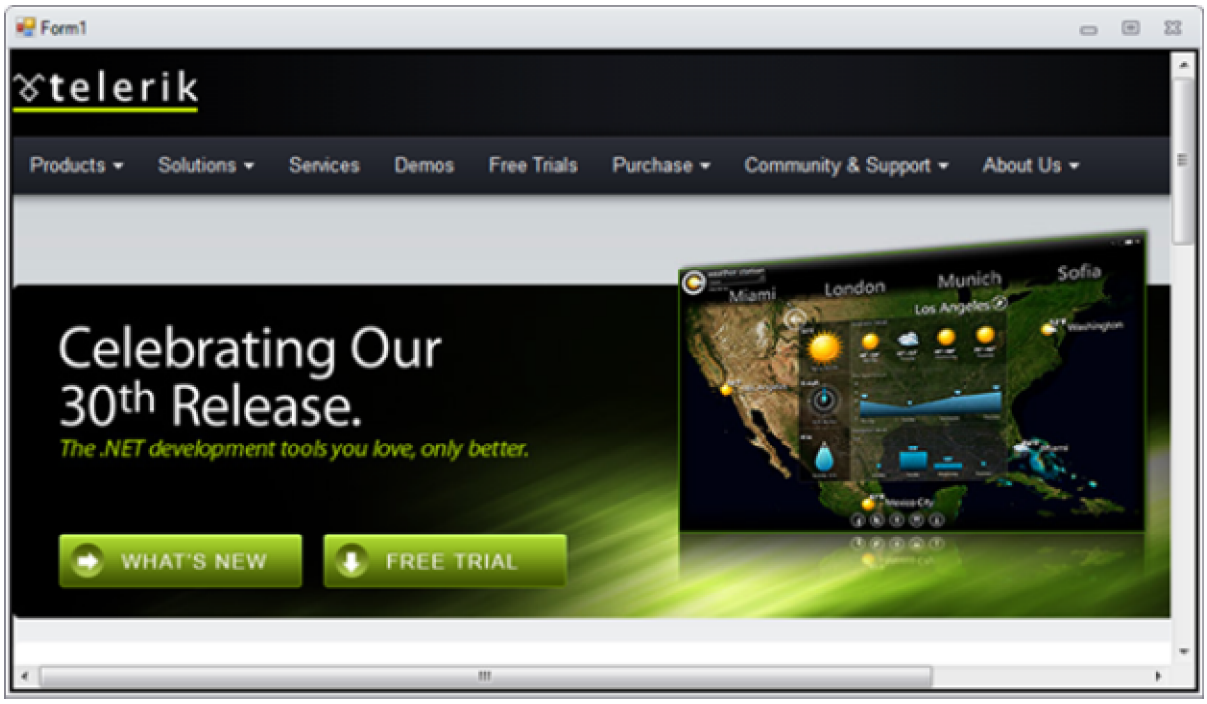
To add tool windows to the layout, select one of the predifined buttons in the Tool Windows pane. To add DocumentWindow click "Add Document Window" button. You drag-and-drop tool and document windows, just as you can at run time. When you are satisfied with your layout, click the **Save and Close** button to close the Advanced Layout Designer and apply the layout to RadDock control on your form. To abandon your changes click the window "x" close button in the upper right hand corner of the form ot click gthe"Discard and Close" button.

RadRotator

The RadRotator Smart Tag has only standard tasks common with all other RadControls. Use the Property window to access the **Items** collection property. Use the drop down button to add and configure RadElements.

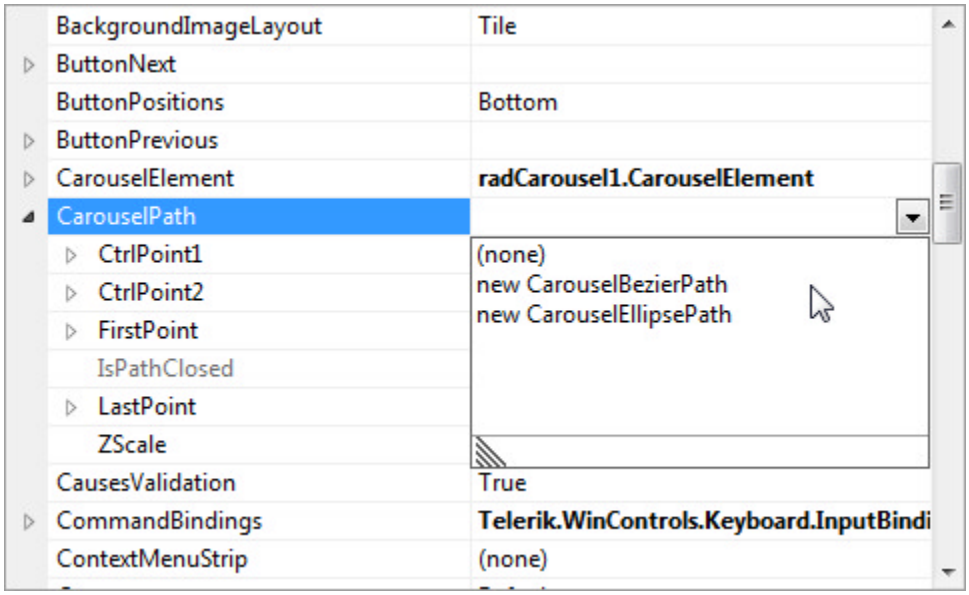


After the rotator is populated, you can navigate through the elements right in the designer using the next and back buttons. Set the **Running** property true to have the items animate at design time.

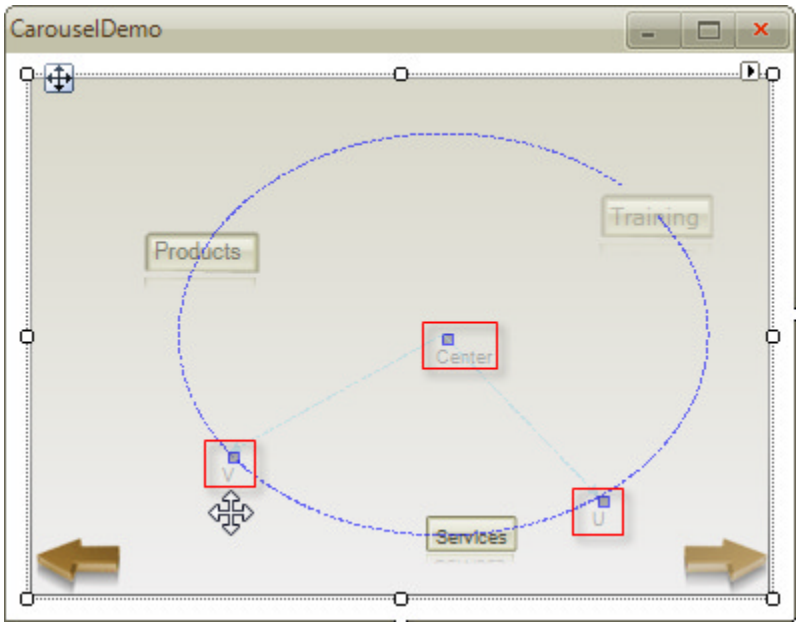


RadCarousel

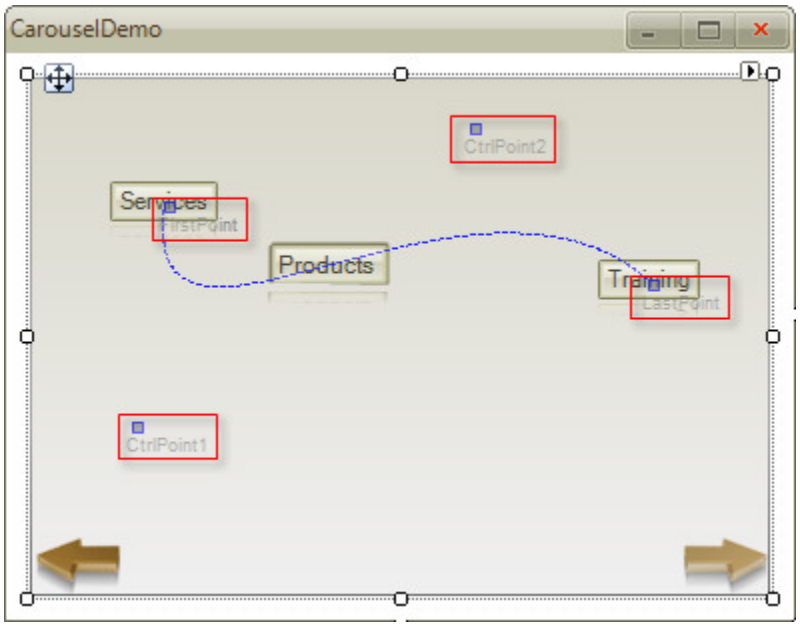
The RadCarousel Smart Tag has a **Edit Items** link to access the **Items** collection. The other key design-time aspect is the **CarouselPath**. In the CarouselPath property you can choose between a Bezier or elliptical path.



By default CarouselPath is a **CarouselEllipsePath**. Drag the **Center** point grip to move the entire carousel control around the form during design-time. Drag the two control points **U** and **V** to alter the tilt and dimensions of the ellipse.



If you choose **CarouselBezierPath**, you get a FirstPoint, LastPoint and two control points "CtrlPoint1" and "CtrlPoint2". You can drag all four points in the designer to define the curve.



You can arrange the points only in 2D. Z coordinates must be set programmatically if needed.

Also note that you can drag the control points over the whole designer area, i.e. you are not limited to the form only.

16.5 Programming the "Real Estate" Controls

Working with RadDock

RadDock controls the overall behavior of ToolWindows and DocumentWindows that it manages. This includes enabling basic capabilities of docking, floating and hiding, handling tabbed document behavior, serialization of

layout and providing access to collections of managed dockable objects.

To create a new **ToolWindow**, construct a **ToolWindow** instance, set properties and call the **RadDock DockWindow()** method, passing a reference to the **ToolWindow** and a **DockPosition** enumeration value.

To create a **DocumentWindow**, construct an instance of **DocumentWindow**, assign properties and call the **RadDock AddDocument()** method, passing the **DocumentWindow** instance.

[VB] Creating and Adding a DockWindows

```
'create tool window
Dim toolWindow As New ToolWindow()
toolWindow.Text = "ToolTabStrip"
'create and add controls to the tool window
Dim browser As New WebBrowser()
browser.Url = New Uri("http://www.telerik.com")
browser.Dock = DockStyle.Fill
toolWindow.Controls.Add(browser)
'dock the tool windows to the top
Me.radDock1.DockWindow(toolWindow, DockPosition.Top)

'create document window
Dim documentWindow As New DocumentWindow()
documentWindow.Text = "Document Container"
'create and add controls to the document window
Dim browser2 As New WebBrowser()
browser2.Url = New Uri("http://www.telerik.com/products/winforms.aspx")
browser2.Dock = DockStyle.Fill
documentWindow.Controls.Add(browser2)
'add the document window to raddock
Me.radDock1.AddDocument(documentWindow)
```

[C#] Creating and Adding a DockWindows

```
//create tool window
ToolWindow toolWindow = new ToolWindow();
toolWindow.Text = "ToolTabStrip";
//create and add controls to the tool window
WebBrowser browser = new WebBrowser();
browser.Url = new Uri("http://www.telerik.com (http://www.telerik.com/)");
browser.Dock = DockStyle.Fill;
toolWindow.Controls.Add(browser);
//dock the tool windows to the top
this.radDock1.DockWindow(toolWindow, DockPosition.Top);
//create document window
DocumentWindow documentWindow = new DocumentWindow();
documentWindow.Text = "Document Container";
//create and add controls to the document window
WebBrowser browser2 = new WebBrowser();
browser2.Url = new Uri("http://www.telerik.com/products/winforms.aspx");
browser2.Dock = DockStyle.Fill;
documentWindow.Controls.Add(browser2);
//add the document window to raddock
this.radDock1.AddDocument(documentWindow);
```



Once you have the RadDock populated, you can retrieve the collection of DockWindow instances by accessing the **DockWindows** property of the control. This property returns a DockWindowCollection with all dockables, even those that are un-pinned or hiddden.

[VB] Finding specified window

```
Dim window As DockWindow = Nothing
For Each currentWindow As DockWindow In radDock1.DockWindows
        If currentWindow.Text.Equals("My Window Name") Then
                window = currentWindow
                Exit For
        End If
Next
```

[C#] Finding specified window

```
DockWindow window = null;
foreach (DockWindow currentWindow in radDock1.DockWindows)
{
if (currentWindow.Text.Equals("My Window Name"))
{
```

```
window = currentWindow;
break;
}
}
```

You can hide, show and make a window float (only ToolWindows) by calling the appropriate method of the DockWindow instance:

[VB] Hiding DockWindow

```
ToolWindow1.Hide()
```

[C#] Hiding DockWindow

```
toolWindow1.Hide();
```

You can close all tabbed documents at once, close a specified window or close collection of dock windows.

[VB] Closing DockWindows

```
'close all dock windows
radDock1.CloseAllWindows()
'close just the specified window
radDock1.CloseWindow(documentWindow1)
'close all dock windows instances that are in the collection
Dim list As New List(Of DockWindow)()
list.Add(toolWindow)
list.Add(documentWindow1)
radDock1.CloseWindows(list)
```

[C#] Closing DockWindows

```
//close all dock windows
radDock1.CloseAllWindows();
//close just the specified window
radDock1.CloseWindow(documentWindow1);
//close all dock windows instances that are in the collection
List<DockWindow> list = new List<DockWindow>();
list.Add(toolWindow);
list.Add(documentWindow1);
radDock1.CloseWindows(list);
```

RadDock has two different methods which you can call if you want to remove a ToolWindow/DocumentWindow. The of from these methods depends on the **CloseAction** property which can have one of these values

- **Close -** When a **DockWindow** has its **CloseAction** set to **Close**, this window is closed and detached (but not disposed) from its **RadDock**. You cannot manage it from **RadDock** collections anymore.
- **Hide -** When a **DockWindow** has its **CloseAction** set to Hide, this window is only closed (hidden) in **RadDock**. However, this window is not detached from its **RadDock** and you can manage it from the **RadDock** collections.
- **CloseAndDispose -** When a **DockWindow** has its **CloseAction** set to **CloseAndDispose**, this window is closed and then disposed.

RemoveWindow

This method closes and detaches (but does not dispose) a DockWindow regardless of its **CloseAction** property. This will happen if you pass only a DockWindow as a parameter. In addition, you can pass a **CloseAction** parameter in the RemoveWindow method. In this case, the RemoveWindow method will act differently in accordance with this parameter.

CloseWindow

This method has a different result on a DockWindow instance, in accordance with the **CloseAction** property of this instance.

Default behavior

DocumentWindow: If you do not set the **CloseAction** for a DocumentWindow and you close this DocumentWindow from the UI, this window is closed with **CloseAction** set to CloseAndDispose, i.e. the DocumentWindow is disposed and you cannot reuse it.
ToolWindow: If you do not set the **CloseAction** for a ToolWindow and you close the ToolWindow from the UI, this window is closed with **CloseAction** set to Hide, i.e. the ToolWindow is not disposed and you can reuse it later.

Other Useful RadDock Properties

RadDock Properties

- **ActiveWindow**: Sets or gets the currently active DockWindow (ToolWindow or DocumentWindow).
- **AutoDetectMdiChildren**: Allows **RadDock** to automatically control MDI child windows of a parent form.
- **DockWindows**: A collection of IDockWindow objects being managed by the RadDock.
- **DockWindows.ToolWindows**: A collection of the ToolWindows being managed by the RadDock.
- **DockWindows.DocumentWindows**: Gets a collection of the DocumentWindows being managed by the RadDock.
- **FloatingWindows**: Gets a collections of the floating windows.
- **Dock**: This property is not specific to **RadDock**, but rather is a general **Controls** property that indicates what are of a parent control will be occupied. This **DockStyle** enumeration can be **Fill, Top, Bottom, Left, Right** and **None**.
- **MdiChildren**: An array of MDI forms parented in the RadDock. This array is accessible at runtime.
- **MainDocumentContainerVisible**: Indicates whether main document container will be initially visible.
- **QuickNavigatorSettings**: this property give you an object of type QuickNavigatorSettings. This object gives you a full control over the Quick Navigator features.

RadDock Methods

- **ActivateWindow(DockWindow window)**: Activates a DockWindow.
- **AddDocument(DockWindow window)**: Adds a DockWindow as a DocumentWindow in RadDock.
- **AutoHideWindow(DockWindow window)**: Makes the specified DockWindow auto-hidden.
- **AutoHideWindows(IEnumerable<DockWindow> windows, AutoHidePosition position)**: Makes the specified DockWindow collection auto-hidden.
- **CloseAllWindows()**: Closes all DockWindows. The CloseAction property specifies whether a ToolWindow/DocumentWindow will be Closed, ClosedAndDisposed or just Hidden.
- **CloseWindow(DockWindow window)**: Closes a specified ToolWindow or DocumentWindow.
- **DockWindow(DockWindow window, DockPosition position)**: Docks a DockWindow (ToolWindow/DocumentWindow) at the specified position.
- **DockWindow(DockWindow window, DockWindow target, DockPosition position)**: Docks a DockWindow at a position relative to the specified target DockWindow.
- **FloatToolTabStrip(ToolTabStrip strip, Rectangle bounds)**: Floats a ToolTabStrip with ToolWindows in a

new form. The form has **bounds** specified by the bounds parameter.

- **FloatWindow(DockWindow window)**: Floats the specified DockWindow in a new form.
- **FloatWindow(DockWindow window, Rectangle bounds)**: Floats the specified DockWindow in a new form. The form has **bounds** specified by the bounds parameter.
- **LoadFromXml(fileName)**: Deserializes **RadDock** layout structure.
- **SaveToXml(fileName)**: Serializes the **RadDock** layout structure. This method does not save content. See Loading and Saving Layouts for more information.
- **RemoveWindow(DockWindow window)**: Removes a **ToolWindow** or **DocumentWindow** from the list of managed dockables.

DockWindows

Each DockWindow instance has information about its own state, position, etc. Here are some especially useful properties.

- **DockState**: Floating, Docked, TabbedDocument, AutoHide (un-pinned), Hidden.
- **DockType**: ToolWindow (i.e. a DockPanel), Document.
- **Image**: displayed in the window title bar.
- **Text**: displayed in the window title bar.

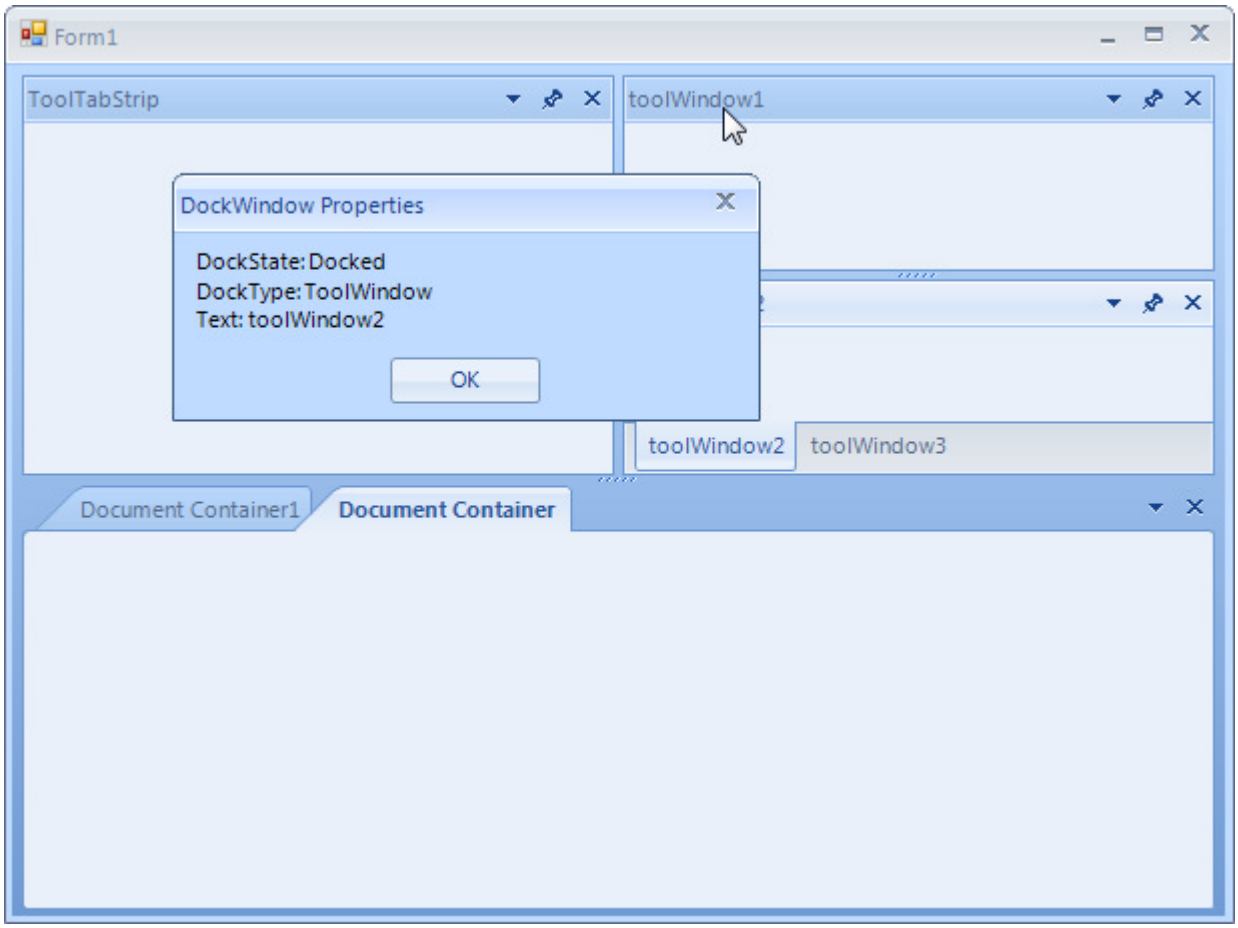
[VB] IDockable Methods

```
If radDock1.ActiveWindow IsNot Nothing Then
    Dim builder As New StringBuilder()
    builder.Append("DockState: " & radDock1.ActiveWindow.DockState.ToString())
    builder.Append(Environment.NewLine)
    builder.Append("DockType: " & radDock1.ActiveWindow.DockType.ToString())
    builder.Append(Environment.NewLine)
    builder.Append("Text: " + radDock1.ActiveWindow.Text)

    RadMessageBox.Show(builder.ToString(), "DockWindow Properties")
End If
```

[C#] IDockable Methods

```
if (radDock1.ActiveWindow != null)
{
StringBuilder builder = new StringBuilder();
builder.Append("DockState: " + radDock1.ActiveWindow.DockState.ToString());
builder.Append(Environment.NewLine);
builder.Append("DockType: " + radDock1.ActiveWindow.DockType.ToString());
builder.Append(Environment.NewLine);
builder.Append("Text: " + radDock1.ActiveWindow.Text);
RadMessageBox.Show(builder.ToString(), "DockWindow Properties");
}
```



RadDock also has a **ActivateWindow()** method that puts focus on the specified window. The example below iterates the managed dockable windows and if it finds one with the text "Products", activates that window.

[VB] Activating DockWindow

```
Dim foundDockable As DockWindow = Nothing
For Each dockable As DockWindow In radDock1.DockWindows
    If dockable.Text.Equals("Products") Then
        foundDockable = dockable
        Exit For
    End If
Next

If foundDockable IsNot Nothing Then
    radDock1.ActivateWindow(foundDockable)
End If
```

[C#] Activating DockWindow

```
DockWindow foundDockable = null;
foreach (DockWindow dockable in radDock1.DockWindows)
{
if (dockable.Text.Equals("Products"))
{
foundDockable = dockable;
break;
}
```

```
}
if (foundDockable != null)
{
radDock1.ActivateWindow(foundDockable);
}
```

In certain cases, you may need to perform specific operations depending on the currently activated HostWindow in regards to the form/user control that it contains.
In order to do this, you should first subscribe to the **ActiveWindowChanged** event and then execute the following code snippet in the **ActiveWindowChanged** event handler:

[VB] ActiveWindowChanged

```
Private Sub radDock1_ActiveWindowChanged(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.Docking.DockWindowEventArgs)
    Dim hostWin As HostWindow = TryCast(e.DockWindow, HostWindow)
    If Not hostWin Is Nothing Then
        If TypeOf hostWin.Content Is VegetablesForm Then
            ' custom implementation here
        End If
    End If
End Sub
```

[C#] ActiveWindowChanged

```
void radDock1_ActiveWindowChanged(object sender,
Telerik.WinControls.UI.Docking.DockWindowEventArgs e)
{
   HostWindow hostWin = e.DockWindow as HostWindow;
   if (hostWin != null)
   {
       if (hostWin.Content is VegetablesForm)
       {
           // custom implementation here
       }
   }
}
```

In order to get a HostWindow that hosts a particular form/user control instance, you should call the GetHostWindows method passing the contained control as a parameter. Supposing that vegetablesForm is an instance of type VegetablesForm, we can use the following code snippet:

[VB] Get HostWindow by its content

```
Dim vegetablesWindow As HostWindow = Me.RadDock1.GetHostWindow(VegetablesForm)
```

[C#] Get HostWindow by its content

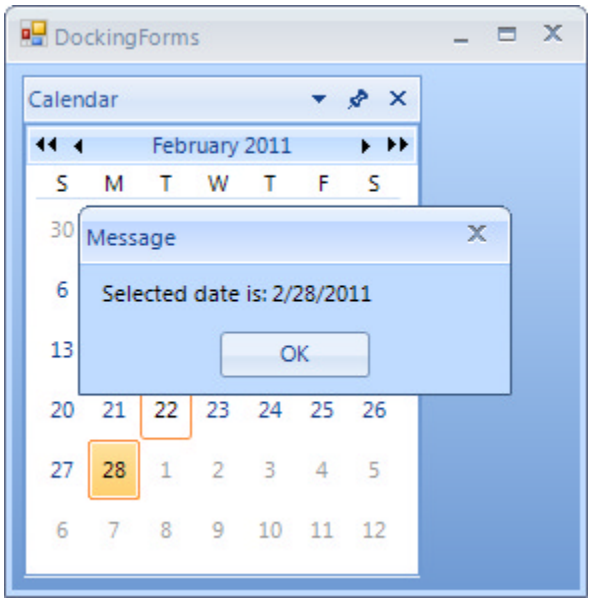
```
HostWindow vegetablesWindow = this.radDock1.GetHostWindow(vegetablesForm);
```

Dock UserControl with custom event

This tutorial walks you through a real-world scenario that you can have in your application. At a high level, this scenario involves:

1. Placing controls on different user controls.
2. Docking the user controls inside RadDock.
3. Implementing events, methods and properties in the user controls which allow you to have communication between the forms/user controls.

In our particular case we dock a user control instance in a RadDock instance. The user control itself contains a RadCalendar. When the user clicks the RadCalendar a custom DateChanged event for the UserControl1 fires, showing a dialog with the selected date. Using this as a model you can create UserControls with custom properties, methods and events for more complex situations.





You can find the complete source for this project at:

\RealEstate\<VB|CS>\DockUserControl

Implement the DockUserControl

1. In Visual Studio create a new Windows Application.
2. Create a new UserControl - from the Solution Explorer, right-click the project node in the Solution Explorer and select "Add >> UserControl...". Name the new user control CalendarPanel.
3. Drop a RadCalendar instance on the CalendarPanel design surface.
4. Set the RadCalendar Dock property to Fill.
5. In the Property Window locate the SelectionChanged event and double-click it to create an event handler.
6. Add a delegate and event to be surfaced by the CalendarPanel control when the date selection changes:

   [VB.NET] Setting up DateChanged Event

```
Public Delegate Sub DateChangedHandler(ByVal [date] As DateTime)
Public Event DateChanged As DateChangedHandler
```

   [C#] Setting up DateChanged Event

```
public delegate void DateChangedHandler(DateTime date);
public event DateChangedHandler DateChanged;
```

7. In the **SelectionChanged** event handler add the following code:

   [VB.NET] Handling the RadCalendar SelectionChanged event

```
Private Sub RadCalendar1_SelectionChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles RadCalendar1.SelectionChanged
    RaiseEvent DateChanged((TryCast(sender, RadCalendar)).SelectedDate)
End Sub
```

[C#] Handling the RadCalendar SelectionChanged event

```
private void radCalendar1_SelectionChanged(object sender, EventArgs e)
{
   if (DateChanged != null)
   {
       DateChanged((sender as Telerik.WinControls.UI.RadCalendar).SelectedDate);
   }
}
```

8. In the Solution Explorer, double-click the main form to open its design surface.
9. Drop a RadDock instance on the opened design surface. Set the RadDock Dock property Fill.
10. Dock the CalendarPanel user control to RadDock in the Form_Load event handler using the following code snippet:

[VB.NET] Docking CalendarPanel user control in RadDoc

```
Dim calendarPanel As New CalendarPanel()
Dim host As HostWindow = Me.RadDock1.DockControl(calendarPanel,
Telerik.WinControls.UI.Docking.DockPosition.Left)
host.Text = "Calendar"
AddHandler calendarPanel.DateChanged, AddressOf calendarPanel_DateChanged
```

[C#] Docking CalendarPanel user control in RadDoc

```
CalendarPanel calendarPanel = new CalendarPanel();
HostWindow host = this.radDock1.DockControl(calendarPanel,
Telerik.WinControls.UI.Docking.DockPosition.Left);
host.Text = "Calendar";
calendarPanel.DateChanged += calendarPanel_DateChanged;
```

11. Add an event handler for the CalendarPanel DateChanged event:

[VB.NET] Handling the Custom DateChanged event

```
Sub calendarPanel_DateChanged(ByVal [date] As DateTime)
    RadMessageBox.Show("Selected date is: " + [date].ToShortDateString())
End Sub
```

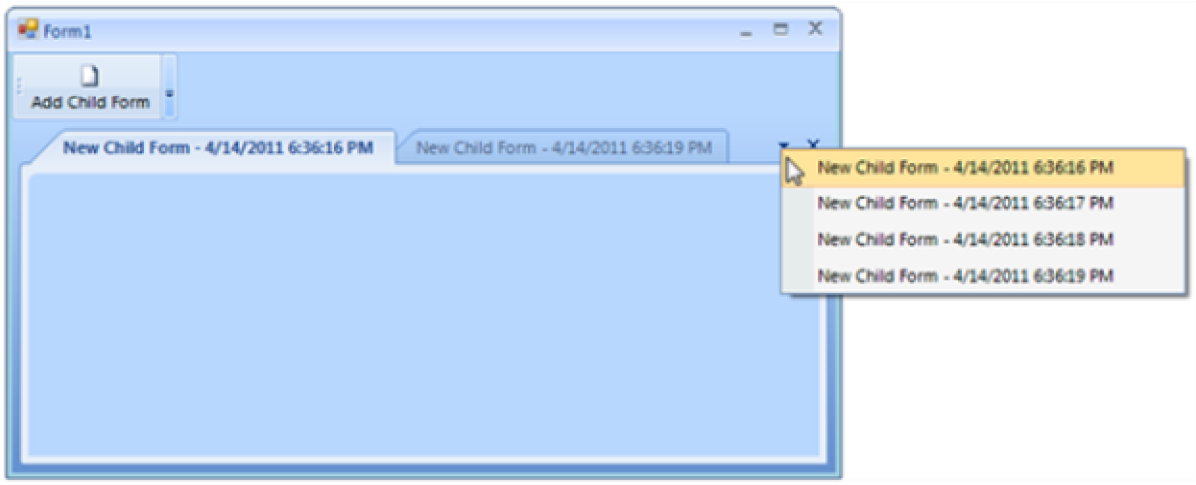
[C#] Handling the Custom DateChanged event

```
void calendarPanel_DateChanged(DateTime date)
{
   RadMessageBox.Show("Selected date is: " + date.ToShortDateString());
}
```

12. Press F5 to run the application. Click the cells in the calendar to display the date in a message dialog.

Automatic Hosting of Multiple Document Interface (MDI) Child Forms

RadDock can automatically host Multiple Document Interface (MDI) child forms as tabbed documents simply by setting the **AutoDetectMdiChildForms** property to True. The rest of your logic will be identical to a standard MDI application where the parent form AutoDetectMdiChildForms is set True, and child forms MdiParent is set to the parent form.



[VB] Hosting MDI Child Forms

```
Private Sub MDIParent_Load(sender As Object, e As EventArgs)
 Me.IsMdiContainer = True
 radDock1.AutoDetectMdiChildForms = True
 End Sub
Private Sub btnAddChildForm_Click(sender As Object, e As EventArgs)
 Dim childForm As New RadForm()
 childForm.MdiParent = Me
 childForm.Text = "New Child Form - " + DateTime.Now.ToString()
 childForm.Show()
End Sub
```

[C#] Hosting MDI Child Forms

```
private void MDIParent_Load(object sender, EventArgs e)
{
 this.IsMdiContainer = true;
 radDock1.AutoDetectMdiChildForms = true;
}
private void btnAddChildForm_Click(object sender, EventArgs e)
{
 RadForm childForm = new RadForm();
 childForm.MdiParent = this;
 childForm.Text = "New Child Form - " + DateTime.Now.ToString();
 childForm.Show();
}
```

Call **ActivateMdiChild()** to select a given child form:

[VB] Activating an MDI Child Form in the DockingManager

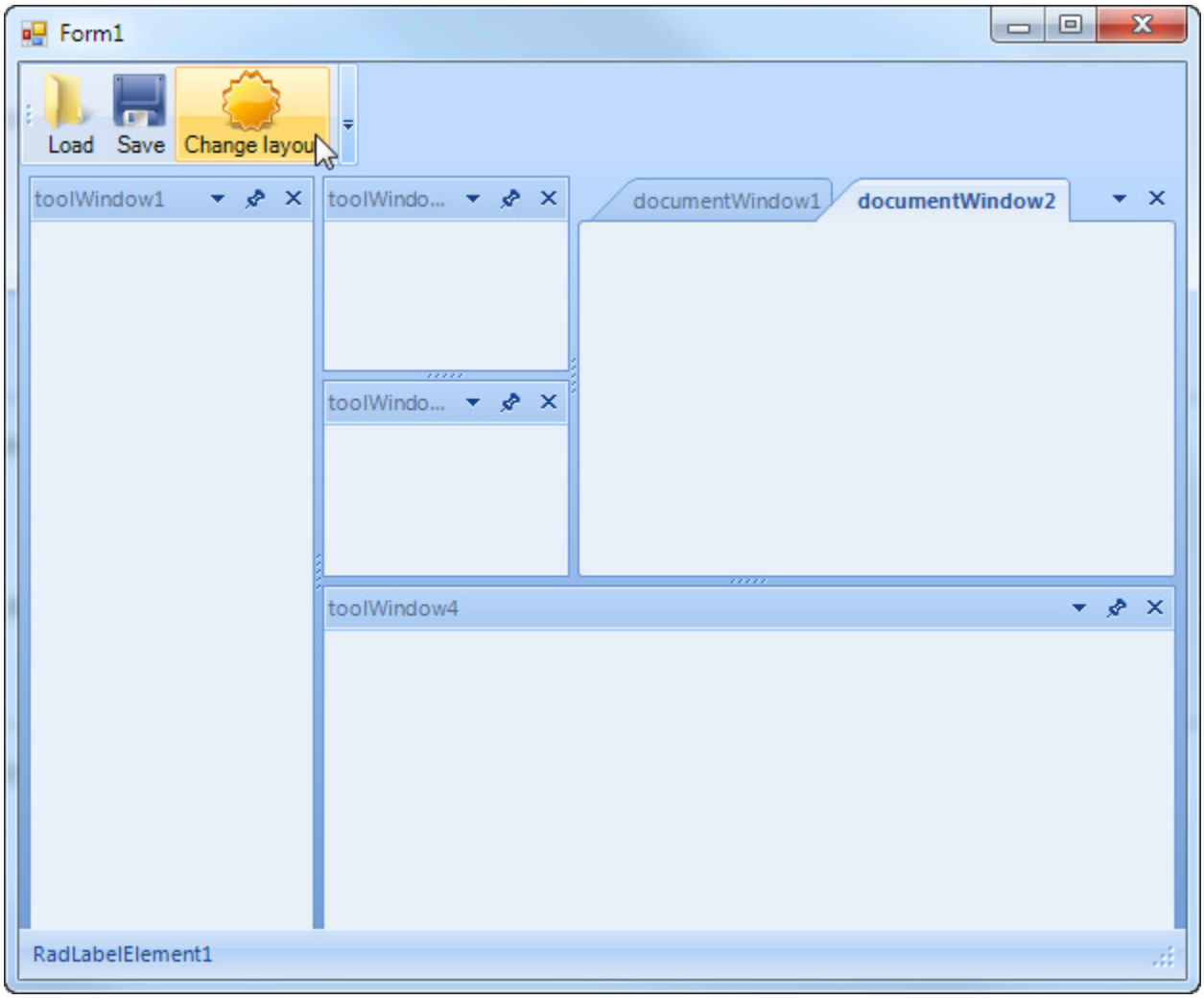
```
radDock1.ActivateMdiChild(childForm)
```

[C#] Activating an MDI Child Form in the DockingManager

```
radDock1.ActivateMdiChild(childForm);
```

Saving and Loading Layout

RadDock records the size and arrangement of all DockWindows within the RadDock using the SaveToXml() method. This saves all ToolWindows and DocumentWindows regardless of state to an XML file. The inverse method, LoadFromXml(), reads the XML file and recreates the DockWindows, their state and placement.





You can find the complete source for this project at:

\RealEstate\<VB|CS>\ProgrammingDock

[VB] Loading and Saving Layout

```
Private Sub btnLoad_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnLoad.Click
Dim dialog As New OpenFileDialog()
dialog.DefaultExt = "xml"
If dialog.ShowDialog() = DialogResult.OK Then
RadDock1.LoadFromXml(dialog.FileName)
lblStatus.Text = "Loaded layout " + dialog.FileName
End If
End Sub
Private Sub btnSave_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnSave.Click
Dim dialog As New SaveFileDialog()
dialog.DefaultExt = "xml"
If dialog.ShowDialog() = DialogResult.OK Then
RadDock1.SaveToXml(dialog.FileName)
```

```
lblStatus.Text = "Saved layout as " + dialog.FileName
End If
End Sub
Private Sub btnLayout_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnLayout.Click
ToolWindow1.Close()
ToolWindow3.Close()
ToolWindow4.Close()
lblStatus.Text = "Layout changed"
End Sub
```

[C#] Loading and Saving Layout

```
private void btnLoad_Click(object sender, EventArgs e)
{
OpenFileDialog dialog = new OpenFileDialog();
dialog.DefaultExt = "xml";
if (dialog.ShowDialog() == DialogResult.OK)
{
radDock1.LoadFromXml(dialog.FileName);
lblStatus.Text = "Loaded layout " + dialog.FileName;
}
}
private void btnSave_Click(object sender, EventArgs e)
{
SaveFileDialog dialog = new SaveFileDialog();
dialog.DefaultExt = "xml";
if (dialog.ShowDialog() == DialogResult.OK)
{
radDock1.SaveToXml(dialog.FileName);
lblStatus.Text = "Saved layout as " + dialog.FileName;
}
}
private void btnLayout_Click(object sender, EventArgs e)
{
toolWindow1.Close();
toolWindow3.Close();
toolWindow4.Close();
lblStatus.Text = "Layout changed";
}
```



Be aware that only the layout structure is persisted. If you want to save and load the content you will need to implement your own custom logic.

RadRotator

The first task working with RadRotator is to load the **Items** collection with any RadElement. You can also add a RadHostItem that wraps some other standard Windows Forms control. After the items are loaded, call the RadRotator **Start()** method to begin animating between items and **Stop()** to end animation. You could instead set the **Running** property to True to begin animating.

When the Running property is set to true or the Start() method is called, the **StartRotation** event fires. Using the arguments passed to StartRotation you can cancel animation before it begins. When the animation is ended, the **StopRotation** event is fired.

While the StartRotation and StopRotation event bracket the entire animation, **BeginRotate** and **EndRotate** events are fired just before and after a transition between rotator items. BeginRotate passes a

BeginRotateEventArgs containing a **From** property with the index of the previous rotator item, a **To** property with the index of the upcoming rotator item that will display next, and a **Cancel** property that can be set to prevent the upcoming rotator item from displaying.

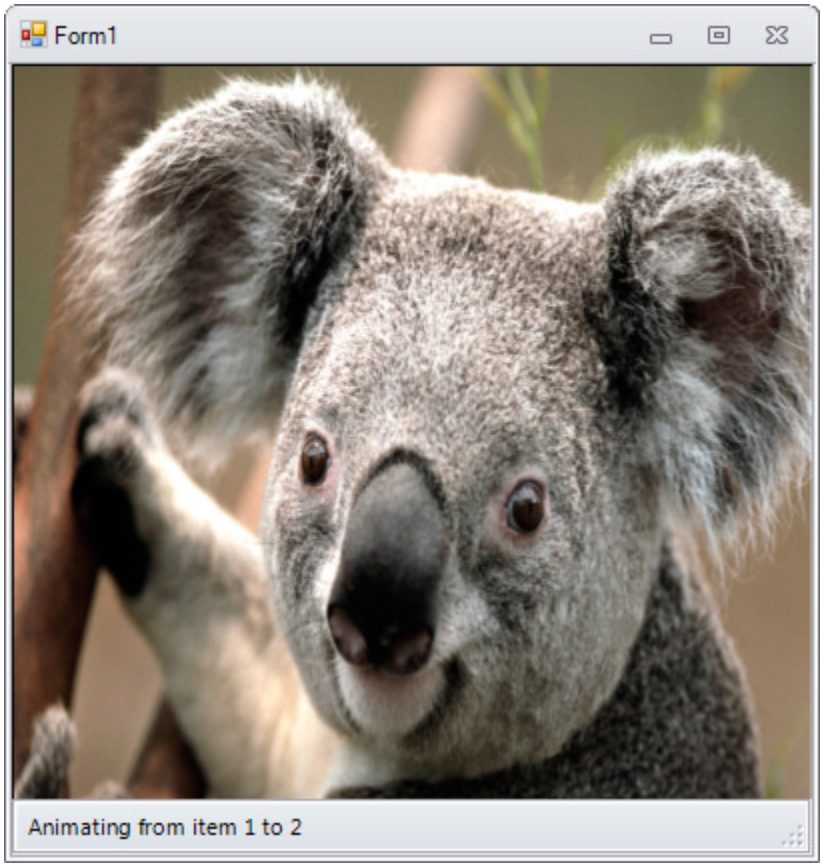
The other significant properties of RadRotator are:

LocationAnimation controls the direction of movement when an item is animated. A positive LocationAnimation.Width property value causes the transition between items to move left, and when negative to move right. A positive LocationAnimation.Height property value causes the transition between items to move up and when negative to move down. For example, a value of -1, -1 makes the image appear from the upper left and travel lower right.

OpacityAnimation when true (the default), causes items to become more transparent as they move off screen, while the item moving onscreen to replace the previous item becomes opaque. The animation appears to smoothly blend from one item to the other. When false, new items move over the top of old items without the fading effect.

CurrentIndex gets or sets the currently displayed rotator item.

The following code example uses some stock images in the Properties.Settings of a project. A number of RadButtonElements are loaded as items and the animation begins. The status strip displays RadRotator events as they fire.



In the code below notice:

- The LoadRotator() creates one of the RadElement types, set the Text, Image and other properties of the element and finally add the element to the RadRotator Items collection.
- In the form Load event handler we're setting the AnimationFrames to be 30 instead of the default 10 to make the animation smoother. We're setting the Interval to "3000" or three seconds. The LocationAnimation is set to -1, -1 so that the images animate down from the upper left to lower right.

[VB] Loading and Configuring RadRotator

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 ' slow, smoother animation than default 10
 radRotator1.AnimationFrames = 30
 ' three seconds between rotations
 radRotator1.Interval = 3000
 ' animates moving almost straight forward
 radRotator1.LocationAnimation = New SizeF(-1, -1)
 LoadRotator()
 radRotator1.Start()
End Sub
Private Sub LoadRotator()
 Dim resourceSet As ResourceSet = Properties.Resources.ResourceManager.GetResourceSet
(CultureInfo.CurrentCulture, True, True)
 Dim enumerator As IDictionaryEnumerator = resourceSet.GetEnumerator()
 ' for each image resources, retrieve the image name and object,
 ' create and populate the menu item, then add to the drop down
 ' button items collection
 While enumerator.MoveNext()
  ' the key will be the name of the image, minus extension,
  ' also remove underline
  Dim key As String = (DirectCast(enumerator.Key, String)).Replace("_"C, " "C)
  ' the value will be the image object
  Dim image As Image = New Bitmap(DirectCast(enumerator.Value, Image), new Size(400,400))
  Dim element As New RadButtonElement()
  element.Image = image
  ' show image only
  element.DisplayStyle = DisplayStyle.Image
  element.Tag = key
  radRotator1.Items.Add(element)
 End While
End Sub
Private Sub radRotator1_BeginRotate(sender As Object, e As BeginRotateEventArgs)
 lblStatus.Text = [String].Format("Animating from item {0} to {1}", e.From, e.[To])
End Sub
Private Sub radRotator1_EndRotate(sender As Object, e As EventArgs)
 lblStatus.Text = "End rotation"
End Sub
Private Sub radRotator1_StartRotation(sender As Object, e As CancelEventArgs)
 lblStatus.Text = "Started rotation"
End Sub
Private Sub radRotator1_StopRotation(sender As Object, e As EventArgs)
 lblStatus.Text = "Stopped rotation"
End Sub
```

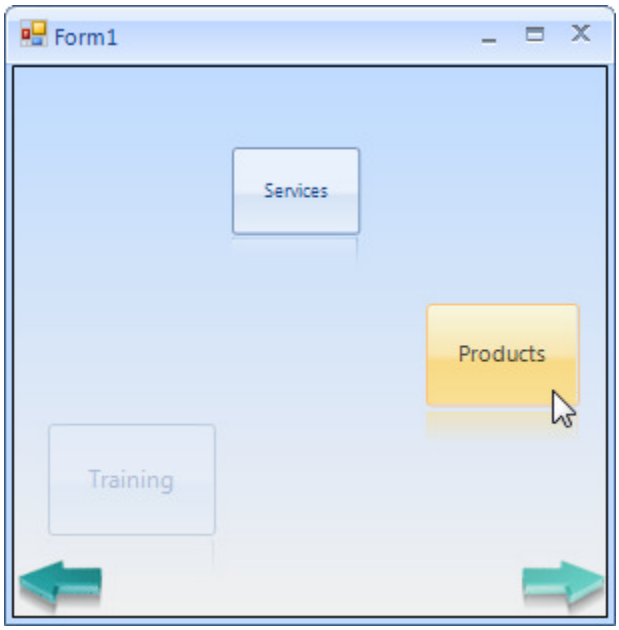
[C#] Loading and Configuring RadRotator

```
public Form1()
{
InitializeComponent();
// slow, smoother animation than default 10
radRotator1.AnimationFrames = 30;
// three seconds between rotations
radRotator1.Interval = 3000;
// animates moving almost straight forward
radRotator1.LocationAnimation = new SizeF(-1, -1);
```

```
LoadRotator();
radRotator1.Start();
}
private void LoadRotator()
{
ResourceSet resourceSet = Properties.Resources.ResourceManager.GetResourceSet(
CultureInfo.CurrentCulture, true, true);
IDictionaryEnumerator enumerator = resourceSet.GetEnumerator();
// for each image resources, retrieve the image name and object,
// create and populate the menu item, then add to the drop down
// button items collection
while (enumerator.MoveNext())
{
// the key will be the name of the image, minus extension,
// also remove underline
string key = ((string)enumerator.Key).Replace('_', ' ');
// the value will be the image object
Image image = new Bitmap((Image)enumerator.Value, new Size(400, 400));
RadButtonElement element = new RadButtonElement();
element.Image = image;
// show image only
element.DisplayStyle = DisplayStyle.Image;
element.Tag = key;
radRotator1.Items.Add(element);
}
}
private void radRotator1_BeginRotate(object sender, BeginRotateEventArgs e)
{
lblStatus.Text = String.Format("Animating from item {0} to {1}", e.From, e.To);
}
private void radRotator1_EndRotate(object sender, EventArgs e)
{
lblStatus.Text = "End rotation";
}
private void radRotator1_StartRotation(object sender, CancelEventArgs e)
{
lblStatus.Text = "Started rotation";
}
private void radRotator1_StopRotation(object sender, EventArgs e)
{
lblStatus.Text = "Stopped rotation";
}
```

RadCarousel

RadCarousel is similar to the RadRotator in that the first job is to fill the Items collection with RadElements. This can be done at design time, programmatically and through data binding. Here's a simple example of adding some RadButtonElements.



[VB] Adding Elements to the RadCarousel Items Collection

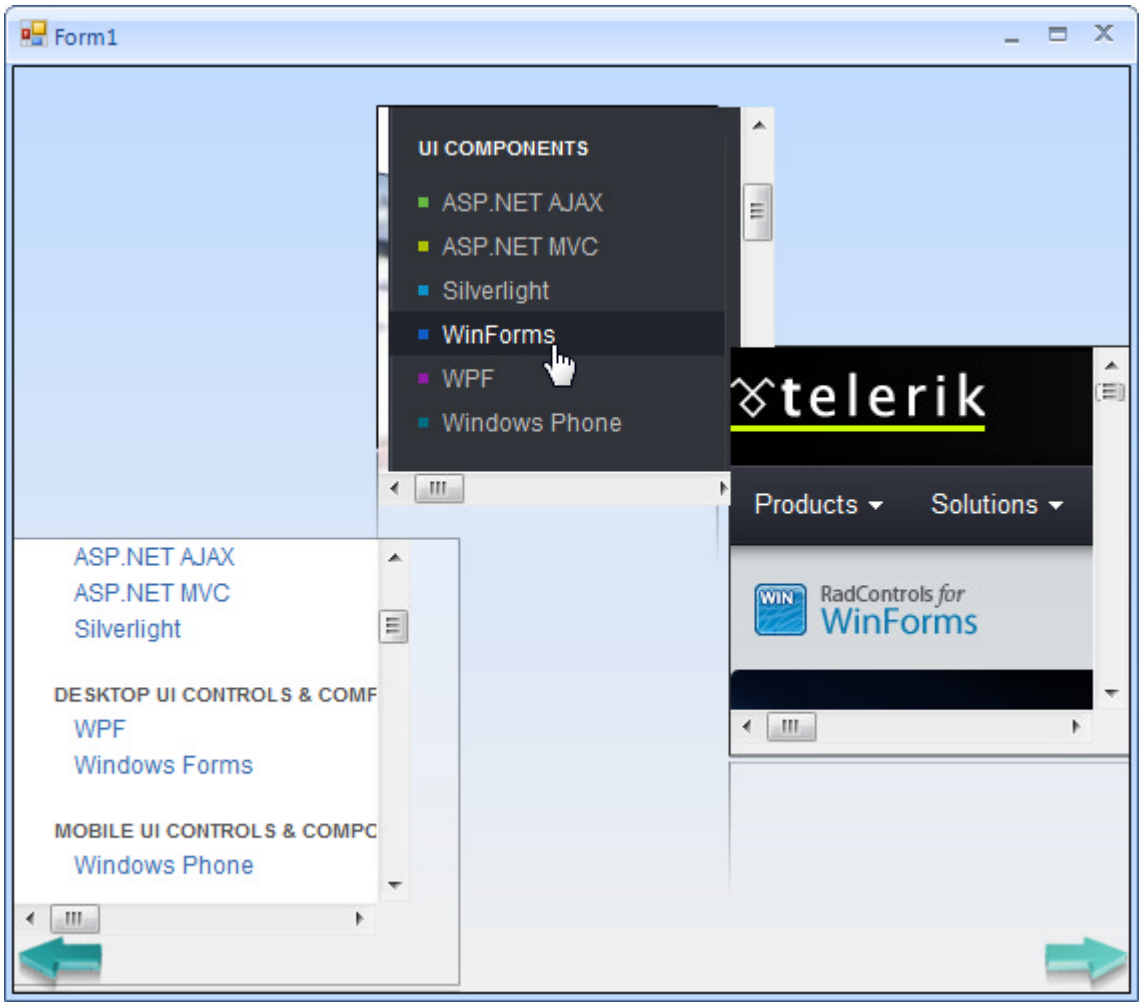
```
Dim btnProducts As New RadButtonElement("Products")
btnProducts.MinSize = New Size(75, 50)
radCarousel1.Items.Add(btnProducts)
Dim btnServices As New RadButtonElement("Services")
btnServices.MinSize = New Size(75, 50)
radCarousel1.Items.Add(btnServices)
Dim btnTraining As New RadButtonElement("Training")
btnTraining.MinSize = New Size(75, 50)
radCarousel1.Items.Add(btnTraining)
```

[C#] Adding Elements to the RadCarousel Items Collection

```
RadButtonElement btnProducts = new RadButtonElement("Products");
btnProducts.MinSize = new Size(75, 50);
radCarousel1.Items.Add(btnProducts);
RadButtonElement btnServices = new RadButtonElement("Services");
btnServices.MinSize = new Size(75, 50);
radCarousel1.Items.Add(btnServices);
RadButtonElement btnTraining = new RadButtonElement("Training");
btnTraining.MinSize = new Size(75, 50);
radCarousel1.Items.Add(btnTraining);
```

Adding Browser Pages

Browser pages can be added by making use of RadWebBrowserElement.



Simply create an instance of RadWebBrowserElement, set its size and Url and add it to the Items collection of RadCarousel.

[VB] Creating RadButtonElement with Web Browser Image

```
Dim webBrowser1 As New RadWebBrowserElement()
webBrowser1.MinSize = New System.Drawing.Size(200, 200)
webBrowser1.WebBrowserItem.Url = New Uri("http://www.telerik.com")
radCarousel1.Items.Add(webBrowser1)

Dim webBrowser2 As New RadWebBrowserElement()
webBrowser2.MinSize = New System.Drawing.Size(200, 200)
webBrowser2.WebBrowserItem.Url = New Uri("http://www.telerik.com/products/winforms.aspx")
radCarousel1.Items.Add(webBrowser2)

Dim webBrowser3 As New RadWebBrowserElement()
webBrowser3.MinSize = New System.Drawing.Size(200, 200)
webBrowser3.WebBrowserItem.Url = New Uri("http://tv.telerik.com/")
radCarousel1.Items.Add(webBrowser3)
```

[C#] Creating RadButtonElement with Web Browser Image

```
RadWebBrowserElement webBrowser1 = new RadWebBrowserElement();
webBrowser1.MinSize = new System.Drawing.Size(200, 200);
webBrowser1.WebBrowserItem.Url = new Uri("http://www.telerik.com
```

```
(http://www.telerik.com/)");
radCarousel1.Items.Add(webBrowser1);
RadWebBrowserElement webBrowser2 = new RadWebBrowserElement();
webBrowser2.MinSize = new System.Drawing.Size(200, 200);
webBrowser2.WebBrowserItem.Url = new Uri("http://www.telerik.com/products/winforms.aspx");
radCarousel1.Items.Add(webBrowser2);
RadWebBrowserElement webBrowser3 = new RadWebBrowserElement();
webBrowser3.MinSize = new System.Drawing.Size(200, 200);
webBrowser3.WebBrowserItem.Url = new Uri("http://tv.telerik.com (http://tv.telerik.com/)/");
radCarousel1.Items.Add(webBrowser3);
```

DataBinding



You can find the complete source for this project at:

\<VB|CS>\RealEstate\RadCarouselProgramming

Databinding takes an additional step compared to other RadControls that need a simple, one-column, set of data. You will need to handle the **NewCarouselItemCreating** event to add elements you want available later when the **ItemDataBound** event fires.

In the form Load event you can hook up your DataSource. In this example we use an RSS (Really Simple Syndication) URL to retrieve an XML file with columns for "title" and "link". The XmlTextReader object from the System.Xml namespace will read directly from an RSS URL. You can feed this reader to a DataSet using its ReadXml() method. Now you have a populated DataSet that can be assigned directly to the RadCarousel DataSource property except that we don't want to display every last record. Extract just a few rows into a generic List of DataRow and assign that list instead.

[VB] Assigning the DataSource

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 radCarousel1.VisibleItemCount = 5
 Dim reader As New XmlTextReader("http://silverlight.net/blogs/MainFeed.aspx")
 Dim ds As New DataSet()
 ds.ReadXml(reader)
 ' get just the top three links
 Dim topRows As New List(Of DataRow)()
 Dim i As Integer = 0
 While i < radCarousel1.VisibleItemCount
  topRows.Add(ds.Tables("item").Rows(i))
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
 radCarousel1.DataSource = topRows
End Sub
```

[C#] Assigning the DataSource

```
private void RadForm1_Load(object sender, EventArgs e)
{
 radCarousel1.VisibleItemCount = 5;
 XmlTextReader reader = new XmlTextReader("http://silverlight.net/blogs/MainFeed.aspx");
 DataSet ds = new DataSet();
 ds.ReadXml(reader);
 // get just the top three links
 List<DataRow> topRows = new List<DataRow>();
 for (int i = 0; i < radCarousel1.VisibleItemCount; i++)
 {
```

```
   topRows.Add(ds.Tables["item"].Rows[i]);
 }
 radCarousel1.DataSource = topRows;
}
```

In the **NewCarouselItemCreating** you can keep the code minimal and just create the element that you want in the carousel. In the **ItemDataBound** you can get the element back through the arguments e.DataBoundItem property and the actual DataRow used to populate the element by way of e.DataItem.



Gotcha! If there are JavaScript errors in the page you're loading it may be hard to determine if there's a bug in your application vs. the web page that's being loaded. Even though there is a WebBrowser ScriptErrorsSuppressed property, it will not completely silence these errors. Be aware that you will still need to go to the Internet Explorer browser options and turn off debugging if there are JavaScript errors in the page.

[VB] Handling the NewCarouselItemCreating and ItemDataBound Events

```
Private Sub radCarousel1_NewCarouselItemCreating(sender As Object, e As
NewCarouselItemCreatingEventArgs)
 e.NewCarouselItem = New RadButtonElement()
End Sub
Private Sub radCarousel1_ItemDataBound(sender As Object, e As ItemDataBoundEventArgs)
 If TypeOf e.DataBoundItem Is RadButtonElement Then
  ' get the button that was created in NewCarouselItemCreating
  Dim button As RadButtonElement = (TryCast(e.DataBoundItem, RadButtonElement))
  ' retrieve the DataRow for this item
  Dim row As DataRow = TryCast(e.DataItem, DataRow)
  Dim url As String = row("link").ToString()
  Dim title As String = row("title").ToString()
  ' configure the button element
  button.Text = title
  button.DisplayStyle = DisplayStyle.Image
  button.AngleTransform = 15
  button.Click += New EventHandler(button_Click)
  ' create the webbrowser component, size it to pick up
  ' as much of the page as possible.
  ' set the url and attach the button element as the tag for
  ' use in the DocumentCompleted event handler.
  Dim browser As New WebBrowser()
  browser.Width = 800
  browser.Height = 600
  browser.Url = New Uri(url)
  browser.ScrollBarsEnabled = False
  browser.Tag = button
  browser.DocumentCompleted += New WebBrowserDocumentCompletedEventHandler
(browser_DocumentCompleted)
 End If
End Sub
Private Sub browser_DocumentCompleted(sender As Object, e As
WebBrowserDocumentCompletedEventArgs)

 ' browser that just completed rendering

Dim browser As WebBrowser = TryCast(sender, WebBrowser)

' the element we want to add an image to
```

```
Dim element As RadButtonElement = TryCast(browser.Tag, RadButtonElement)

' temporary bitmap to draw to

Dim bitmap As New Bitmap(browser.Bounds.Width, browser.Bounds.Height)

' draw the browser image to the bitmap

browser.DrawToBitmap(bitmap, browser.Bounds)

' scrunch the image down and place it in the element image

element.Image = bitmap.GetThumbnailImage(150, 150, Nothing, New IntPtr())

End Sub

Private Sub button_Click(sender As Object, e As EventArgs)

 RadMessageBox.Show(TryCast(sender, RadButtonElement).Text)

End Sub
```

[C#] Handling the NewCarouselItemCreating and ItemDataBound Events

```
private void radCarousel1_NewCarouselItemCreating(object sender,
NewCarouselItemCreatingEventArgs e)
{
 e.NewCarouselItem = new RadButtonElement();
}
private void radCarousel1_ItemDataBound(object sender, ItemDataBoundEventArgs e)
{
 if (e.DataBoundItem is RadButtonElement)
 {
   // get the button that was created in NewCarouselItemCreating
   RadButtonElement button = (e.DataBoundItem as RadButtonElement);
   // retrieve the DataRow for this item
   DataRow row = e.DataItem as DataRow;
   string url = row["link"].ToString();
   string title = row["title"].ToString();
   // configure the button element
   button.Text = title;
   button.DisplayStyle = DisplayStyle.Image;
   button.AngleTransform = 15;
   button.Click += new EventHandler(button_Click);
   // create the webbrowser component, size it to pick up
   // as much of the page as possible.
   // set the url and attach the button element as the tag for
   // use in the DocumentCompleted event handler.
   WebBrowser browser = new WebBrowser();
   browser.Width = 800;
   browser.Height = 600;
   browser.Url = new Uri(url);
   browser.ScrollBarsEnabled = false;
   browser.Tag = button;
   browser.DocumentCompleted +=
     new WebBrowserDocumentCompletedEventHandler(browser_DocumentCompleted);
```

```
 }
}
void browser_DocumentCompleted(object sender, WebBrowserDocumentCompletedEventArgs e)
{
// browser that just completed rendering
WebBrowser browser = sender as WebBrowser;
// the element we want to add an image to
RadButtonElement element = browser.Tag as RadButtonElement;
// temporary bitmap to draw to
Bitmap bitmap = new Bitmap(browser.Bounds.Width, browser.Bounds.Height);
// draw the browser image to the bitmap
browser.DrawToBitmap(bitmap, browser.Bounds);
// scrunch the image down and place it in the element image
element.Image = bitmap.GetThumbnailImage(150, 150, null, new IntPtr());
}
void button_Click(object sender, EventArgs e)
{
RadMessageBox.Show((sender as RadButtonElement).Text);
}
```



Other Useful RadCarousel Properties

Buttons

The buttons that trigger animation can be aligned to the Bottom, Left, Right or Top using the **ButtonPositions** property. To access the individual buttons and their visual sub-properties, use **ButtonNext** and **ButtonPrevious**.

Carousel Path

You can cause the carousel to animate continually as a slideshow by setting the **EnableLooping** property to True. **AutoLoopDirection** can be **Forward** or **Backward.** When the user moves the mouse over the carousel, the carousel should pause briefly; **AutoLoopPauseCondition** controls pausing and can be set to **None**, **OnMouseOverCarousel** (default) or **OnMouseOverItem**. **AutoLoopPauseInterval** is the number of seconds before the carousel resumes looping.

When **EnableRelativePath** is true, RadCarousel will recalculate and scale the path proportionally when the form is resized.

Item Interaction

ItemClickDefaultAction controls if the carousel animates when the user clicks an item. When set to **SelectItem**, clicking on a carousel item causes the item carousel to animate. When set to **None**, the items remain static in the carousel.

ItemReflectionPercentage is the percentage amount of reflection displayed, relative to the size of the item.

SelectedIndex is similar to the same property in a list box by indicating the ordinal position of the currently selected item.

16.6 Summary

In this chapter you learned some stylish and unique ways of organizing form space using RadDock to handle dockable windows and tabbed documents, RadCarousel to animate a series of clickable images and RadRotator to display constantly changing personalized content.

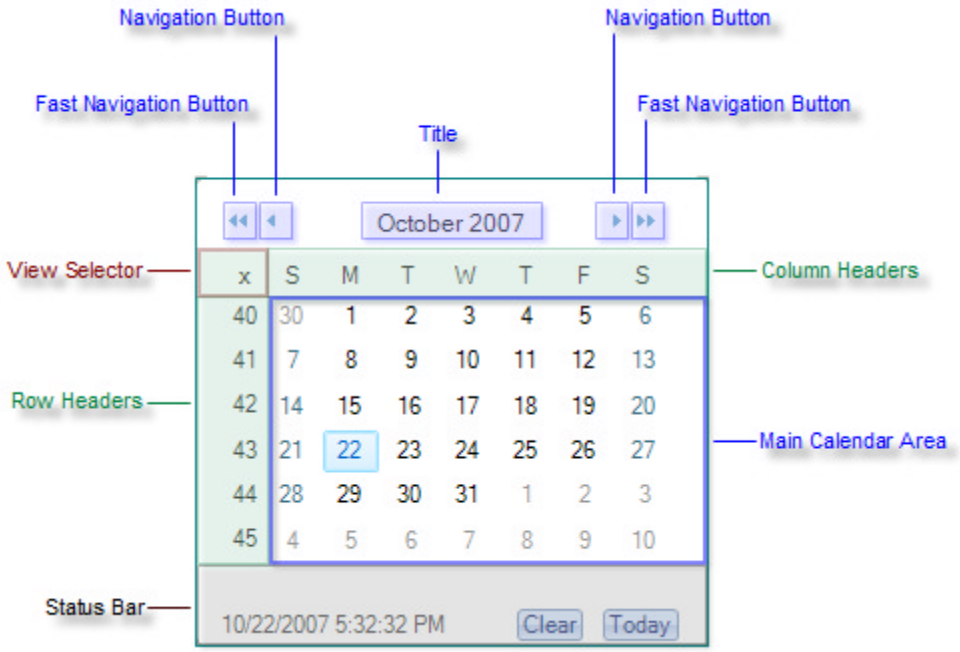
17 Calendar

17.1 Objectives

- Become familiar with RadCalendar extensive customization features.
- Work with selected days.
- Work with the "Special Days" collection to mark specific days or repeating events.

17.2 Introduction

RadCalendar is a lightweight yet highly advanced date input control for Windows Forms. Zoom, select multiple dates, view several months at once, use it as a date picker, or even add a Hebrew calendar to your forms.

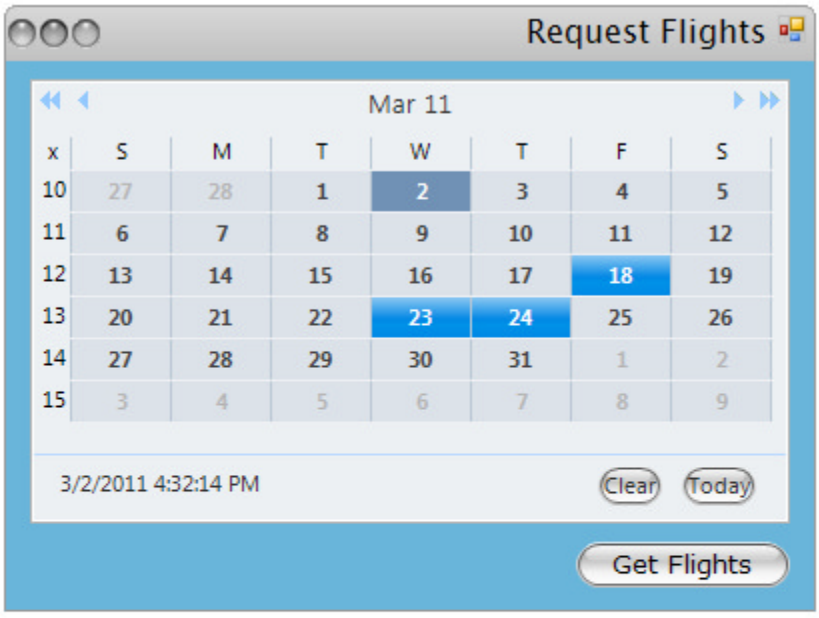


Just about everything on the calendar can be customized to suit your purposes.

- **Title** displays the selected date range and navigation buttons. The appearance of both title and the navigation buttons can be customized.
- **Navigation Buttons** allow you to move to the previous/next month or jump X number of months at a time.
- The **Main Calendar Area** renders a single or multiple month "calendar view". This area includes all of the individual days and the row & column selectors.
- **Row and Column Headers** let your user select a whole row or column of dates at a time by clicking on the respective day or week button.
- Use the **View Selector** button to select all cells in the current month. If multiple months are being displayed at one time, there will be one view selector per month.
- The **StatusBar** footer area below the Main Calendar Area can be used for visual customization or for displaying additional information about the selected item.
- When the user clicks the Title, the **Fast Navigation Popup** displays a window with the months and years surrounding the selected date (not visible on the screenshot)

17.3 Getting Started

RadCalendar allows extensive modifications of appearance and behavior. In this "Getting Started" walk through you will configure the calendar to display row headers, a footer and to allow multiple selections. A button click will collect the selected dates and display them.





Preparing the Project



You can find the complete source for this project at:

\Calendar\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. Add the **AquaTheme** from the ToolBox to the form.
6. In the Properties window, set the form **Size** property to "400,300", **Text** property to "Request Flights" and **ThemeName** to **Aqua,** and **BackColor** to 105,181,218.
7. Change the new RadForm1 to be the startup form.

RadCalendar Setup

1. From the Toolbox, add a **RadCalendar** to the form. Size the calendar to allow a little room on the top and sides and enough room for a button to appear below it. Set RadCalendar Properties:
   - **Anchor** = "Top, Bottom, Left Right"
   - **ShowRowHeader** = True
   - **ShowViewSelector** = True
   - **AllowMultipleSelect** = True
   - **AllowColumnHeaderSelectors** = True
   - **AllowRowHeaderSelectors** = True
   - **AllowViewSelector** = True
   - **ShowFooter** = True

   Add a RadButton below the calendar, name it "btnGetFlights" and set the Text to "Get Flights".

2. Double-click the button to create a **Click** event handler. Add the code below that iterates the SelectedDates collection of DateTime objects and builds a string to display in a RadMessageBox.

   [VB] Handling the Click Event

```
Private Sub btnGetFlights_Click(sender As Object, e As EventArgs)
 Dim builder As New StringBuilder()
 For Each dateTime As DateTime In radCalendar1.SelectedDates
  builder.Append(dateTime.ToLongDateString())
  builder.Append(Environment.NewLine)
 Next
RadMessageBox.SetThemeName("Aqua")
RadMessageBox.Show(builder.ToString(), "Look for flights on these days?",
MessageBoxButtons.YesNo, RadMessageIcon.Question)
End Sub
```

   [C#] Handling the Click Event

```
private void btnGetFlights_Click(object sender, EventArgs e)
{
 StringBuilder builder = new StringBuilder();
 foreach (DateTime dateTime in radCalendar1.SelectedDates)
 {
   builder.Append(dateTime.ToLongDateString());
   builder.Append(Environment.NewLine);
 }
RadMessageBox.SetThemeName("Aqua");
 RadMessageBox.Show(builder.ToString(), "Look for flights on these days?",
   MessageBoxButtons.YesNo, RadMessageIcon.Question);
}
```

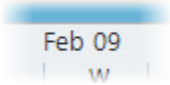
3. Run the application and verify that you can select multiple days, that you can select the entire month using the view selector and that you can select rows and columns using the corresponding selectors.

   Try selecting the dates in a random order. Notice that the DateTime objects in SelectedDates are in the order they were selected.

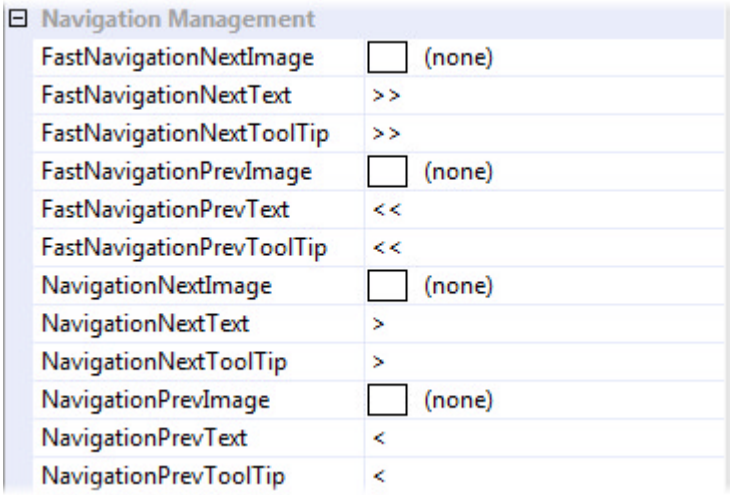
17.4 Using the Design Time Interface

Title and Navigation Properties

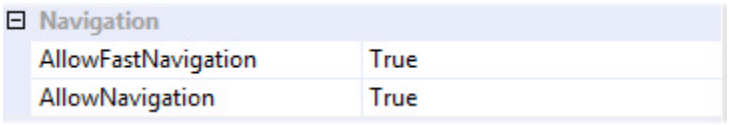
TitleFormat is a format string that controls the title caption at the top of the calendar. The property should contain either a format specifier character or a custom format pattern. For more information, search the MSDN for "System.Globalization.DateTimeFormatInfo". By default this property uses formatting string of 'MMMM yyyy'. For example, if you change TitleFormat to "MMM yy" you will see a three character month and two digit year. **TitleAlign** moves the title caption Left, Right or Center.



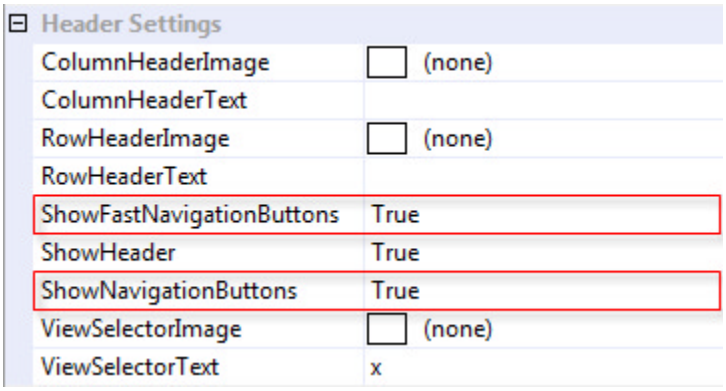
Image, Text and ToolTip properties of navigation buttons found to the left and right of the title can be found in the **Navigation Management** group of properties.



Toggle the ability to use the navigation and fast navigation using the Navigation properties **AllowNavigation** and **AllowFastNavigation.**



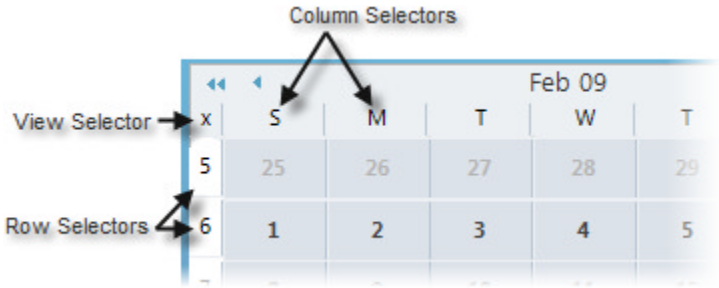
In a separate group of properties called Header Settings, you control if the navigation buttons are even shown using **ShowFastnavigationButtons** and **ShowFastnavigationButtons**.



By default the navigation buttons move you forward and back one month at a time and the fast navigation buttons move you 3 months at a time. The FastNavigationStep step is the number of months that the fast navigation buttons move you at a time.

Row, Column and View

By default the column headers display as the days of the week just below the title area. You can also display the row headers that display the week numbers of the year along the left side of the calendar day cells by toggling **ShowRowheaders**. The days of the week display if **ShowColumnHeaders** is true (the default). Set **ShowViewSelector** True to display the little "x" selector at the upper left of the calendar day cells.

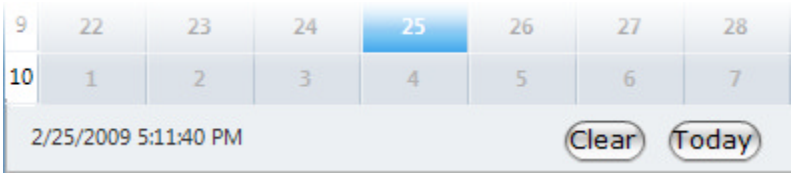


To allow an entire row, column or all day cells for the view to be selected at one time, toggle the **AllowRowHeaderSelectors**, **AllowColumnHeaderSelectors** and **AllowViewSelector** behavior properties.

> If you want to set **AllowRowHeaderSelectors, AllowColumnHeaderSelectors** and **AllowViewSelector** to True, you must also set **AllowMultipleSelect** to be True.

Footer

The footer is located below the month day cells and is visible if the **ShowFooter** property is True. The footer shows a date and time that automatically updates, a Clear button that clears all selections in the current view and a Today button that selects the current date. Access footer buttons through the **ClearButton** and **TodayButton** objects, e.g. ClearButton.Text.

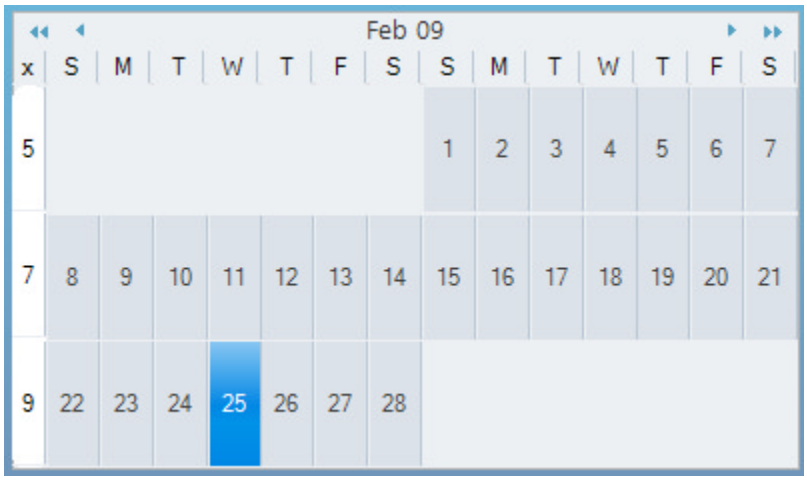


FocusedDate

The **FocusedDate** property determines the current view. For example, if you want to show the month of January 2000 when you first display the calendar, then set the FocusedDate to 1/1/2000.

Day Cell Layout

You can configure non-standard arrangements of days using the integer **Columns** and **Rows** properties to be other than the typical 7 days 6 week rows configuration. Set the **ShowOtherMonthDays** property False to hide days that aren't part of the current month view.

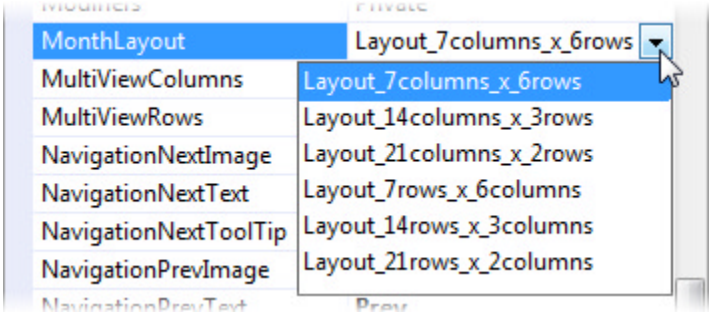


Orientation

Orientation by default is **Horizontal**, with the days progressing from left to right or right to left (see Globalization for more information). Orientation can also be **Vertical** to list the days from top to bottom or bottom to top.

Month Layout

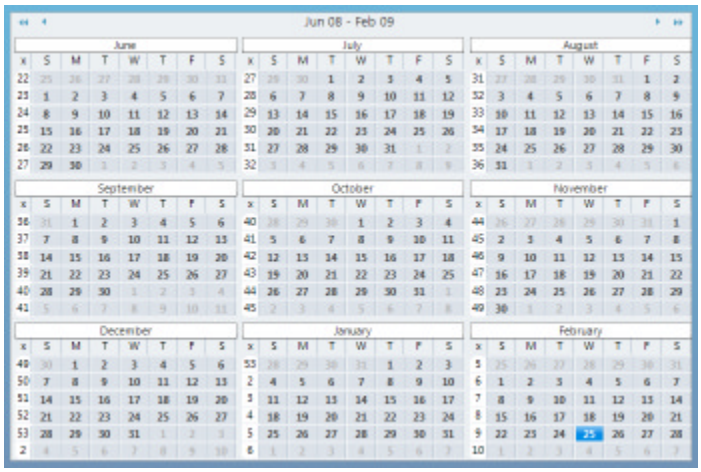
Columns and rows and be displayed in a number of **MonthLayout** configurations.



Multiple Month Views

RadCalendar can be configured to show more than one month in the calendar area. Set the **AllowMultipleView** property to true, then set integer properties **MultiViewColumns** and **MultiViewRows** to the number of calendar month views you wish to have displayed.

In multi-view mode, the position of the month view that contains the focused date is controlled by the **CurrentViewRow** and **CurrentViewColumn** properties. In this screenshot MultiViewColumns and MultiViewRows are both set to "3". CurrentViewRow and CurrentViewColumn are both set to "2", which puts the focus on the lower right hand month view.



Limiting Date Selection

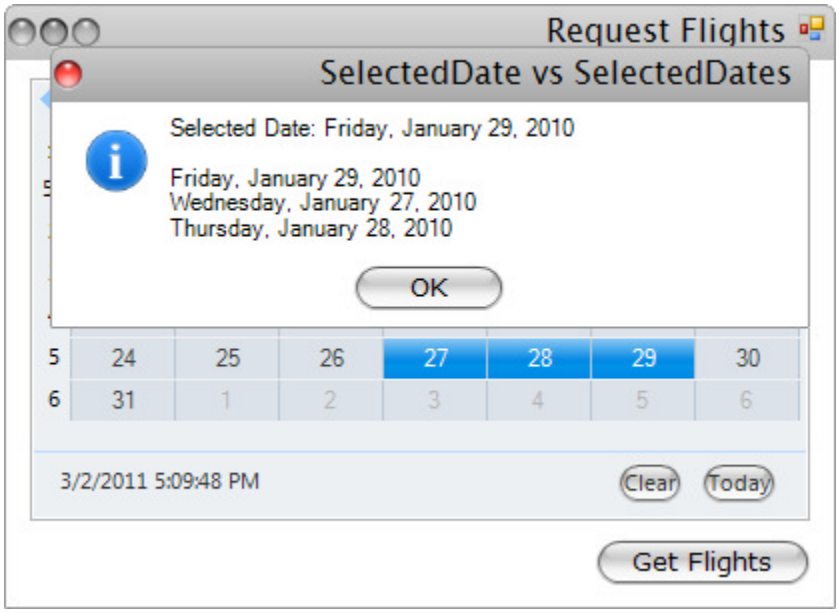
Use the RangeMinDate and RangeMaxDate properties to limit the upper and lower bounds of selectable dates.

17.5 Programming the Calendar

Selected Dates

You may have noticed both **SelectedDate** (DateTime) and **SelectedDates** (DateTime[]) RadCalendar properties. They are closely related. When the **AllowMultipleSelect** property is set to false, a mode that allows only a single date selection, SelectedDate and SelectedDates[0] have the same value and SelectedDates.Count equals 1. When AllowMultipleSelect is set to true, SelectedDate will still SelectedDates[0] have the same value, but the SelectedDates.Count may have more than one element.

Here's an example where dates have been selected in the order "29", "28", "27" and are stored in SelectedDates in that order. Also notice that SelectedDate and SelectedDates[0] are the same.



[VB] Comparing the SelectedDate and SelectedDates Properties

```
Dim builder As New StringBuilder()
builder.Append("Selected Date: ")
builder.Append(radCalendar1.SelectedDate.ToLongDateString())
```

```
builder.Append(Environment.NewLine)
builder.Append(Environment.NewLine)
For Each dateTime As DateTime In radCalendar1.SelectedDates
 builder.Append(dateTime.ToLongDateString())
 builder.Append(Environment.NewLine)
Next
RadMessageBox.Show(builder.ToString(), "SelectedDate vs SelectedDates",
MessageBoxButtons.OK, RadMessageIcon.Info)
```

[C#] Comparing the SelectedDate and SelectedDates Properties

```
StringBuilder builder = new StringBuilder();
builder.Append("Selected Date: ");
builder.Append(radCalendar1.SelectedDate.ToLongDateString());
builder.Append(Environment.NewLine);
builder.Append(Environment.NewLine);
foreach (DateTime dateTime in radCalendar1.SelectedDates)
{
 builder.Append(dateTime.ToLongDateString());
 builder.Append(Environment.NewLine);
}
RadMessageBox.Show(builder.ToString(), "SelectedDate vs SelectedDates",
 MessageBoxButtons.OK, RadMessageIcon.Info);
```



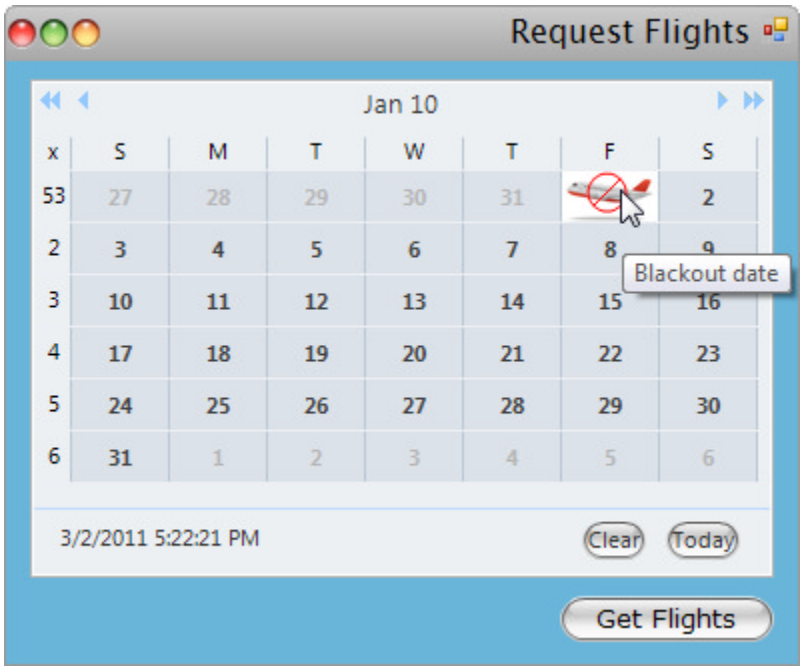
You can find the complete source for this project at:

\<VB|CS>\Programming

Using the Special Days Collection

To customize RadCalendar for your business or product you will want to customize particular days to show holidays, appointments or events. Populate the **SpecialDays** collection to highlight these days. SpecialDays is a collection of **RadCalendarDay**, not simply DateTime. Each RadCalendarDay has properties for **Date**, **Image**, if the day is **Recurring** and a **TemplateItem** property that can be used as a container for other controls. You can also query the **IsToday** and **IsWeekend** boolean properties.

Here's an example that adds a "Blackout date" as a special day that shows a "no fly" icon for New Years day.



You can pass a DateTime to the RadCalendarDay in one of the constructor overloads to automatically assign the Date. The **FocusedDate** property assignment moves focus to the month of January.

[VB] Adding a Special Day

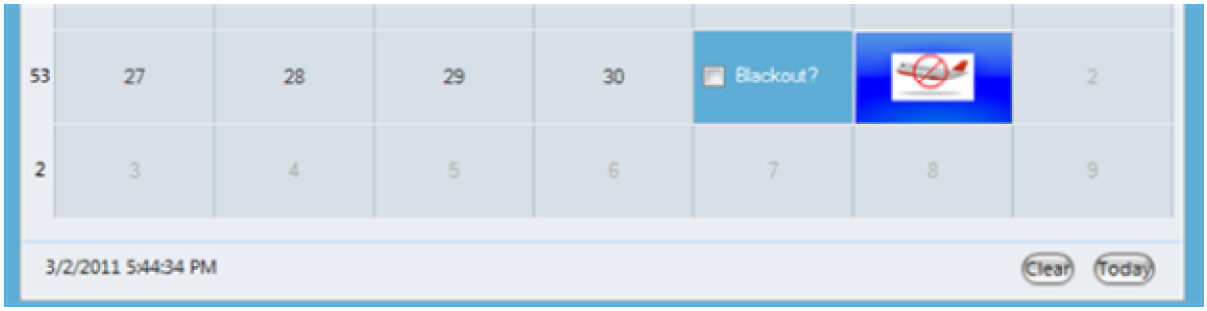
```
' assign New Years as a special day
Dim day As New RadCalendarDay(New DateTime(2010, 1, 1))
radCalendar1.SpecialDays.Add(day)
radCalendar1.SpecialDays(0).Image = imageList1.Images("blackout.ico")
radCalendar1.FocusedDate = day.[Date]
```

[C#] Adding a Special Day

```
// assign New Years as a special day
RadCalendarDay day = new RadCalendarDay(new DateTime(2010, 1, 1));
radCalendar1.SpecialDays.Add(day);
radCalendar1.SpecialDays[0].Image = imageList1.Images["blackout.ico"];
radCalendar1.FocusedDate = day.Date;
```

Templates

Assign a RadHostItem to the **ItemTemplate** if you want to include any Controls including standard Windows forms controls and RadControls. The example below creates a special day for New Years Eve day and assigns a standard CheckBox to the RadHostItem, then assigns the RadHostItem to the ItemTemplate property.



[VB] Assigning a TemplateItem

```
Dim checkbox As New CheckBox()
checkbox.Text = "Blackout?"
checkbox.Padding = New Padding(5)
Dim hostItem As New RadHostItem(checkbox)
Dim newYearsEveDay As New RadCalendarDay(New DateTime(2009, 12, 31))
radCalendar1.SpecialDays.Add(newYearsEveDay)
newYearsEveDay.TemplateItem = hostItem
radCalendar1.FocusedDate = newYearsEveDay.[Date]
```

[C#] Assigning a TemplateItem

```
CheckBox checkbox = new CheckBox();
checkbox.Text = "Blackout?";
checkbox.Padding = new Padding(5);
RadHostItem hostItem = new RadHostItem(checkbox);
RadCalendarDay newYearsEveDay = new RadCalendarDay(new DateTime(2009, 12, 31));
radCalendar1.SpecialDays.Add(newYearsEveDay);
newYearsEveDay.TemplateItem = hostItem;
radCalendar1.FocusedDate = newYearsEveDay.Date;
```



You can find the complete source for this project at:

\<VB|CS>\Programming

Recurring Days

A RadCalendarDay object can be configured as a repeating event. Set the **Recurring** property to:

- **DayInMonth:** Only the day part of the date is taken into account. The event repeats every month on the same day.
- **DayAndMonth:** The month and the day part of the date are taken into account. The event repeats in a specific month on the same day.
- **Week:** The week day is taken into account. The event repeats on a specific day of the week.
- **WeekAndMonth:** The week day and the month are taken into account. The event repeats in a specific week day in a specific month.
- **Today:** Gives the ability to control the visual appearance of today's day.
- **None:** Default value - means that the day in question is a single point event, no recurrence.

[VB] Adding Recurring Dates

```
Dim fridays As New RadCalendarDay(New DateTime(2010, 1, 8))
fridays.Recurring = RecurringEvents.Week
radCalendar1.SpecialDays.Add(fridays)
```

[C#] Adding Recurring Dates

```
RadCalendarDay fridays = new RadCalendarDay(new DateTime(2010, 1, 8));
fridays.Recurring = RecurringEvents.Week;
radCalendar1.SpecialDays.Add(fridays);
```

You can find the complete source for this project at:

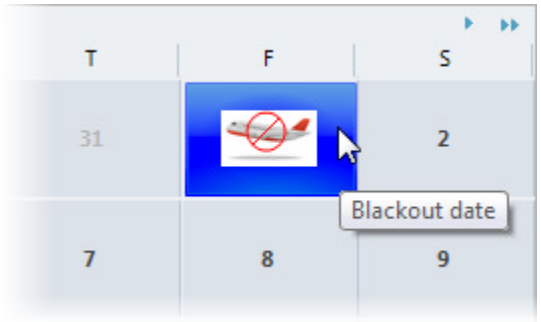
\<VB|CS>\Programming

Calendar Events

ElementRender

You can dial-in the appearance of each individual calendar cell element using the ElementRender event. ElementRender is fired after the generation of every calendar cell object and just before it gets rendered. It is the last place where changes to the already constructed calendar cells can be made.

ElementRender arguments include a **Day** (RadCalendarDay) and a visual **Element** that is about to be rendered. The example below determines if the Day is from the SpecialDays collection, then uses the properties of Element to paint a glossy gradient on the background, makes the ForeColor transparent (to hide the day number) and adds a tooltip.



[VB] Handling the ElementRender Event

```
Private Sub radCalendar1_ElementRender(sender As Object, e As RenderElementEventArgs)
 Dim isSpecialDay As Boolean = radCalendar1.SpecialDays.Find(e.Day.[Date]).Length > 0
 If isSpecialDay Then
  e.Element.BackColor = Color.FromArgb(105, 181, 218)
  e.Element.BackColor2 = Color.FromArgb(112, 145, 181)
  e.Element.BackColor3 = Color.Blue
  e.Element.NumberOfColors = 3
  e.Element.GradientStyle = GradientStyles.OfficeGlassRect
  e.Element.GradientPercentage = 80
  e.Element.ForeColor = Color.Transparent
  e.Element.ToolTipText = "Blackout date"
 End If
End Sub
```

[C#] Handling the ElementRender Event

```
private void radCalendar1_ElementRender(object sender, RenderElementEventArgs e)
{
 bool isSpecialDay = radCalendar1.SpecialDays.Find(e.Day.Date).Length > 0;
 if (isSpecialDay)
 {
   e.Element.BackColor = Color.FromArgb(105, 181, 218);
   e.Element.BackColor2 = Color.FromArgb(112, 145, 181);
   e.Element.BackColor3 = Color.Blue;
   e.Element.NumberOfColors = 3;
   e.Element.GradientStyle = GradientStyles.OfficeGlassRect;
   e.Element.GradientPercentage = 80;
   e.Element.ForeColor = Color.Transparent;
```

```
    e.Element.ToolTipText = "Blackout date";
  }
}
```



If you have cell customzations that need to show when the calendar first loads, call the calendar **InvalidateCalendar()** method to explicitly rebuild. This call is appropriate whenever you have batch updates to calendar properties that occur outside the control.

SelectionChanging and SelectionChanged

SelectionChanging gives you the opportunity to prevent the user from selecting a day in the calendar. The event arguments Dates property is an array of DateTime showing the date selections up to that point. SelectionChanged fires after the selection has already occurred and doesn't offer the opportunity to cancel.

[VB] Handling the SelectionChanging Event

```
Private Sub radCalendar1_SelectionChanging(sender As Object, e As SelectionEventArgs)
 If e.Dates.Count = 3 Then
  e.Cancel = True
  RadMessageBox.Show("Select no more than three travel dates")
 End If
End Sub
```

[C#] Handling the SelectionChanging Event

```
private void radCalendar1_SelectionChanging(object sender, SelectionEventArgs e)
{
 if (e.Dates.Count == 3)
 {
   e.Cancel = true;
   RadMessageBox.Show("Select no more than three travel dates");
 }
}
```

ViewChanging and ViewChanged

ViewChanging fires when navigating to a different date range.

[VB] Handling the ViewChanging Event

```
Private Sub radCalendar1_ViewChanging(sender As Object, e As ViewChangingEventArgs)
 If e.View.ViewStartDate > radCalendar1.FocusedDate.AddDays(90) Then
  e.Cancel = True
  RadMessageBox.Show("You may not reserve flights more than 90 days in advance")
 End If
End Sub
```

[C#] Handling the ViewChanging Event

```
private void radCalendar1_ViewChanging(object sender, ViewChangingEventArgs e)
{
 if (e.View.ViewStartDate > radCalendar1.FocusedDate.AddDays(90))
 {
   e.Cancel = true;
   RadMessageBox.Show("You may not reserve flights more than 90 days in advance");
 }
}
```

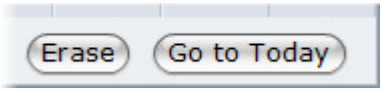


You can find the complete source for this project at:

\<VB|CS>\Programming

Visual Customization

Most aspects of the calendar UI are surfaced directly through properties. For example, the footer Clear and Today buttons can be changed at design time or in code simply by using the corresponding properties TodayButton and ClearButton:



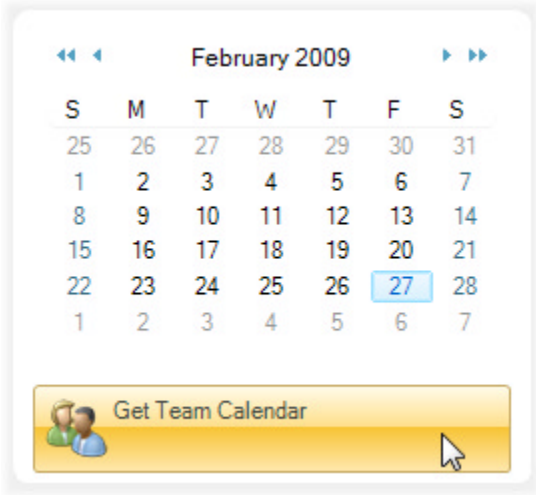
[VB] Setting Footer Button Properties

```
radCalendar1.TodayButton.Text = " Go to Today "
radCalendar1.ClearButton.Text = " Erase "
```

[C#] Setting Footer Button Properties

```
radCalendar1.TodayButton.Text = " Go to Today ";
radCalendar1.ClearButton.Text = " Erase ";
```

What if you want to replace the footer with some custom information or elements? We can shut off both buttons and the updating clock information on the left of the footer and add a custom element in its place. Here we set the Visibility property of all three elements to Collapsed so they don't take up any space, then add a new RadButtonElement that expands to fill the entire footer.



[VB] Adding a Custom Element

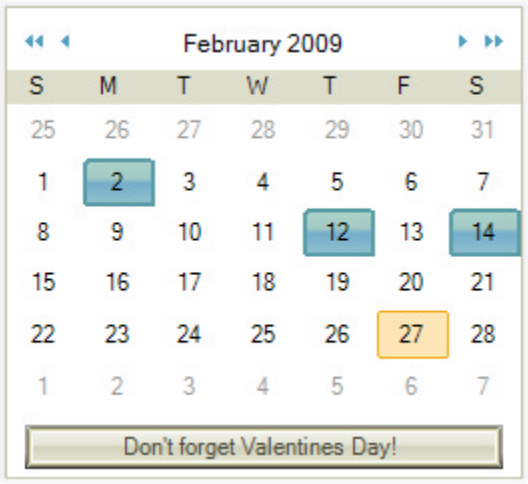
```
Private Sub Form1_Load(sender As Object, e As System.EventArgs)
 radCalendar1.ShowFooter = True
 radCalendar1.ClearButton.Visibility = ElementVisibility.Collapsed
 radCalendar1.TodayButton.Visibility = ElementVisibility.Collapsed
 ' get the footer layout panel
 Dim layoutPanel As DockLayoutPanel = TryCast(radCalendar1.CalendarElement.Children
(0).Children(0).Children(0), DockLayoutPanel)
 ' get the updating date and time label and hide that too
 layoutPanel.Children(2).Visibility = ElementVisibility.Collapsed
 ' create a new custom element and add it
 Dim button As New RadButtonElement()
```

```
 button.Image = Properties.Resources.users.ToBitmap()
 button.Text = "Get Team Calendar"
 button.TextImageRelation = TextImageRelation.ImageBeforeText
 button.Padding = New Padding(5, 5, 0, 5)
 layoutPanel.Children.Add(button)
End Sub
```

[C#] Adding a Custom Element

```
private void Form1_Load(object sender, System.EventArgs e)
{
 radCalendar1.ShowFooter = true;
 radCalendar1.ClearButton.Visibility = ElementVisibility.Collapsed;
 radCalendar1.TodayButton.Visibility = ElementVisibility.Collapsed;
 // get the footer layout panel
 DockLayoutPanel layoutPanel =
   radCalendar1.CalendarElement.Children[0].Children[0].Children[0] as DockLayoutPanel;
 // get the updating date and time label and hide that too
 layoutPanel.Children[2].Visibility = ElementVisibility.Collapsed;
 // create a new custom element and add it
 RadButtonElement button = new RadButtonElement();
 button.Image = Properties.Resources.users.ToBitmap();
 button.Text = "Get Team Calendar";
 button.TextImageRelation = TextImageRelation.ImageBeforeText;
 button.Padding = new Padding(5, 5, 0, 5);
 layoutPanel.Children.Add(button);
}
```

Perhaps you can find more creative ways to combine controls like this Rotator located in the footer of the calendar that announces Special Days. The calendar below has special days for February 2 (Groundhog Day!), Feb 12 (Abe Lincolns Birthday) and Feb 14 (Don't forget the flowers...). The code to is much like the previous example except that a RadRotatorElement is added instead of a RadButtonElement, and we iterate the calendar SpecialDays collection, adding RadButtonElements to the rotation for each day.



[VB] Adding a Rotator Element to the Footer

```
Private Sub Form1_Load(sender As Object, e As System.EventArgs)
 radCalendar1.ShowFooter = True
 radCalendar1.ClearButton.Visibility = ElementVisibility.Collapsed
 radCalendar1.TodayButton.Visibility = ElementVisibility.Collapsed
 ' get the footer layout panel
 Dim layoutPanel As DockLayoutPanel = TryCast(radCalendar1.CalendarElement.Children
```

```
(0).Children(0).Children(0), DockLayoutPanel)
 ' get the updating date and time label and hide that too
 layoutPanel.Children(2).Visibility = ElementVisibility.Collapsed
 Dim rotator As New RadRotatorElement()
 For Each day As RadCalendarDay In radCalendar1.SpecialDays
  Dim button As New RadButtonElement()
  button.Text = day.ToolTip
  button.DisplayStyle = DisplayStyle.Text
  rotator.Items.Add(button)
 Next
 rotator.Interval = 4000
 rotator.Running = True
 layoutPanel.Children.Add(rotator)
End Sub
```

[C#] Adding a Rotator Element to the Footer

```
private void Form1_Load(object sender, System.EventArgs e)
{
 radCalendar1.ShowFooter = true;
 radCalendar1.ClearButton.Visibility = ElementVisibility.Collapsed;
 radCalendar1.TodayButton.Visibility = ElementVisibility.Collapsed;
 // get the footer layout panel
 DockLayoutPanel layoutPanel =
   radCalendar1.CalendarElement.Children[0].Children[0].Children[0] as DockLayoutPanel;
 // get the updating date and time label and hide that too
 layoutPanel.Children[2].Visibility = ElementVisibility.Collapsed;
 RadRotatorElement rotator = new RadRotatorElement();
 foreach (RadCalendarDay day in radCalendar1.SpecialDays)
 {
   RadButtonElement button = new RadButtonElement();
   button.Text = day.ToolTip;
   button.DisplayStyle = DisplayStyle.Text;
   rotator.Items.Add(button);
 }
 rotator.Interval = 4000;
 rotator.Running = true;
 layoutPanel.Children.Add(rotator);
}
```



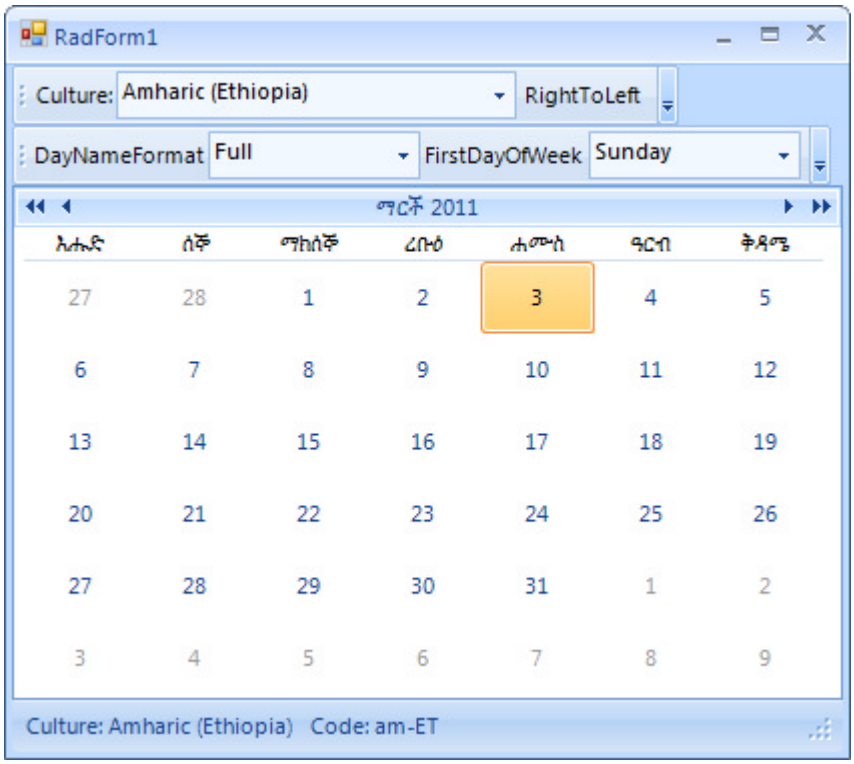
You can find the complete source for this project at:

\<VB|CS>\Footer

Globalization

RadCalendar provides built in internationalization support to build world-ready applications.

RadCalendar supports the **System.Globalization** namespace. System.Globalization consists of classes (CompareInfo, CultureInfo, RegionInfo, etc.) that contain culture-related information, such as the language, country/region, calendars in use, format patterns for dates, currency, and numbers, and the sort order for strings.



You can find the complete source for this project at:

\Calendar\<VB|CS>\Globalization

Setting the **Culture** property automatically re-labels in the language and format of the culture. Culture is a System.Globalization namespace **CultureInfo** object that can be set to known culture codes. In the screenshot above the calendar is set to Bengali, a culture code of "bn-BD". You can do the same by creating a new CultureInfo object and passing any available culture code in the constructor or use the static CultureInfo.**GetCultureInfo()** to return a CultureInfo.

[VB] Assigning the Culture

```
radCalendar1.Culture = New CultureInfo("bn-BD")
```

[C#] Assigning the Culture

```
radCalendar1.Culture = new CultureInfo("bn-BD");
```

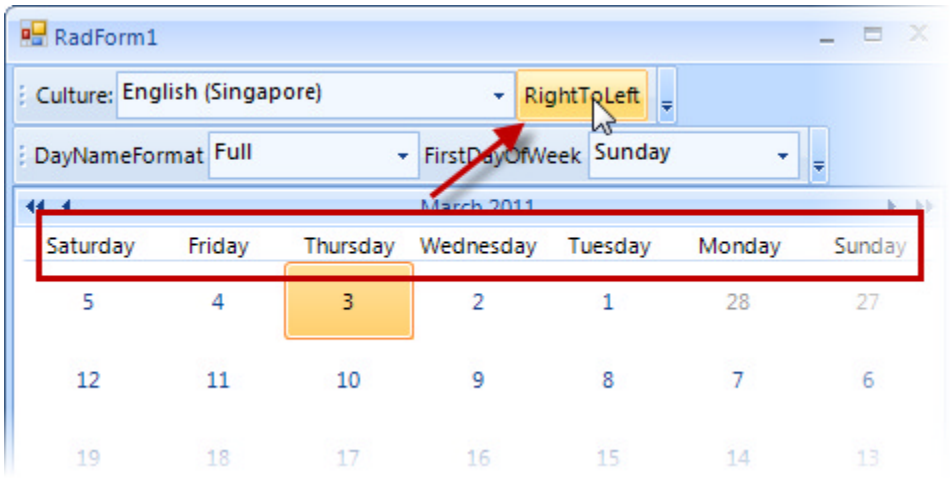
For languages that run right-to-left, simply assign the calendar **RightToLeft** property a value of **Yes**, **No**, or **Inherit** (inherits the direction of the parent control).

[VB] Assigning the RightToLeft Property

```
radCalendar1.RightToLeft = If(cbRightToLeft.Checked, RightToLeft.Yes, RightToLeft.No)
```

[C#] Assigning the RightToLeft Property

```
radCalendar1.RightToLeft = cbRightToLeft.Checked ? RightToLeft.Yes : RightToLeft.No;
```



Assign a **FirstDayOfWeek** enumeration member a value of **Monday..Sunday** or **Default** to be handled by the **System.Globalization.Calendar** object.

[VB] Assigning the FirstDayOfWeek

```
radCalendar1.FirstDayOfWeek = FirstDayOfWeek.Monday
```

[C#] Assigning the FirstDayOfWeek

```
radCalendar1.FirstDayOfWeek = FirstDayOfWeek.Monday;
```

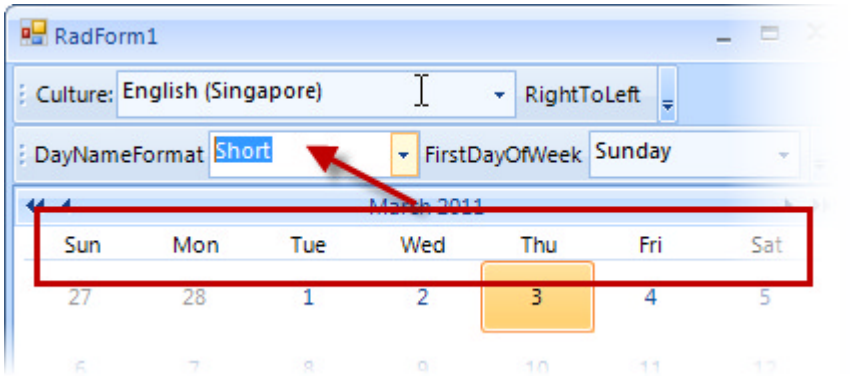
The format for the day column headings is handled by the **DayNameFormat** property and corresponding enumeration:

[VB] Assigning the DayNameFormat

```
radCalendar1.DayNameFormat = DayNameFormat.[Short]
```

[C#] Assigning the DayNameFormat

```
radCalendar1.DayNameFormat = DayNameFormat.Short;
```



The other "format" properties that relate to globalization take formatting strings:

- **TitleFormat**: The default value is "MMMM yyyy" shows the full month and year.
- **DayCellFormat**: The default value is "%d" prints the simple integer day of the month number in each cell.
- **CellToolTipFormat**: The default value is "dddd.MMMM dd, yyyy".

One other miscellaneous property is the **DateRangeSeparator** that appears between start and end months in a multi-view title. Thie property defaults to " - ".

For an overview on globalization in general, see the MSDN article "**Introduction to International**

Applications Based on the .NET Framework (http://msdn.microsoft.com/en-us/library/t18274tk (vs.80).aspx)".

17.6 Summary

This chapter focused on using the extensive customization features of RadCalendar to completely tailor the calendar header, calendar footer, number of displayed months, the number of rows and columns, date ranges, selected and focused dates, the first day of the week as well as the title and navigation areas of the calendar. You learned how to work with the selected days in the calendar. The chapter also explained how to work with the "special days" collection to mark specific days or repeating events.

18 Shortcuts

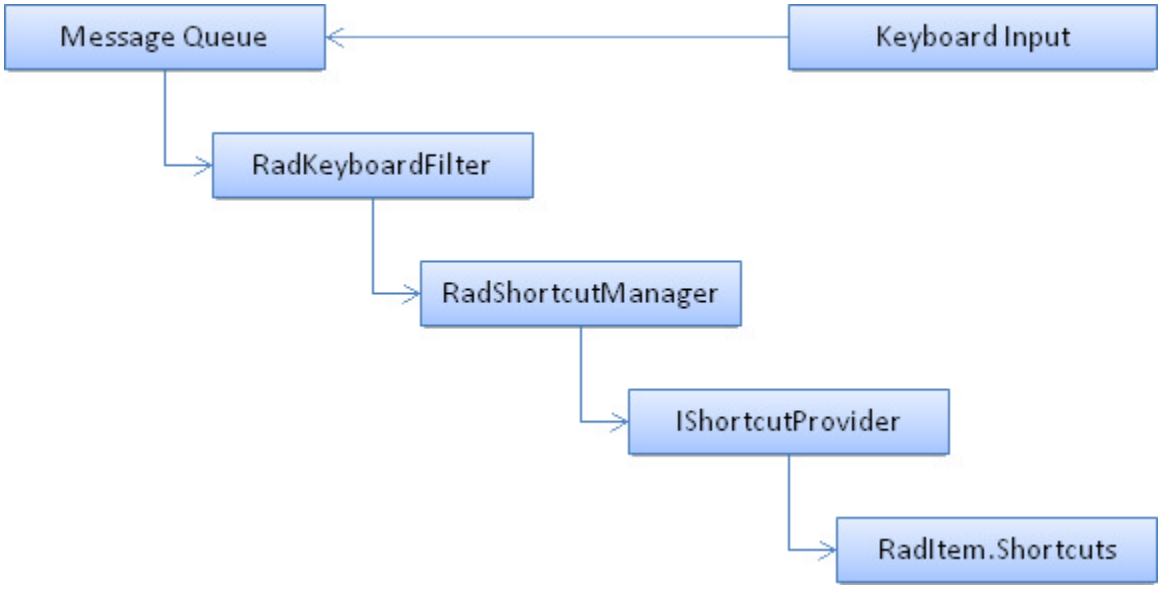
18.1 Objectives

- Learn how to create shortcuts and global shortcuts
- Learn how to create shortcuts for RadControls
- Learn how to create shortcuts for menu items
- Explore the RadShortcut functionality

18.2 Introduction

Almost each application uses the so called “Shortcuts” – a keyboard combination that triggers a specific action. For a valid shortcut is considered any keyboard combination where a Modifier Key (CTRL, ALT and SHIFT or a combination of these) is down and other key(s) is pressed. This semantic is available out-of-the-box in our framework and allows you to seamlessly plug any valid keys combination as an action accelerator. Supported are also the so called multi-keys shortcuts where the Keys member of each shortcut may be more than one key – e.g. Ctrl + A, S is recognized by the framework. Shortcuts without modifier keys are also supported, but they should be used with caution, since they may be in conflict with other controls which intercept keyboard input.

The following diagram illustrates the object model and class hierarchy that lie behind each **RadShortcut**:



The entire mechanism is controlled by several interfaces and classes which provide very easy and intuitive way to register different shortcuts.

- **RadKeyboardFilter**: This class listens to all keyboard messages that come on the application’s message queue and delegates them to each IKeyboardListener registered.
- **RadShortcutManager**: A concrete implementation of IKeyboardListener interface which is plugged into the RadKeyboardFilter instance. Stores a collection of IShortcutProvider instances, recognizes a valid shortcut combination, provided by any of the registered IShortcutProvider and calls the IShortcutProvider.OnShortcut(RadShortcut) callback of the matched provider.
- **RadItem.Shortcuts**: Each RadItem implements the IShortcutProvider interface and allows number of shortcuts to be associated with it. When a keyboard combination matches any of the item’s shortcuts, the

item's Click event is raised.

18.3 Creating Shortcuts

Creating shortcuts for RadMenuItems

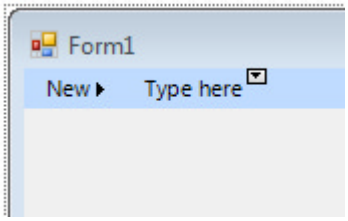


You can find the complete source for this project at:

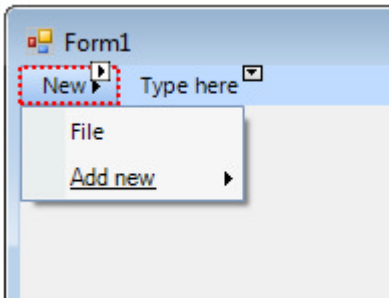
\Shortcuts\<VB|CS>\MenuItemShortcuts

The following application will demontrate how to assign shortcut to RadMenuItems:

1. Create a new Windows Forms application.
2. Drag RadMenu from the toolbox to the form.
3. Click the RadMenu that reads "Type here" and enter "New". This will automatically create a menu item object "radMenuItem1".



4. Click and the newly created item to open the dropdown menu and create a submenu item. By default, the text of the new item will be set to radMenuItem2. Change it to File from the Properties Window:



5. Double click both menu items to create their Click event handlers.
6. In both event handlers add code to show a message box when the item is clicked:

[VB.NET] Handle RadMenuItem click event

```
Private Sub RadMenuItem1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs
Handles RadMenuItem1.Click
RadMessageBox.Show("New")
End Sub
Private Sub RadMenuItem2_Click(ByVal sender As System.Object, ByVal e As System.EventArgs
Handles RadMenuItem2.Click
RadMessageBox.Show("File")
End Sub
```

[C#] Handle RadMenuItem click event

```
private void radMenuItem1_Click(object sender, EventArgs e)
{
RadMessageBox.Show("New");
```

```
}
private void radMenuItem2_Click(object sender, EventArgs e)
{
RadMessageBox.Show("File");
}
```

7. Now we just need to add the shortcuts to the desired items

[VB.NET] Adding shortcuts

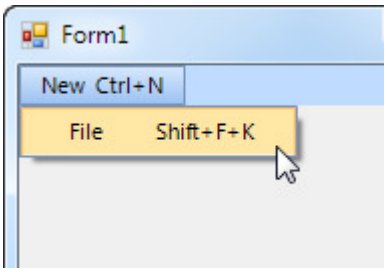
```
Me.RadMenuItem1.Shortcuts.Add(New RadShortcut(Keys.Control, Keys.N))
Me.RadMenuItem2.Shortcuts.Add(New RadShortcut(Keys.Shift, Keys.F, Keys.K))
```

[C#] Adding shortcuts

```
this.radMenuItem1.Shortcuts.Add(new RadShortcut(Keys.Control, Keys.N));
this.radMenuItem2.Shortcuts.Add(new RadShortcut(Keys.Shift, Keys.F, Keys.K)
```

> In the constructor of **RadShortcut**, you should first pass the *key modifier* as a parameter and then an *array of the key mappings*.

8. As a result, you will get this picture runtime. As you can see, the shortcuts are automatically displayed for the RadMenuItems:



Creating shortcuts for RadButtons

You can find the complete source for this project at:

\Shortcuts\<VB|CS>\ButtonsShortcuts

As stated in the Introduction article, RadShortcuts are assigned at RadItem level. That said, you can't assign a shortcut directly to a RadControl (RadButton for example). Instead, you should assign the shortcut to tMyShorthe main element of a RadControl. For example, considering our buttons, you can assign shortcuts to them as it is shown below:

Example Title

```
Me.RadButton1.ButtonElement.Shortcuts.Add(New RadShortcut(Keys.Control, Keys.B))
Me.RadRadioButton1.ButtonElement.Shortcuts.Add(New RadShortcut(Keys.Control, Keys.R))
Me.RadSplitButton1.DropDownButtonElement.Shortcuts.Add(New RadShortcut(Keys.Control,
Keys.S))
Me.RadToggleButton1.ButtonElement.Shortcuts.Add(New RadShortcut(Keys.Control, Keys.T))
```

Example Title

```
this.radButton1.ButtonElement.Shortcuts.Add(new RadShortcut(Keys.Control, Keys.B));
this.radRadioButton1.ButtonElement.Shortcuts.Add(new RadShortcut(Keys.Control, Keys.R));
```

```
this.radSplitButton1.DropDownButtonElement.Shortcuts.Add(new RadShortcut(Keys.Control,
Keys.S));
this.radToggleButton1.ButtonElement.Shortcuts.Add(new RadShortcut(Keys.Control, Keys.T));
```

18.4 Global Shortcuts

RadItem allows you to add shortcuts which will generate a Click event for that item, allowing you to perform specific action, associated with that item. This approach however has some requirements such as you need a RadItem instance and some restrictions such as each shortcut, registered with RadItem is processed only if the item resides on the currently active form within the Application. In order to create a custom shortcuts behavior, you may provide your own IShortcutProvider implementation and to handle its callback methods to provide completely customized shortcut support for your Application, including “Global” shortcuts – that is a shortcut which is not bound to the currently active form. The following code snippet demonstrates how this can be done:



You can find the complete source for this project at:

\Shortcuts\<VB|CS>\GlobalShortcuts

As stated in the Introduction article, RadShortcuts

1. Implement a custom IShortcutProvider
   [VB.NET] Implement IShortcutProvider

```
Public Class MyShortcutProvider
    Implements IShortcutProvider
    Private shortcuts_Renamed As RadShortcutCollection
    Private registered As Boolean

    Public Sub New()
        Me.shortcuts_Renamed = New RadShortcutCollection(Me)
    End Sub

    Public Sub OnPartialShortcut(ByVal e As PartialShortcutEventArgs) Implements
IShortcutProvider.OnPartialShortcut
        'This callback is called when a key that partially matches a registered shortcut
pressed
        'For example if we have CTRL + A, S pressing CTRL + A will execute this callback
        'You will need to set the PartialShortcutEventArgs.Handled to true if you want to
wait for the complete keyboard combination
        e.Handled = True
    End Sub

    Public Sub OnShortcut(ByVal e As ShortcutEventArgs) Implements
IShortcutProvider.OnShortcut
        'A keyboard combination for a specific shortcut is pressed.
        MessageBox.Show("Shortcut [" & e.Shortcut.GetDisplayText() & "] is executed.")
        'Mark the event arguments as "Handled" so that this shortcut is no further
processed.
        e.Handled = True
    End Sub

    Public Sub OnShortcutsChanged() Implements IShortcutProvider.OnShortcutsChanged
```

```
        'Called by the Shortcuts collection when a shortcut is either added or removed fr
the collection
        'This is used for optimization purposes - e.g. is we do not have shortcuts
registered,
        'we do not need to be registered with RadShortcutManager
        If Me.shortcuts_Renamed.Count > 0 Then
            If (Not Me.registered) Then
                RadShortcutManager.Instance.AddShortcutProvider(Me)
                Me.registered = True
            End If
        Else
            If Me.registered Then
                RadShortcutManager.Instance.RemoveShortcutProvider(Me)
                Me.registered = False
            End If
        End If
    End Sub

    Public ReadOnly Property Shortcuts() As RadShortcutCollection Implements
IShortcutProvider.Shortcuts
        Get
            Return Me.shortcuts_Renamed
        End Get
    End Property
End Class
```

[C#] Implement IShortcutProvider

```
public class MyShortcutProvider : IShortcutProvider
{
   private RadShortcutCollection shortcuts;
   private bool registered;

   public MyShortcutProvider()
   {
       this.shortcuts = new RadShortcutCollection(this);
   }

   #region IShortcutProvider Members

   public void OnPartialShortcut(PartialShortcutEventArgs e)
   {
       //This callback is called when a key that partially matches a registered shortcut
pressed
       //For example if we have CTRL + A, S pressing CTRL + A will execute this callback
       //You will need to set the PartialShortcutEventArgs.Handled to true if you want to
wait for the complete keyboard combination
       e.Handled = true;
   }

   public void OnShortcut(ShortcutEventArgs e)
   {
       //A keyboard combination for a specific shortcut is pressed.
       MessageBox.Show("Shortcut [" + e.Shortcut.GetDisplayText() + "] is executed.");
       //Mark the event arguments as "Handled" so that this shortcut is no further
```

```
processed.
        e.Handled = true;
    }

    public void OnShortcutsChanged()
    {
        //Called by the Shortcuts collection when a shortcut is either added or removed fr
the collection
        //This is used for optimization purposes - e.g. is we do not have shortcuts
registered,
        //we do not need to be registered with RadShortcutManager
        if (this.shortcuts.Count > 0)
        {
            if (!this.registered)
            {
                RadShortcutManager.Instance.AddShortcutProvider(this);
                this.registered = true;
            }
        }
        else
        {
            if (this.registered)
            {
                RadShortcutManager.Instance.RemoveShortcutProvider(this);
                this.registered = false;
            }
        }
    }

    public RadShortcutCollection Shortcuts
    {
        get
        {
            return this.shortcuts;
        }
    }

    #endregion
}
```

2. And here is how we utilize the custom shortcut provider:

[VB.NET] Add global shortcut

```
MyShortcutProvider provider;

public Form2()
{
    InitializeComponent();

    provider = new MyShortcutProvider();
    provider.Shortcuts.Add(new RadShortcut(Keys.Control, Keys.A, Keys.S));
}
```

[C#] Add global shortcut

```
MyShortcutProvider provider;

public Form1()
{
   InitializeComponent();

   provider = new MyShortcutProvider();
   provider.Shortcuts.Add(new RadShortcut(Keys.Control, Keys.A, Keys.S));
}
```



The shortcut providers that implement IShortcutProvider interface are registered as WeakReferences. So, if you declare the MyShortcutProvider variable in the scope of the constructor or in another method (for example Form_Load) the Garbage collection may collect the shortcut provider reference at a certain moment. To prevent this from happening, you need to declare the MyShortcutProvider variable globally.

19 Scheduler

19.1 Objectives

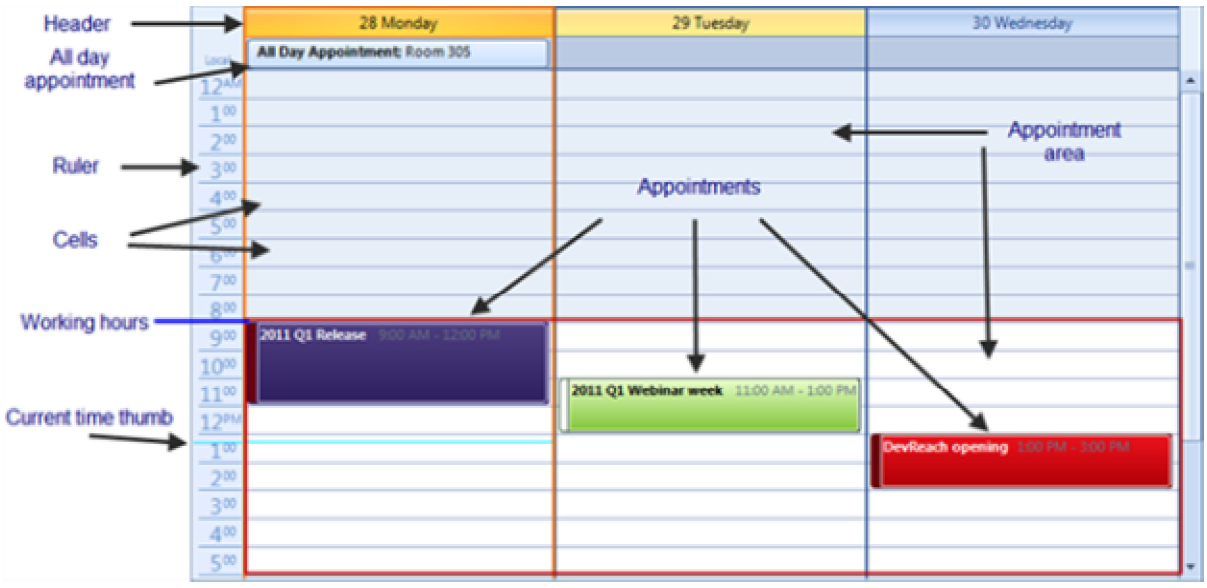
- Learn how to bind data to the scheduler.
- Learn how to add, remove and iterate appointments.
- Learn how to work with resources.
- Learn how to modify the view (i.e. week, day, month) and manipulate appointment status.
- Learn how to drag and drop appointments into the scheduler.

19.2 Introduction

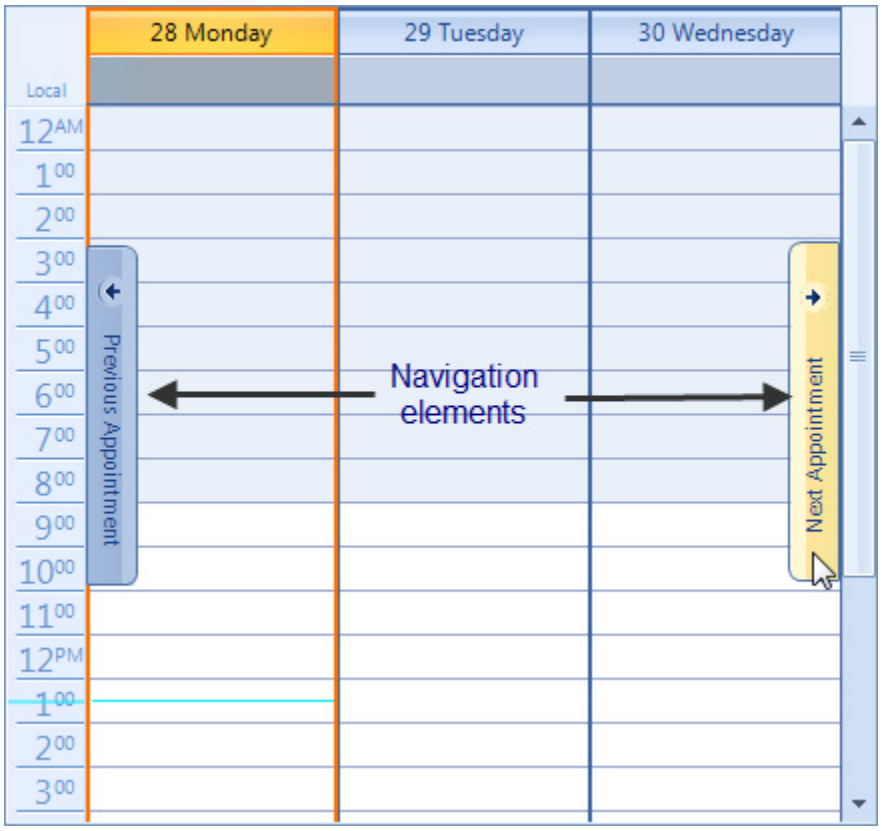
RadScheduler is a highly-customizable appointment presentation component that offers rich Outlook®-style functionality. The product delivers swift performance, simplified deployment and limitless customization capabilities and is suitable for use in large-scale enterprise applications.

The structure of the RadScheduler control varies slightly, depending on the View. The RadScheduler structure below has the same elements for day and week type views.

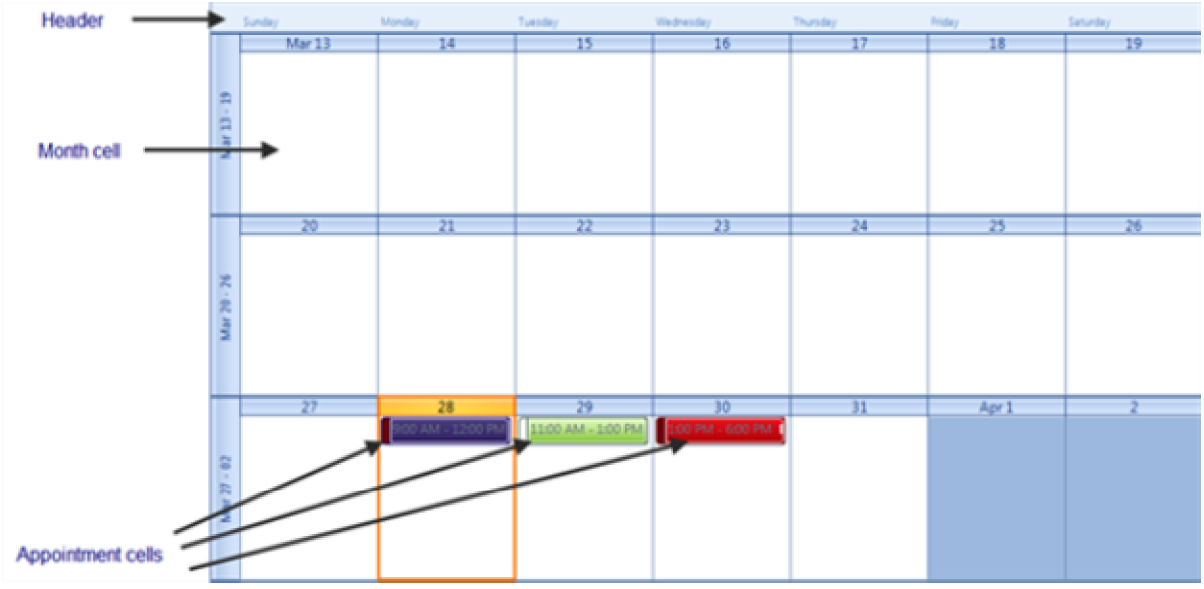
- In the left border of the scheduler is the **Ruler** area. The current time is marked within the ruler by the **Current Time Thumb**.
- The top of the scheduler is the **Header** and displays dates that appointments can be created in.
- Below the Header are the **All Day Appointments** that have no specific duration in hours**.**
- Underneath the All Day Appointments is the **Appointment Area** that contains any number of **Appointments**.



In addition, **Navigation Elements** let the user jump forward or backwards to the nearest appointment.

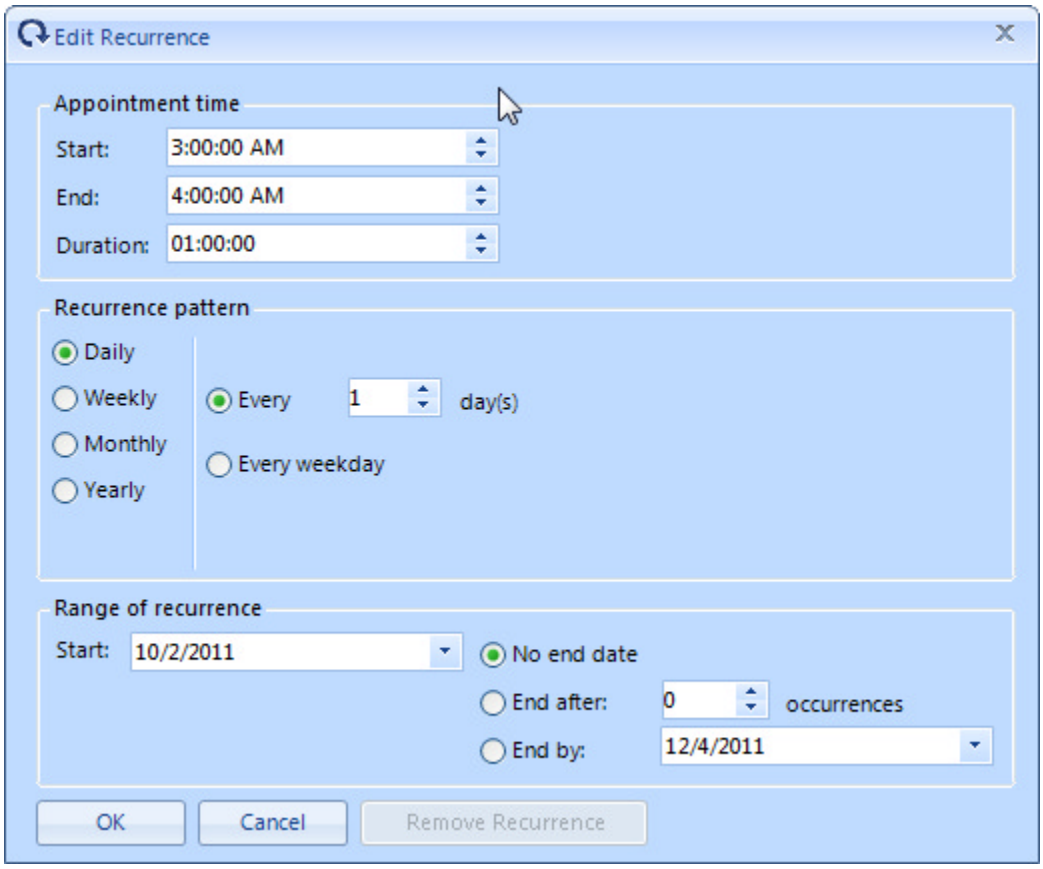


The month view of the scheduler simply has a **Header**, **Month Cells** for every day in the month and zero or more **Appointment Cells** in each month cell.



Recurrence

Appointments can recur daily, weekly, monthly and yearly. The user can set the interval using the "Recurrence" button in the "Edit Appointment" dialog. Here they can set the appointment duration, the frequency appointments should occur and the range that occurrences should fall within. These settings can also be configured programmatically.



Appointments can be configured programmatically to recur every minute or hour.

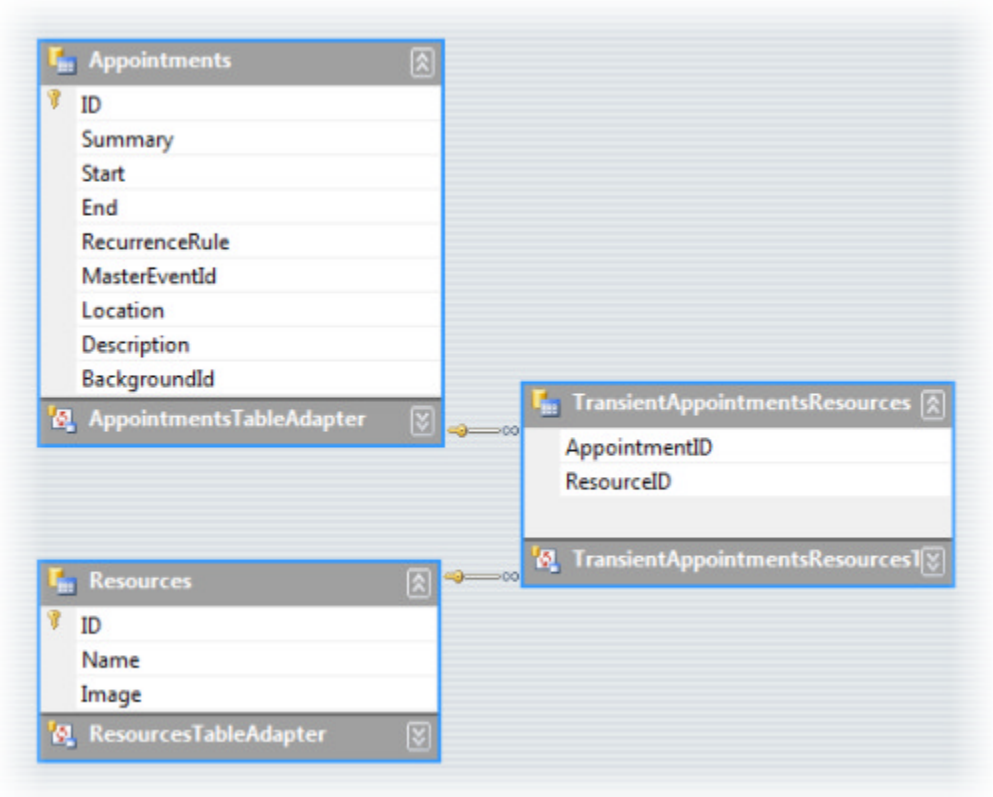
Binding Data

RadScheduler's DataSource property lets you bind to objects that inherit the base abstract **SchedulerDataSource** class. The **SchedulerBindingDataSource** implementation of SchedulerdataSource binds traditional data stores like lists of business objects and database tables. You can find the SchedulerBindingDataSource component in the Toolbox.

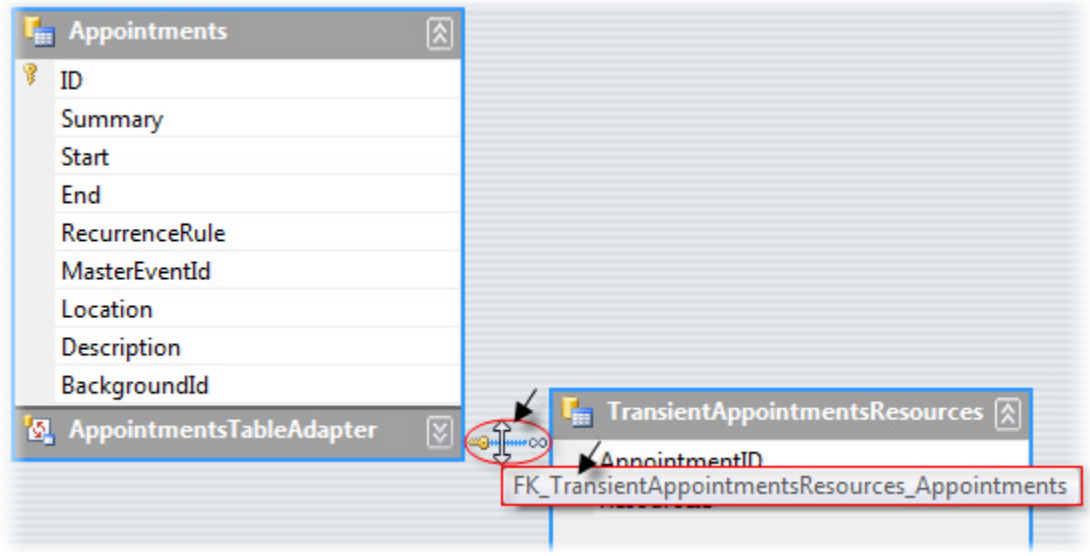
To have the data source make any sense to the SchedulerBindingDataSource you also need to define mappings that link the expected information to columns in a database table or business object. There are two mapping objects that need to be loaded up and assigned, the **AppointmentMappingInfo** and **ResourceMappingInfo**. AppointmentMappingInfo has a series of string properties that point to all appointment related data such as start date/time, end date/time, a description, etc. The ResourcesMappingInfo object has a Name and Image.

Here's a dataset design view for the SchedulerData.mdf database file that ships with RadControls for Winforms. You can see all the fields that can be defined for appointments and resources. Also notice a "join table" named "TransientAppointmentsResources" that assigns a particular resource to an appointment.

Notice that while the database structure allows for multiple resources, the built-in scheduler dialog only allows a single resource to be selected at a time.



The foreign key relationship between "TransientAppointmentsResources" and the "Appointments" table will also be defined in the AppointmentMappingInfo object.



You can find SchedulerData.mdb (Access) and SchedulerData.mdf (MS SQL database file) in the installation directory under \Examples\QuickStart\DataSources.

Here's some example code that demonstrates loading the AppointmentMappingInfo fields with column names in a database table. The **Resources** property is assigned the name of the foreign key that joins the Appointments and the "join table' that sits between the appointments and resources table. The **ResourceID** property is assigned to the column in the "join table" of a unique resource identifier.

RadScheduler works from a provider model so that in the future, custom appointment and resource providers

can be plugged in to replace the built-in providers. The AppointmentMappingInfo is assigned to the Mapping property of the schedulers datasource event provider. Review the code below to see how this is done when binding to a database tables.

[VB] Mapping Appointment Data

```
Dim appointmentMappingInfo As New AppointmentMappingInfo()
appointmentMappingInfo.Start = "Start"
appointmentMappingInfo.[End] = "End"
appointmentMappingInfo.Summary = "Summary"
appointmentMappingInfo.Description = "Description"
appointmentMappingInfo.Location = "Location"
appointmentMappingInfo.BackgroundId = "BackgroundID"
appointmentMappingInfo.StatusId = "StatusID"
appointmentMappingInfo.RecurrenceRule = "RecurrenceRule"
appointmentMappingInfo.Resources = "FK_TransientAppointmentsResources_Appointments"
appointmentMappingInfo.ResourceId = "ResourceID"
schedulerBindingDataSource1.EventProvider.Mapping = appointmentMappingInfo
```

[C#] Mapping Appointment Data

```
AppointmentMappingInfo appointmentMappingInfo = new AppointmentMappingInfo();
appointmentMappingInfo.Start = "Start";
appointmentMappingInfo.End = "End";
appointmentMappingInfo.Summary = "Summary";
appointmentMappingInfo.Description = "Description";
appointmentMappingInfo.Location = "Location";
appointmentMappingInfo.BackgroundId = "BackgroundID";
appointmentMappingInfo.StatusId = "StatusID";
appointmentMappingInfo.RecurrenceRule = "RecurrenceRule";
appointmentMappingInfo.Resources = "FK_TransientAppointmentsResources_Appointments";
appointmentMappingInfo.ResourceId = "ResourceID";
schedulerBindingDataSource1.EventProvider.Mapping = appointmentMappingInfo;
```

The resource mapping has fewer columns and doesn't need any information about joining to the appointments table.

[VB] Mapping Resource Data

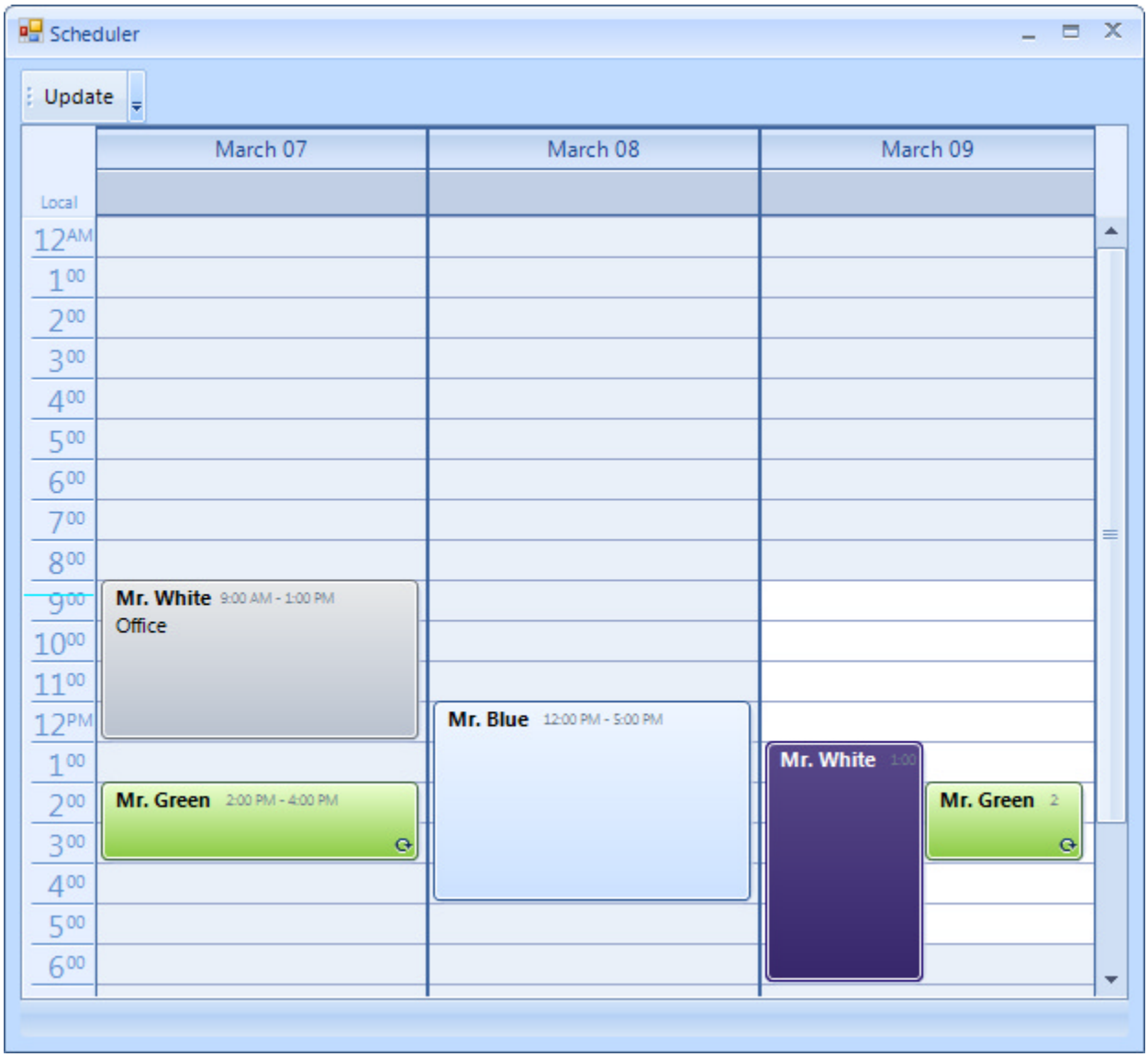
```
Dim resourceMappingInfo As New ResourceMappingInfo()
resourceMappingInfo.Id = "ID"
resourceMappingInfo.Name = "Name"
schedulerBindingDataSource1.ResourceProvider.Mapping = resourceMappingInfo
```

[C#] Mapping Resource Data

```
ResourceMappingInfo resourceMappingInfo = new ResourceMappingInfo();
resourceMappingInfo.Id = "ID";
resourceMappingInfo.Name = "Name";
schedulerBindingDataSource1.ResourceProvider.Mapping = resourceMappingInfo;
```

19.3 Getting Started

This walkthrough will show a minimal RadScheduler setup where we will bind to the supplied SchedulerData.mdf that ships with RadControls for WinForms. We will map the the necessary database table columns and also add a little code to persist scheduling changes made by the user to the database.



Scheduler Without Binding

You can find the complete source for this project at:

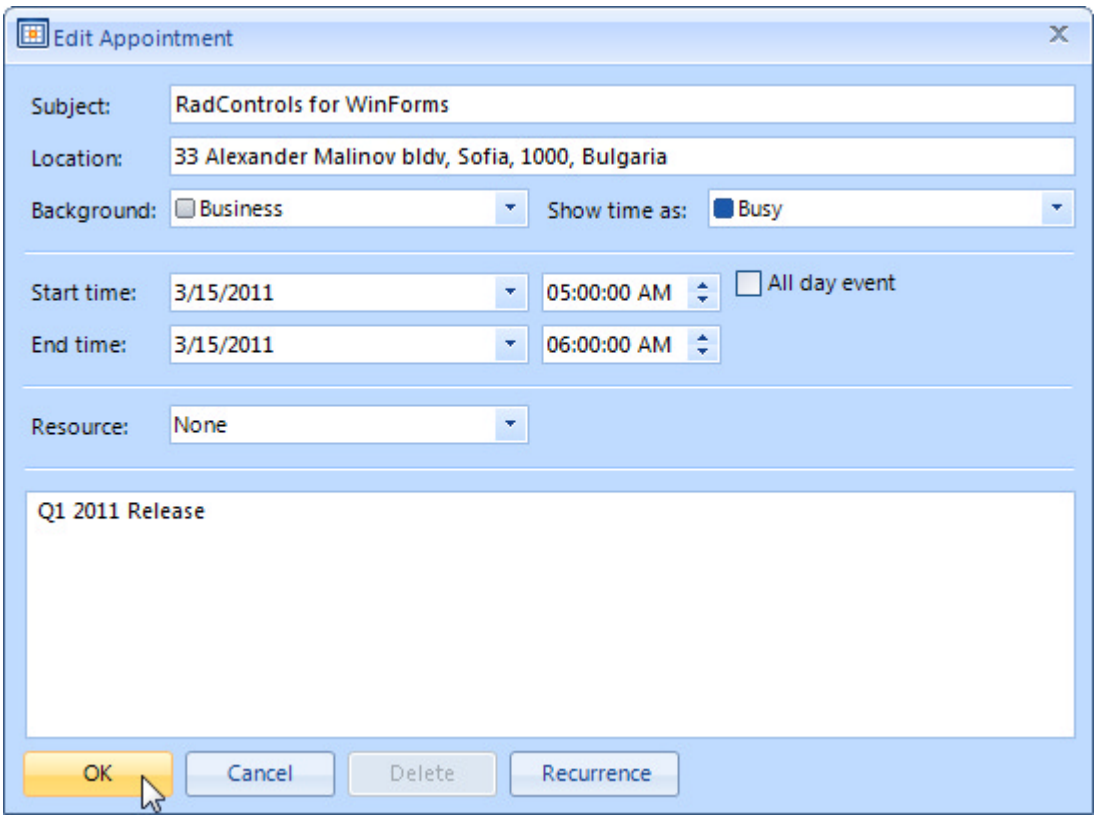
\Scheduler\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. In the Properties window, set the form **Size** property to "500,400".
6. Add a **RadStatusStrip** to the form. Add a RadLabelElement to the status strip. Set the **Name** property to "lblStatus" and the **Text** to "".
7. Add a RadCommandBar to the form. Add a **CommandBarButton** to the new row. Set the button element

Name to "btnUpdate" and **Text** to "Update".

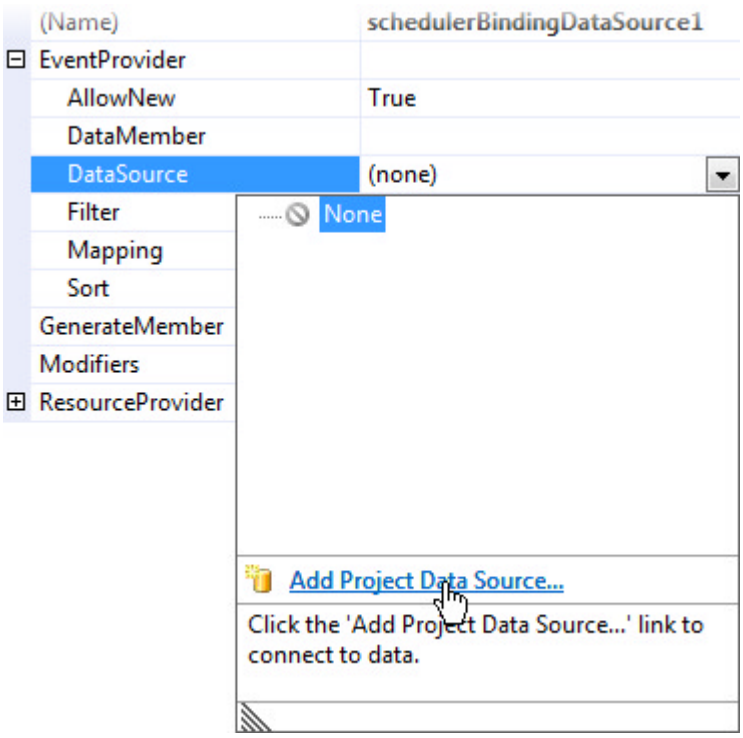
8. Change the new RadForm1 to be the startup form.
9. From the Toolbox, add a **RadScheduler** to the form and set the **Dock** property to "Fill".
10. Press **Ctrl-F5** to run the application. Experiment with the scheduler functionality. Double-Click a scheduler cell and add an appointment. Notice the effect of the "Background" and "Show time as" entry fields.

> At this point the scheduler is un-bound so changes will not persist.

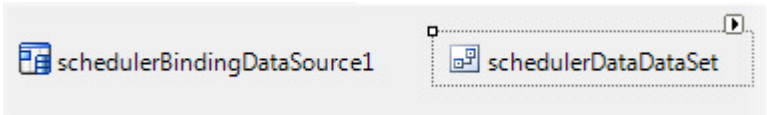


Binding the Scheduler

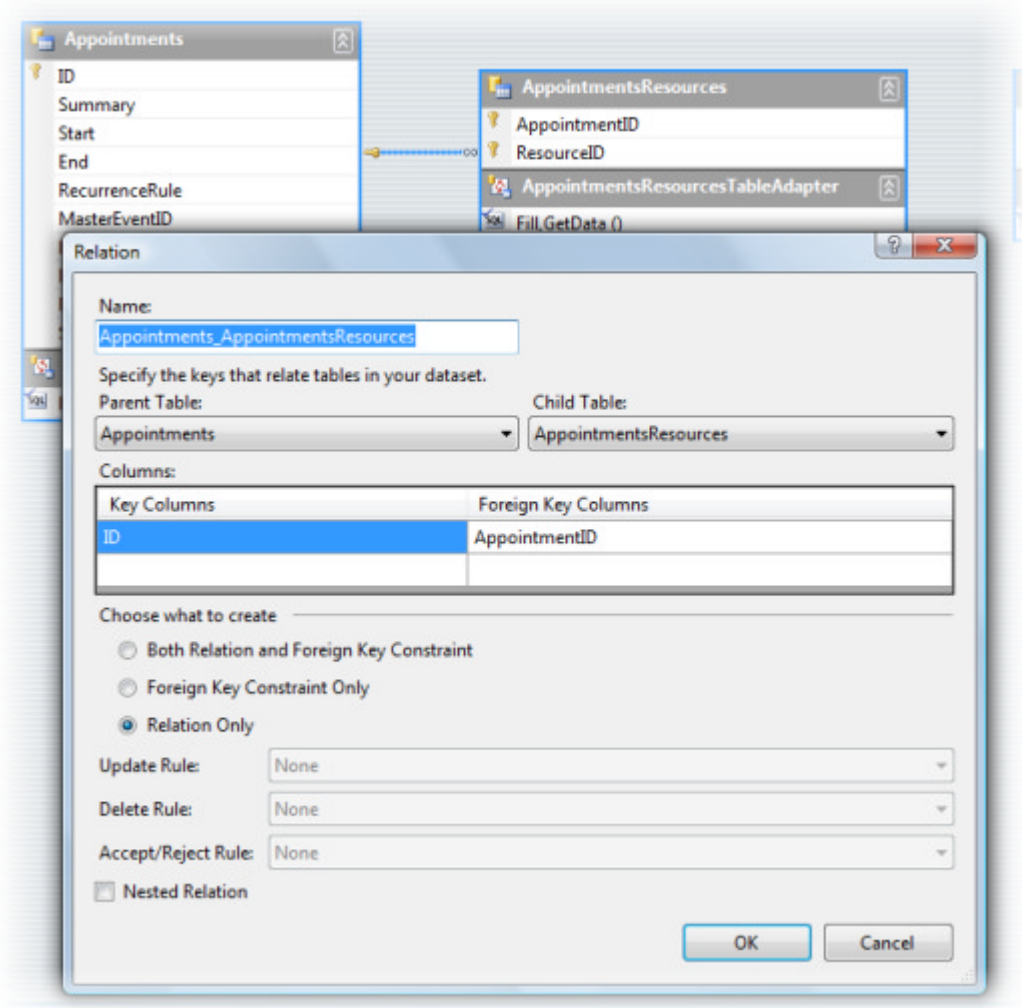
1. Add a **SchedulerBindingDataSource** from the Toolbox to the form.
2. In the Properties window, open the **EventProvider** property. Drop down the **DataSource** sub-property list and select **Add Project DataSource...** This step will display the Data Source Configuration Wizard dialog.



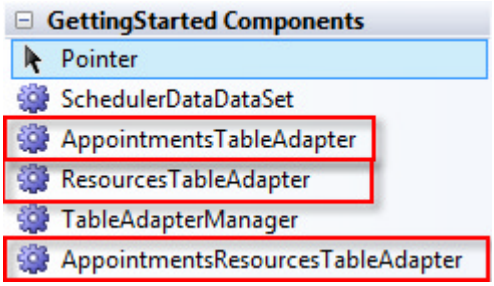
3. In the "Choose a Data Source Type" page of the wizard, select the **Database** type and click the **Next** button to continue.
4. In the "Choose a Data Connection" page of the wizard, add a connection to the SchedulerData.mdf file supplied with RadControls for WinForms distribution.
    - Click the **New Connection...** button to display the Add Connection dialog.
    - Click the **Change** button and select the Access Database File option.
    - Set the Database File Name to the Schedulerdata.mdb file located in the RadControls for Winforms installation directory under \Examples\QuickStart\DataSources.
    - Click **OK** to close the Add Connecction dialog.
5. Back in the "Choose a Data Connection" page of the wizard, click the **Next** button to continue.
6. A dialog will popup to ask where the file should be located. Choose **No** to leave the file located in the \Examples\QuickStart\DataSources directory.
7. In the "Save the Connection String in the Application Configuration File" click the **Next** button to continue.
8. In the "Choose Your Database Objects" page of the wizard, select the "Tables" checkbox to automatically select the Appointments, Resources and AppointmentsResources tables.
9. Click **Finish** to complete the wizard and close the dialog. This step will create a DataSet component and add it to the component tray below the form designer.



10. Click the "SchedulerDataDataSet" Smart Tag and select **Edit in Dataset Designer**. Add a relation between the Appointments and AppointmentsResources tables and name it "Appointments_AppointmentsResources". Make the "Key Columns" entry be "ID" from the Appointments table and set "Foreign Key Columns" from the AppointmentsResources table to "AppointmentID".



11. Build the project. This step will create several useful adapter components we can use to fill the dataset.



12. From these new components, add the AppointmentsTableAdapter
13. Add the code below for the form's Load event handler to fill the table in the dataset.

[VB] Filling the DataSet

```
appointmentsTableAdapter1.Fill(schedulerDataDataSet.Appointments)
resourcesTableAdapter1.Fill(schedulerDataDataSet.Resources)
appointmentsResourcesTableAdapter1.Fill(schedulerDataDataSet.AppointmentsResources)
```

[C#] Filling the DataSet

```
appointmentsTableAdapter1.Fill(schedulerDataDataSet.Appointments);
```

```
resourcesTableAdapter1.Fill(schedulerDataDataSet.Resources);
appointmentsResourcesTableAdapter1.Fill(
 schedulerDataDataSet.AppointmentsResources);
```

14. Add code to the end of the Load event handler that configures the Appointment provider. The code will map the Appointment columns, the relationship between the appointment and resources tables and assign the EventProvider DataSource.

[VB] Configure the Appointment Provider

```
' create and assign appointment mapping
Dim appointmentMappingInfo As New AppointmentMappingInfo()
appointmentMappingInfo.Start = "Start"
appointmentMappingInfo.[End] = "End"
appointmentMappingInfo.Summary = "Summary"
appointmentMappingInfo.Description = "Description"
appointmentMappingInfo.Location = "Location"
appointmentMappingInfo.BackgroundId = "BackgroundID"
appointmentMappingInfo.StatusId = "StatusID"
appointmentMappingInfo.RecurrenceRule = "RecurrenceRule"
appointmentMappingInfo.Resources = "Appointments_AppointmentsResources"
appointmentMappingInfo.ResourceId = "ResourceID"
schedulerBindingDataSource1.EventProvider.Mapping = appointmentMappingInfo
schedulerBindingDataSource1.EventProvider.DataSource = schedulerDataDataSet.Appointments
```

[C#] Configure the Appointment Provider

```
// create and assign appointment mapping
AppointmentMappingInfo appointmentMappingInfo = new AppointmentMappingInfo();
appointmentMappingInfo.Start = "Start";
appointmentMappingInfo.End = "End";
appointmentMappingInfo.Summary = "Summary";
appointmentMappingInfo.Description = "Description";
appointmentMappingInfo.Location = "Location";
appointmentMappingInfo.BackgroundId = "BackgroundID";
appointmentMappingInfo.StatusId = "StatusID";
appointmentMappingInfo.RecurrenceRule = "RecurrenceRule";
appointmentMappingInfo.Resources = "Appointments_AppointmentsResources";
appointmentMappingInfo.ResourceId = "ResourceID";
schedulerBindingDataSource1.EventProvider.Mapping = appointmentMappingInfo;
schedulerBindingDataSource1.EventProvider.DataSource = schedulerDataDataSet.Appointments;
```

15. Configure the resources provider mapping:

[VB] Configure the Resources Provider

```
' create and assign resource mapping
Dim resourceMappingInfo As New ResourceMappingInfo()
resourceMappingInfo.Id = "ID"
resourceMappingInfo.Name = "ResourceName"
schedulerBindingDataSource1.ResourceProvider.Mapping = resourceMappingInfo
schedulerBindingDataSource1.ResourceProvider.DataSource = schedulerDataDataSet.Resources
```

[C#] Configure the Resources Provider

```
// create and assign resource mapping
ResourceMappingInfo resourceMappingInfo = new ResourceMappingInfo();
resourceMappingInfo.Id = "ID";
resourceMappingInfo.Name = "ResourceName";
schedulerBindingDataSource1.ResourceProvider.Mapping = resourceMappingInfo;
schedulerBindingDataSource1.ResourceProvider.DataSource = schedulerDataDataSet.Resources;
```

16. Finally, assign the configured SchedulerBindingDataSource to the RadScheduler's DataSource property.

[VB] Assigning the RadScheduler DataSource

```
' assign mapped binding data source
radScheduler1.DataSource = schedulerBindingDataSource1
```

[C#] Assigning the RadScheduler DataSource

```
// assign mapped binding data source
radScheduler1.DataSource = schedulerBindingDataSource1;
```

Updating the Database

1. The preceding takes care of binding the data, but we need to update the dataset to persist the data all the way back to the database. Add code to the Click event handler for the "Update" button in the tool strip.

[VB] Update the Dataset

```
Private Sub btnUpdate_Click(sender As Object, e As EventArgs)
 ' save scheduler changes
 appointmentsTableAdapter1.Update(schedulerDataDataSet.Appointments)
 appointmentsResourcesTableAdapter1.Update(schedulerDataDataSet.AppointmentsResources)
 lblStatus.Text = "Updated scheduler at " + DateTime.Now.ToString()
End Sub
```

[C#] Update the Dataset

```
private void btnUpdate_Click(object sender, EventArgs e)
{
 // save scheduler changes
 appointmentsTableAdapter1.Update(schedulerDataDataSet.Appointments);
 appointmentsResourcesTableAdapter1.Update(schedulerDataDataSet.AppointmentsResources);
 lblStatus.Text = "Updated scheduler at " + DateTime.Now.ToString();
}
```

2. Run the application. Add some appointments, then click the Update button. Re-run the application to verify the appointments are persisted in the database.

19.4 Using the Design Time Interface

The most important property aside from those dealing with data binding is **ActiveViewType** which can be **Day**, **MultiDay**,**TimeLline, Week**, **WorkWeek** or **Month**. Like RadCalendar, the **FocusedDate** is a DateTime that determines the visible day, week or month currently displayed in the scheduler.

Here are some of the other helpful properties that can be set at design time:

- **AccessibleInterval** is a the duration of time that the user is allowed to use.

AccessibleInterval	[3/17/2009 12:00:00 AM - 3/19/2009 8:58:23 AM]
Duration	4.00:00:00
End	3/21/2009
Start	3/17/2009


The active view cannot be changed if the new view is not in the accessible interval.

- **AppointmentTitleFormat** is fed to a String.Format() call and supplies parameters for Start, End, Subject and Location in that order. So for example you could assign a string with a format like this:

Starts: {0} Ends: {1} Subject: {2} Location: {3}

- **AllowAppointmentMove** and **AllowAppointmentResize** control if the user can drag or resize an appointment at runtime.
- To toggle visibility of appointment status, all day appointment status and navigation elements use the **ShowAppointmentStatus**, **ShowAllDayAppointmentStatus** and **ShowNavigationElements** properties.

19.5 Programming the Scheduler

Working with Appointments

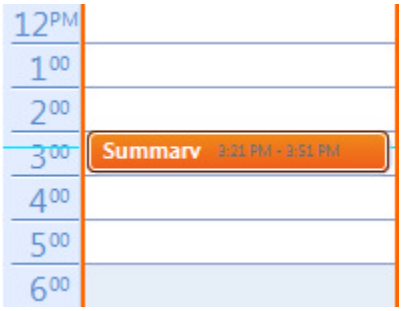
Use one of many overloads to build a new Appointment instance. The example below creates an appointment that starts "Now", extends for one half hour and has summary and descriptions. You can use the StatusId and BackgroundId appointment properties to provide visual feedback in the left edge and background of each appointment.

[VB] Add a new appointment

```
Dim appointment As New Appointment(DateTime.Now, TimeSpan.FromMinutes(30), "Summary",
"Description")
appointment.StatusId = CType(AppointmentStatus.Busy, Int32)
appointment.BackgroundId = CType(AppointmentBackground.NeedsPreparation, Int32)
radScheduler1.Appointments.Add(appointment)
```

[C#] Add a new appointment

```
Appointment appointment = new Appointment(DateTime.Now, TimeSpan.FromMinutes(30), "Summary",
"Description");
appointment.StatusId = (int)AppointmentStatus.Busy;
appointment.BackgroundId = (int)AppointmentBackground.NeedsPreparation;
radScheduler1.Appointments.Add(appointment);
```



Working with Views

The scheduler displays dates and times using a "view" that can be "Day", "MultiDay", "Week", "WorkWeek" or "Month".

> The difference between a day view and a multi-day view is that while the day view is constrained to showing a single sequence of consecutive days (for example 7th through 10th, or 10th thru 12th, or just the 15th), the multi-day view can display all of the above sequences at once.

At any one time the scheduler displays a view using a descendent of the SchedulerView class: **SchedulerDayView**, **SchedulerWeekView, SchedulerTimelineView** and **SchedulerMonthView**. Each view has special properties particular to the view. Use the RadScheduler **GetDayView()**, **GetWeekView(), GetTimelineView()** and **GetMonthView()** methods to get the respective views. Here's an example that retrieves the day view and sets the ruler to start at the second hour and stop at the fifth hour:

[VB] Using the GetDayView() Method

```
Dim dayView As SchedulerDayView = radScheduler1.GetDayView()
dayView.RulerStartScale = 2
dayView.RulerEndScale = 5
```

[C#] Using the GetDayView() Method

```
SchedulerDayView dayView = radScheduler1.GetDayView();
dayView.RulerStartScale = 2;
dayView.RulerEndScale = 5;
```

After running the code, the day view for the scheduler looks like this screenshot:



Change between views by changing the **ActiveViewType** property to one of the **SchedulerViewType** enumeration members.

[VB] Assigning the ActiveViewType

```
radScheduler1.ActiveViewType = SchedulerViewType.Day
```

[C#] Assigning the ActiveViewType

```
radScheduler1.ActiveViewType = SchedulerViewType.Day;
```

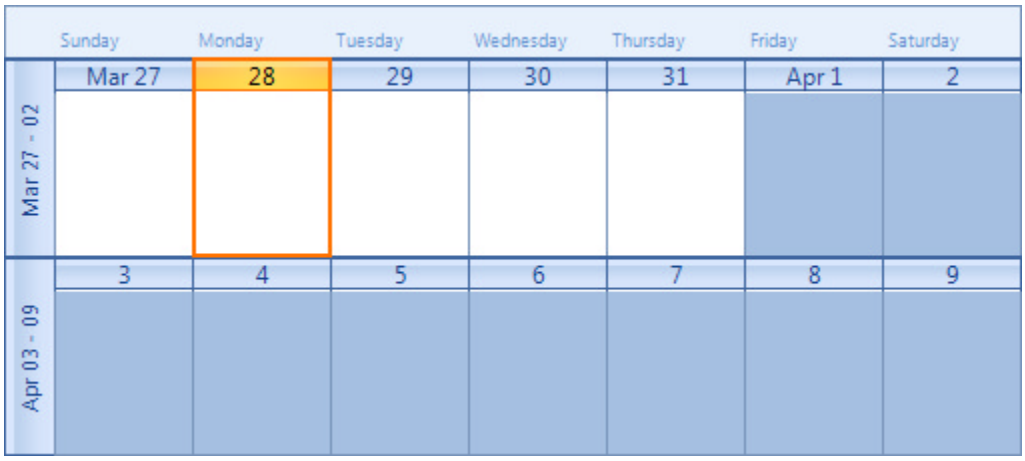
Retrieve the view that is currently being displayed by using the ActiveView property, cast it to be the ActiveViewType

[VB] Using the ActiveView Property

```
radScheduler1.ActiveViewType = SchedulerViewType.Month
(TryCast(radScheduler1.ActiveView, SchedulerMonthView)).WeekCount = 2
```

[C#] Using the ActiveView Property

```
radScheduler1.ActiveViewType = SchedulerViewType.Month;
(radScheduler1.ActiveView as SchedulerMonthView).WeekCount = 2;
```



Detect changes to the view by handling the **ActiveViewChanging** and **ActiveViewChanged** events. As always, the "Changing" event arguments provide the ability to cancel the view change, but also the "old" and "new" views before and after the view changes transpires:

[VB] Handling the ActiveViewChanging Event

```
Private Sub radScheduler1_ActiveViewChanging(sender As Object, e As
SchedulerViewChangingEventArgs)
 lblStatus.Text = [String].Format("Old: {0} New: {1}", e.OldView.ViewType.ToString(),
e.NewView.ViewType.ToString())
End Sub
```

[C#] Handling the ActiveViewChanging Event

```
private void radScheduler1_ActiveViewChanging(object sender, SchedulerViewChangingEventArgs
e)
{
 lblStatus.Text = String.Format("Old: {0}   New: {1}",
   e.OldView.ViewType.ToString(), e.NewView.ViewType.ToString());
}
```

In this walkthrough you will dynamically change the view, change some of the view specific properties and handle the ActiveViewChanging event.

Project Setup



You can find the complete source for this project at:

\Scheduler\<VB|CS>\Views

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. Add the **DesertTheme** from the ToolBox to the form.
6. In the Properties window, set the form **ThemeName** to **Desert**.
7. Add a **RadStatusStrip** to the form. Set the **ThemeName** to **Desert**. Add a RadLabelElement to the status strip. Set the **Name** property to "lblStatus" and the **Text** to "".
8. Add a RadCommandBar to the form. Add elements to the command bar strip:

- Using the context menu, add a **CommandBarDropDownList** to the new row. Set the **Name** to "ddActiveViewType" and **Text** to "".
- Using the context menu, add a **CommandBarDropDownList** to the new row. Set the **Name** to "ddRange" and **Text** to "".
- Using the context menu, add a **CommandBarDropDownList** to the new row. Set the **Name** to "ddCount" and **Text** to "".

9. Change the new RadForm1 to be the startup form.
10. From the Toolbox, add a **RadScheduler** to the form and set the **Dock** property to "Fill" and the **ThemeName** to **Desert**.
11. Add code to the form load that will add values to the combo boxes in the tool strip for **SchedulerViewType** and **ScaleRange** enumerations. Also, add a simple range of integers to the "count" combo box.

[VB] Handling the Load Event

```
Private Sub RadForm1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
For Each viewType As SchedulerViewType In System.Enum.GetValues(GetType(SchedulerViewTyp
Dim item As RadListDataItem = New RadListDataItem()
item.Text = viewType.ToString()
item.Value = viewType
ddActiveViewType.Items.Add(item)
Next viewType
For Each range As ScaleRange In System.Enum.GetValues(GetType(ScaleRange))
Dim item As RadListDataItem = New RadListDataItem()
item.Text = range.ToString()
item.Value = range
ddRange.Items.Add(item)
Next range
For i As Integer = 1 To 9
Dim item As RadListDataItem = New RadListDataItem()
item.Text = i.ToString()
item.Value = i
ddCount.Items.Add(item)
Next i
End Sub
```

[C#] Handling the Load Event

```
private void RadForm1_Load(object sender, EventArgs e)
{
foreach (SchedulerViewType viewType in Enum.GetValues(typeof(SchedulerViewType)))
{
RadListDataItem item = new RadListDataItem();
item.Text = viewType.ToString();
item.Value = viewType;
ddActiveViewType.Items.Add(item);
}
foreach (ScaleRange range in Enum.GetValues(typeof(ScaleRange)))
{
RadListDataItem item = new RadListDataItem();
item.Text = range.ToString();
item.Value = range;
ddRange.Items.Add(item);
}
```

```
for (int i = 1; i < 10; i++)
{
RadListDataItem item = new RadListDataItem();
item.Text = i.ToString();
item.Value = i;
ddCount.Items.Add(item);
}
}
```

12. Next add a SelectedIndexChanged event handler for the cbActiveViewType combo box:

[VB] Set the Current ActiveViewType

```
Private Sub ddActiveViewType_SelectedIndexChanged(ByVal sender As System.Object, ByVal e
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles
ddActiveViewType.SelectedIndexChanged
Dim dropDownList As RadDropDownListElement = TryCast(sender, RadDropDownListElement)
radScheduler1.ActiveViewType = CType(dropDownList.SelectedValue, SchedulerViewType)
End Sub
```

[C#] Set the Current ActiveViewType

```
private void ddActiveViewType_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
RadDropDownListElement dropDownList = sender as RadDropDownListElement;
radScheduler1.ActiveViewType = (SchedulerViewType)(dropDownList.SelectedValue);
}
```

13. Add another SelectedIndexChanged event handler for the cbRange combo box element:

[VB] Set the RangeFactor

```
Private Sub ddRange_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles ddRange.SelectedIndexChange
Dim dropDownList As RadDropDownListElement = TryCast(sender, RadDropDownListElement)
Dim range As ScaleRange = CType(dropDownList.SelectedValue, ScaleRange)
' set the appropriate range factor based on the type of view
Select Case radScheduler1.ActiveViewType
Case SchedulerViewType.Day
TryCast(radScheduler1.ActiveView, SchedulerDayView).RangeFactor = range
Case SchedulerViewType.MultiDay
TryCast(radScheduler1.ActiveView, SchedulerMultiDayView).RangeFactor = range
Case SchedulerViewType.Week, SchedulerViewType.WorkWeek
TryCast(radScheduler1.ActiveView, SchedulerWeekView).RangeFactor = range
End Select
End Sub
```

[C#] Set the RangeFactor

```
private void ddRange_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
RadDropDownListElement dropDownList = sender as RadDropDownListElement;
ScaleRange range = (ScaleRange)(dropDownList.SelectedValue);
// set the appropriate range factor based on the type of view
switch (radScheduler1.ActiveViewType)
{
case SchedulerViewType.Day:
(radScheduler1.ActiveView as SchedulerDayView).RangeFactor = range;
```

```
break;
case SchedulerViewType.MultiDay:
(radScheduler1.ActiveView as SchedulerMultiDayView).RangeFactor = range;
break;
case SchedulerViewType.Week:
case SchedulerViewType.WorkWeek:
(radScheduler1.ActiveView as SchedulerWeekView).RangeFactor = range;
break;
}
}
```

14. Add another SelectedIndexChanged event handler for the cbCount combo box element:

[VB] Setting the DayCount or WeekCount

```
Private Sub ddCount_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs) Handles ddCount.SelectedIndexChange
Dim dropDownList As RadDropDownListElement = TryCast(sender, RadDropDownListElement)
Dim count As Integer = CInt(Fix(dropDownList.SelectedValue))
' set the Day or WeekCount based on the current view
Select Case radScheduler1.ActiveViewType
Case SchedulerViewType.Day
TryCast(radScheduler1.ActiveView, SchedulerDayView).DayCount = count
Case SchedulerViewType.MultiDay
TryCast(radScheduler1.ActiveView, SchedulerMultiDayView).DayCount = count
Case SchedulerViewType.Month
TryCast(radScheduler1.ActiveView, SchedulerMonthView).WeekCount = count
End Select
End Sub
```

[C#] Setting the DayCount or WeekCount

```
private void ddCount_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
RadDropDownListElement dropDownList = sender as RadDropDownListElement;
int count = (int)dropDownList.SelectedValue;
// set the Day or WeekCount based on the current view
switch (radScheduler1.ActiveViewType)
{
case SchedulerViewType.Day:
(radScheduler1.ActiveView as SchedulerDayView).DayCount = count;
break;
case SchedulerViewType.MultiDay:
(radScheduler1.ActiveView as SchedulerMultiDayView).DayCount = count;
break;
case SchedulerViewType.Month:
(radScheduler1.ActiveView as SchedulerMonthView).WeekCount = count;
break;
}
}
```

15. Handle the RadScheduler ActiveViewChanging event. Use the SchedulerViewChangingEventArgs OldView and NewView to display in the status label.

[VB] Handling the ActiveViewChanging Event

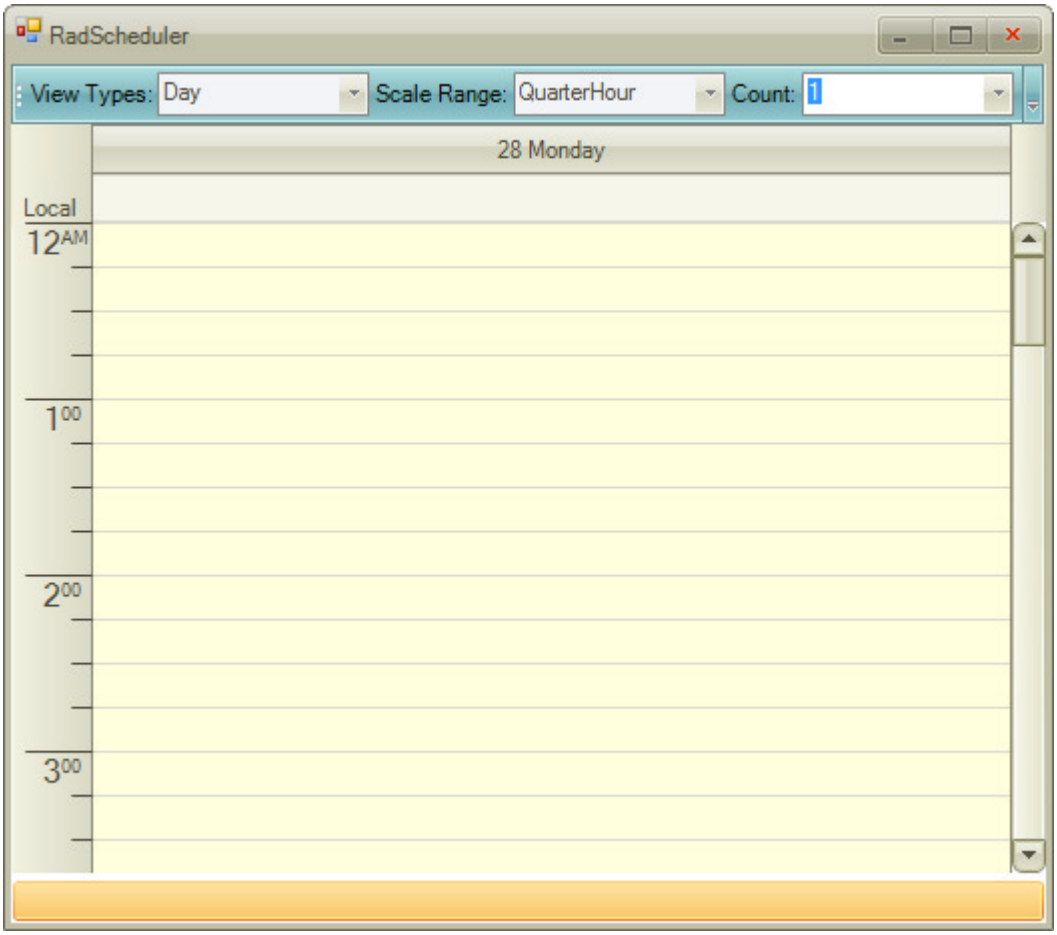
```
Private Sub radScheduler1_ActiveViewChanging(ByVal sender As Object, ByVal e As
SchedulerViewChangingEventArgs) Handles radScheduler1.ActiveViewChanging
```

```
lblStatus.Text = String.Format("Old: {0} New: {1}", e.OldView.ViewType.ToString(),
e.NewView.ViewType.ToString())
End Sub
```

[C#] Handling the ActiveViewChanging Event

```
private void radScheduler1_ActiveViewChanging(object sender, SchedulerViewChangingEventA
e)
{
lblStatus.Text = String.Format("Old: {0} New: {1}",
e.OldView.ViewType.ToString(), e.NewView.ViewType.ToString());
}
```

16. Run the application and test the various combinations of settings.



Using Resources

Add resources by using the Resources collection **Add()** method and passing a new Resource instance. Pass a unique ID and a description in the constructor.

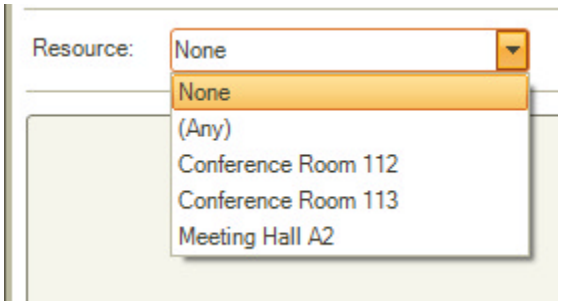
[VB] Adding Resources

```
Private Sub LoadResources()
 Dim descriptions As String() = {"Conference Room 112", "Conference Room 113", "Meeting Hall
A2"}
 Dim count As Integer = 1
 For Each description As String In descriptions
  radScheduler1.Resources.Add(New Resource(System.Math.Max
```

```
(System.Threading.Interlocked.Increment(count),count - 1), description))
 Next
End Sub
```

[C#] Adding Resources

```
private void LoadResources()
{
 string[] descriptions = { "Conference Room 112", "Conference Room 113", "Meeting Hall
A2" };
 int count = 1;
 foreach (string description in descriptions)
 {
   radScheduler1.Resources.Add(new Resource(count++, description));
 }
}
```



A Resource object also has an **Image** property that is displayed in the resource selection control.

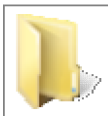


Binding to Business Objects

What if your scheduling data originates from somewhere other than an easily accessible database? An API that accesses a legacy system or an email based system are two examples that might fit this description. RadScheduler allows binding to objects of any arbitrary structure.

Binding to objects follows the same basic pattern as binding to database tables. You must assign a collection of objects to an instance of SchedulerBindingDataSource. You also need to define mappings so that the appointment data expected in the scheduler (Start, End, Subject, etc.) is satisfied by specific properties in the bound objects.

The code below is an example appointment. Keep in mind that the particular construction of the MyAppointment class and the names of its properties are arbitrary. The mappings will decide where properties are used. Notice that the object implements the **INotifyPropertyChanged** interface. Without this interface implementation the populated appointment object data will not show up in the scheduler.



You can find the complete source for this project at:

\Scheduler\<VB|CS>\BusinessObjects

[VB] The Sample Appointment Class

```
Imports System
Imports System.ComponentModel
Namespace BusinessObjects
 Public Class MyAppointment
  Implements INotifyPropertyChanged
  Private start As DateTime = DateTime.Now
  Private [end] As DateTime = DateTime.Now
  Private subject As String = String.Empty
  Private description As String = String.Empty
  Private location As String = String.Empty
  Private id As Guid = Guid.NewGuid()
  Public Sub New()
  End Sub
  Public Sub New(start As DateTime, [end] As DateTime, subject As String, description As
String, location As String)
   Me.start = start
   Me.[end] = [end]
   Me.subject = subject
   Me.description = description
   Me.location = location
  End Sub
  Public Property Id() As Guid
   Get
    Return Me.id
   End Get
   Set
    If Me.id <> value Then
     Me.id = value
     Me.OnPropertyChanged("Id")
    End If
   End Set
  End Property
  Public Property Start() As DateTime
   Get
```

```
    Return Me.start
   End Get
   Set
    If Me.start <> value Then
     Me.start = value
     Me.OnPropertyChanged("Start")
    End If
   End Set
  End Property
  Public Property [End]() As DateTime
   Get
    Return Me.[end]
   End Get
   Set
    If Me.[end] <> value Then
     Me.[end] = value
     Me.OnPropertyChanged("End")
    End If
   End Set
  End Property
  Public Property Subject() As String
   Get
    Return Me.subject
   End Get
   Set
    If Me.subject <> value Then
     Me.subject = value
     Me.OnPropertyChanged("Subject")
    End If
   End Set
  End Property
  Public Property Description() As String
   Get
    Return Me.description
   End Get
   Set
    If Me.description <> value Then
     Me.description = value
     Me.OnPropertyChanged("Description")
    End If
   End Set
  End Property
  Public Property Location() As String
   Get
    Return Me.location
   End Get
   Set
    If Me.location <> value Then
     Me.location = value
     Me.OnPropertyChanged("Location")
    End If
   End Set
  End Property
  #region INotifyPropertyChanged Members
  Public Event PropertyChanged As PropertyChangedEventHandler
```

```
  Protected Overridable Sub OnPropertyChanged(propertyName As String)
   If Me.PropertyChanged <> Nothing Then
    Me.PropertyChanged(Me, New PropertyChangedEventArgs(propertyName))
   End If
  End Sub
#End Region
 End Class
End Namespace
```

[C#] The Sample Appointment Class

```
using System;
using System.ComponentModel;
namespace BusinessObjects
{
 public class MyAppointment : INotifyPropertyChanged
 {
   private DateTime start = DateTime.Now;
   private DateTime end = DateTime.Now;
   private string subject = string.Empty;
   private string description = string.Empty;
   private string location = string.Empty;
   private Guid id = Guid.NewGuid();
   public MyAppointment()
   {
   }
   public MyAppointment(DateTime start, DateTime end, string subject, string description,
string location)
   {
     this.start = start;
     this.end = end;
     this.subject = subject;
     this.description = description;
     this.location = location;
   }
   public Guid Id
   {
     get
     {
       return this.id;
     }
     set
     {
       if (this.id != value)
       {
         this.id = value;
         this.OnPropertyChanged("Id");
       }
     }
   }
   public DateTime Start
   {
     get
     {
       return this.start;
     }
```

```
    set
    {
      if (this.start != value)
      {
        this.start = value;
        this.OnPropertyChanged("Start");
      }
    }
  }
  public DateTime End
  {
    get
    {
      return this.end;
    }
    set
    {
      if (this.end != value)
      {
        this.end = value;
        this.OnPropertyChanged("End");
      }
    }
  }
  public string Subject
  {
    get
    {
      return this.subject;
    }
    set
    {
      if (this.subject != value)
      {
        this.subject = value;
        this.OnPropertyChanged("Subject");
      }
    }
  }
  public string Description
  {
    get
    {
      return this.description;
    }
    set
    {
      if (this.description != value)
      {
        this.description = value;
        this.OnPropertyChanged("Description");
      }
    }
  }
  public string Location
```

```
    {
      get
      {
        return this.location;
      }
      set
      {
        if (this.location != value)
        {
          this.location = value;
          this.OnPropertyChanged("Location");
        }
      }
    }
    #region INotifyPropertyChanged Members
    public event PropertyChangedEventHandler PropertyChanged;
    protected virtual void OnPropertyChanged(string propertyName)
    {
      if (this.PropertyChanged != null)
      {
        this.PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
      }
    }
    #endregion
  }
}
```

To use your custom object, create MyAppointment instances and place them in a generic list before mapping and binding to the SchedulerBindingDataSource component.

[VB] Binding to a List of Business Objects

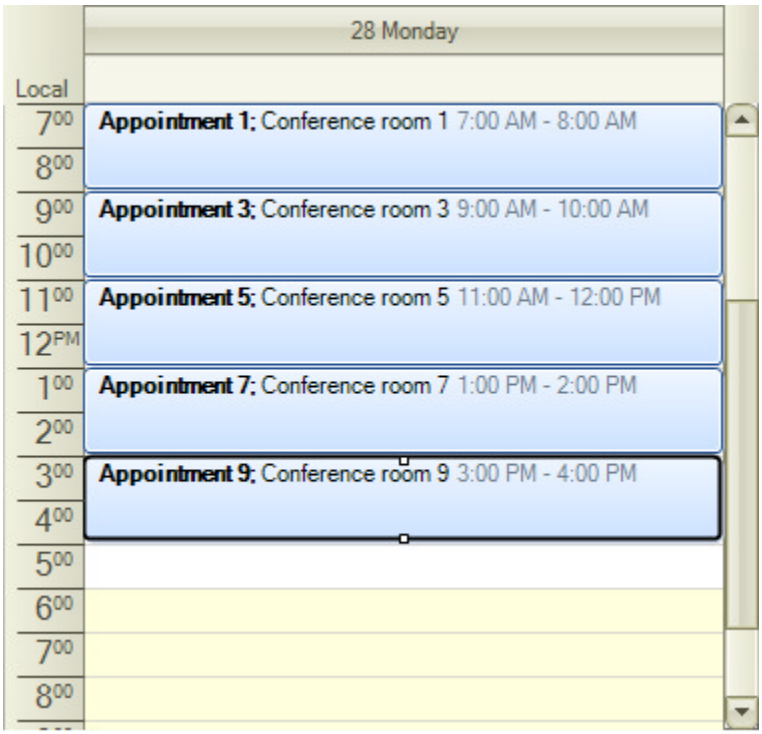
```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 ' create a list of MyAppointment objects
 Dim appointments As New BindingList(Of MyAppointment)()
 Dim i As Integer = 0
 While i < 10
  ' add every other appointment, populate with sample data
  If (i Mod 2) = 0 Then
   Dim appointmentNumber As Integer = i + 1
   Dim myAppointment As New MyAppointment(DateTime.Now.AddHours(appointmentNumber),
DateTime.Now.AddHours(appointmentNumber + 2), "Appointment " + appointmentNumber.ToString(),
"Description for Appointment " + appointmentNumber.ToString(), "Conference room " +
appointmentNumber.ToString())
   appointments.Add(myAppointment)
  End If
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
 ' create and configure a scheduler binding source
 Dim dataSource As New SchedulerBindingDataSource()
 ' map the MyAppointment properties to the scheduler
 Dim appointmentMappingInfo As New AppointmentMappingInfo()
 appointmentMappingInfo.Start = "Start"
 appointmentMappingInfo.[End] = "End"
 appointmentMappingInfo.Summary = "Subject"
 appointmentMappingInfo.Description = "Description"
 appointmentMappingInfo.Location = "Location"
```

```
 appointmentMappingInfo.UniqueId = "Id"
 dataSource.EventProvider.Mapping = appointmentMappingInfo
 ' assign the generic List of MyAppointment as the EventProvider data source
 dataSource.EventProvider.DataSource = appointments
 Me.radScheduler1.DataSource = dataSource
End Sub
```

[C#] Binding to a List of Business Objects

```
private void RadForm1_Load(object sender, EventArgs e)
{
 // create a list of MyAppointment objects
 BindingList<MyAppointment> appointments = new BindingList<MyAppointment>();
 for (int i = 0; i < 10; i++)
 {
   // add every other appointment, populate with sample data
   if ((i % 2) == 0)
   {
     int appointmentNumber = i + 1;
     MyAppointment myAppointment =
       new MyAppointment(
         DateTime.Now.AddHours(appointmentNumber),
         DateTime.Now.AddHours(appointmentNumber + 2),
         "Appointment " + appointmentNumber.ToString(),
         "Description for Appointment " + appointmentNumber.ToString(),
         "Conference room " + appointmentNumber.ToString());
     appointments.Add(myAppointment);
   }
 }
 // create and configure a scheduler binding source
 SchedulerBindingDataSource dataSource = new SchedulerBindingDataSource();
 // map the MyAppointment properties to the scheduler
 AppointmentMappingInfo appointmentMappingInfo = new AppointmentMappingInfo();
 appointmentMappingInfo.Start = "Start";
 appointmentMappingInfo.End = "End";
 appointmentMappingInfo.Summary = "Subject";
 appointmentMappingInfo.Description = "Description";
 appointmentMappingInfo.Location = "Location";
 appointmentMappingInfo.UniqueId = "Id";
 dataSource.EventProvider.Mapping = appointmentMappingInfo;
 // assign the generic List of MyAppointment as the EventProvider data source
 dataSource.EventProvider.DataSource = appointments;
 this.radScheduler1.DataSource = dataSource;
}
```

When the application is run, a series of MyAppointment objects show up in the scheduler.



Recurring Appointments

Create specialized **RecurrenceRule** descendants to implement recurrence for specific intervals:

- MinutelyRecurrenceRule
- HourlyRecurrenceRule
- DailyRecurrenceRule
- WeeklyRecurrenceRule
- MonthlyRecurrenceRule
- YearlyRecurrenceRule

One of several constructor overloads lets you set the start time, duration and number of occurences. Then the rule can be assigned to the appointments **RecurrenceRule** property. The snippet below defines a rule that starts "now" and recurs every two hours and stops after the tenth occurrence.

[VB] Adding a RecurrenceRule

```
radScheduler1.Appointments(0).RecurrenceRule = New HourlyRecurrenceRule(DateTime.Now, 2, 10)
```

[C#] Adding a RecurrenceRule

```
radScheduler1.Appointments[0].RecurrenceRule =
 new HourlyRecurrenceRule(DateTime.Now, 2, 10);
```

The Appointment **Occurrences** property lets you iterate a list of **IEvent** instances. To get only some occurrences between specific starting and stopping times, use the Appointment **GetOccurrences()** method.

[VB] Iterating Occurrences

```
' iterate all appointment occurrances
  '...
For Each ev As IEvent In recurringAppointment.Occurrences
Next
' iterate only occurrances after 10am
```

```
Dim occurrencesAfter10AM As IEnumerable(Of IEvent) = recurringAppointment.GetOccurrences(New
DateTime(2008, 10, 1, 10, 0, 0), DateTime.Now)
  '...
For Each ev As IEvent In occurrencesAfter10AM
Next
```

[C#] Iterating Occurrences

```
// iterate all appointment occurrances
foreach (IEvent ev in recurringAppointment.Occurrences)
{
 //...
}
// iterate only occurrances after 10am
IEnumerable<IEvent> occurrencesAfter10AM = recurringAppointment.GetOccurrences(
 new DateTime(2011, 10, 1, 10, 0, 0), DateTime.Now);
foreach (IEvent ev in occurrencesAfter10AM)
{
 //...
}
```

When the user changes a specific occurrence and not the entire series, an "Exception" is created. "Exceptions" in this context refer to "Exceptions to a rule", not the .NET Exception class related to error handling. You can create exceptions programmatically by adding to the IEvent **MasterEvent.Exceptions** collection. The snippet below changes the background and status of an IEvent instance and adds the IEvent to its own MasterEvent Exceptions collection.

[VB] Adding Rule Exceptions

```
myEvent.BackgroundId = DirectCast(AppointmentBackground.Important, Integer)
myEvent.StatusId = DirectCast(AppointmentStatus.Tentative, Integer)
myEvent.MasterEvent.Exceptions.Add(myEvent)
```

[C#] Adding Rule Exceptions

```
myEvent.BackgroundId = (int)AppointmentBackground.Important;
myEvent.StatusId = (int)AppointmentStatus.Tentative;
myEvent.MasterEvent.Exceptions.Add(myEvent);
```

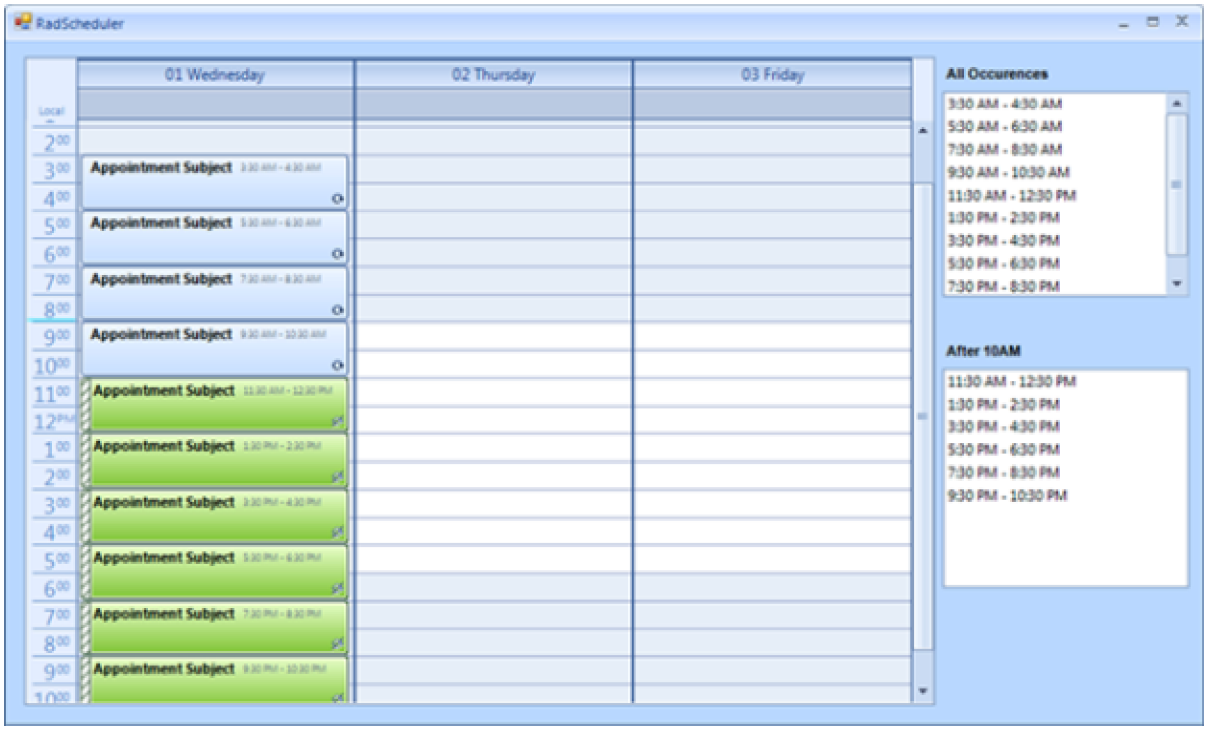
Recurrence Rule Walkthrough

This example will create a single appointment, then define a recurrence rule that occurs every two hours for ten occurrences. In the example you will change the background and status for a subset of appointments that occur after 10AM.



You can find the complete source for this project at:

\Scheduler\<VB|CS>\Recurring



1. In a new application, add a RadScheduler and two RadListControls to the form. Place the RadScheduler on the left half of the form and the two RadListControls on the right half of the form, one above the other. Name the first list box "lcAll" and the second "lcAfter10".
2. Add the code below to the form's Load Event handler:

[VB] Adding and Traversing Appointment Recurrences

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 Dim startDate As New DateTime(2008, 10, 1, 3, 30, 0)
 ' Create a sample appointment that starts at 10/1/2008 3:30 AM and lasts half an hour.
 Dim recurringAppointment As New Appointment(startDate, TimeSpan.FromHours(1), "Appointme
Subject")
 ' Create a recurrence rule to repeat the appointment every 2 hours for 10 occurrences.
 Dim rrule As New HourlyRecurrenceRule(recurringAppointment.Start, 2, 10)
 'Assign the hourly recurrence rule to the appointment
 recurringAppointment.RecurrenceRule = rrule
 radScheduler1.Appointments.Add(recurringAppointment)
 ' iterate all appointment occurrances
 For Each ev As IEvent In recurringAppointment.Occurrences
  lbAll.Items.Add(New RadListBoxItem(ev.Start.ToShortTimeString() + " - " + ev.
[End].ToShortTimeString()))
 Next
 ' iterate only occurrances after 10am
 Dim occurrencesAfter10AM As IEnumerable(Of IEvent) = recurringAppointment.GetOccurrences
(New DateTime(2008, 10, 1, 10, 0, 0), DateTime.Now)
 For Each ev As IEvent In occurrencesAfter10AM
  lbAfter10.Items.Add(New RadListBoxItem(ev.Start.ToShortTimeString() + " - " + ev.
[End].ToShortTimeString()))
  ' set the background id to "Important" and make this occurence an "Exception"
  ev.BackgroundId = DirectCast(AppointmentBackground.Important, Integer)
  ev.StatusId = DirectCast(AppointmentStatus.Tentative, Integer)
```

```
  ev.MasterEvent.Exceptions.Add(ev)
 Next
 radScheduler1.FocusedDate = startDate
End Sub
```

[C#] Adding and Traversing Appointment Recurrences

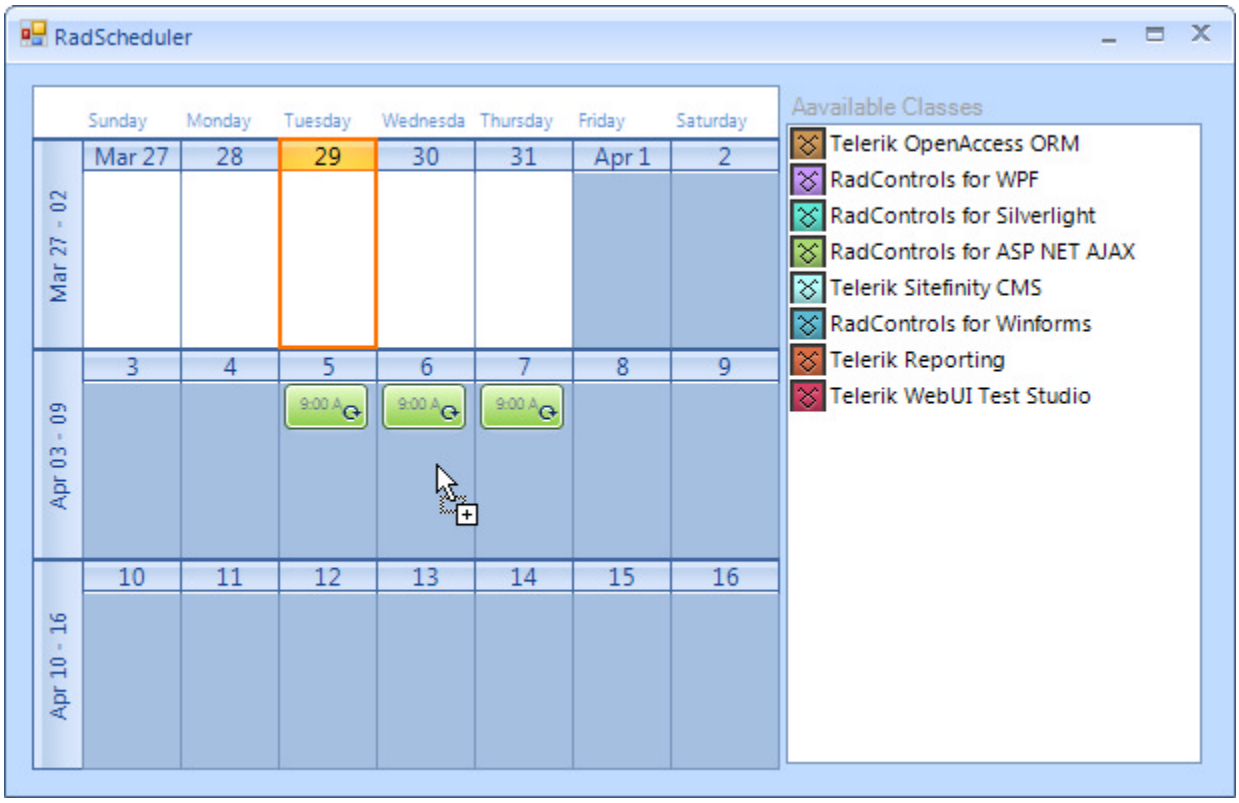
```
private void RadForm1_Load(object sender, EventArgs e)
{
 DateTime startDate = new DateTime(2008, 10, 1, 3, 30, 0);
 // Create a sample appointment that starts at 10/1/2008 3:30 AM and lasts half an hour.
 Appointment recurringAppointment = new Appointment(startDate,
   TimeSpan.FromHours(1.0), "Appointment Subject");
 // Create a recurrence rule to repeat the appointment every 2 hours for 10 occurrences.

 HourlyRecurrenceRule rrule =
   new HourlyRecurrenceRule(recurringAppointment.Start, 2, 10);
 //Assign the hourly recurrence rule to the appointment
 recurringAppointment.RecurrenceRule = rrule;
 radScheduler1.Appointments.Add(recurringAppointment);
 // iterate all appointment occurrances
 foreach (IEvent ev in recurringAppointment.Occurrences)
 {
   lbAll.Items.Add(
     new RadListBoxItem(ev.Start.ToShortTimeString() + " - " +
       ev.End.ToShortTimeString()));
 }
 // iterate only occurrances after 10am
 IEnumerable<IEvent> occurrencesAfter10AM = recurringAppointment.GetOccurrences(
   new DateTime(2008, 10, 1, 10, 0, 0), DateTime.Now);
 foreach (IEvent ev in occurrencesAfter10AM)
 {
   lbAfter10.Items.Add(
     new RadListBoxItem(ev.Start.ToShortTimeString() + " - " +
       ev.End.ToShortTimeString()));
   // set the background id to "Important" and make this occurence an "Exception"
   ev.BackgroundId = (int)AppointmentBackground.Important;
   ev.StatusId = (int)AppointmentStatus.Tentative;
   ev.MasterEvent.Exceptions.Add(ev);
 }
 radScheduler1.FocusedDate = startDate;
}
```

3. Run the application. Notice that the background and status for appointments after 10am are changed to reflect changes made to members of the collection returned by GetOccurrences().

Drag And Drop

Dragging appointment subjects from a RadListBox uses similar techniques as explained in the list boxes chapter. Here we'll talk about the portion that differs, i.e. when the item is dropped we will need to find the scheduler cell. Once you have the scheduler cell you can get the date and create an appointment for it. In this example we also make the appointment recur for three days.





You can find the complete source for this project at:

\Scheduler\<VB|CS>\DragAndDrop

The first part that differs is the RadScheduler DragDrop event handler. Here you need to get the location of the mouse and convert it to a point that the scheduler can use to get the cell element underneath the mouse. This MonthCellElement is passed to a private method GetCellAppointment() that we will write next.

[VB] Drop to the Month Cell Element

```
Private Sub radScheduler1_DragDrop(ByVal sender As Object, ByVal e As DragEventArgs) Handles
radScheduler1.DragDrop
Dim scheduler As RadScheduler = TryCast(sender, RadScheduler)
' get mouse location on the screen
Dim mouseLocation As System.Drawing.Point = New System.Drawing.Point(e.X, e.Y)
' convert to location on the sceduler
Dim schedulerLocation As System.Drawing.Point = scheduler.SchedulerElement.PointFromScreen
(mouseLocation)
' get the cell element for the location
Dim element As RadElement = scheduler.ElementTree.GetElementAtPoint(schedulerLocation)
' cast cell element to MonthCellElement and create an appointment for cell date
If (Not element Is Nothing) AndAlso (TypeOf element Is MonthCellElement) Then
Dim monthCellElement As MonthCellElement = TryCast(element, MonthCellElement)
Dim appointment As Appointment = GetCellAppointment(monthCellElement)
scheduler.Appointments.Add(appointment)
End If
' finished drop, clear temporary variables
_dragRectangle = Rectangle.Empty
_currentItem = Nothing
```

```
End Sub
```

[C#] Drop to the Month Cell Element

```
private void radScheduler1_DragDrop(object sender, DragEventArgs e)
{
RadScheduler scheduler = sender as RadScheduler;
// get mouse location on the screen
Point mouseLocation = new Point(e.X, e.Y);
// convert to location on the sceduler
Point schedulerLocation = scheduler.SchedulerElement.PointFromScreen(mouseLocation);
// get the cell element for the location
RadElement element = scheduler.ElementTree.GetElementAtPoint(schedulerLocation);
// cast cell element to MonthCellElement and create an appointment for cell date
if ((element != null) && (element is MonthCellElement))
{
MonthCellElement monthCellElement = element as MonthCellElement;
Appointment appointment = GetCellAppointment(monthCellElement);
scheduler.Appointments.Add(appointment);
}
// finished drop, clear temporary variables
_dragRectangle = Rectangle.Empty;
_currentItem = null;
}
```

The helper method GetCellAppointment() takes a MonthCellElement and creates an appointment using the MonthCellElement Date. The appointment gets its Summary property content from the dragged RadListBox item. The method also assigns a DailyRecurrenceRule that runs for three, eight hour days.

```
Private Function GetCellAppointment(ByVal monthCellElement As MonthCellElement) As Appointment
Dim appointment As Appointment = New Appointment()
' start at 9:00AM on the drop target cell date
appointment.Start = monthCellElement.Date.AddHours(9)
' class is 8 hours long
appointment.Duration = TimeSpan.FromHours(8)
' copy the listbox item itext as the summary
appointment.Summary = _currentItem.Text
' fill other misc properties
appointment.Location = "Telerik Training Center"
appointment.BackgroundId = AppointmentBackground.Important
appointment.StatusId = AppointmentStatus.Busy
' the class will run for three, eight hour days
appointment.RecurrenceRule = New DailyRecurrenceRule(appointment.Start, 1, 3)
Return appointment
End Function
```

[C#] Creating an Appointment for the MonthCellElement

```
private Appointment GetCellAppointment(MonthCellElement monthCellElement)
{
Appointment appointment = new Appointment();
// start at 9:00AM on the drop target cell date
```

```
appointment.Start = monthCellElement.Date.AddHours(9);
// class is 8 hours long
appointment.Duration = TimeSpan.FromHours(8);
// copy the list control item itext as the summary
appointment.Summary = _currentItem.Text;
// fill other misc properties
appointment.Location = "Telerik Training Center";
appointment.BackgroundId = (int)AppointmentBackground.Important;
appointment.StatusId = (int)AppointmentStatus.Busy;
// the class will run for three, eight hour days
appointment.RecurrenceRule = new DailyRecurrenceRule(appointment.Start, 1, 3);
return appointment;
}
```

19.6 Summary

This chapter explored RadScheduler capabilities: how to bind data to the scheduler, add, remove and iterate appointments, work with resources, modify the view (i.e. week, day, month) and manipulate appointment status. The chapter also demonstrated how to drag and drop appointments into the scheduler.

20 Chart

20.1 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.

20.2 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.

20.3 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:

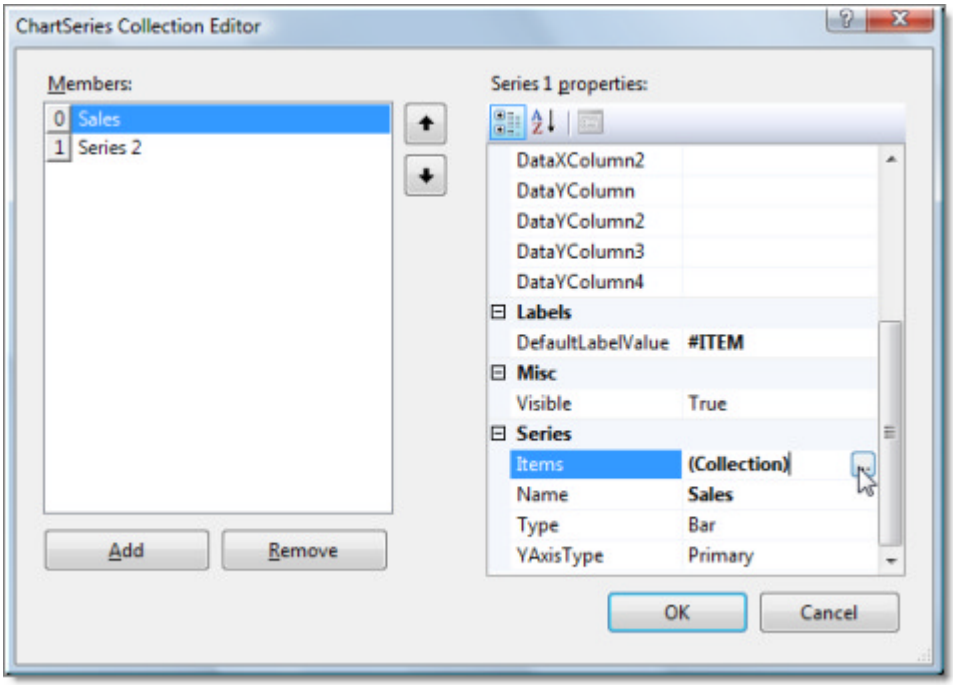
\Chart\<VB|CS>\GettingStarted1

Prepare the Project

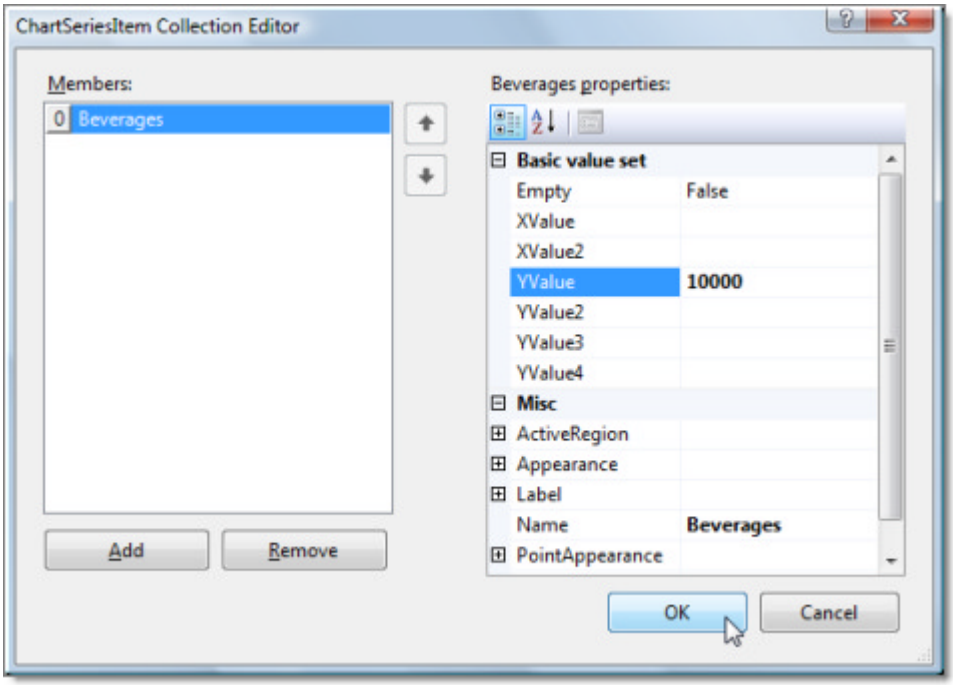
1. Create a new Windows Forms application.
2. On the default form, place a RadChart control.

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.

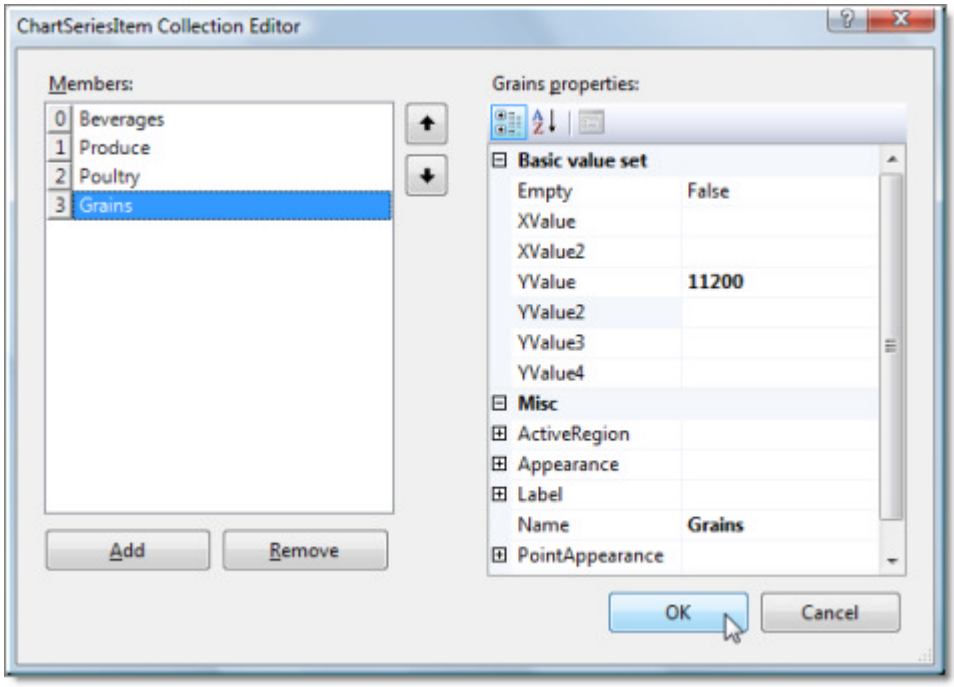


5. Click the ellipses button of the Items property to open the ChartSeriesItem Collection Editor.
6. Click the **Add** button to add a new Item.
7. In the property window for the new item, change the Name property to "Beverages".
8. Change the **YValue** property to "10000".

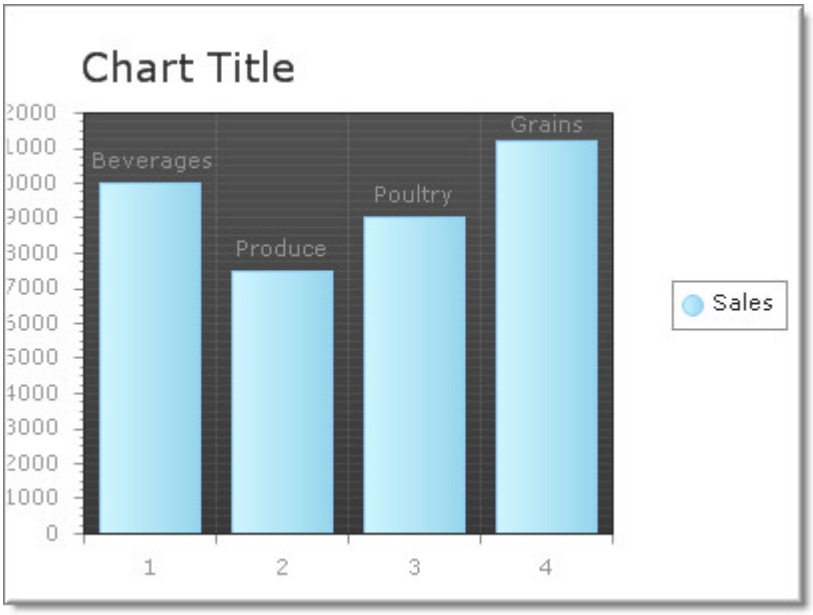


9. Repeat the Add Item steps to add 3 new items. Replace the properties for the three new items as follows:
   - Label: Produce, YValue: 7500
   - Label: Poultry, YValue: 9000

- Label: Grains, YValue: 11200



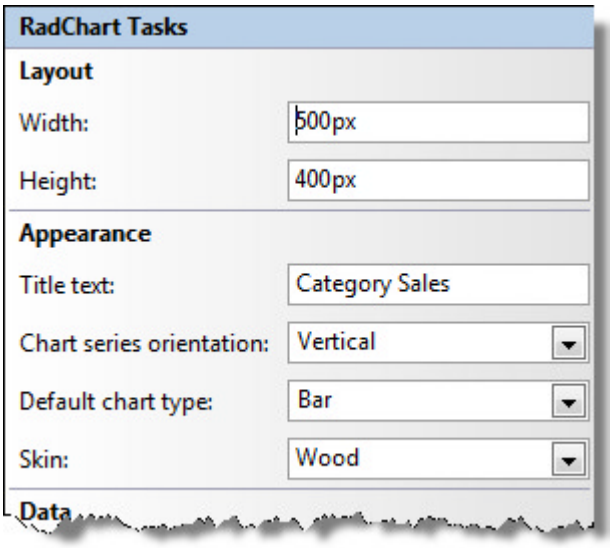
10. Click **OK** to close the ChartSeriesItem Collection Editor.
11. Click "Series 2" in the ChartSeries Collection Editor.
12. Click the **Remove** button to remove Series 2.
13. Click the **OK** button to close the ChartSeries Collection Editor.
14. The chart will display the new data using the default formatting.



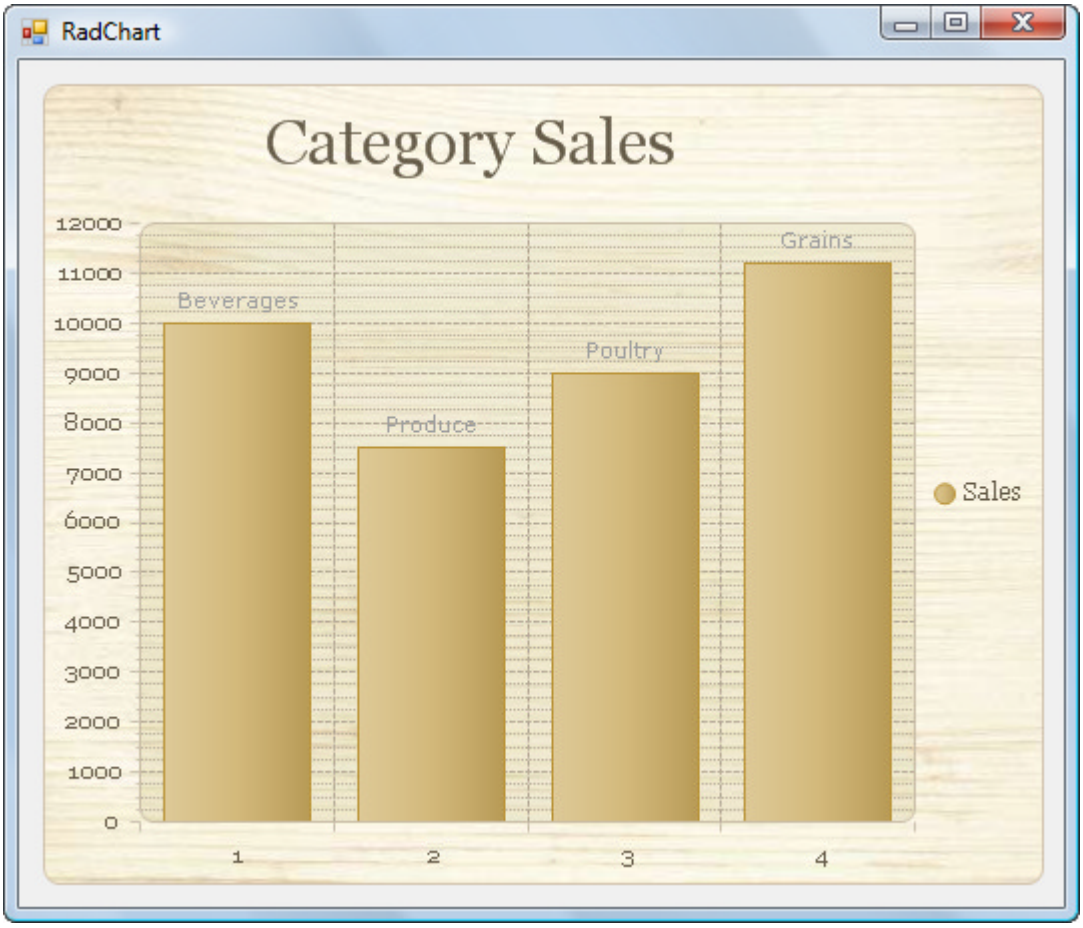
Format the Chart Using the SmartTag

1. Click the RadChart's Smart Tag

2. Change the Layout section **Width** to "500px" and **Height** to "400px".
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". *Notice that the AutoLayout feature of RadChart makes the best use of the form real-estate and expands the chart content as much as possible.*
5. Press **Ctl-F5** to run the application. 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:

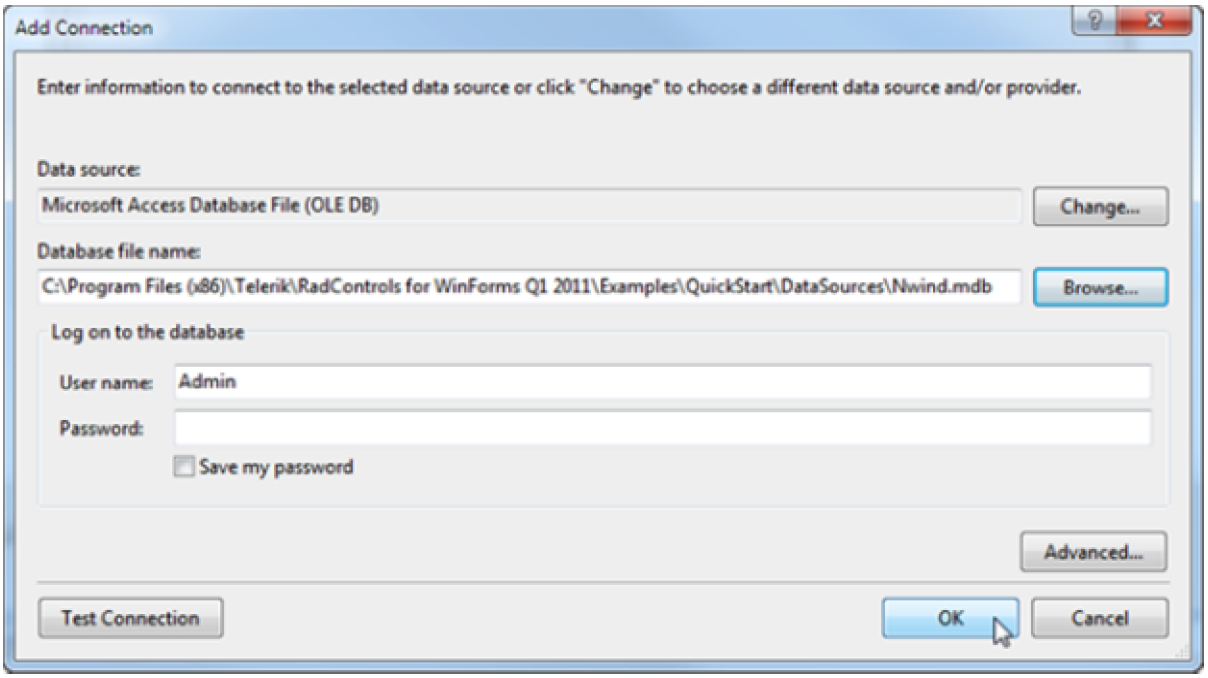
\Chart\<VB|CS>\GettingStarted2

Prepare the Project

1. Create a new Windows Forms application.
2. From the Toolbox drag a **RadChart** component to the default form.

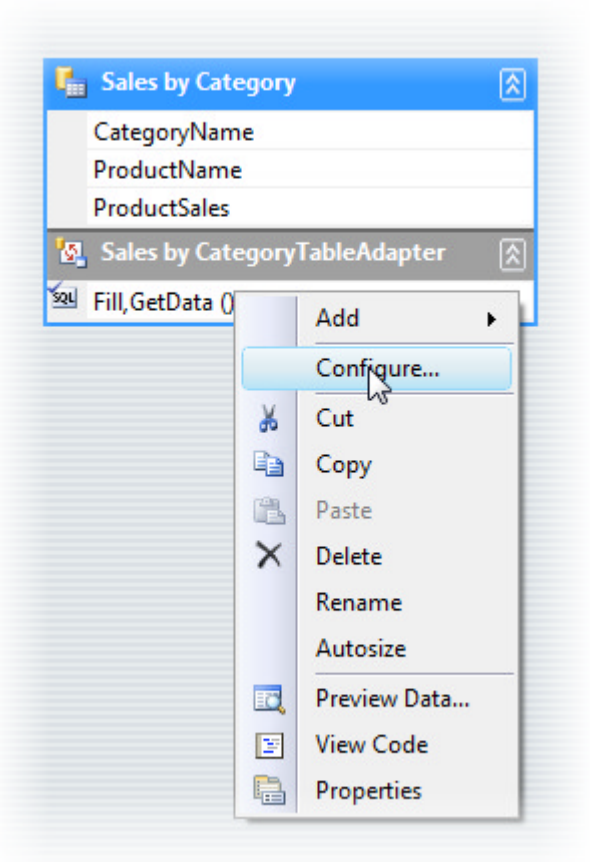
Configure the RadChart Using the Wizard

1. Configure the data source:
    - From the Smart Tag open the **Choose Data Source** drop down and select **Add Project Data Source...** from the list.
    - In the "Choose a Data Source Type" page of the wizard, select the Database icon, then click the **Next** button to continue.
    - In the "Choose your data connection" page of the wizard, click the **New Connection...** button. This step will bring up the Add Connection dialog.
    - In the Add Connection dialog, the Data Source entry should read "Microsoft Access Database File (OLE DB)" (see screenshot below). If it is not, click the **Change...** button and select "Microsoft Access Database File (OLE DB)" from the list.
    - Also in the Add Connection dialog, click the **Browse...** button, navigate to the RadControls for Winforms installation and locate the \Examples\QickStart\DataSources directory. Open the "Nwind.mdb" file.
    - Click **OK** to close the Add Connection dialog.



- Back at the "Choose Your Data Connection" page of the wizard, click the **Next** button to continue.
- You will receive a dialog popup with message "The connection you selected is a local file...". Click **Yes** to close the dialog and copy the data file locally.
- In the **Choose Your Database Objects** page, open the treeview "Views" node, locate the "Sales by Category" view and check the "CategoryName", "ProductName" and "ProductSales" columns.
- Click the **Finish** button to close the dialog. *This last step will create a DataSet, TableAdapter and BindingSource and place them in the components tray.*

2. Customize the data source query statement:
   - Locate the "sales_by_CategoryTableAdapter" component in the component tray. Open the Smart Tag and select **Edit Queries in DataSet Designer...**
   - Locate the "Fill, GetData()" element of the "Sales by CategoryTableAdapter", right-click and select **Configure...** from the drop down menu. This step will display the TableAdapter Configuration Wizard dialog.



   - In the TableAdapter Configuration Wizard "Enter a SQL Statement" page, enter the SQL below, then click **Finish** to close the dialog. *This query groups the sales figure by category and provides total sales for each group.*

     [SQL] Retrieve sales by category data

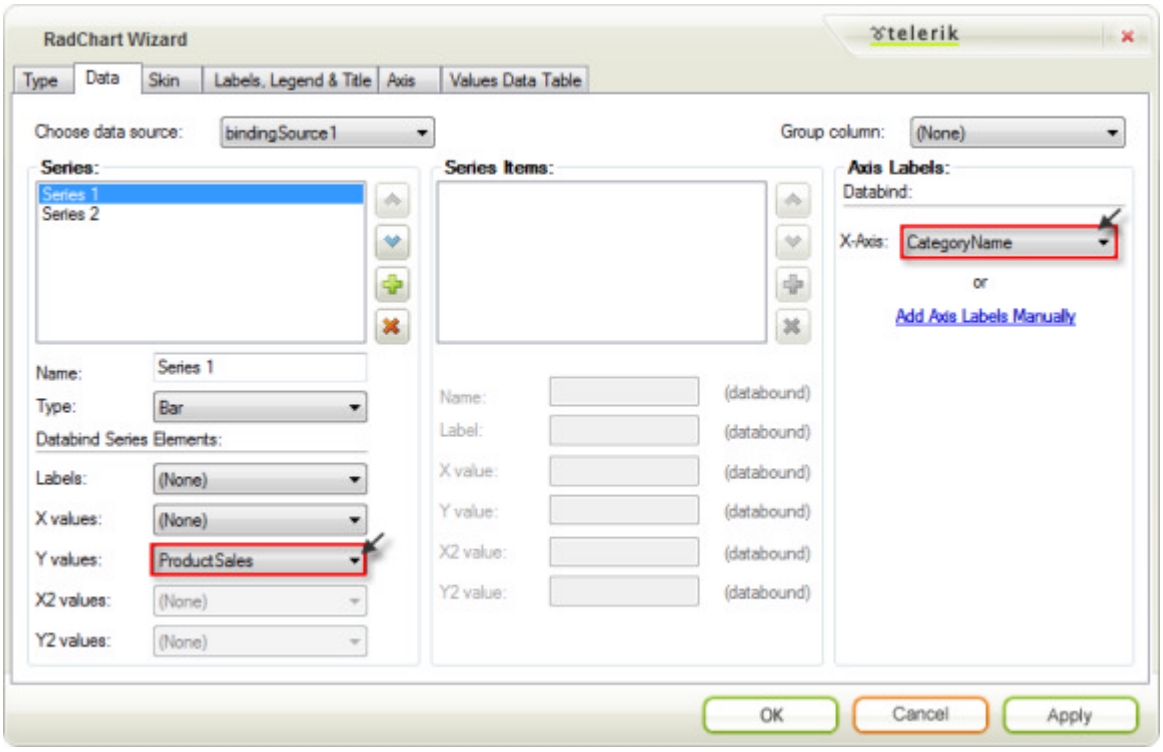
```
SELECT CategoryName, SUM(ProductSales) AS ProductSales
FROM [Sales by Category]
GROUP BY CategoryName
ORDER BY SUM(ProductSales) DESC
```

3. From the Smart Tag select the **Chart Wizard** link from the Setting section.

4. In the Type tab of the wizard, select the **Horizontal** orientation and the **Bar** chart type.



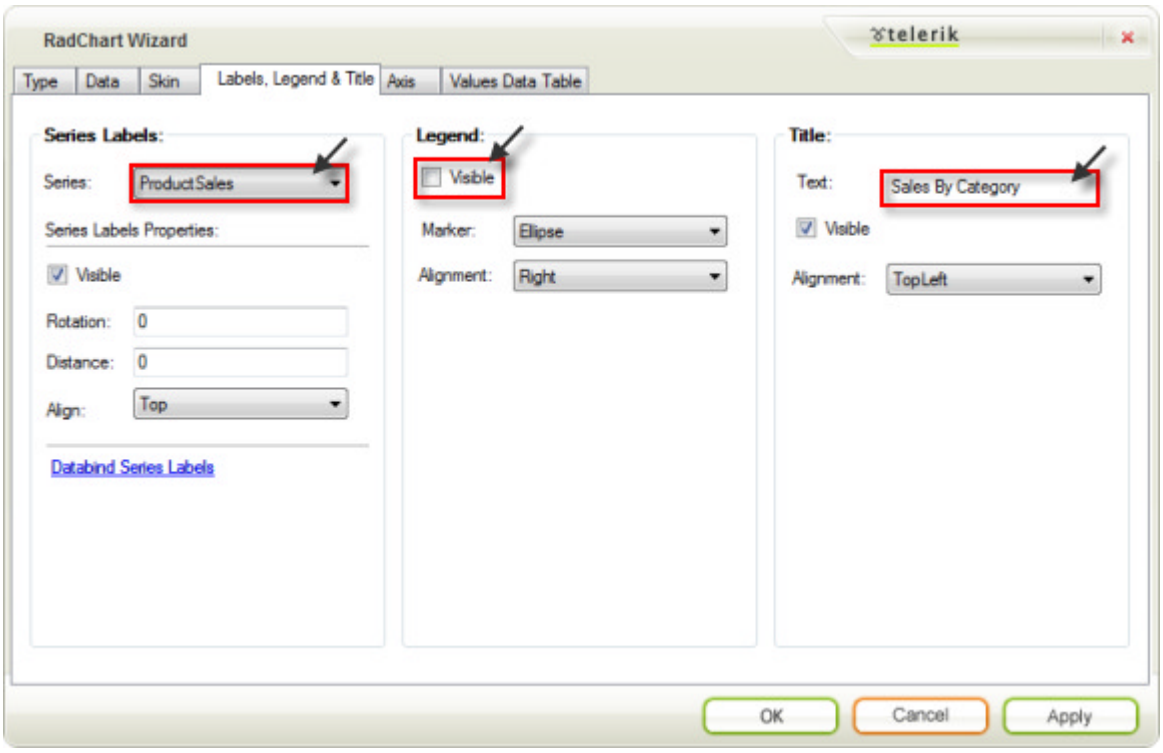
5. In the Data tab of the wizard set the **Y values** drop down list to "ProductSales" and the **X-Axis** to "CategoryName". *In a horizontal bar chart, the X-Axis will list the category names from top to bottom on the left hand side of the chart.*



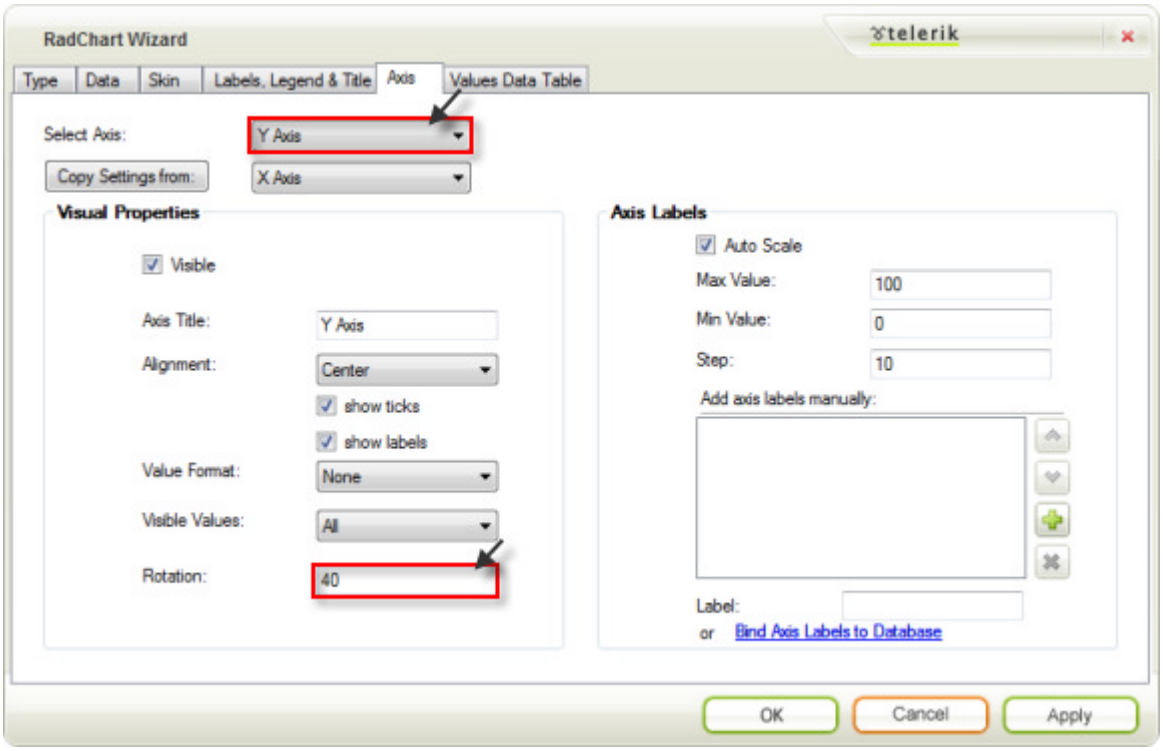
6. On the Skin tab of the wizard, select the DeepGreen skin.



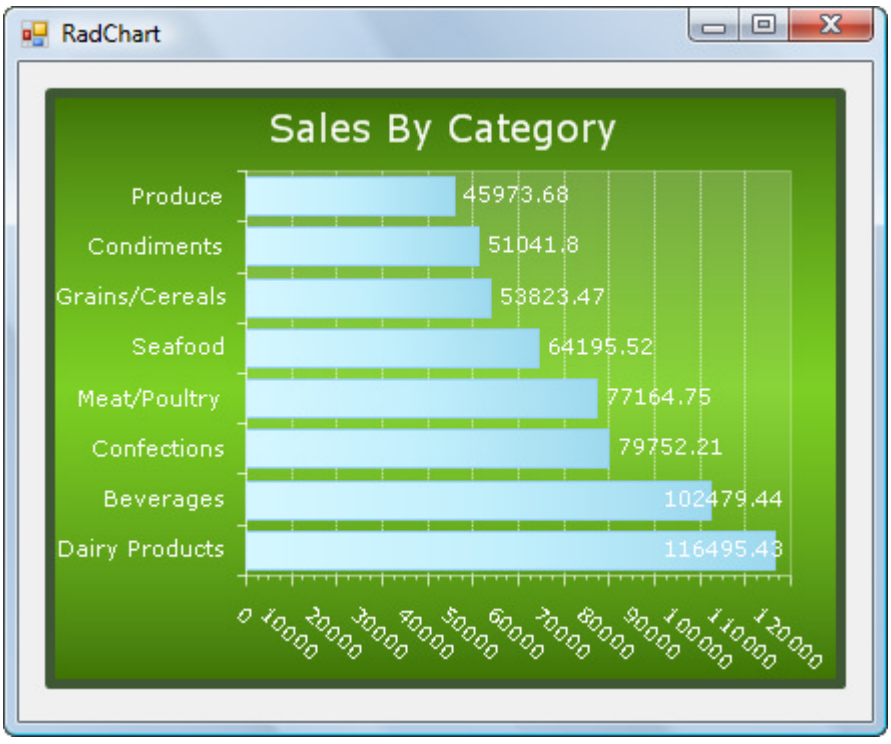
7. In the Labels, Legend & Title tab, set the **Series Labels** to "ProductSales" from the drop down list, de-select the Legend **Visible** check box. Set the **Title Text** to "Sales by Category"



8. In the Axis tab of the wizard, select the "Y Axis" from the drop down list. Then set the Rotation to "40". *This will spin the labels clockwise 40 degrees so that they will not overlap.*



9. Click the **OK** button to close the wizard.
10. In the Properties window set the RadCalendar **AutoLayout** property to "true".
11. Press **Ctl-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. Also notice that the labels at the bottom of the chart are spun 40 degrees clockwise to prevent overlap.



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.

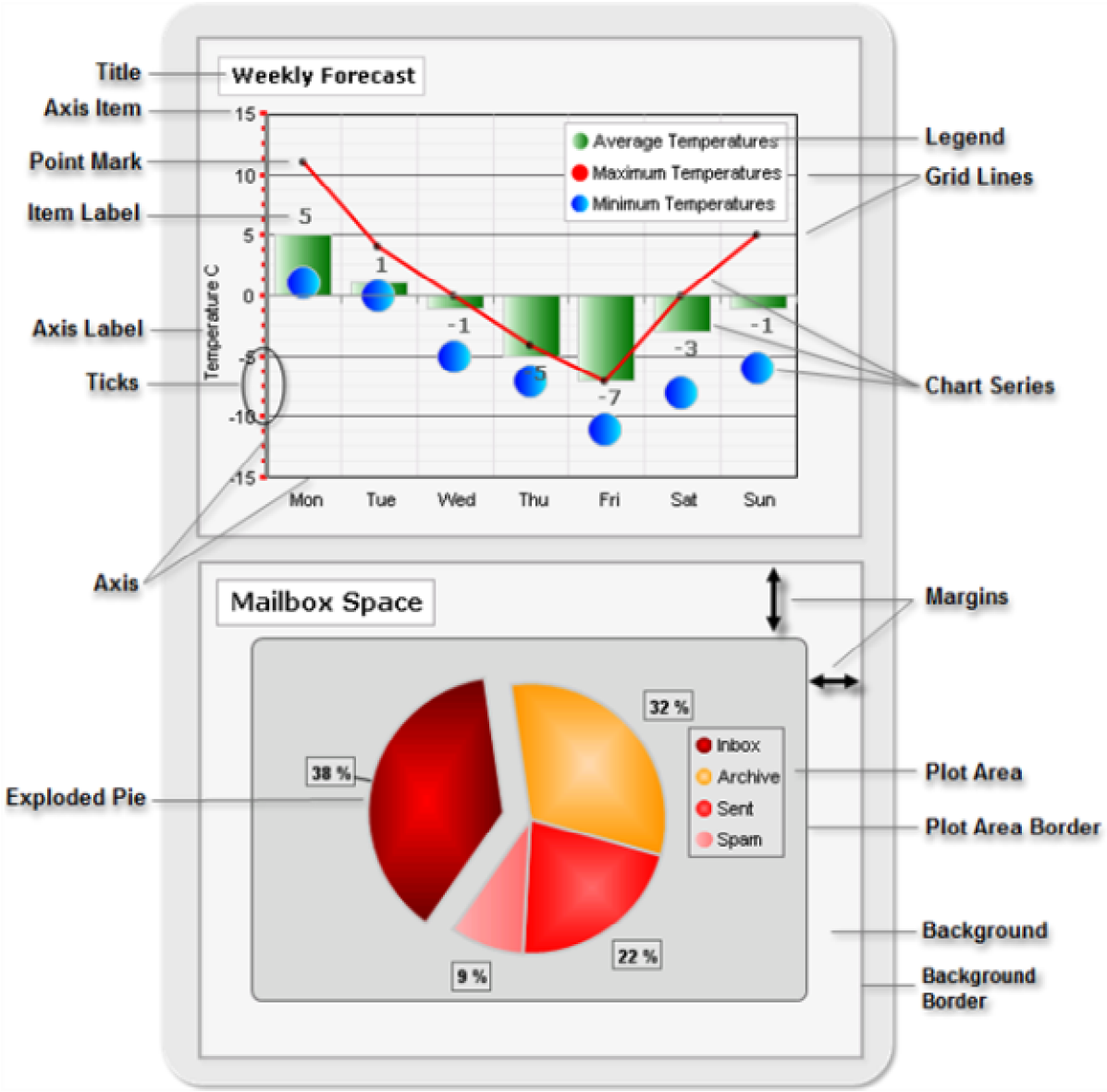


Chart Background

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.

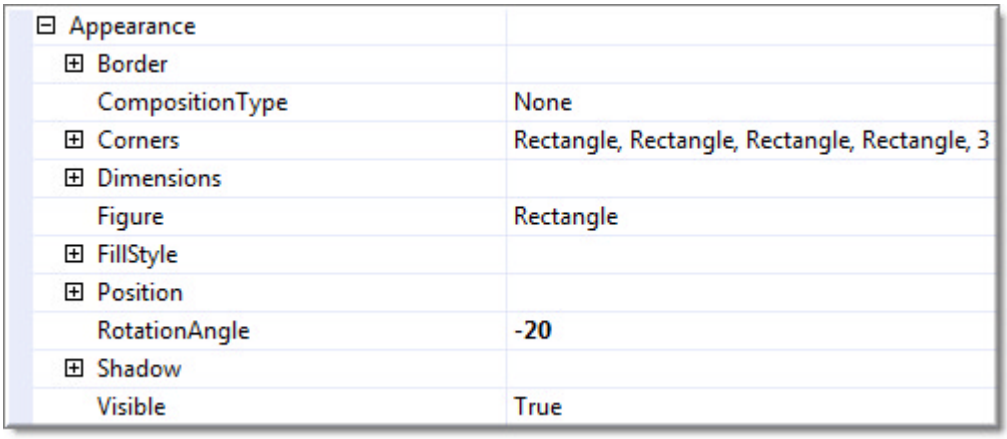
Title, Legend and Labeling

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 they show up in many aspects of the chart.

- **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.



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 Appearance property 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.

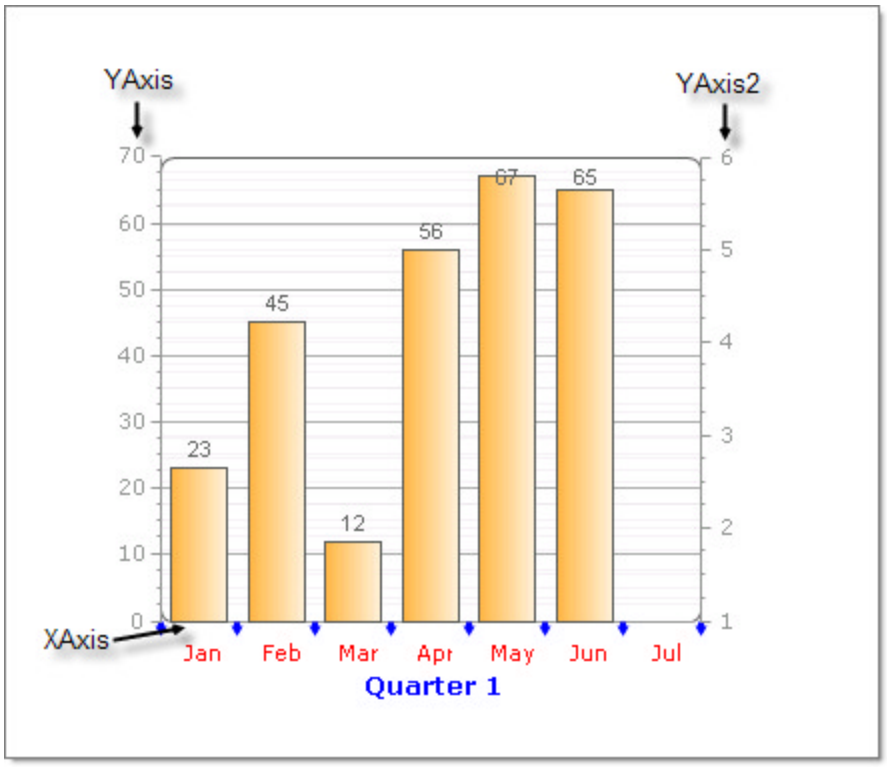
TextBlock	
Appearance	
AutoTextWrap	Auto
Border	
Color	**AliceBlue**
PenStyle	Solid
Visible	True
Width	1
Corners	Rectangle, Rectangle, Rectangle, Rectangle, 3
Dimensions	
FillStyle	
MaxLength	255
Position	
Shadow	
TextProperties	
Visible	True
Text	**Top 10 Product Categories**
Visible	True
Visible	True

The TextBlock.Appearance.Border property setting was applied to the ChartTitle to get this appearance:



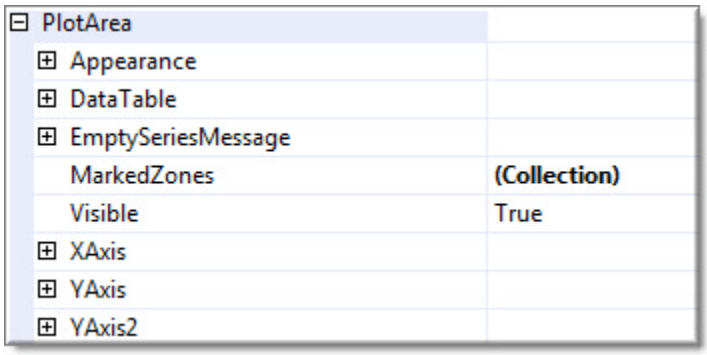
Axis

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.

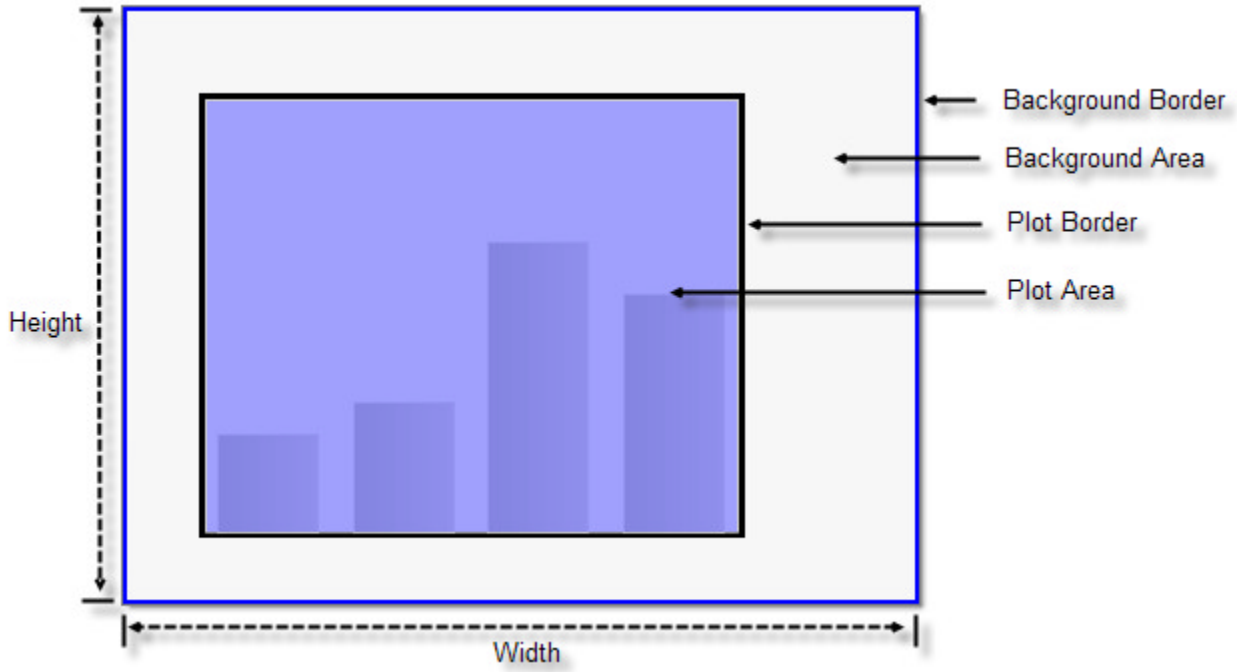


Plot Area

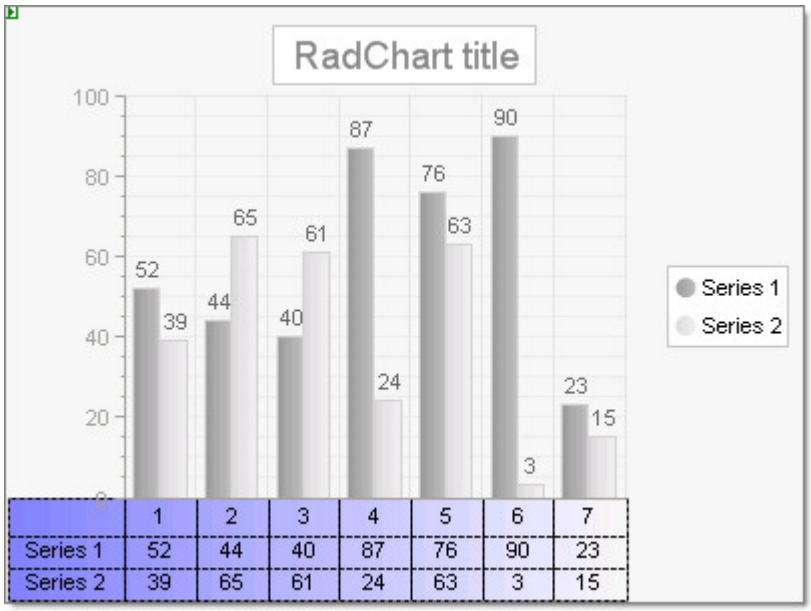
The plot area is the working rectangular area between X and Y axes where data is displayed. This property is a major jumping off point for configuring the axis of the chart.



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.

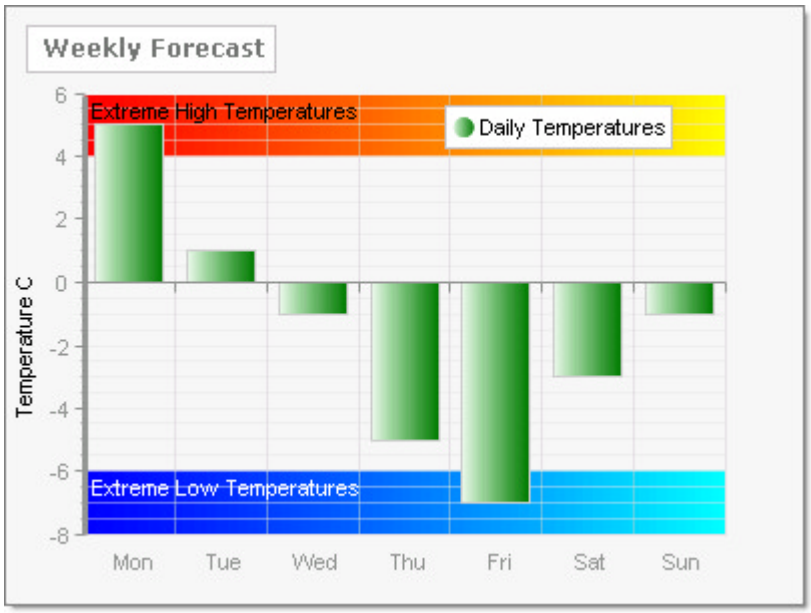
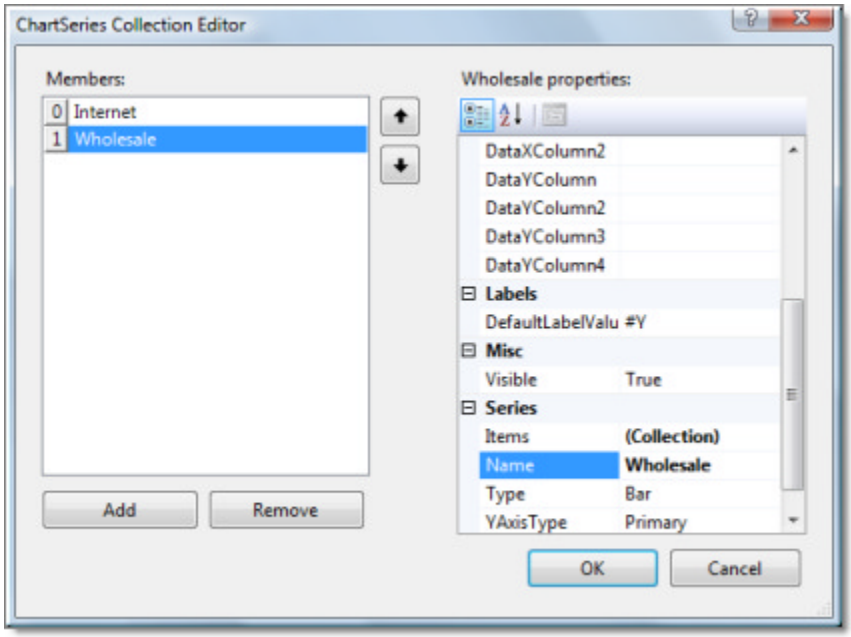


Chart Series

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.



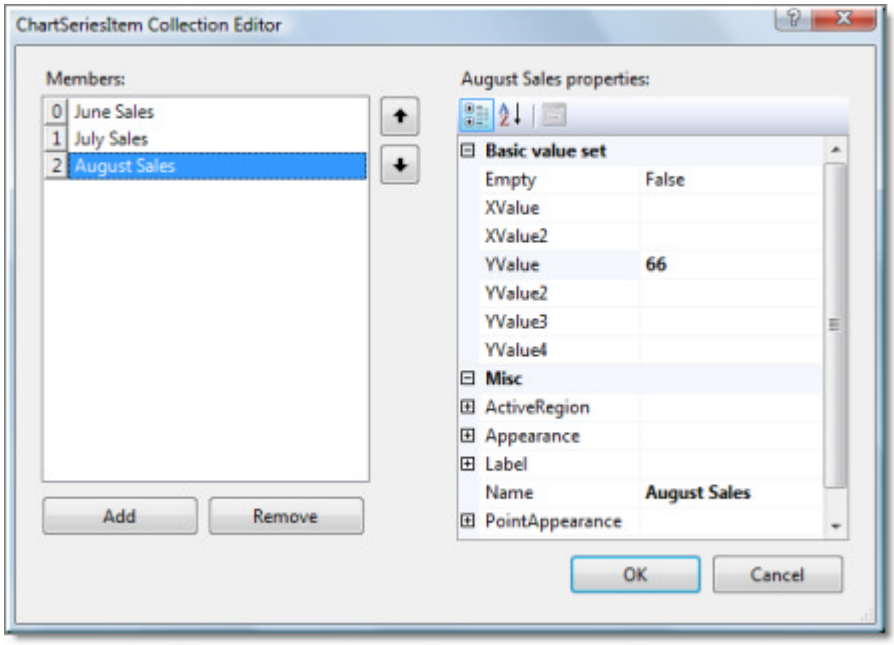
The **DefaultLabelValue** holds label formatting options for the series:

- 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%



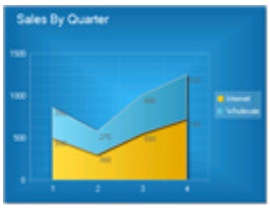
Stacked Bar 100% shows the combined contribution of values as percentages where the combined total for each category is 100 percent. Use when the relationship between values in a category is more significant than the amounts.

Area



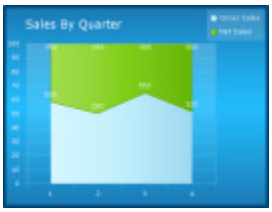
The Area chart consists of a series of data points joined by a line and where the area below the line is filled. Area charts are appropriate for visualizing data that fluctuates over a period of time and can be useful for emphasizing trends.

Stacked Area



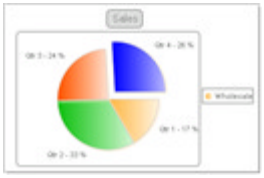
The Stacked Area chart is a variation of the Area chart that display trends of the contribution of each value over time (or across categories). The areas are stacked so that each series adjoins but does not overlap the preceding series. Area charts are appropriate for visualizing data that fluctuates over a period of time and where the entire area for all series data must be visible at one time.

Stacked Area 100%



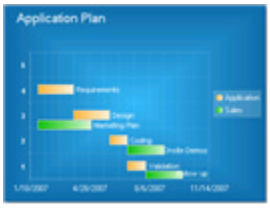
Stacked Area 100% charts are a variation of Stacked Area charts that present values for trends as percentages, totaling to 100% for each category. Use this chart type to visualize data that fluctuates over a period of time and where the relationship between values in a category is more significant than the amounts.

Pie



The Pie chart shows slices representing fractional parts of a whole.

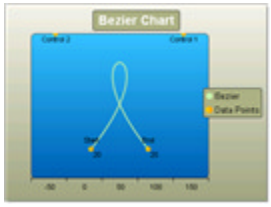
Gantt



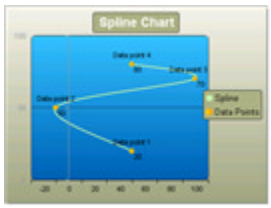
Gantt charts, also known as Time charts, display separate events as bars along a time scale. These charts are often used for project/time planning, where data can be plotted using a date-time scale or other numeric scale.

Bezier

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.

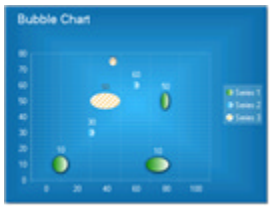


Spline



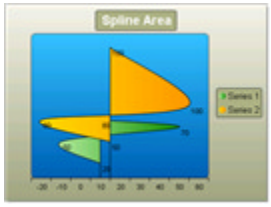
Spline charts allow you to take a limited set of known data points and approximate intervening values. The Spline chart, like the Bezier, is often used for data modelling by taking a limited number of data points and interpolating or estimating the intervening values.

Bubble



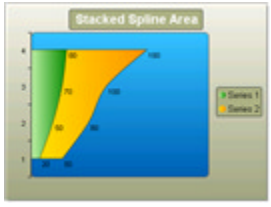
The Bubble chart is an extension of the Point chart but each point can be a circle or oval of any size or dimension. The bubble size may be used to convey larger values. The Bubble chart is often used for scientific data modeling or financial data.

Spline Area



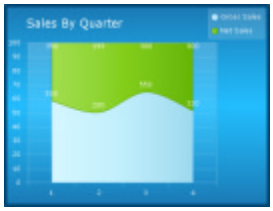
The Spline Area chart type defines one or more spline curves and fills in the area defined by the spline. This chart type can also can be used for data modelling in that it takes a limited number of data points and interpolates the intervening values.

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.

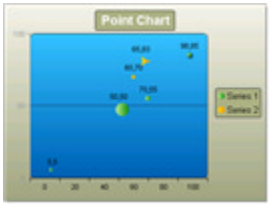
Stacked Spline Area 100%



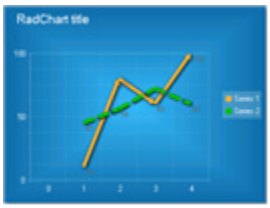
The Stacked Spline Area 100% chart is a variation of the Spline Area chart. The areas are stacked so that each series adjoins but does not overlap the preceding series and where the combined total for each category is 100 percent. The Stacked Spline Area 100% chart can also can be used for data modelling 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. Use this chart type when the relationship between values in a category is more significant than the amounts.

Point

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 used with time dependent data where a Line chart is more suited.



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



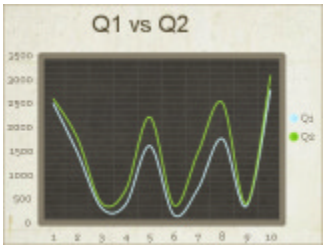
The CandleStick chart combines bar and line chart styles to show a range of value movement over time. Dark colored bars show downward trends, light colored bars show upward trends and the line through the center (the "wick") shows the extreme high and low values. Use this chart type to visualize price 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

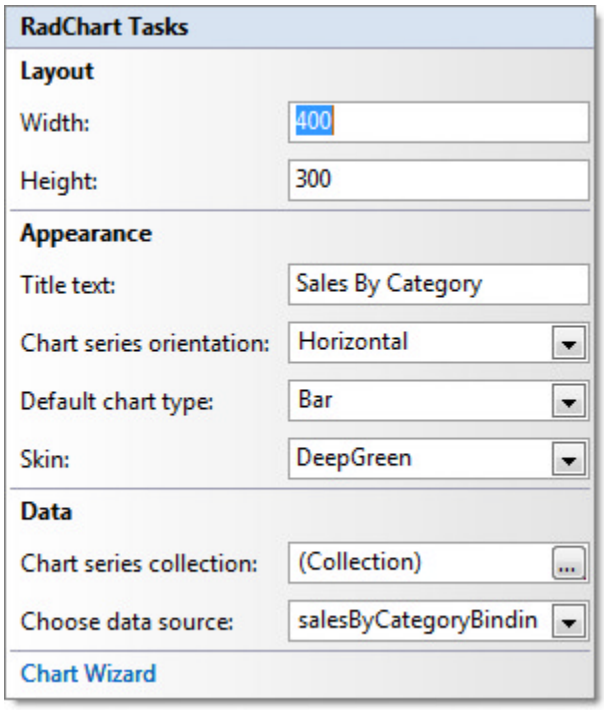


The Stacked Spline chart, line the Stacked Line, lets you have multiple series of Y values. It can take a limited number of data points and interpolate the intervening values.

20.4 Using the Design Time Interface

Smart Tag

The RadChart Smart Tag contains short cuts to frequently used functionality.



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.

Chart Wizard

Click this link to display the Chart Wizard.

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:

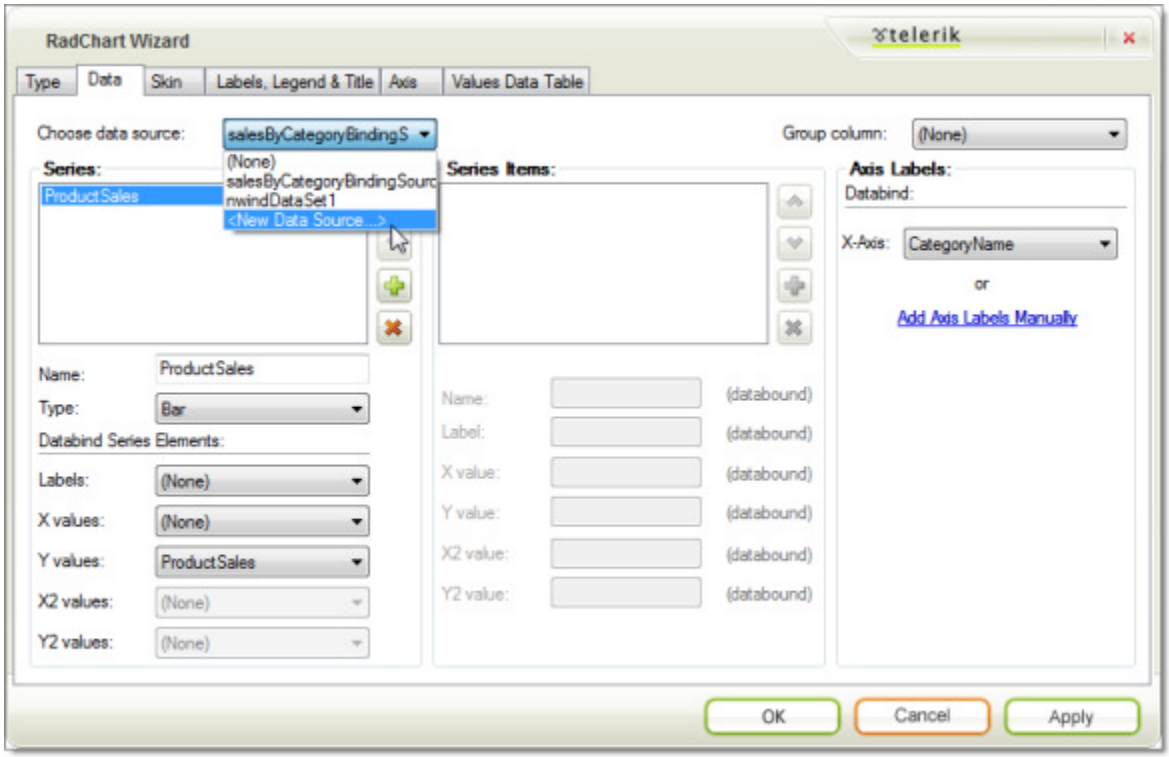
Type Tab

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.

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. 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

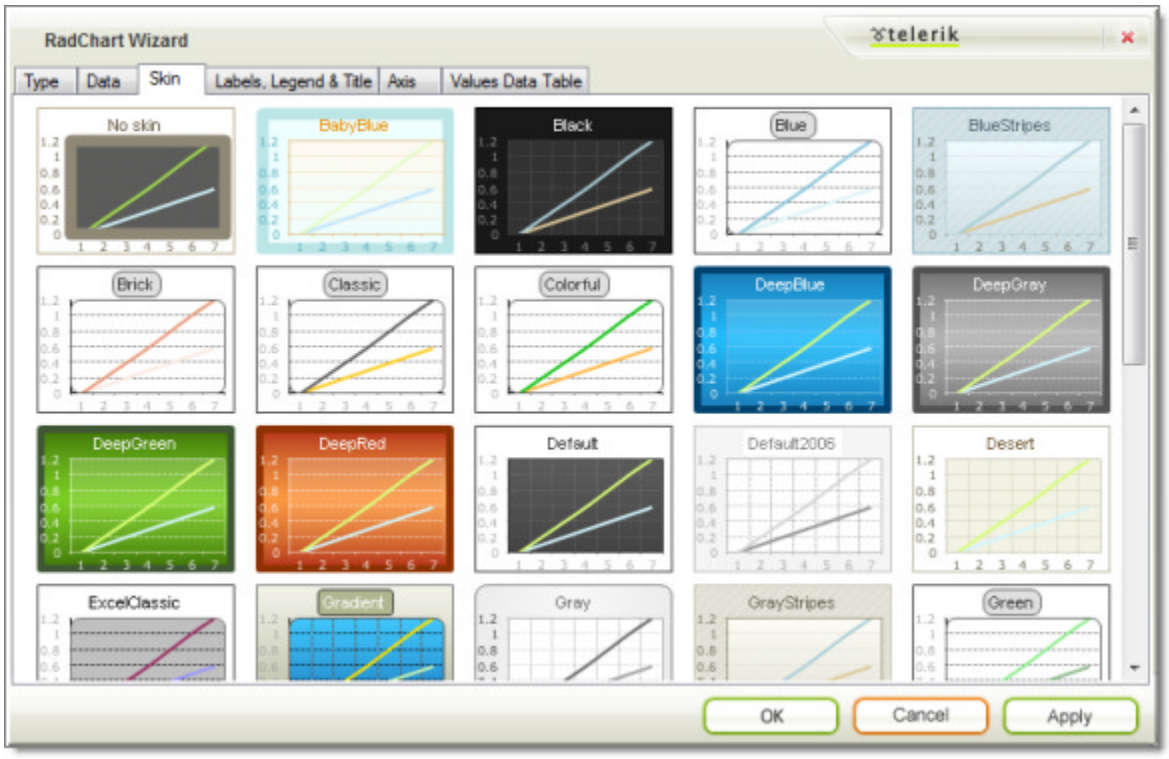
For each series you select in the Series Items list, you can add, edit, delete and reorder entries. Use the plus and minus buttons to add and delete series items. Use the up and down arrows to move series items up or down in the list. For each item you can set the Name, Label and X and Y Values. X2 and 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.

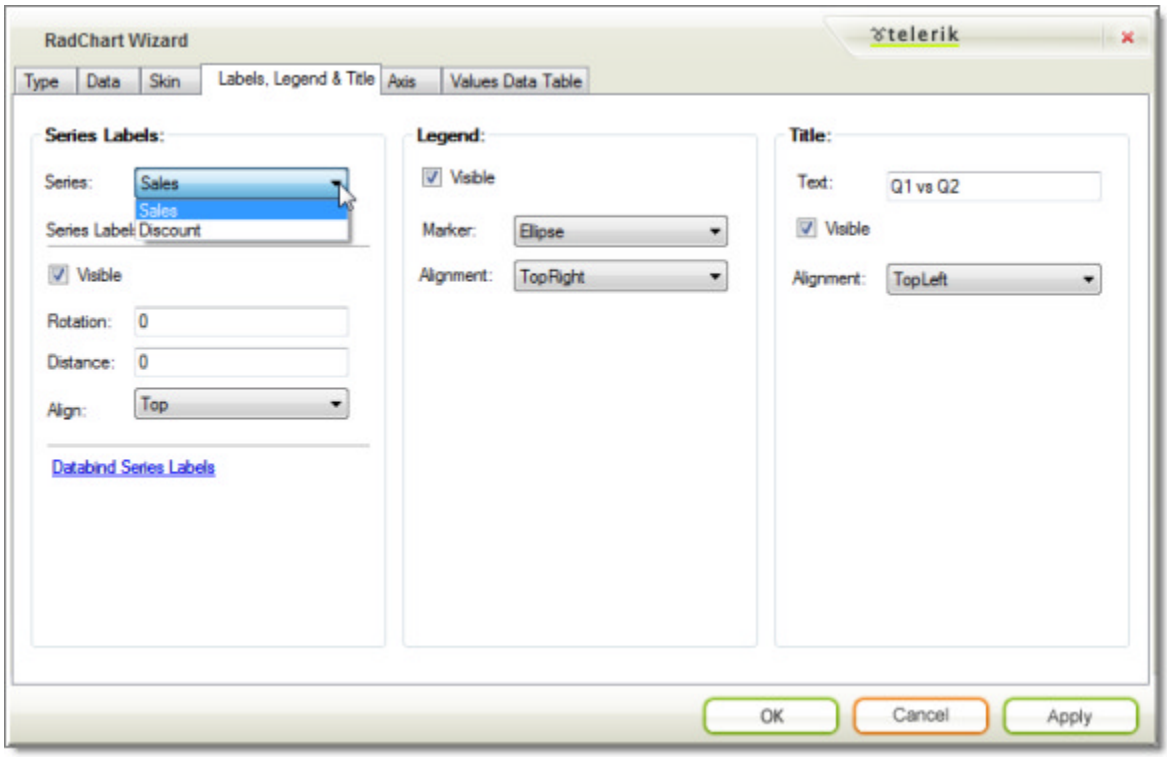
Skin Tab

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 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 in 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

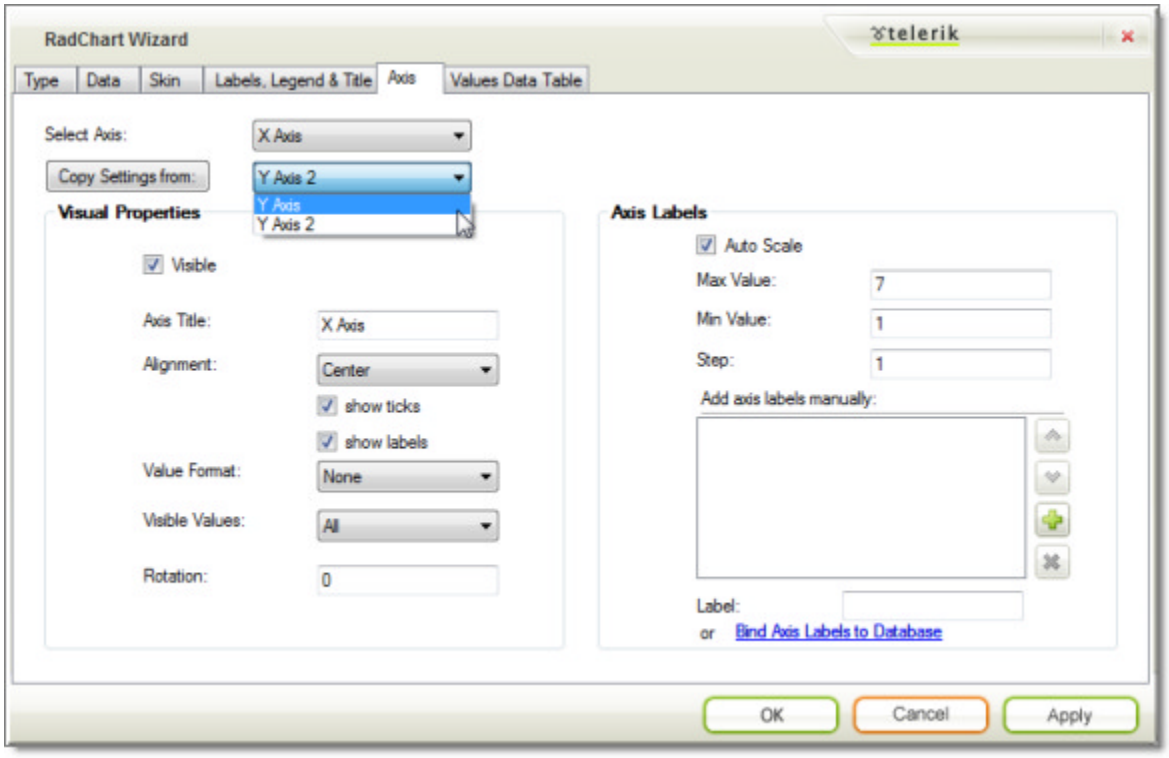
Un-select the Visible check box to hide the legend. Use the Marker drop down to select from a predefined list of shapes, e.g. Cross, Diamond, Ellipse, Rectangle, etc. Use the Alignment drop down to move the legend position between None, Left, Top, Bottom, Center, TopRight, TopLeft, BottomRight and BottomLeft.

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

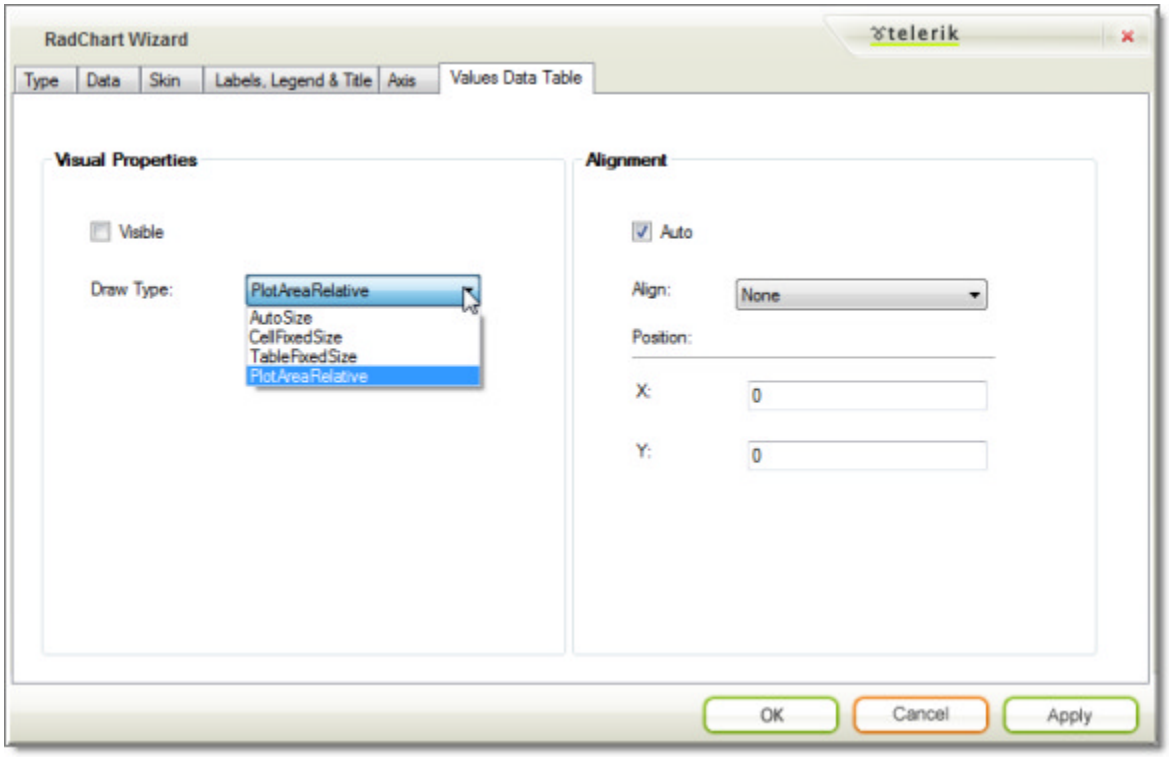
The Visual Properties section of the page controls properties for the axis as a whole. Uncheck the **Visible** checkbox to hide the entire axis (including labels and tick marks). The **Axis Title** text populates a single label that appears for the axis as a whole. Use the **Alignment** property to place 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, DataSource/DataMember, Legend, PlotArea and the Series collection. Within the Series collection

are Items collections that define the individual data points in the series.



The RadChart.Series collection contains individual ChartSeries items and each ChartSeries instance contains an Items collection that defines the individual ChartSeriesItems in the series (each series item maps to a single data point). The hierarchy is something like the outline below:

RadChart

Series

Items

<Data point 1>

<Data point 2>

<Data point N..>

Other helpful properties are...

AutoLayout: The AutoLayout feature of RadChart makes the best use of the form real-estate and expands the chart content as much as possible. This boolean property is turned off by default.

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.

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.

20.5 Programming the Chart

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

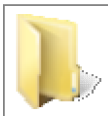
```
' add an item with a Y value
chartSeries.AddItem(5)
' add an empty item
```

```
Dim isEmpty As Boolean = True
Dim item As New ChartSeriesItem(isEmpty)
chartSeries.AddItem(item)
```

[C#] Adding a Chart Series Item

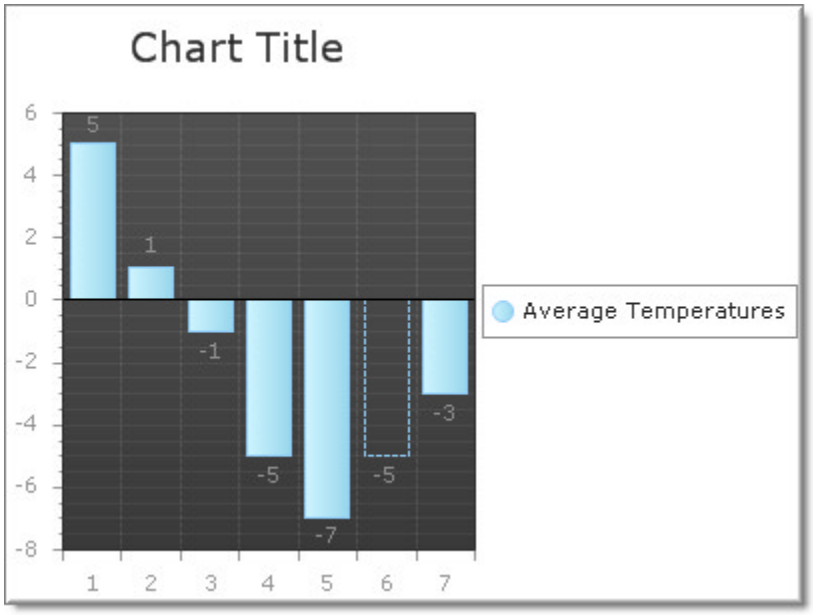
```
// add an item with a Y value
chartSeries.AddItem(5);
// add an empty item
bool isEmpty = true;
ChartSeriesItem item = new ChartSeriesItem(isEmpty);
chartSeries.AddItem(item);
```

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 and the reference to Telerik.Charting has been included.



You can find the complete source for this project at:

\Chart\<VB / CS>\Programming1



[VB] Adding a Chart Series and Items

```
Private Sub Form1_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)
```

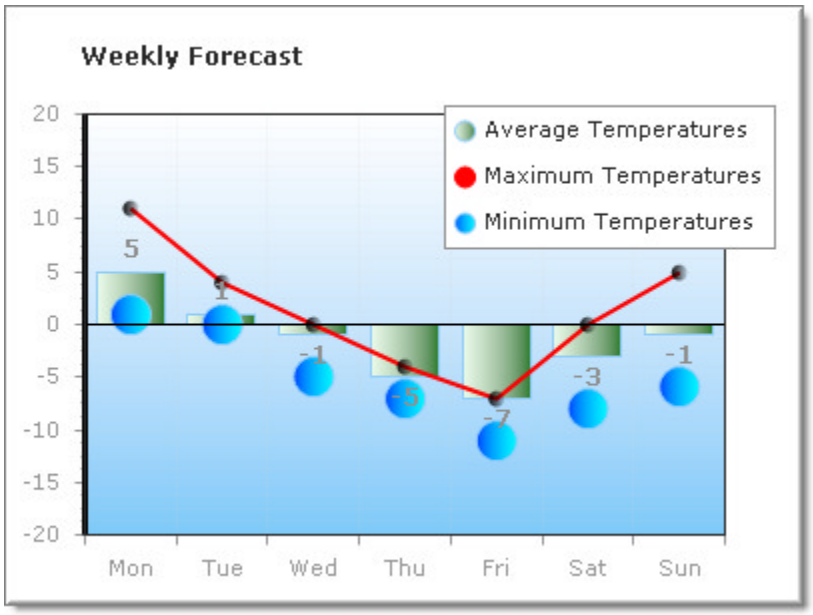
```
 chartSeries.AddItem(item)
 chartSeries.AddItem(-3)
 ' Add the series to the chart, chart to page.
 radChart1.Series.Add(chartSeries)
End Sub
```

[C#] Adding a Chart Series and Items

```
private void Form1_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. This project will also answer questions you may run across...

- 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 "all the values seem to scrunch up too high in 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:

\Chart\<VB|CS\Programming2

1. Create an Windows Forms Application.
2. From the Toolbox drag a **RadChart** component to the form.
3. Begin coding the Form_Load handler by setting up the label and chart title.

For the legend you need to shut off the Appearance.Position.Auto property 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 Form_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 Form_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";

 //...
}
```

4. Next, add code to the Form_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:

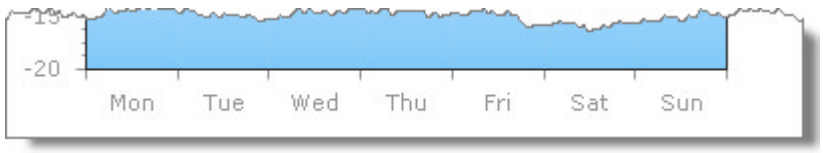
[VB] Configure the PlotArea

```
' 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;
```

5. 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"
```

6. **[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";
```

7. Configure the YXis 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";
```

8. 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;
```

9. Add the code below to the end of the Form_Load event handler: Add the data points to the first series by using the **AddItem()** method of the chart series and passing Y values.

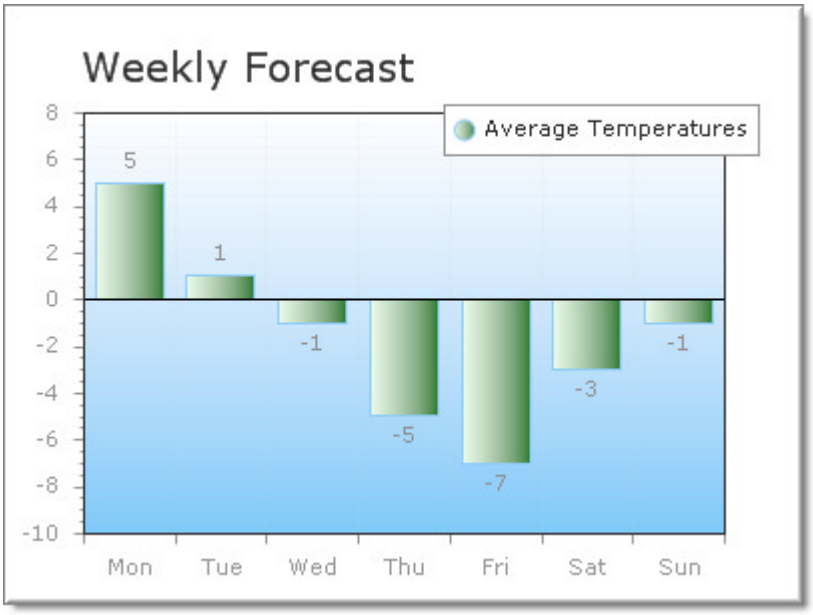
[VB] Adding Chart Series Items

```
' 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)
radChart1.Series.Add(chartSeries);
```

[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);
radChart1.Series.Add(chartSeries);
```

The chart should now look something like the screenshot below:



10. Add the code below to the end of the Form_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".

[VB] 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;
```

11. Add the code below to the end of the Form_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

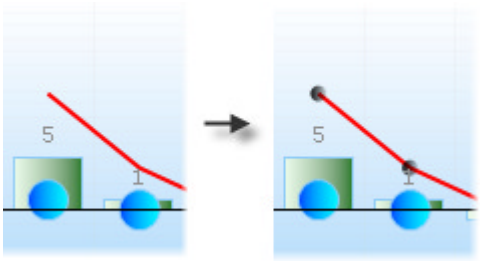
```
' Define the items in the series
chartSeries2.AddItem(11)
```

```
chartSeries2.AddItem(4)
chartSeries2.AddItem(0)
chartSeries2.AddItem(-4)
chartSeries2.AddItem(-7)
chartSeries2.AddItem(0)
chartSeries2.AddItem(5)
radChart1.Series.Add(chartSeries2)
```

[C#] 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);
radChart1.Series.Add(chartSeries2);
```

12. Add the code below to the end of the Form_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;
```

13. 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)
radChart1.Series.Add(chartSeries3;
' 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);
radChart1.Series.Add(chartSeries3;
// 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;
```

14. Press **Ctl-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 the **DataSource** property. The control-specific differences are in the properties used to specify what columns are bound to particular displays and behaviors in the chart.


RadChart uses the same basic underlying engine for Windows, Web and Reporting versions.

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 there meaning is...

- YValue = Open
- YValue2 = Close
- YValue 3 = Max
- YValue 4 = Min

The ChartSeries **DataLabelsColumn** supplies the text that displays on top of each series item. The XAxis also has a DataLabelsColumn property that displays text next to each X Axis item.

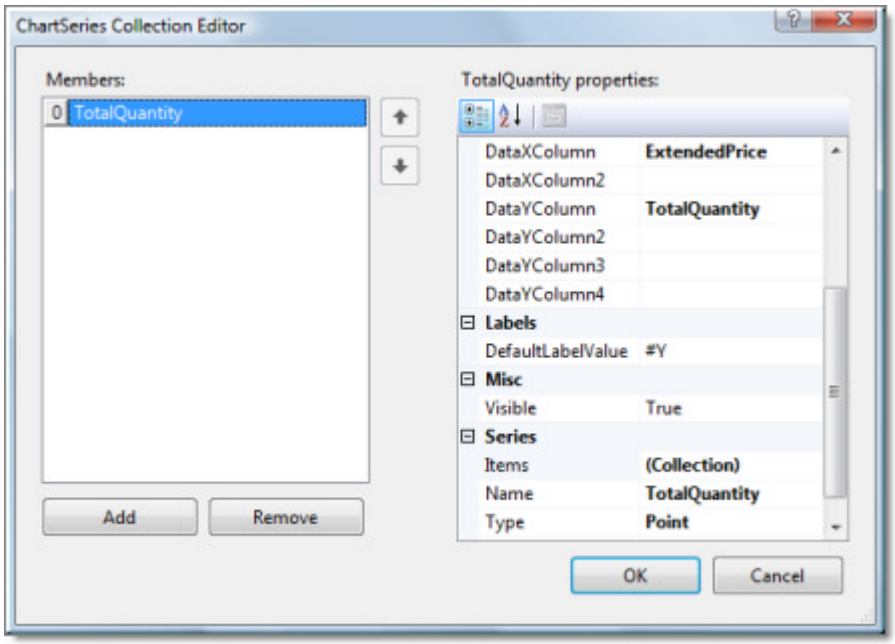
Data Binding Events

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 particularly data point (i.e. row) to make very specific changes to your SeriesItem.

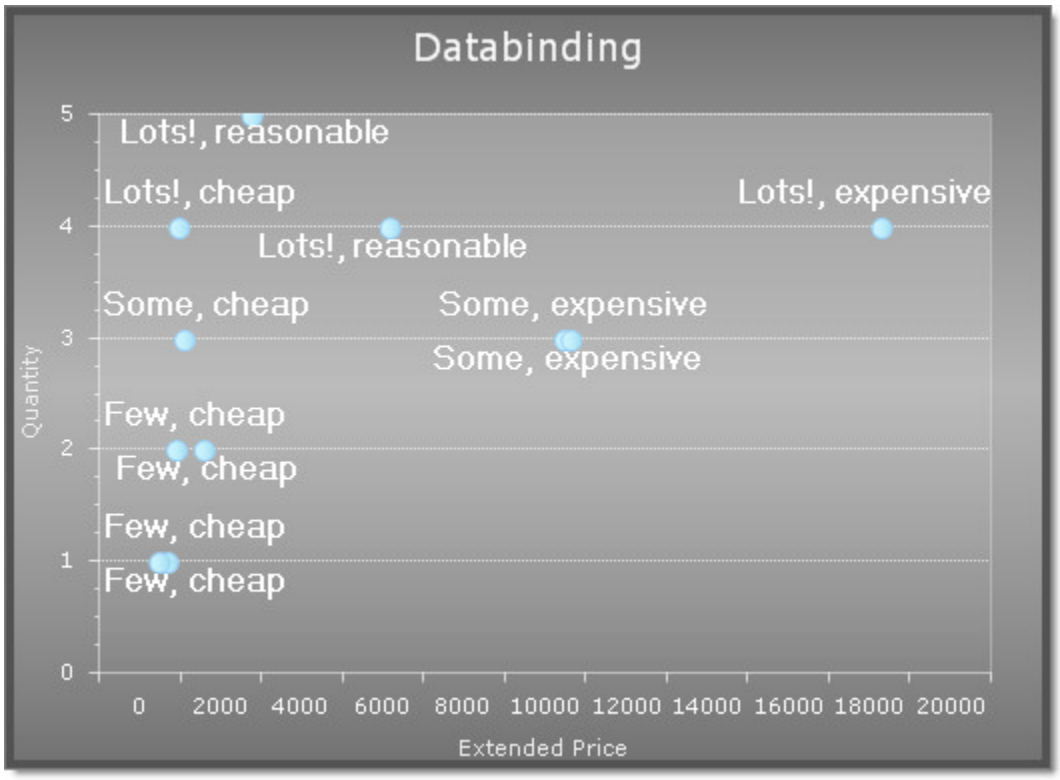
Here's an example where we bind to the Furniture.mdb file and query the Products table:

SELECT SalesRepresentative, SUM(Quantity) AS TotalQuantity, SUM(Quantity * Price) AS ExtendedPrice
FROM Products GROUP BY SalesRepresentative

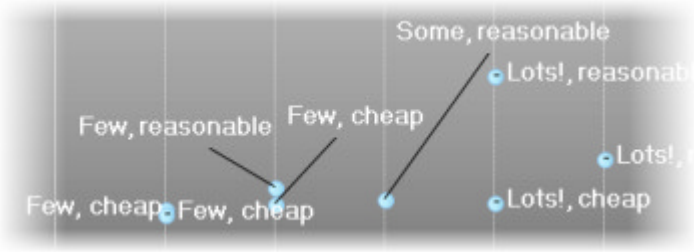
This point chart plots each point where the "ExtendedPrice" appears along the X axis and the "TotalQuantity" along the Y axis.



The resulting chart looks something like the example below:



You can run into a problem if IntelligentLabelsEnabled is set to true, moving the labels too far away from their respective data points, thus making the chart harder to interpret. Setting the ChartSeries Appearance.ShowLabelConnectors property to true displays the lines between the labels and the data points.



As the items are bound, the labels are formatted based on the ranges of values the data points fall within.

[VB] 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 < 2000 Then
   priceLabel = "cheap"
ElseIf price < 10000 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 < 2000)
 {
   priceLabel = "cheap";
 }
 else if (price < 10000)
 {
   priceLabel = "reasonable";
 }
 else
 {
   priceLabel = "expensive";
 }
 e.SeriesItem.Label.TextBlock.Appearance.TextProperties.Font =
   new System.Drawing.Font("Ariel", 12, System.Drawing.FontStyle.Bold);
 e.SeriesItem.Label.TextBlock.Text = quantityLabel + ", " + priceLabel;
}
```



You can find the complete source for this project at:

\Chart\<VB|CS>\Databinding

Grouping Data Bound Items

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 "2011" and "2012".
- The DataGroupColumn property is "Year".

...then there will be two series, one for "2011" and the second for "2012".

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.

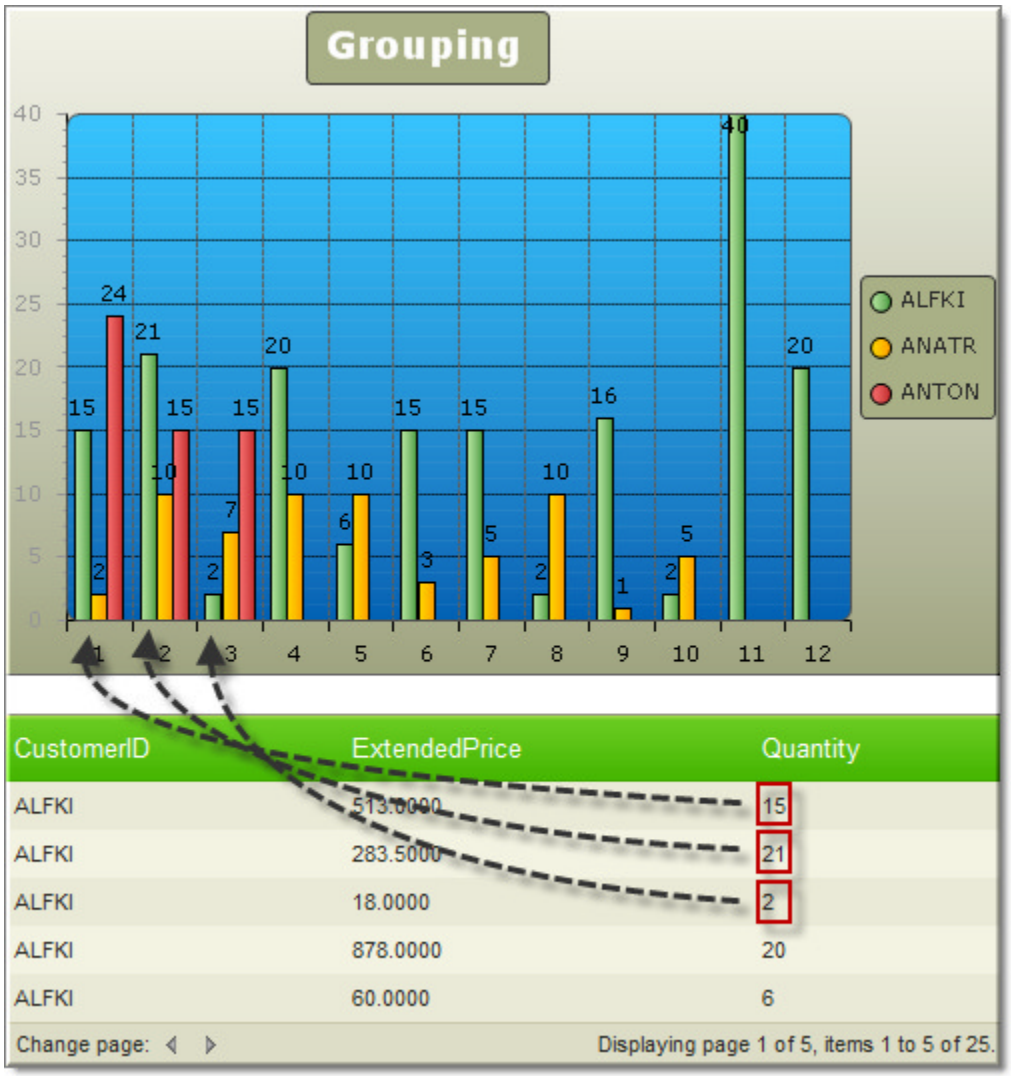
[T-SQL] Selecting Invoice Data

```
SELECT TOP (25) CustomerID, ExtendedPrice, Quantity FROM Invoices ORDER BY CustomerID
```



The "ORDER BY" clause is important 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 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: 2011" and "Year: 2012". 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 chart.

You can find the complete source for this project at:

\Chart\<VB|CS>\Grouping

1. Create a new Windows Forms application.
2. From the Toolbox drag a **RadChart** component to the form.
3. Add a simple "Sales" class either to the form code or to a separate Sales.cs file with the following class definition:

4

[VB] The Sales Class Definition

```
Public Class Sales
 Public Sub New(ByVal year As Integer, ByVal quarter As Integer, ByVal value As Double)
  Year = year
  Quarter = quarter
  Value = value
 End Sub
 Public Property Year() As Integer
  Get
  End Get
  Set
  End Set
 End Property
 Public Property Quarter() As Integer
  Get
  End Get
  Set
  End Set
 End Property
 Public Property Value() As Double
  Get
  End Get
  Set
  End Set
 End Property
End Class
```

4. **[C#] The Sales Class Definition**

```
public class Sales
{
 public Sales(int year, int quarter, double value)
 {
   Year = year;
   Quarter = quarter;
   Value = value;
 }
 public int Year { get; set; }
 public int Quarter { get; set; }
 public double Value { get; set; }
}
```

5. In the Form_Load, populate a generic List of Sales objects:

[VB] Populate and Group Chart Data

```
Protected Sub Form1_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(2011, 1, 5))
 sales.Add(New Sales(2011, 2, 2))
 sales.Add(New Sales(2011, 3, 3))
 sales.Add(New Sales(2011, 4, 1))
 sales.Add(New Sales(2012, 1, 4))
 sales.Add(New Sales(2012, 2, 3))
 sales.Add(New Sales(2012, 3, 8))
 sales.Add(New Sales(2012, 4, 2))
```

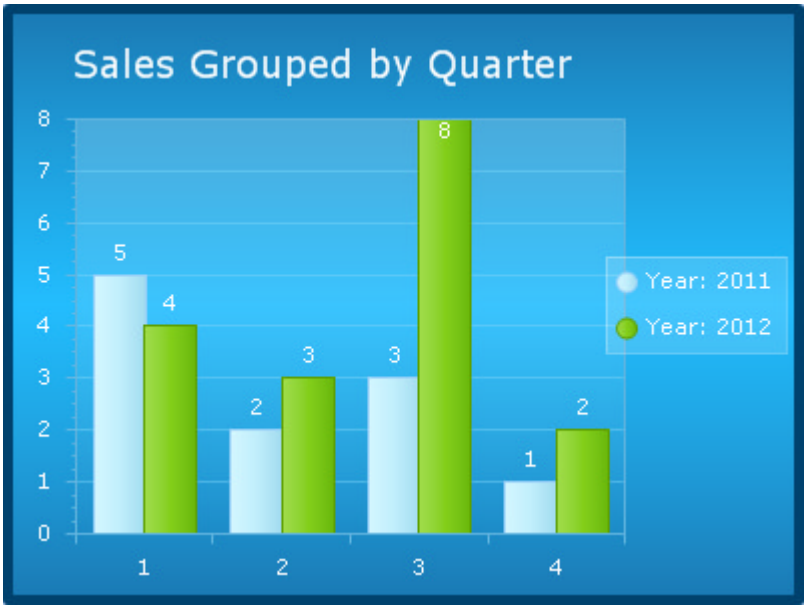
```
 ' Setup the chart appearance and title
 radChart1.Skin = "DeepBlue"
 radChart1.ChartTitle.TextBlock.Text = "Sales Grouped by Quarter"
 ' Remove the default series
 radChart1.Series.Clear()
 ' Create and add a new Bar series type
 Dim chartSeries As New ChartSeries("Sales", ChartSeriesType.Bar)
 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

End Sub
```

[C#] Populate and Group Chart Data

```
protected void Form1_Load(object sender, EventArgs e)
{
 // Populate the generic list of Sales
 List<Sales> sales = new List<Sales>();
 sales.Add(new Sales(2011, 1, 5));
 sales.Add(new Sales(2011, 2, 2));
 sales.Add(new Sales(2011, 3, 3));
 sales.Add(new Sales(2011, 4, 1));
 sales.Add(new Sales(2012, 1, 4));
 sales.Add(new Sales(2012, 2, 3));
 sales.Add(new Sales(2012, 3, 8));
 sales.Add(new Sales(2012, 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;
}
```

6. Press **Ctl-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...

Displaying Date Time Values on Numeric Axes

One of the top most frequently asked questions is "how do I display date and time values along the axes"? The secret is in the DateTime **ToAODate()** method that converts a DateTime to a OLE Automation double value. Use the OLE Automation dates to populate axes. The steps for each axis:

- Set the axis **AutoScale** property off so you can manually populate.
- Set the axis Appearance.**ValueFormat** using the **ChartValueFormat** enumeration. For example, you can use the ShortDate or ShortTime to format the axis.
- Add a range of DateTime values, being careful to convert them with the ToAODate() method.

Here's a short code example that labels the XAxis with a series of dates:

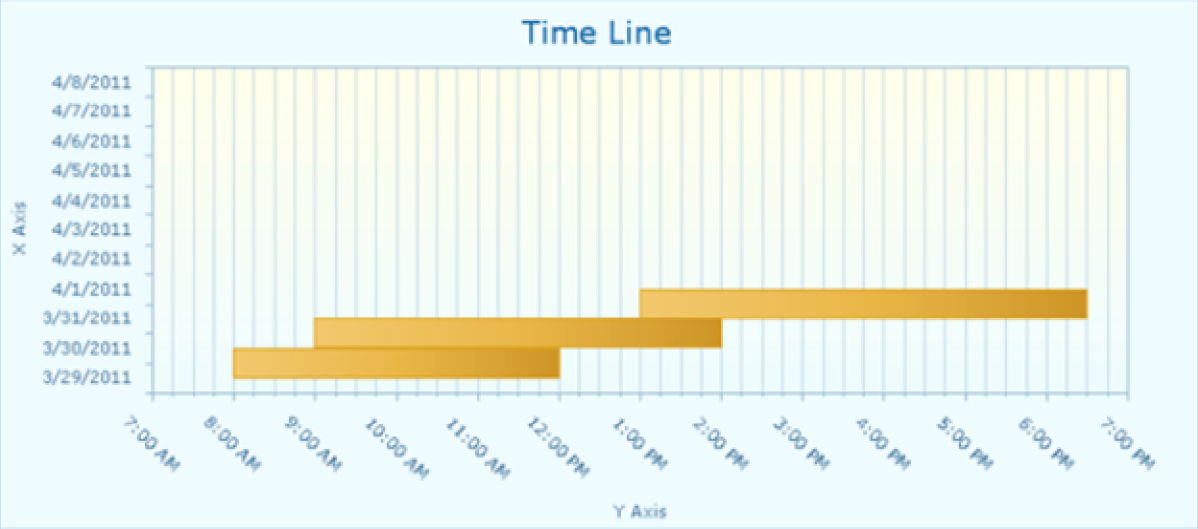
[VB] Adding a Range of Dates

```
radChart1.PlotArea.XAxis.AutoScale = False
radChart1.PlotArea.XAxis.Appearance.ValueFormat = ChartValueFormat.ShortDate
radChart1.PlotArea.XAxis.AddRange(today.ToOADate(), today.AddDays(10).ToOADate(), 1)
```

[C#] Adding a Range of Dates

```
radChart1.PlotArea.XAxis.AutoScale = false;
radChart1.PlotArea.XAxis.Appearance.ValueFormat = ChartValueFormat.ShortDate;
radChart1.PlotArea.XAxis.AddRange(today.ToOADate(), today.AddDays(10).ToOADate(), 1);
```

Here's a slightly longer example that shows a gantt chart with a series of dates on the X axis and a series of times on the Y axis.



You can find the complete source for this project at:

\Chart\<VB|C#>\DateTimeOnNumericAxis

1. In a new WinForms project, add a RadChart to the default form.
2. In the code behind, add a private method GetDataTable() that will generate and return a four column table with sample values. Notice that the data values are also converted to OLE Automation types before storing.

[VB] Generating the Data

```
' generate a four column table with sample
' date and time data stored as doubles
Private Function GetDataTable() As DataTable
 Dim input As Double(,) = New Double(3, 2) {{8, 12}, {9, 14}, {13, 18.5}}
 Dim table As New DataTable()
 table.Columns.Add("X", GetType(Double))
 table.Columns.Add("X2", GetType(Double))
 table.Columns.Add("Y", GetType(Double))
 table.Columns.Add("Y2", GetType(Double))
 Dim start As DateTime = DateTime.Today
 Dim i As Integer = 0
 While i < 3
  Dim row As DataRow = table.NewRow()
  ' convert all dates and times to OLE Automation dates
  ' which are stored as doubles
  row("X") = start.AddDays(i).ToOADate()
  row("X2") = start.AddDays(i + 1).ToOADate()
  row("Y") = start.AddHours(input(i, 0)).ToOADate()
  row("Y2") = start.AddHours(input(i, 1)).ToOADate()
  table.Rows.Add(row)
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
 Return table
End Function
```

[C#] Generating the Data

```
// generate a four column table with sample
```

```
// date and time data stored as doubles
private DataTable GetDataTable()
{
 double[,] input = new double[3, 2] {
     { 8, 12 }, { 9, 14}, { 13, 18.5}
 };
 DataTable table = new DataTable();
 table.Columns.Add("X", typeof(double));
 table.Columns.Add("X2", typeof(double));
 table.Columns.Add("Y", typeof(double));
 table.Columns.Add("Y2", typeof(double));
 DateTime start = DateTime.Today;
 for (int i = 0; i < 3; i++)
 {
   DataRow row = table.NewRow();
   // convert all dates and times to OLE Automation dates
   // which are stored as doubles
   row["X"] = start.AddDays(i).ToOADate();
   row["X2"] = start.AddDays(i + 1).ToOADate();
   row["Y"] = start.AddHours(input[i, 0]).ToOADate();
   row["Y2"] = start.AddHours(input[i, 1]).ToOADate();
   table.Rows.Add(row);
 }
 return table;
}
```

3. In the form Load event, configure both axes using the ShortDate format for the X axis and the ShortTime format for the Y axis. Notice at the top of the method where we make a little room for the labels by removing the legend, setting AutoLayout true to automatically use the extra space. Also notice that the axis labels are made visible and the YAxis label is rotated 45 degrees.

[VB] Configuring the Chart And Binding the Data

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 radChart1.AutoLayout = True
 radChart1.Legend.Visible = False
 radChart1.PlotArea.YAxis.AxisLabel.Visible = True
 radChart1.PlotArea.YAxis.Appearance.LabelAppearance.RotationAngle = 45
 radChart1.PlotArea.XAxis.AxisLabel.Visible = True
 Dim today As DateTime = DateTime.Today
 ' Setup chart orientation
 radChart1.SeriesOrientation = ChartSeriesOrientation.Horizontal
 ' Setup XAxis
 radChart1.PlotArea.XAxis.Appearance.ValueFormat = ChartValueFormat.ShortDate
 radChart1.PlotArea.XAxis.AutoScale = False
 radChart1.PlotArea.XAxis.AddRange(today.ToOADate(), today.AddDays(10).ToOADate(), 1)
 ' Setup YAxis
 radChart1.PlotArea.YAxis.Appearance.ValueFormat = ChartValueFormat.ShortTime
 radChart1.PlotArea.YAxis.AutoScale = False
 radChart1.PlotArea.YAxis.AddRange(today.AddHours(7).ToOADate(), today.AddHours(18).ToOAD
(), 1 / 24)
 ' Setup Series
 radChart1.Series(0).Type = ChartSeriesType.Gantt
 radChart1.Series(0).DataXColumn = "X"
 radChart1.Series(0).DataXColumn2 = "X2"
 radChart1.Series(0).DataYColumn = "Y"
 radChart1.Series(0).DataYColumn2 = "Y2"
```

```
 ' Data Bind
 radChart1.Series(0).Appearance.LabelAppearance.Visible = False
 radChart1.DataSource = GetDataTable()
 radChart1.DataBind()
End Sub
```

[C#] Configuring the Chart And Binding the Data

```
private void RadForm1_Load(object sender, EventArgs e)
{
 radChart1.AutoLayout = true;
 radChart1.Legend.Visible = false;
 radChart1.PlotArea.YAxis.AxisLabel.Visible = true;
 radChart1.PlotArea.YAxis.Appearance.LabelAppearance.RotationAngle = 45;
 radChart1.PlotArea.XAxis.AxisLabel.Visible = true;
 DateTime today = DateTime.Today;
 // Setup chart orientation
 radChart1.SeriesOrientation = ChartSeriesOrientation.Horizontal;
 // Setup XAxis
 radChart1.PlotArea.XAxis.Appearance.ValueFormat = ChartValueFormat.ShortDate;
 radChart1.PlotArea.XAxis.AutoScale = false;
 radChart1.PlotArea.XAxis.AddRange(today.ToOADate(), today.AddDays(10).ToOADate(), 1);
 // Setup YAxis
 radChart1.PlotArea.YAxis.Appearance.ValueFormat = ChartValueFormat.ShortTime;
 radChart1.PlotArea.YAxis.AutoScale = false;
 radChart1.PlotArea.YAxis.AddRange(today.AddHours(7).ToOADate(), today.AddHours(18).ToOAD
(), 1d / 24d);
 // Setup Series
 radChart1.Series[0].Type = ChartSeriesType.Gantt;
 radChart1.Series[0].DataXColumn = "X";
 radChart1.Series[0].DataXColumn2 = "X2";
 radChart1.Series[0].DataYColumn = "Y";
 radChart1.Series[0].DataYColumn2 = "Y2";
 // Data Bind
 radChart1.Series[0].Appearance.LabelAppearance.Visible = false;
 radChart1.DataSource = GetDataTable();
 radChart1.DataBind();
}
```

4. Run the application.

Events

Use the **BeforeLayout** event when you want to safely customize layout-related properties. BeforeLayout will correctly display without clipping because layout arrangements occur after this event. All the elements of the chart exist before this event fires, so you can make changes to items, axis, legend, etc., as in this example below.

The form Load event makes a little extra room for the new labels. The BeforeLayout event handler sets the chart legend border and cycles through the Y axis text blocks, sets a custom string and rotates the axis item 20 degrees.

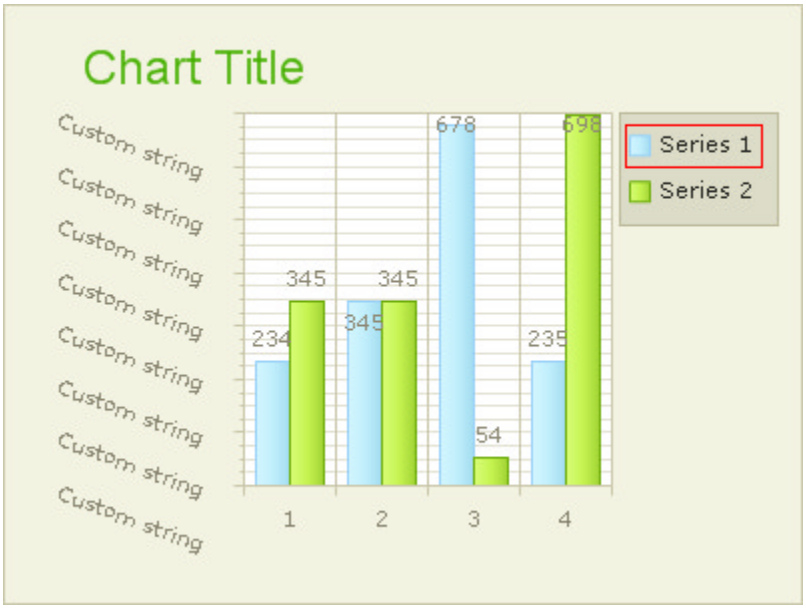
[VB] Handling the BeforeLayout Event

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 radChart1.PlotArea.Appearance.Dimensions.Margins.Left = New Telerik.Charting.Styles.Unit
(30, Telerik.Charting.Styles.UnitType.Percentage)
```

```
 radChart1.PlotArea.Appearance.Dimensions.Margins.Bottom = New Telerik.Charting.Styles.Unit
(20, Telerik.Charting.Styles.UnitType.Percentage)
 radChart1.IntelligentLabelsEnabled = True
End Sub
Private Sub radChart1_BeforeLayout(sender As Object, e As EventArgs)
 radChart1.Legend.Items(0).Appearance.Border.Color = Color.Red
 radChart1.Legend.Items(0).Appearance.Border.Visible = True
 For Each axisItem As ChartAxisItem In Me.radChart1.PlotArea.YAxis.Items
  axisItem.TextBlock.Text = "Custom string"
  axisItem.Appearance.RotationAngle = 20
 Next
End Sub
```

[C#] Handling the BeforeLayout Event

```
private void Form1_Load(object sender, EventArgs e)
{
 radChart1.PlotArea.Appearance.Dimensions.Margins.Left =
   new Telerik.Charting.Styles.Unit(30, Telerik.Charting.Styles.UnitType.Percentage);
 radChart1.PlotArea.Appearance.Dimensions.Margins.Bottom =
   new Telerik.Charting.Styles.Unit(20, Telerik.Charting.Styles.UnitType.Percentage);
 radChart1.IntelligentLabelsEnabled = true;
}
private void radChart1_BeforeLayout(object sender, EventArgs e)
{
 radChart1.Legend.Items[0].Appearance.Border.Color = Color.Red;
 radChart1.Legend.Items[0].Appearance.Border.Visible = true;
 foreach (ChartAxisItem axisItem in this.radChart1.PlotArea.YAxis.Items)
 {
   axisItem.TextBlock.Text = "Custom string";
   axisItem.Appearance.RotationAngle = 20;
 }
}
```

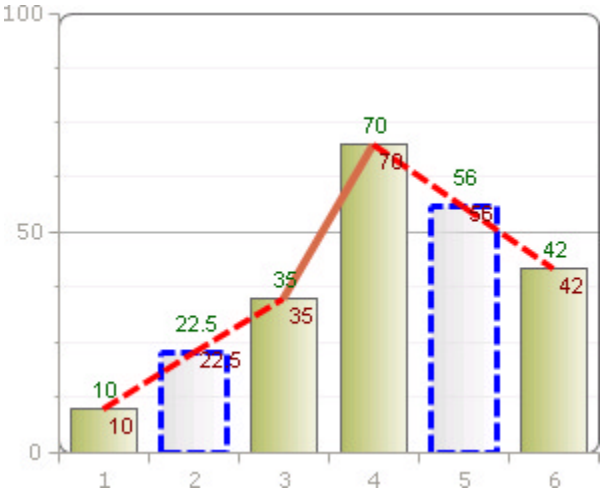


20.6 Unique RadChart Features

RadChart has some unique features that we haven't run into yet that you should be aware of:

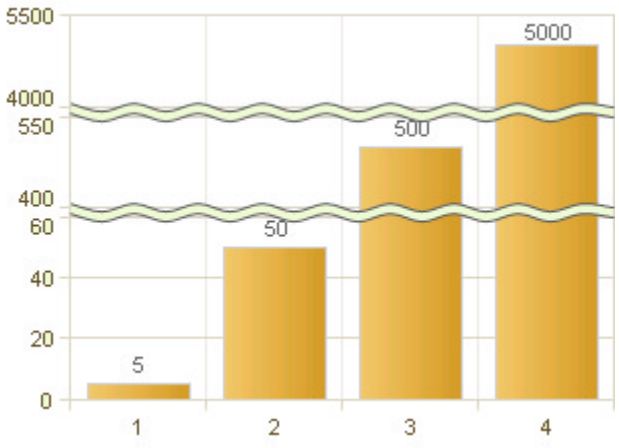
Empty Values

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.



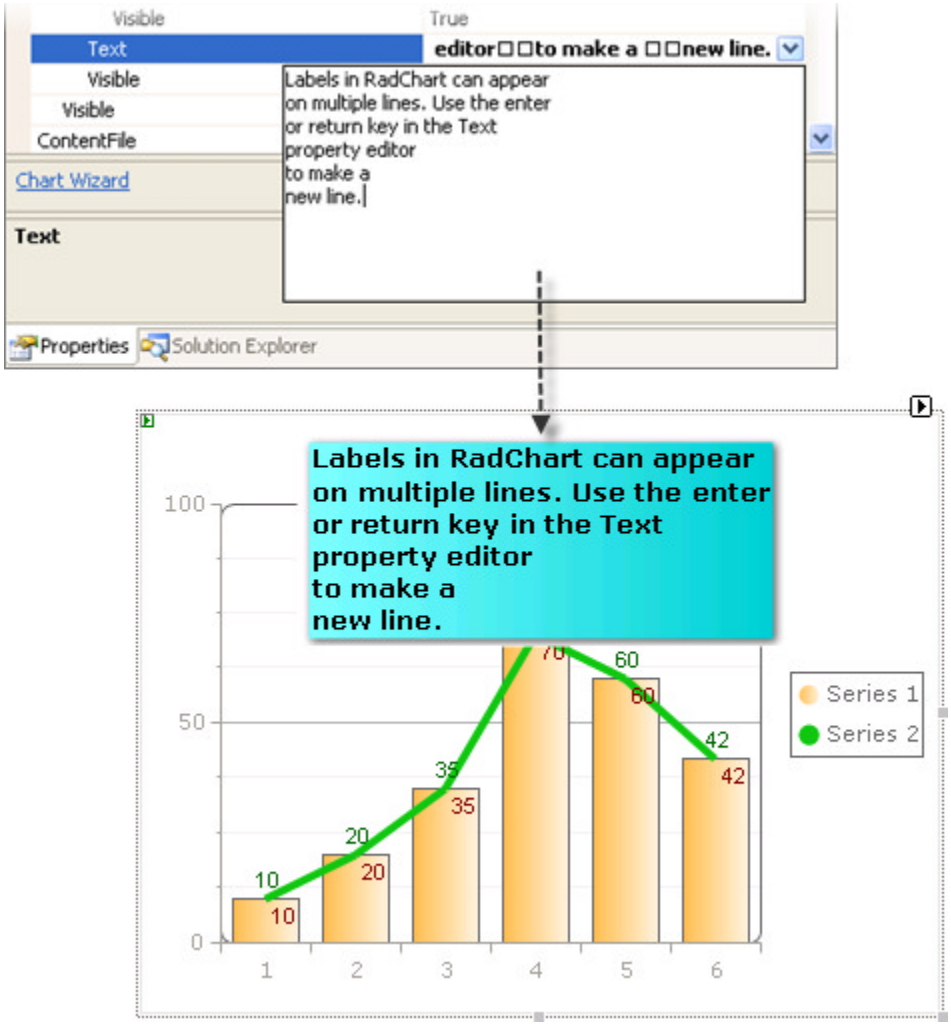
Scale Breaks

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

"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.

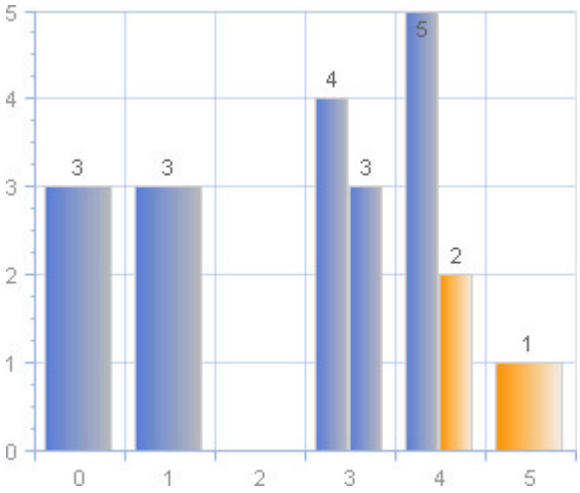
The screen shot below was produced using the X and Y values from this table:

Series 1 (Blue)

YValue	XValue
3	0
3	1
4	3
3	3
5	4

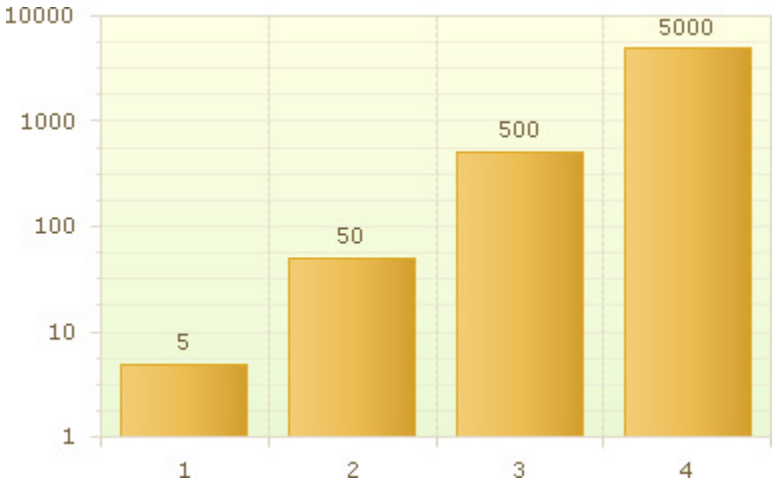
Series 2 (Orange)

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.



20.7 How To

Drill-Down

RadChart can be configured to handle "drill-down" functionality where the initial presentation of the chart presents the "Mile High View" of data where the information shows the most general, highest level scope of data. Then the user clicks on a chart element and "drills down" to the next level of specificity.

For example, if we have revenue by Year, Quarter and Month, the initial view will be the greatest time span, which is year. The year is divided into quarters. By clicking on an item on the chart you can see the values correspoding to the quarters of the selected year. Each quarter consists of 3 months. After choosing a quarter for the selected year you can see the values for each month of the quarter.

To test this you can use the Chart.mdb file in the RadControls for Winforms installation directory under \Examples\DataSources. Chart.mdb contains a "Revenue" table. Select the Year, Quarter, Month and Revenue columns.



Prepare the Project

1. Create a new Windows Forms application.
2. From the Toolbox drag a **RadChart** component to the default form.

Configure the RadChart Using the Wizard

1. Configure the data source:
    - From the Smart Tag open the **Choose Data Source** drop down and select **Add Project Data Source...** from the list.
    - In the "Choose a Data Source Type" page of the wizard, select the Database icon, then click the **Next** button to continue.
    - In the "Choose your data connection" page of the wizard, click the **New Connection...** button. This step will bring up the Add Connection dialog.
    - In the Add Connection dialog, the Data Source entry should read "Microsoft Access Database File (OLE DB)" (see screenshot below). If it is not, click the **Change...** button and select "Microsoft Access Database File (OLE DB)" from the list.
    - Also in the Add Connection dialog, click the **Browse...** button, navigate to the RadControls for Winforms installation and locate the \Examples\QuickStart\DataSources directory. Open the "Chart.mdb" file.

- Click **OK** to close the Add Connection dialog.
- Back at the "Choose Your Data Connection" page of the wizard, click the **Next** button to continue.
- You will receive a dialog popup with message "The connection you selected is a local file...". Click **Yes** to close the dialog and copy the data file locally.
- In the **Choose Your Database Objects** page, open the treeview "Tables" node, locate the "Revenue" table and check the "Year", "Quarter", "Month" and "Revenue" columns.
- Click the **Finish** button to close the dialog. *This last step will create a DataSet, TableAdapter and BindingSource and place them in the components tray.*

  The next few steps will create table adapter methods used to fill the dataset by data for the year, month and quarter. These methods will be called from within your code when the chart is clicked.

2. Create a "FillByYear" data source query statement:
   1. Locate the "revenueTableAdapter" component in the component tray. Open the Smart Tag and select **Edit Queries in DataSet Designer...**
   2. Right-click the "FillBy" element and select **Configure...** from the context menu.
   3. On the "Specify a SQL SELECT statement" page of the wizard, enter the following SQL statement and click the **Next** button to continue.

      [SQL] FillByYear

      ```
      SELECT [YEAR], SUM(Revenue) AS Revenue
      FROM Revenue
      GROUP BY [YEAR]
      ```

   4. On the "Choose Methods to Generate" page of the wizard, check the "Fill a DataTable" option and set the Method name to "FillByYear". Un-check the "Return a DataTable" option. Click the **Finish** button to close the dialog.
3. Add a "FillByQuarter" data source query statement:
   1. Right-click the RevenueTableAdapter and choose Add Query... from the context menu. This will display the TableAdapter Query Configuration Wizard.
   2. On the "Choose a Command Type" page of the wizard, select the "Use a SELECT statement to load data" option. Click the **Next** button to continue.
   3. On the "Choose a Query Type" page of the wizard, select the "SELECT which returns rows" option. Click the **Next** button to continue.
   4. On the "Specify a SQL SELECT statement" page of the wizard, enter the following SQL statement and click the **Next** button to continue.

      [SQL] FillByQuarter

      ```
      SELECT SUM(Revenue) AS Revenue, Quarter
      FROM Revenue
      WHERE [YEAR] = ?
      GROUP BY Quarter
      ```

   5. On the Choose Methods to Generate page of the wizard, check the "Fill a DataTable" option and set the Method name to "FillByQuarter". Un-check the "Return a DataTable" option. Click the **Finish** button to close the dialog.
4. Add a "FillByMonth" data source query statement:
   1. Right-click the RevenueTableAdapter and choose Add Query... from the context menu. This will display the TableAdapter Query Configuration Wizard.
   2. On the "Choose a Command Type" page of the wizard, select the "Use a SELECT statement to load

data" option. Click the **Next** button to continue.

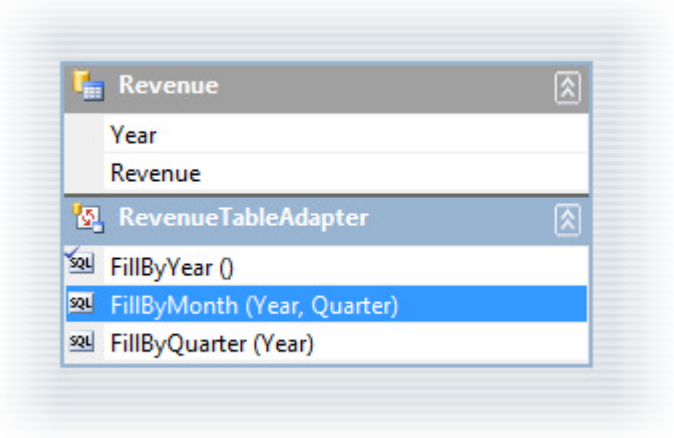
3. On the "Choose a Query Type" page of the wizard, select the "SELECT which returns rows" option. Click the **Next** button to continue.
4. On the "Specify a SQL SELECT statement" page of the wizard, enter the following SQL statement and click the **Next** button to continue.

   [SQL] FillByMonth

```
SELECT SUM(Revenue) AS Revenue, [MONTH]
FROM Revenue
WHERE ([YEAR] = ?) AND (Quarter = ?)
GROUP BY [MONTH]
```

5. On the Choose Methods to Generate page of the wizard, check the "Fill a DataTable" option and set the Method name to "FillByMonth". Un-check the "Return a DataTable" option. Click the **Finish** button to close the dialog.

   The RevenueTableAdapter designer should look something like the screenshot below:



5. In the code for the form, add the following private variable to store the current year when its selected from the chart:

   [VB] Storing the Current Year

```
'stores the currently selected year
Private _year As Integer
```

   [C#] Storing the Current Year

```
//stores the currently selected year
private int _year;
```

6. Add the following to the Form_Load event. *This set of statements will fill the dataset initially with the "by year" data and configure the chart. Notice that the XAxis AutoScale is turned off so that MinValue and MaxValue settings can take effect.*

   [VB] Handle the Form_Load Event

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
 Me.revenueTableAdapter.FillByYear(Me.chartDataSet.Revenue)
 ' remove the second, unused series
 radChart1.Series.RemoveAt(1)
```

```
 radChart1.ChartTitle.TextBlock.Text = "Years"
 radChart1.PlotArea.XAxis.IsZeroBased = False
 radChart1.PlotArea.XAxis.AutoScale = False
 radChart1.PlotArea.XAxis.MinValue = 2003
 radChart1.PlotArea.XAxis.MaxValue = 2005
 'attach the datasource to the chart
 radChart1.DataManager.DataSource = Me.chartDataSet.Revenue
 radChart1.Series(0).DataYColumn = "Revenue"
 radChart1.Series(0).DataXColumn = "Year"
 radChart1.Series(0).Name = "Years"
End Sub
```

[C#] Handle the Form_Load Event

```
private void Form1_Load(object sender, EventArgs e)
{
 this.revenueTableAdapter.FillByYear(this.chartDataSet.Revenue);
 // remove the second, unused series
 radChart1.Series.RemoveAt(1);
 radChart1.ChartTitle.TextBlock.Text = "Years";
 radChart1.PlotArea.XAxis.IsZeroBased = false;
 radChart1.PlotArea.XAxis.AutoScale = false;
 radChart1.PlotArea.XAxis.MinValue = 2003;
 radChart1.PlotArea.XAxis.MaxValue = 2005;
 //attach the datasource to the chart
 radChart1.DataManager.DataSource = this.chartDataSet.Revenue;
 radChart1.Series[0].DataYColumn = "Revenue";
 radChart1.Series[0].DataXColumn = "Year";
 radChart1.Series[0].Name = "Years";
}
```

7. Double-click the chart to create an Click event handler and populate the handler with the code below. *This code uses the ChartClickEventArgs passed in, finds the ChartSeriesItem that was clicked on and checks that the item's parent is "Years". If that condition is true, then we have just drilled down to the "Quarters", otherwise we're looking at "Months". Depending on which level we're at, clear the series, set the DataYColumn to the appropriate data column name and bind the chart.*

   [VB] Handling the Click Event

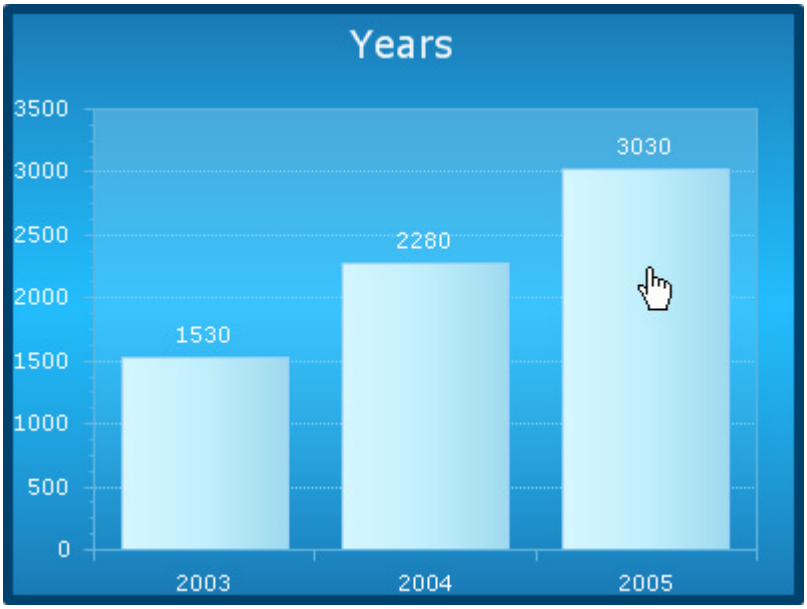
```
Private Sub radChart1_Click(ByVal sender As Object, ByVal args As
Telerik.Charting.ChartClickEventArgs)
 If TypeOf args.Element.ActiveRegion.Parent Is ChartSeriesItem AndAlso (TryCast
(args.Element.ActiveRegion.Parent, ChartSeriesItem)).Parent.Name = "Years" Then
  radChart1.ChartTitle.TextBlock.Text = "Quarters"
  'a bar representing the revenue for a year was clicked
  _year = (TryCast(args.Element, ChartSeriesItem)).Index + 2003
  'fill revenue values for the year whose bar was clicked
  revenueTableAdapter.FillByQuarter(Me.chartDataSet.Revenue, _year)
  'setup chart properties with appopriate axis labels
  radChart1.PlotArea.XAxis.DataLabelsColumn = "Quarter"
  radChart1.Series(0).Clear()
  radChart1.Series(0).DataYColumn = "Revenue"
  radChart1.Series(0).Name = "Quarters"
  radChart1.DataBind()
ElseIf TypeOf args.Element Is ChartSeriesItem AndAlso (TryCast(args.Element,
ChartSeriesItem)).Parent.Name = "Quarters" Then
  radChart1.ChartTitle.TextBlock.Text = "Months";
  'a bar representing the revenue for a year was clicked
```

```
  Dim quarter As Integer = (TryCast(args.Element, ChartSeriesItem)).Index + 1
  'fill revenue values for the year whose bar was clicked
  revenueTableAdapter.FillByMonth(Me.chartDataSet.Revenue, _year, quarter)
  'setup chart properties with appopriate axis labels
  radChart1.PlotArea.XAxis.DataLabelsColumn = "Month"
  radChart1.Series(0).Clear()
  radChart1.Series(0).DataYColumn = "Revenue"
  radChart1.Series(0).Name = "Months"
  radChart1.DataBind()
  'Detach Click event handler for the innermost chart
  'Telerik.WinControls.UI.RadChart.ChartClickEventHandler
  RemoveHandler radChart1.Click, AddressOf radChart1_Click
 End If
End Sub
```

[C#] Handling the Click Event

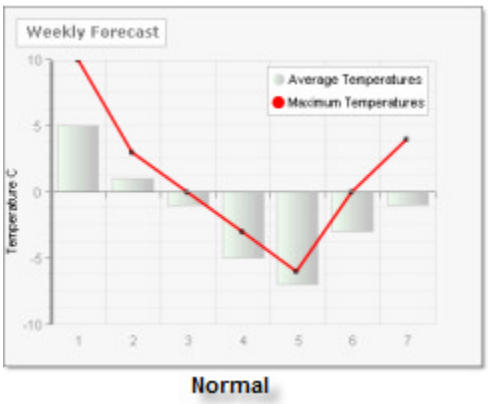
```
private void radChart1_Click(object sender, Telerik.Charting.ChartClickEventArgs args)
{
 if (args.Element.ActiveRegion.Parent is ChartSeriesItem &&
   (args.Element.ActiveRegion.Parent as ChartSeriesItem).Parent.Name == "Years")
 {
   radChart1.ChartTitle.TextBlock.Text = "Quarters";
   //a bar representing the revenue for a year was clicked
   _year = (args.Element as ChartSeriesItem).Index + 2003;
   //fill revenue values for the year whose bar was clicked
   revenueTableAdapter.FillByQuarter(this.chartDataSet.Revenue, _year);
   //setup chart properties with appopriate axis labels
   radChart1.PlotArea.XAxis.DataLabelsColumn = "Quarter";
   radChart1.Series[0].Clear();
   radChart1.Series[0].DataYColumn = "Revenue";
   radChart1.Series[0].Name = "Quarters";
   radChart1.DataBind();
 }
 else if (args.Element is ChartSeriesItem &&
    (args.Element as ChartSeriesItem).Parent.Name == "Quarters")
 {
   radChart1.ChartTitle.TextBlock.Text = "Months";
   //a bar representing the revenue for a year was clicked
   int quarter = (args.Element as ChartSeriesItem).Index + 1;
   //fill revenue values for the year whose bar was clicked
   revenueTableAdapter.FillByMonth(this.chartDataSet.Revenue, _year, quarter);
   //setup chart properties with appopriate axis labels
   radChart1.PlotArea.XAxis.DataLabelsColumn = "Month";
   radChart1.Series[0].Clear();
   radChart1.Series[0].DataYColumn = "Revenue";
   radChart1.Series[0].Name = "Months";
   radChart1.DataBind();
   //Detach Click event handler for the innermost chart
   //Telerik.WinControls.UI.RadChart.ChartClickEventHandler
   radChart1.Click -= new RadChart.ChartClickEventHandler(radChart1_Click);
 }
}
```

8. Press **Ctrl-F5** to run the application.

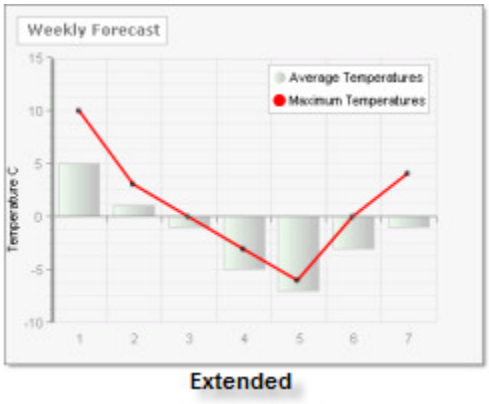


Extend the Displayable Area

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.



Normal



Extended

20.8 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 created and configured many of the chart elements programmatically, including the chart series, items, legend and chart title.

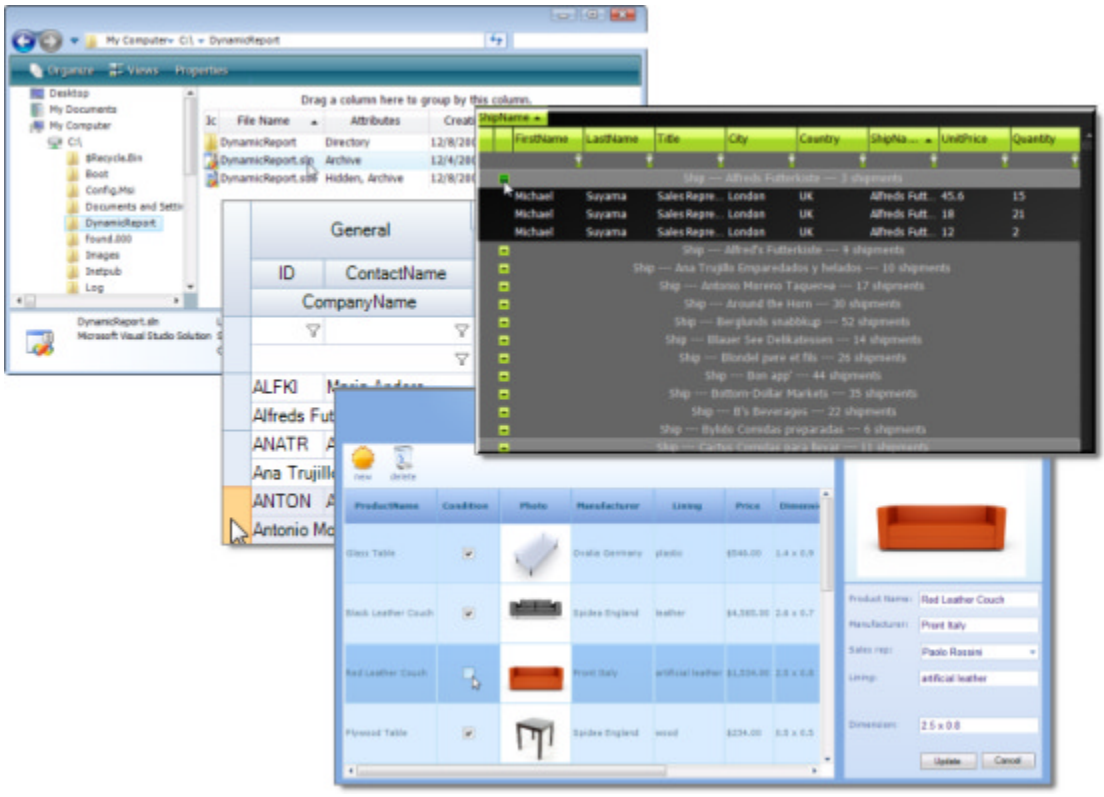
21 Grid

21.1 Objectives

- Get familiar with the RadGridView control by binding to a data source and configuring the grid using the Property Builder.
- Learn how to add columns for any data type at design-time and using code.
- Learn how to group, filter and sort data based on user input at runtime, design-time configuration and using code.
- Display hierarchical data from multiple related tables in the grid.
- Get low-level programmatic control over the RadGridView update process using Virtual Mode.
- Learn to use RadGridView with LINQ data sources to implement dynamic paging, sorting, and filtering on large datasets.
- Export RadGridView data.

21.2 Introduction

RadGridView is a powerful, highly performant grid component from Telerik developed on top of the innovative Telerik Presentation Foundation, which allows for unprecedented combination of performance, extensibility, customizability, and ease of use.



Some of the key features of RadGridView are:

- **Hierarchical data presentation** - RadGridView has the ability to represent hierarchical master-detail data. Its hierarchical schema could be set up either at design-time, at runtime using the control API, or handled automatically for you based on the structure of the data.

- **Easily customizable theming mechanism** - RadGridView is shipped with a consistent set of themes that allow you to easily build slick interfaces. Or you can use the codeless visual approach to build a new custom theme.
- **Grouping** - RadGridView allows easy implementation of multilevel grouping of data from a single table. Simply drag the column header(s) to the group panel on the top to define the grouping order and hierarchy. You can also programmatically group data using group-by expressions. Another unique feature is the ability to sort grouped data by single or multiple columns. Grouping programmatically also allows you to perform aggregate operations (e.g. sum, min, max, count, first and last) and to output custom formatted strings into the group heading.
- **Multi-column sorting** - in addition to the simple one-column sorting RadGridView allows you to sort data by several columns just like in Microsoft Excel. With the help of sorting expressions, you can also do custom sorting.
- **Filtering** - RadGridView can perform filtering operations for all columns that support filtering. For each column there will be a drop down menu with the available filter expressions
- **Column and Row resizing** - RadGridView supports convenient column and row resizing with features like real-time resizing, best fit sizing, automatic resizing of grid rows based on their content, and text wrapping.
- **Column reordering with drag-and-drop** - column reordering is a nice interface feature which lets users reorder columns, based on their personal preference. Telerik RadGridView allows users to quickly reorder columns at runtime by simply drag-and-dropping their headers, with visual indication of the header being dragged.
- **Keyboard navigation** - RadGridView can be navigated using the keyboard. You can focus on a grid with the TAB key, navigate through the rows, and edit cells.
- **Rich set of column types** - RadGridView supports the most commonly used column types to provide editing functionality.
- **Pinned rows and Pinned columns support** - RadGridView enhances the simple scrolling by supporting pinned rows and columns. You can scroll the grid data, while pinned rows stay visible and next to the header row; pinned columns stay visible and on the left of the grid.
- **Conditional formatting** - RadGridView enables you to apply conditional formatting to grid elements for enhanced readability and usability of the displayed data. Conditional formatting can be applied programmatically or by the user at run-time.
- **Context menu support** - The context menu provides extra usability and richness to any application. The default RadGridView context menu provides support for sorting, formatting, grouping, column selection and automatic column width adjustment. You can extend the context menu to add your own menu items and display your menus conditionally.
- **Saving and loading layout functionality**

21.3 Getting Started

RadGridView control's many features include flexible setup during design time, quick and easy binding to database data, using auto generated columns from the DataSource, and extensive Property Builder options. This lab will introduce the Telerik RadGridView control, demonstrate connecting to a DataSource, using the PropertyBuilder and applying a theme.



You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\RadGridViewIntro

This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**

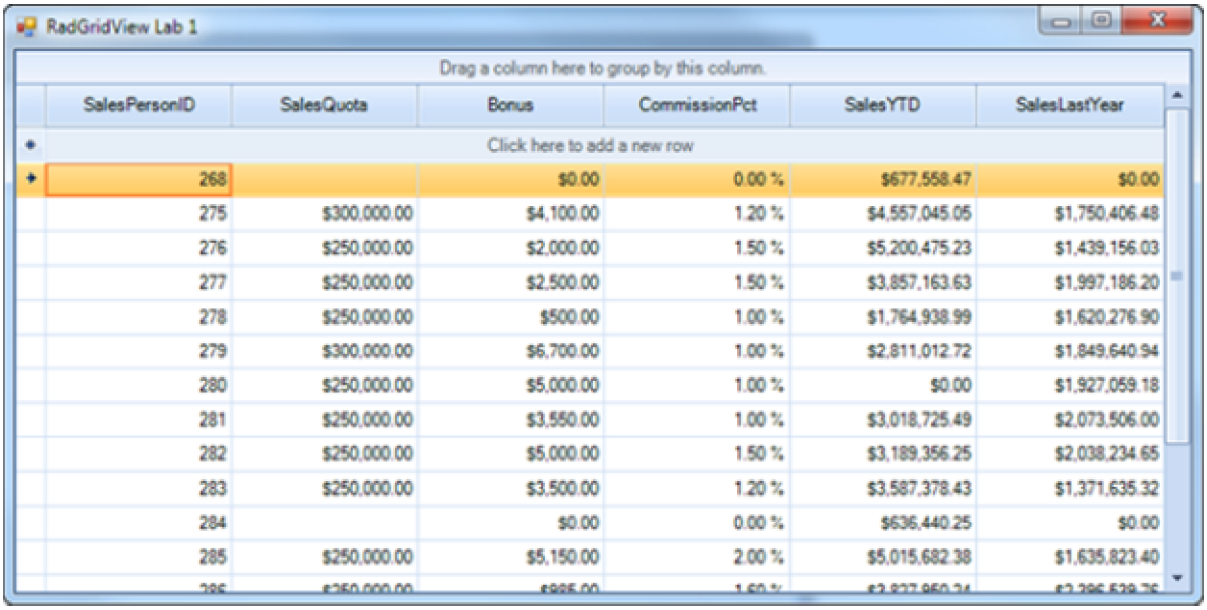
1. Start a new Windows Forms Project. Change the form size to 800x400, or something sufficiently large to view several columns of data.
2. Drag a RadGridView grid onto the form, and set the **Dock** property to "Fill".
3. Open the RadGridView's Smart Tag, and under "Choose DataSource" select "Add Project DataSource".
4. Use the DataSource wizard to set up a DataBase DataSource to the AdventureWorks database, and the Sales.SalesPerson table. Select all the columns in the table, and save the BindingSource to the project.

   The RadGridView should now be configured to show all of the columns in the table.
5. Use the Properties window for the RadGridView to set the **AutoSizeColumnsMode** property to "Fill".
6. Use the Smart Tag to open the Property Builder and uncheck the following columns: "TerritoryID", "rowguid", and "ModifiedDate" from the columns list.
7. Also in the Property Builder, Advanced tab, set the FormatString of the SalesQuota, Bonus, SalesYTD, and SalesLastYear to "{0:C}", the FormatString of CommissionPct to "{0:P}".
8. Close the Property Builder.

   The RadGridView property builder has a WYSIWYG-like capability: When you make modifications to the "preview" grid, changes are persisted in the designer. For example:

   - Show/hide columns with the Column Chooser, resize, rearrange columns, pin/unpin
   - Group-by column(s), filter, sort
   - Conditional formatting

   Using the Property Builder you can make changes quickly and get instant visual feedback. Using the "Advanced Settings" tab Properties Grid you can "fine tune" your settings. For example you can resize columns in the Property Builder, but if you wish your column to be exactly 250px wide you can use the Advanced Settings Properties Grid, or the standard Visual Studio Properties window.
9. Press Ctl-F5 to run the project and view the data in the RadGridView.



21.4 Column Manipulation

Additional columns can be added to the RadGridView table either visually by using the Property Builder, or programmatically. Setting the column expression and using different column data types allows for further flexibility within the RadGridView control, including the ability to add columns which are calculated at runtime from values in existing fields. This lab will demonstrate these techniques.

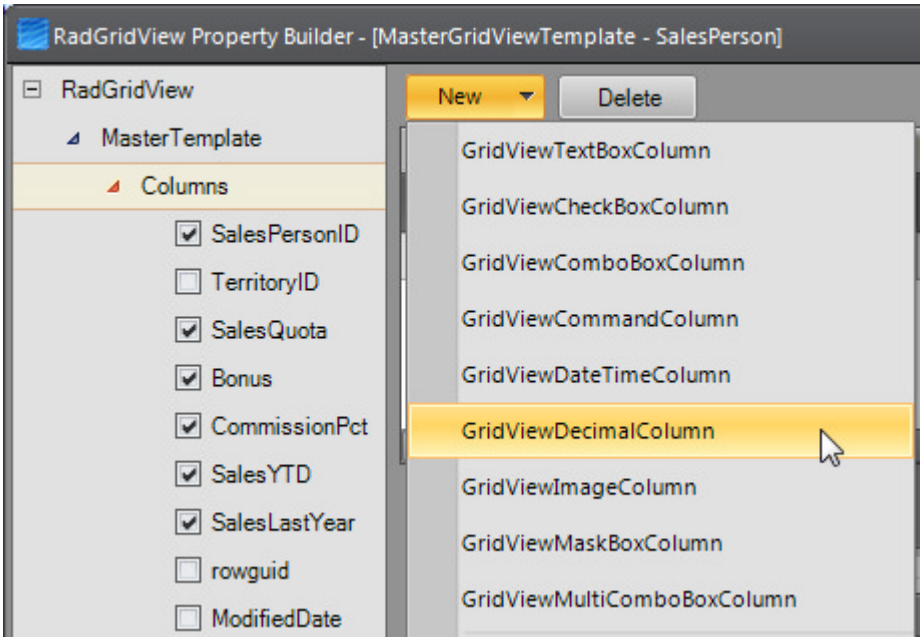


You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\AddingColumns

This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**

1. Start a new Windows Forms Project. Change the form size to 800x400, or something sufficiently large to view several columns of data.
2. Drag a RadGridView grid onto the form, and set the **Dock** property to "Fill".
3. Open the RadGridView's Smart Tag, and under "Choose DataSource" select "Add Project DataSource".
4. Use the DataSource wizard to set up a DataBase DataSource to the AdventureWorks database, and the Sales.SalesPerson table. Select all the columns in the table, and save the BindingSource to the project.
5. Use the Smart Tag to open the Property Builder.
6. Uncheck the following columns: "TerritoryID", "rowguid", and "ModifiedDate" from the columns list.
7. Optionally, you can change the Header Text of the columns to more meaningful names.
8. Use the Property Builder to add a new GridViewDecimalColumn to the grid. **Note**: Be sure to set focus to the grid itself by clicking on one of the Column names in the left column before trying to add a column. Name the new column's UniqueName to "PercentQuota" and the Header Text to "Percent of Quota".



9. Close the Property Builder.
10. Add an namespace reference to the code-behind of the form.

[VB] Adding Namespace Reference

```
Imports Telerik.WinControls.UI
```

[C#] Adding Namespace Reference

```
using Telerik.WinControls.UI;
```

11. Add the code below to the Load method of the form.

Notice that the "PercentQuota" expression is assigned in code. Then a new column is created completely from scratch and added to the columns collection. This code programmatically creates a CheckBox column which shows if a salesperson has reached over 15% of their sales quota for the year. Then column formats are set and finally the GridViewTemplate BestFitColumns() method is called. BestFitColumns calculates the best fit for each column based on the header text and data width so that neither header nor data is obscured.

[VB] Handling the Form Load Event

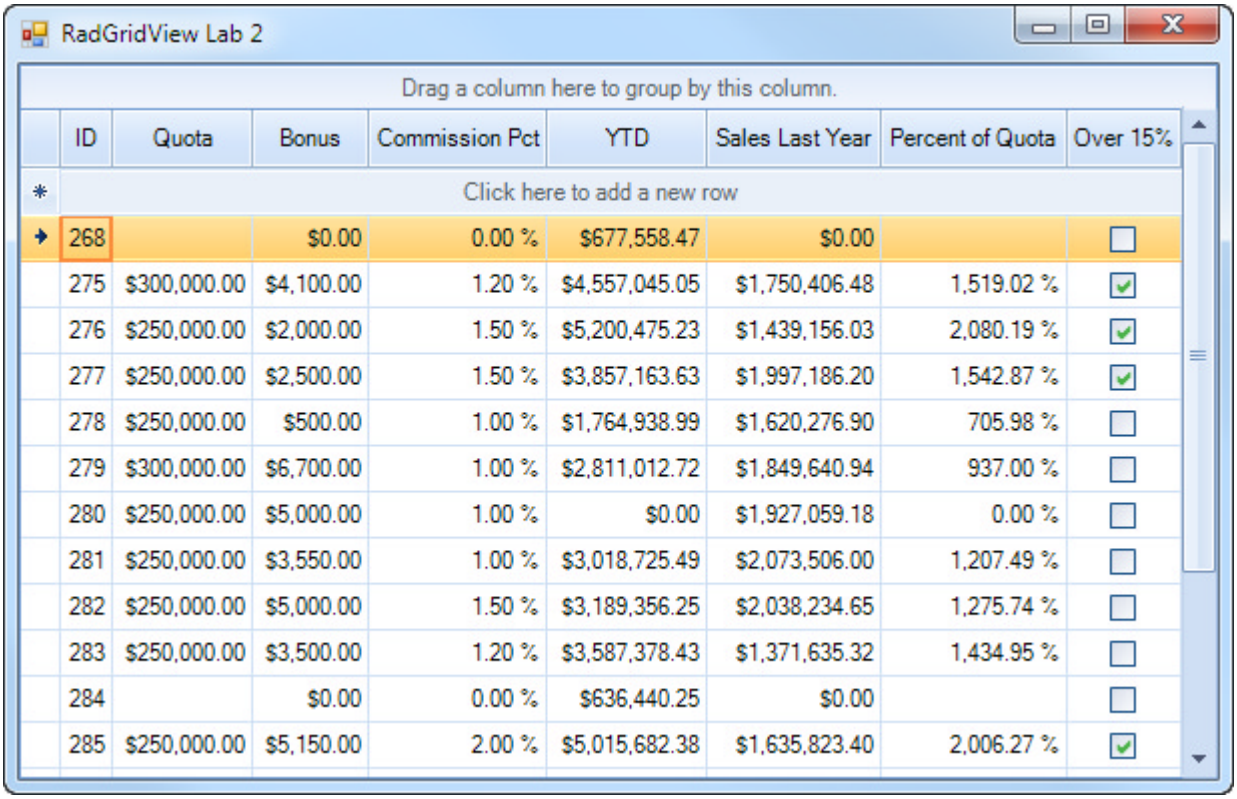
```
Private Sub RadGridViewLab2_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
' load the dataset
Me.salesPersonTableAdapter.Fill(Me.adventureWorksDataSet.SalesPerson)
' assign the "PercentQuota" expression, and set format to be a percentage
Me.radGridView1.Columns("PercentQuota").Expression = "SalesYTD/SalesQuota"
Me.radGridView1.Columns("PercentQuota").FormatString = "{0:P}"
' create a new "Over 15 Percent" checkbox
Dim checkboxColumn As GridViewCheckBoxColumn = New GridViewCheckBoxColumn()
checkboxColumn.Name = "CheckBoxColumn"
checkboxColumn.HeaderText = "Over 15%"
checkboxColumn.Width = 60
radGridView1.Columns.Add(checkboxColumn)
checkboxColumn.Expression = "PercentQuota > 15"
' set column formats
```

```
Me.radGridView1.Columns("SalesQuota").FormatString = "{0:C}"
Me.radGridView1.Columns("Bonus").FormatString = "{0:C}"
Me.radGridView1.Columns("SalesYTD").FormatString = "{0:C}"
Me.radGridView1.Columns("SalesLastYear").FormatString = "{0:C}"
Me.radGridView1.Columns("CommissionPct").FormatString = "{0:P}"
' get the best fit for each column based on header text and data
radGridView1.MasterTemplate.BestFitColumns()
End Sub
```

[C#] Handling the Form Load Event

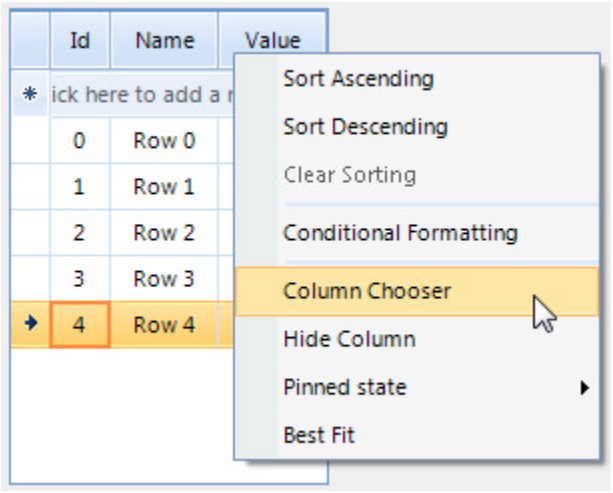
```
private void RadGridViewLab2_Load(object sender, EventArgs e)
{
// load the dataset
this.salesPersonTableAdapter.Fill(this.adventureWorksDataSet.SalesPerson);
// assign the "PercentQuota" expression, and set format to be a percentage
this.radGridView1.Columns["PercentQuota"].Expression = "SalesYTD/SalesQuota";
this.radGridView1.Columns["PercentQuota"].FormatString = "{0:P}";
// create a new "Over 15 Percent" checkbox
GridViewCheckBoxColumn checkboxColumn = new GridViewCheckBoxColumn();
checkboxColumn.Name = "CheckBoxColumn";
checkboxColumn.HeaderText = "Over 15%";
checkboxColumn.Width = 60;
radGridView1.Columns.Add(checkboxColumn);
checkboxColumn.Expression = "PercentQuota > 15";
// set column formats
this.radGridView1.Columns["SalesQuota"].FormatString = "{0:C}";
this.radGridView1.Columns["Bonus"].FormatString = "{0:C}";
this.radGridView1.Columns["SalesYTD"].FormatString = "{0:C}";
this.radGridView1.Columns["SalesLastYear"].FormatString = "{0:C}";
this.radGridView1.Columns["CommissionPct"].FormatString = "{0:P}";
// get the best fit for each column based on header text and data
radGridView1.MasterTemplate.BestFitColumns();
}
```

12. Run the project to view the column created in design view and the column created programmatically with the rest of the data.

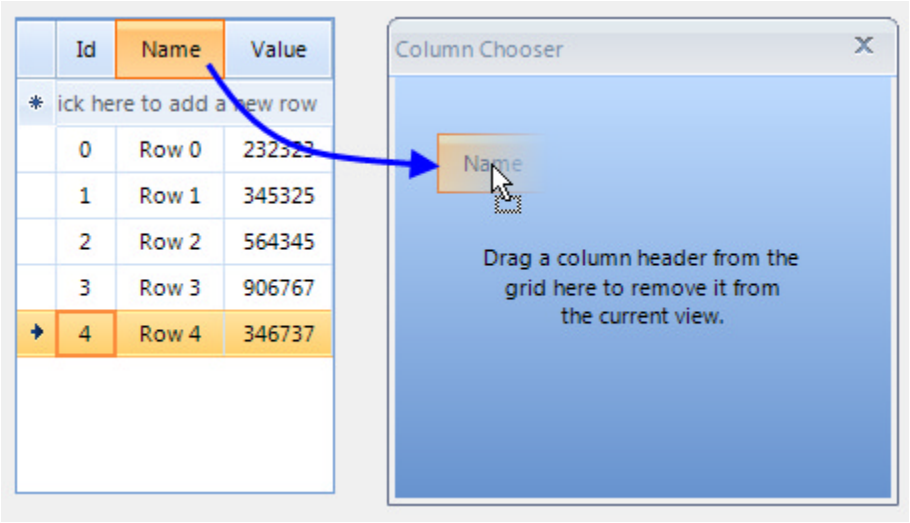


21.5 Column Chooser

The user can manage the visibility of all columns at runtime using the Column Chooser window. The user right-clicks one of the header cells and selects Column Chooser from the context menu:



The user can then drag column headings to the Column Chooser window.



The columns that are allowed to be visible in the Column Chooser can be configured at runtime. To prevent the column from being shown in the chooser, set the column's **VisibleInColumnChooser** property to false. If you want to prevent the column from being dragged off the grid, set the **AllowHide** property to false.

Gotcha! If VisibleInColumnChooser is set to False and AllowHide is true, the column will disappear when dragged onto the column chooser and the user will not be able to restore the column to the grid.

[VB] Disallowing a Column Dragging to Chooser

```
Private Sub Form1_Load(sender As Object, e As EventArgs)
 Me.productTableAdapter.Fill(Me.adventureWorksDataSet.Product)
 radGridView1.Columns(1).VisibleInColumnChooser = False
 radGridView1.Columns(1).AllowHide = False
End Sub
```

[C#] Disallowing a Column Dragging to Chooser

```
private void Form1_Load(object sender, EventArgs e)
{
 this.productTableAdapter.Fill(this.adventureWorksDataSet.Product);
 radGridView1.Columns[1].VisibleInColumnChooser = false;
 radGridView1.Columns[1].AllowHide = false;
}
```

You can disable the chooser all together for any grid view template by setting the **AllowColumnChooser** property. This removes the column chooser from the grid's context menu.

[VB] Disabling the Column Chooser

```
radGridView1.MasterTemplate.AllowColumnChooser = False
```

[C#] Disabling the Column Chooser

```
radGridView1.MasterTemplate.AllowColumnChooser = false;
```

Alternatively, you can display the chooser programmatically using the ColumnChooser **Show()** method.

[VB] Displaying the ColumnChooser

```
radGridView1.ColumnChooser.Show()
```

[C#] Displaying the ColumnChooser

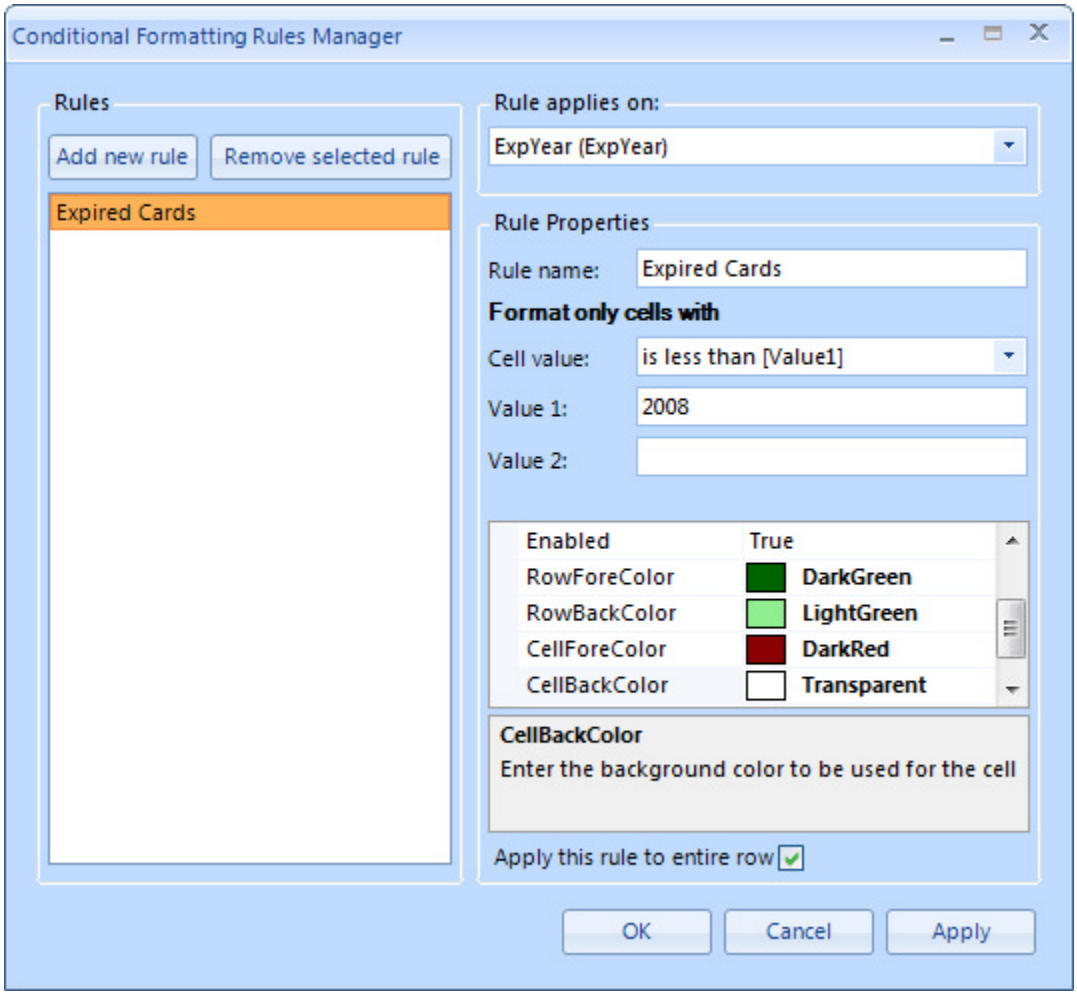
```
radGridView1.ColumnChooser.Show();
```

21.6 Conditional Formatting

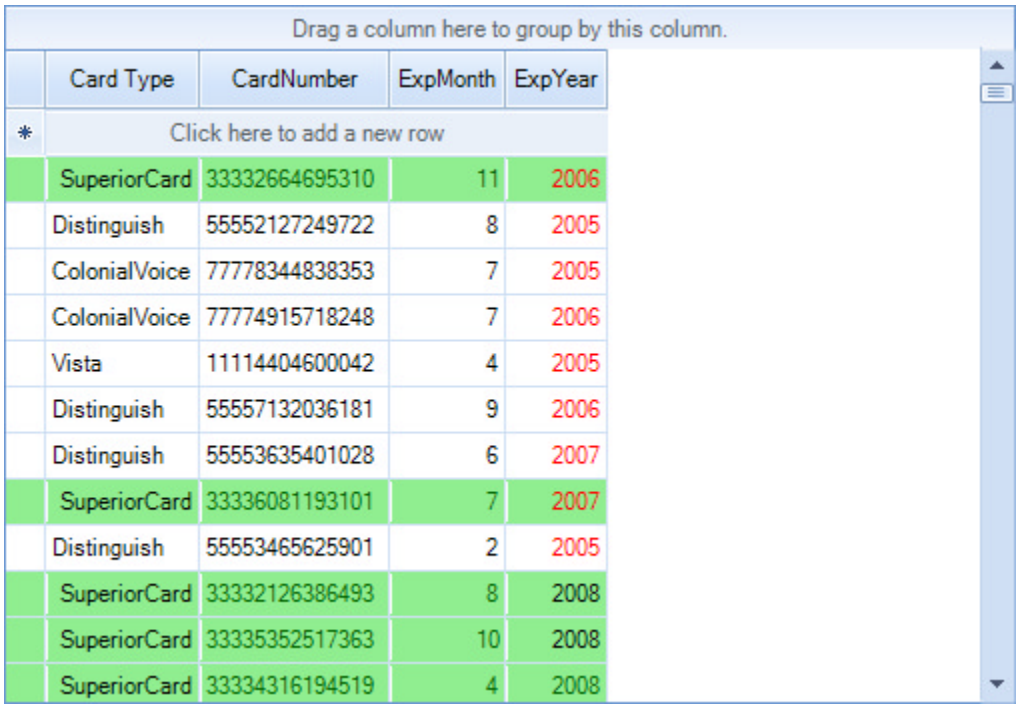
Cells and rows can be styled dynamically based on cell values. You can use the grids conditional formatting capability for many scenarios and fall back to cell formatting for unique or complex situations. Let's look first at the built-in conditional formatting mechanism. The user can initiate a "Conditional Formatting Rules Manager" dialog by right-clicking the grid and selecting "Conditional Formatting" from the context menu. Rules can also be created and applied in code.

User Initiated Conditional Formatting

The screenshot below shows the Conditional Formatting Rules Manager dialog. Each rule defined here has a column that the rule applies to, a comparison condition and one or two values (depending on the type of condition). Also notice the "Apply this rule to entire row" checkbox at the bottom of the form. The rule described here is named "Expired Cards" and fires when the "ExpYear" column has a value less than "2008".



If a rule evaluates to true, formatting is applied to the row and cell.



You can show the dialog programmatically by creating a **ConditionalFormattingForm** instance and calling the **Show()** method. The first parameter in the constructor takes the GridViewTemplate to apply formatting against. The second can be a GridViewDataColumn that the rule will apply to, or this can be left null.

You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\ConditionalFormatting

This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**

[VB] Showing the Conditional Formatting Dialog

```
Private Sub btnShowDialog_Click(sender As Object, e As EventArgs)
 Dim cf As New ConditionalFormattingForm(radGridView1.MasterTemplate, Nothing)
 cf.Show()
End Sub
```

[C#] Showing the Conditional Formatting Dialog

```
private void btnShowDialog_Click(object sender, EventArgs e)
{
 ConditionalFormattingForm cf =
   new ConditionalFormattingForm(radGridView1.MasterTemplate, null);
 cf.Show();
}
```

In fact, you can remove or replace items in the grid's context menu with your own items. The example below swaps out the "Conditional Formatting" item with a "My Conditional Formatting" item that displays the conditional formatting dialog as always topmost.

[VB] Replacing Grid Context Menu Items

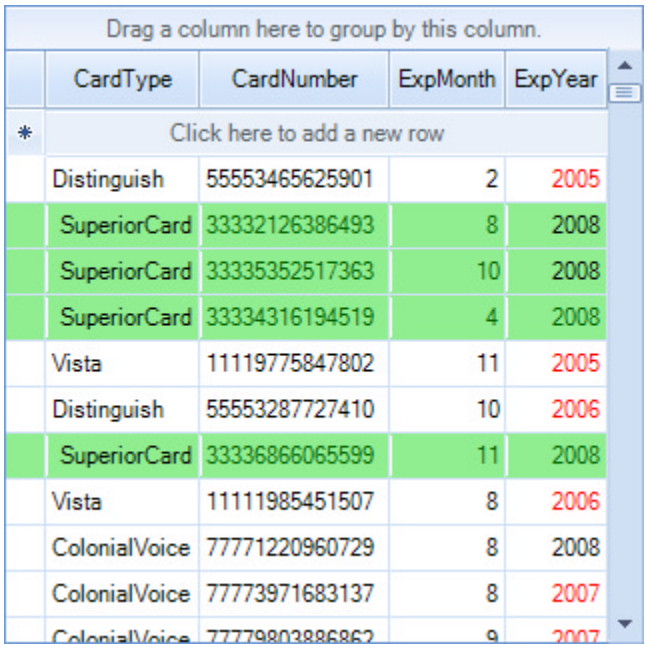
```
Private Sub radGridView1_ContextMenuOpening(sender As Object, e As
ContextMenuOpeningEventArgs)
 Dim i As Integer = 0
 While i < e.ContextMenu.Items.Count
  Dim item As RadMenuItemBase = DirectCast(e.ContextMenu.Items(i), RadMenuItemBase)
  If item.Text = "Conditional Formatting" Then
   e.ContextMenu.Items.Remove(item)
   Dim newItem As New RadMenuItem("My Conditional Formatting")
   newItem.Click += New EventHandler(newItem_Click)
   e.ContextMenu.Items.Insert(4, newItem)
   Exit While
  End If
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
End Sub
Sub newItem_Click(sender As Object, e As EventArgs)
 Dim cf As New ConditionalFormattingForm(radGridView1.MasterGridViewTemplate, Nothing)
 cf.TopMost = True
 cf.Show()
 cf.Focus()
End Sub
```

[C#] Replacing Grid Context Menu Items

```
private void radGridView1_ContextMenuOpening(object sender,
ContextMenuOpeningEventArgs e)
{
 for (int i = 0; i < e.ContextMenu.Items.Count; i++)
 {
   RadMenuItemBase item = (RadMenuItemBase)e.ContextMenu.Items[i];
   if (item.Text == "Conditional Formatting")
   {
     e.ContextMenu.Items.Remove(item);
     RadMenuItem newItem = new RadMenuItem("My Conditional Formatting");
     newItem.Click += new EventHandler(newItem_Click);
     e.ContextMenu.Items.Insert(4, newItem);
     break;
   }
 }
}
void newItem_Click(object sender, EventArgs e)
{
 ConditionalFormattingForm cf =
   new ConditionalFormattingForm(radGridView1.MasterGridViewTemplate, null);
 cf.TopMost = true;
 cf.Show();
 cf.Focus();
}
```

Programmatic Conditional Formatting

To create the parallel to the conditional formatting rules created by using the dialog, create ConditionalFormattingObject instances. The constructor takes the name of the rule, the condition type (i.e. Equal, Less Than, etc), the first value, the second value and if formatting should be applied to the entire row. The example below creates a rule that detects cards of type "SuperiorCard" and applies formatting only to the cell. A second rule looks for the "ExpYear" to be less than 2008 and formats the entire row.



[VB] Adding Rules Programmatically

```
' create a rule to highlight "SuperiorCard" card types
' -highlight the entire row
Dim cfo As New ConditionalFormattingObject("SuperiorCards", ConditionTypes.Equal,
"SuperiorCard", "", True)
cfo.RowBackColor = Color.LightGreen
cfo.RowForeColor = Color.DarkGreen
cfo.TextAlignment = ContentAlignment.MiddleRight
radGridView1.Columns("CardType").ConditionalFormattingObjectList.Add(cfo)
' hightlight cards with expiry year less than 2008.
' -highlight only the cell
Dim cfo2 As New ConditionalFormattingObject("Expired Cards", ConditionTypes.Less, "2008",
"", False)
cfo2.CellBackColor = Color.Transparent
cfo2.CellForeColor = Color.Red
cfo2.TextAlignment = ContentAlignment.MiddleRight
radGridView1.Columns("ExpYear").ConditionalFormattingObjectList.Add(cfo2)
```

[C#] Adding Rules Programmatically

```
// create a rule to highlight "SuperiorCard" card types
// -highlight the entire row
ConditionalFormattingObject cfo =
 new ConditionalFormattingObject("SuperiorCards", ConditionTypes.Equal, "SuperiorCard", "",
true);
cfo.RowBackColor = Color.LightGreen;
cfo.RowForeColor = Color.DarkGreen;
cfo.TextAlignment = ContentAlignment.MiddleRight;
radGridView1.Columns["CardType"].ConditionalFormattingObjectList.Add(cfo);
// hightlight cards with expiry year less than 2008.
// -highlight only the cell
ConditionalFormattingObject cfo2 =
 new ConditionalFormattingObject("Expired Cards", ConditionTypes.Less, "2008", "", false);
cfo2.CellBackColor = Color.Transparent;
cfo2.CellForeColor = Color.Red;
```

```
cfo2.TextAlignment = ContentAlignment.MiddleRight;
radGridView1.Columns["ExpYear"].ConditionalFormattingObjectList.Add(cfo2);
```

21.7 Cell and Row Formatting

If the built-in conditional formatting mechanism is insufficient because:

- The conditions are too complex.
- You want to format other than just cell and row colors and alignment.
- You need to compare values in other cells as part of evaluating the condition.
- You need special effects like alternate row colors.

...then the **CellFormatting and RowFormatting** events provide complete control over any visual aspect of the cell element and allows you to construct arbitrarily complex evaluations using any programmatic tool in the .NET arsenal. RowFormating and CellFormating events fire every time a visible row or cell is created. Even though you can create complex code in these event handlers, have a care for performance as these events will fire for all visible rows or cells.

The events are fired in the following order:

- The RowFormatting event is called after applying all theme properties and updating row elements state.
- The CellFormatting event is fired for every cell in the row.

To be more effective and to save memory, RadGridView uses virtualization of its elements. This means that a limited number of elements are created at a time. Only the rows that are currently visible on the screen have associated visual elements. When the grid is scrolled up the visual elements are reused. The formatting events are fired only for rows that have associated visual elements.

Cell Formatting

The example below makes extensive visual changes to the cell element based on the current column's data (CardType = "SuperiorCard") and another column (ExpYear < 2008).

The example first tests that the column is a **GridViewDataColumn** type. This lets us get the GridViewDataColumn Name so we can know what column we're on. Use CellElement.RowInfo.Cells ["MyColumnName"] to get at other columns. Another important point is that if you want to change the BackColor properties of the CellElement, you should set the **DrawFill** property to True.

Gotcha!

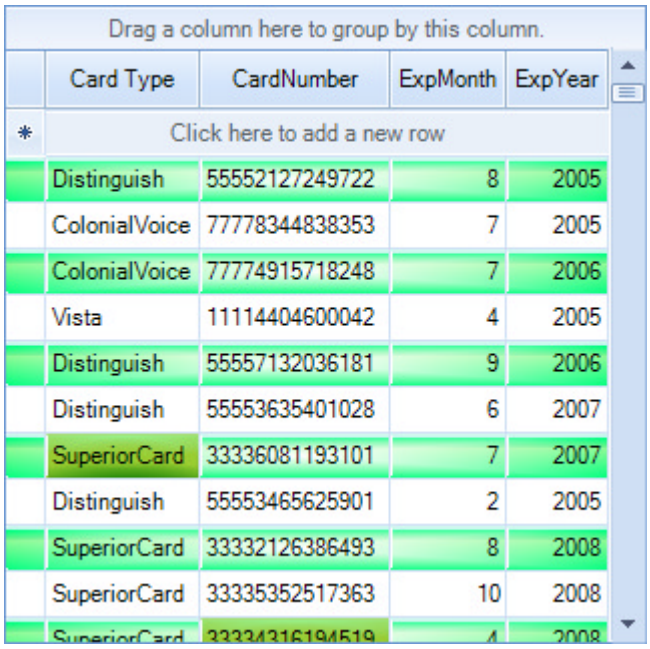
Be sure to set DrawFill back to false whenever you're not specifically accessing the BackColor. Forgetting to do this will cause anomalous painting of other cells in the column as you scroll.

You should always reset the modified properties when a condition is not met. In short, you should always have an "else" clause for every "if" clause.

You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\RowCellFormatting

This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**



[VB] Handling the CellFormatting Event

```
Private Sub radGridView1_CellFormatting(ByVal sender As Object, ByVal e As
CellFormattingEventArgs) Handles radGridView1.CellFormatting
' make sure the column is not the header and is a GridViewDataColumn
If TypeOf e.CellElement.ColumnInfo Is GridViewDataColumn AndAlso Not(TypeOf e.CellElement Is
GridHeaderCellElement) Then
Dim column As GridViewDataColumn = TryCast(e.CellElement.ColumnInfo, GridViewDataColumn)
' only make changes to the CardType column
If column.Name.Equals("CardType") AndAlso e.CellElement.RowInfo.Cells("ExpYear").Value IsNot
Nothing Then
' get value of the "ExpYear"
Dim year As Integer = CShort(Fix(e.CellElement.RowInfo.Cells("ExpYear").Value))
' make changes based on another column ("ExpYear") where the current
' column contains "SuperiorCard"
If (year < 2008) AndAlso (e.CellElement.Text.Equals("SuperiorCard")) Then
' must set DrawFill for background colors to show!
e.CellElement.DrawFill = True
' change any CellElement property
e.CellElement.BackColor = Color.PaleGreen
e.CellElement.BackColor2 = Color.LightGreen
e.CellElement.BackColor3 = Color.YellowGreen
e.CellElement.BackColor4 = Color.Green
e.CellElement.NumberOfColors = 4
e.CellElement.GradientStyle = GradientStyles.OfficeGlassRect
Else
' must reset the visual modifications
e.CellElement.ResetValue(LightVisualElement.DrawFillProperty, ValueResetFlags.Local)
e.CellElement.ResetValue(LightVisualElement.BackColorProperty, ValueResetFlags.Local)
e.CellElement.ResetValue(LightVisualElement.BackColor2Property, ValueResetFlags.Local)
e.CellElement.ResetValue(LightVisualElement.BackColor3Property, ValueResetFlags.Local)
e.CellElement.ResetValue(LightVisualElement.BackColor4Property, ValueResetFlags.Local)
e.CellElement.ResetValue(LightVisualElement.NumberOfColorsProperty, ValueResetFlags.Local)
e.CellElement.ResetValue(LightVisualElement.GradientStyleProperty, ValueResetFlags.Local)
End If
```

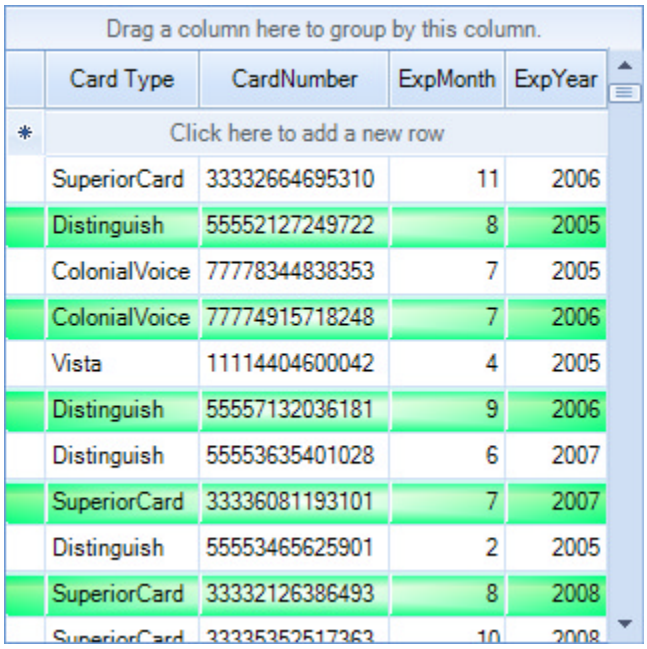
```
End If
End If
End Sub
```

[C#] Handling the CellFormatting Event

```
private void radGridView1_CellFormatting(object sender, CellFormattingEventArgs e)
{
// make sure the column is not the header and is a GridViewDataColumn
if (e.CellElement.ColumnInfo is GridViewDataColumn && !(e.CellElement is
GridHeaderCellElement))
{
GridViewDataColumn column = e.CellElement.ColumnInfo as GridViewDataColumn;
// only make changes to the CardType column
if (column.Name.Equals("CardType") && e.CellElement.RowInfo.Cells["ExpYear"].Value != null)
{
// get value of the "ExpYear"
int year = (Int16)e.CellElement.RowInfo.Cells["ExpYear"].Value;
// make changes based on another column ("ExpYear") where the current
// column contains "SuperiorCard"
if ((year < 2008) && (e.CellElement.Text.Equals("SuperiorCard")))
{
// must set DrawFill for background colors to show!
e.CellElement.DrawFill = true;
// change any CellElement property
e.CellElement.BackColor = Color.PaleGreen;
e.CellElement.BackColor2 = Color.LightGreen;
e.CellElement.BackColor3 = Color.YellowGreen;
e.CellElement.BackColor4 = Color.Green;
e.CellElement.NumberOfColors = 4;
e.CellElement.GradientStyle = GradientStyles.OfficeGlassRect;
}
else
{
//must reset the visual modifications
e.CellElement.ResetValue(LightVisualElement.DrawFillProperty, ValueResetFlags.Local);
e.CellElement.ResetValue(LightVisualElement.BackColorProperty, ValueResetFlags.Local);
e.CellElement.ResetValue(LightVisualElement.BackColor2Property, ValueResetFlags.Local);
e.CellElement.ResetValue(LightVisualElement.BackColor3Property, ValueResetFlags.Local);
e.CellElement.ResetValue(LightVisualElement.BackColor4Property, ValueResetFlags.Local);
e.CellElement.ResetValue(LightVisualElement.NumberOfColorsProperty, ValueResetFlags.Local);
e.CellElement.ResetValue(LightVisualElement.GradientStyleProperty, ValueResetFlags.Local);
}
}
}
}
```

Row Formatting

The RowFormatting event is particularly useful for applying alternating row styles to your grid. Use the RowElement **IsOdd** property to detect every other row. Use the RowElement to style the entire row using any of the TPF visual properties, e.g. back colors, gradients, etc. As with cell formatting, you can make the logic as complex as you need.



[VB] Handling the RowFormatting Event

```
Private Sub radGridView1_RowFormatting(ByVal sender As Object, ByVal e As
RowFormattingEventArgs) Handles radGridView1.RowFormatting
If toggleButtonEnableAlternationRowColor.ToggleState = ToggleState.On Then
If e.RowElement.IsOdd Then
e.RowElement.DrawFill = True
e.RowElement.BackColor = Color.LightGreen
e.RowElement.BackColor2 = Color.PaleTurquoise
e.RowElement.BackColor3 = Color.PaleGreen
e.RowElement.BackColor4 = Color.SpringGreen
e.RowElement.GradientStyle = GradientStyles.Vista
Else
e.RowElement.ResetValue(LightVisualElement.BackColorProperty, ValueResetFlags.Local)
e.RowElement.ResetValue(LightVisualElement.BackColor2Property, ValueResetFlags.Local)
e.RowElement.ResetValue(LightVisualElement.BackColor3Property, ValueResetFlags.Local)
e.RowElement.ResetValue(LightVisualElement.BackColor4Property, ValueResetFlags.Local)
e.RowElement.ResetValue(LightVisualElement.GradientStyleProperty, ValueResetFlags.Local)
e.RowElement.ResetValue(LightVisualElement.DrawFillProperty, ValueResetFlags.Local)
End If
End If
End Sub
```

[C#] Handling the RowFormatting Event

```
private void radGridView1_RowFormatting(object sender, RowFormattingEventArgs e)
{
if (toggleButtonEnableAlternationRowColor.ToggleState == ToggleState.On)
{
if (e.RowElement.IsOdd)
{
e.RowElement.DrawFill = true;
e.RowElement.BackColor = Color.LightGreen;
e.RowElement.BackColor2 = Color.PaleTurquoise;
e.RowElement.BackColor3 = Color.PaleGreen;
e.RowElement.BackColor4 = Color.SpringGreen;
```

```
e.RowElement.GradientStyle = GradientStyles.Vista;
}
else
{
e.RowElement.ResetValue(LightVisualElement.BackColorProperty, ValueResetFlags.Local);
e.RowElement.ResetValue(LightVisualElement.BackColor2Property, ValueResetFlags.Local);
e.RowElement.ResetValue(LightVisualElement.BackColor3Property, ValueResetFlags.Local);
e.RowElement.ResetValue(LightVisualElement.BackColor4Property, ValueResetFlags.Local);
e.RowElement.ResetValue(LightVisualElement.GradientStyleProperty, ValueResetFlags.Local);
e.RowElement.ResetValue(LightVisualElement.DrawFillProperty, ValueResetFlags.Local);
}
}
}
```

Another way you can get alternating row color automatically is to set the EnableAlternatingRowColor property to true. The color used on every other row can be set by setting the AlternatingRowColor property of the TableElement.

[VB] Enabling Alternating Row Color

```
radGridView1.EnableAlternatingRowColor = True

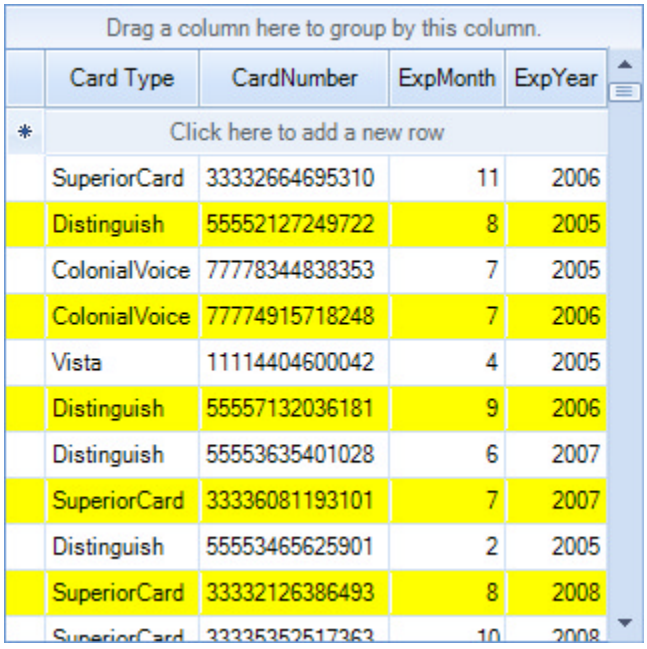
radGridView1.TableElement.AlternatingRowColor = Color.Yellow
```

[C#] Enabling Alternating Row Color

```
radGridView1.EnableAlternatingRowColor = true;

radGridView1.GridElement.AlternatingRowColor = Color.Yellow;
```

The visual results are a little flashy, but these properties can be set at design time without having to code.



21.8 Custom Cells and Elements

Overview

If the existing cells are too limited for a specific job you can create a custom cell type and add any elements that suit your purpose. To do this you need to inherit from the **GridDataCellElement** and override some of its

methods. Thanks to the UI virtualization mechanism, of RadGridView only the currently visible cells are created and they are further reused when needed. In order to avoid your misusage of your custom cell is it best to create a custom column that will be compatible only with the defined custom cell.

Additionally, in order to prevent troubles because of the virtualization it is best to create a custom column where you

The first step is to create your GridDataCellElement descendant. A typical implementation would include a public constructor that passes the column and row and a override of the CreateChildElements method so you can create your own elements inside the cell.

[VB] Minimal Implementation of GridDataCellElement Descendant

```
Public Class MyCell
 Inherits GridDataCellElement
 Public Sub New(column As GridViewColumn, row As GridRowElement)
  MyBase.New(column, row)
 End Sub
 Protected Overloads Overrides Sub CreateChildElements()
  MyBase.CreateChildElements()
 End Sub
End Class
```

[C#] Minimal Implementation of GridDataCellElement Descendant

```
public class MyCell : GridDataCellElement
{
 public ColorSwatchCellElement(GridViewColumn column, GridRowElement row) :
   base(column, row)
 {
 }
 protected override void CreateChildElements()
 {
   base.CreateChildElements();
 }
}
```

Other GridDataCellElement methods you may want to override:

- **SetContentCore(object value)**: To set the cell text.
- **SetContent()**: To create or modify any cell content.
- **ArrangeOverride(SizeF finalSize)**: To handle layout requirements specific to your cell type.
- **ThemeEffectiveType** - to make the element themable
- **IsCompatible** - defines with which column types the cell is compatible with

The example below uses the "Color" column from the Products table and creates a special ColorSwatchCellElement custom cell type. Clicking on the cell displays the color dialog and lets the user change colors.



Defining a Custom Cell Type



You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\CustomCells

1. In a new WinForms application, drop a **RadGridView** on the default form and set the **Dock** property to "Fill".
2. Bind the AdventureWorks "Products" table to the grid and include only the "ProductID", "Name" and "Color" columns.
3. First create a new class "ColorSwatchElement.cs". Change the declaration of the class to be public, descend from GridDataCellElement and add a constructor that takes a GridViewColumn and GridRowElement as parameters.

   [VB] Initial ColorSwatchCellElement Definition

```
Public Class ColorSwatchCellElement
 Inherits GridDataCellElement
 Public Sub New(column As GridViewColumn, row As GridRowElement)
  MyBase.New(column, row)
 End Sub
End Class
```

   [C#] Initial ColorSwatchCellElement Definition

```
public class ColorSwatchCellElement : GridDataCellElement
{
 public ColorSwatchCellElement(GridViewColumn column, GridRowElement row) :
   base(column, row)
 {
 }
}
```

4. Add a new private member "_swatch" and make it a **LightVisualElement** type. Then override the CreateChildElements() method and create an instance of the LightVisualElement and set properties before adding to the **Children** collection:
   - **DrawFill** = True
   - **Font** = a new 7 point "Arial" font

- **Shape** = new RoundRectShape with a 5 point radius
- **GradientStyle** = "Linear"
- **NumberOfColors** = 2
- **MouseDown** = Remove and assign a new MouseDown event handler

[VB] Overiding the CreateChildElements Method

```
Private _swatch As LightVisualElement
Protected Overloads Overrides Sub CreateChildElements()
 MyBase.CreateChildElements()
 _swatch = New LightVisualElement()
 _swatch.DrawFill = True
 _swatch.Font = New Font("Arial", 7F, FontStyle.Regular)
 _swatch.Shape = New RoundRectShape(5)
 _swatch.GradientStyle = GradientStyles.Linear
 _swatch.NumberOfColors = 2
RemoveHandler _swatch.MouseDown, AddressOf _swatch_MouseDown
 _swatch.MouseDown += New MouseEventHandler(_swatch_MouseDown)
 Me.Children.Add(_swatch)
End Sub
```

[C#] Overiding the CreateChildElements Method

```
private LightVisualElement _swatch;
protected override void CreateChildElements()
{
 base.CreateChildElements();
 _swatch = new LightVisualElement();
 _swatch.DrawFill = true;
 _swatch.Font = new Font("Arial", 7F, FontStyle.Regular);
 _swatch.Shape = new RoundRectShape(5);
 _swatch.GradientStyle = GradientStyles.Linear;
 _swatch.NumberOfColors = 2;
_swatch.MouseDown -= new MouseEventHandler(_swatch_MouseDown);
 _swatch.MouseDown += new MouseEventHandler(_swatch_MouseDown);
 this.Children.Add(_swatch);
}
```

5. Override the ArrangeOverride method in order to size the element according to the cell size

[VB.NET] Overriding ArrangeOverride

```
Protected Overrides Function ArrangeOverride(finalSize As SizeF) As SizeF
        MyBase.ArrangeOverride(finalSize)
        'size the element according to the cell size
        _swatch.Arrange(New RectangleF(3, 3, finalSize.Width - 6, finalSize.Height - 6))
        Return finalSize
End Function
```

[C#] Overriding ArrangeOverride

```
protected override SizeF ArrangeOverride(SizeF finalSize)
{
base.ArrangeOverride(finalSize);
//size the element according to the cell size
_swatch.Arrange(new RectangleF(3, 3, finalSize.Width - 6, finalSize.Height - 6));
return finalSize;
}
```

6. Override the **SetContentCore()** method. Set the cell **Text** property blank.

[VB] Overriding the SetContentCore Method

```
Protected Overloads Overrides Sub SetContentCore(value As Object)
 ' blank out the cell's text, only show on swatch element
 Me.Text = ""
End Sub
```

[C#] Overriding the SetContentCore Method

```
protected override void SetContentCore(object value)
{
 // blank out the cell's text, only show on swatch element
 this.Text = "";
}
```

7. Override the **SetContent()** method.

In this method you want to set the "swatch" BackColors and Text properties to a default values of "White" and a blank string. Then test if the cell's Value has something in it and that the Value is not equal to the string "Transparent". Set BackColor2 using the Color.FromName() method and passing the cell value. Also set the Text to the color name stored in the cell's Value.

In the latter part of the method, set the column width to the width of the color name string if it is larger.

[VB] Overriding the SetContent Method

```
Public Overloads Overrides Sub SetContent()
 MyBase.SetContent()
 ' set default properties
_swatch.BackColor = Color.White
 _swatch.BackColor2 = Color.White
 _swatch.Text = [String].Empty
 ' only process if there is a value and its not "Transparent"
 If Not Me.Value.Equals(System.DBNull.Value) AndAlso (Not Me.Value.ToString().Equals
("Transparent")) Then
  Dim colorName As String = Me.Value.ToString()
  ' convert to a Color type and assign to background color
  _swatch.BackColor2 = Color.FromName(colorName)
  ' set the readable color name
  _swatch.Text = colorName
 End If

If Me.ElementTree IsNot Nothing Then
' get the size of the color name string
Dim size As SizeF = SizeF.Empty
Using g As Graphics = Me.ElementTree.Control.CreateGraphics()
size = g.MeasureString(_swatch.Text, Me.Font)
End Using
' get the current column width
Dim width As Integer = Me.GridControl.Columns(Me.ColumnIndex).Width
' set column width to use the column or measured
' color name string width, whichever is widest
Me.GridControl.Columns(Me.ColumnIndex).Width = Math.Max(width, CInt(Fix(size.Width)))
End If
End Sub
```

[C#] Overriding the SetContent Method

```
public override void SetContent()
{
 base.SetContent();
 // set default properties
_swatch.BackColor = Color.White;
 _swatch.BackColor2 = Color.White;
 _swatch.Text = String.Empty;
 // only process if there is a value and its not "Transparent"
 if (!this.Value.Equals(System.DBNull.Value) &&
    (!this.Value.ToString().Equals("Transparent")))
 {
   string colorName = this.Value.ToString();
   // convert to a Color type and assign to background color
   _swatch.BackColor2 = Color.FromName(colorName);
   // set the readable color name
   _swatch.Text = colorName;
 }
if (this.ElementTree != null)
{
 // get the size of the color name string
 SizeF size = SizeF.Empty;
 using (Graphics g = this.ElementTree.Control.CreateGraphics())
 {
   size = g.MeasureString(_swatch.Text, this.Font);
 }
 // get the current column width
 int width = this.GridControl.Columns[this.ColumnIndex].Width;
 // set column width to use the column or measured
 // color name string width, whichever is widest
 this.GridControl.Columns[this.ColumnIndex].Width = Math.Max(width, (int)size.Width);
}
}
```

8. Handle the MouseDown event for the "_swatch" element. First configure a RadColorDialog to show only the Web tab heading and change the heading to 'Swatch Colors'. Set the initial color to whatever color is already in the swatch background, show the dialog and if the user clicks OK, accept the selected color, convert it to a known color name and assign the color name to the cell Value property. GetColorName() is a private method that we will define next.

[VB] Handling the MouseDown Event

```
Sub _swatch_MouseDown(sender As Object, e As MouseEventArgs)
 ' configure color dialog to only show web colors
 Dim colorDialog As New RadColorDialog()
 colorDialog.ColorDialogForm.ShowBasicColors = False
 colorDialog.ColorDialogForm.ShowProfessionalColors = False
 colorDialog.ColorDialogForm.ShowSystemColors = False
 colorDialog.ColorDialogForm.ShowCustomColors = False
 colorDialog.ColorDialogForm.ShowHEXColorValue = False
 colorDialog.ColorDialogForm.AllowColorPickFromScreen = False
 colorDialog.ColorDialogForm.AllowColorSaving = False
 colorDialog.ColorDialogForm.WebTabHeading = "Swatch Colors"
 ' set initial dialog color
 colorDialog.SelectedColor = _swatch.BackColor2
 If colorDialog.ShowDialog() = DialogResult.OK Then
  ' set value to known color name of selected color
```

```
  Me.Value = GetColorName(colorDialog.SelectedColor)
 End If
End Sub
```

[C#] Handling the MouseDown Event

```
void _swatch_MouseDown(object sender, MouseEventArgs e)
{
 // configure color dialog to only show web colors
 RadColorDialog colorDialog = new RadColorDialog();
 colorDialog.ColorDialogForm.ShowBasicColors = false;
 colorDialog.ColorDialogForm.ShowProfessionalColors = false;
 colorDialog.ColorDialogForm.ShowSystemColors = false;
 colorDialog.ColorDialogForm.ShowCustomColors = false;
 colorDialog.ColorDialogForm.ShowHEXColorValue = false;
 colorDialog.ColorDialogForm.AllowColorPickFromScreen = false;
 colorDialog.ColorDialogForm.AllowColorSaving = false;
 colorDialog.ColorDialogForm.WebTabHeading = "Swatch Colors";
 // set initial dialog color
 colorDialog.SelectedColor = _swatch.BackColor2;
 if (colorDialog.ShowDialog() == DialogResult.OK)
 {
   // set value to known color name of selected color
   this.Value = GetColorName(colorDialog.SelectedColor);
 }
}
```

9. Implement the GetColorName() helper method to return a meaningful color name for a given Color. Notice that we only recognize web color names and bypass the system colors in the KnownColor enumeration.

> Color.ToString() will just return a string representation of the actual RGB Hexidecimal values, not a string representation of the "known color".

[VB] Implementing the GetColorName Helper Method

```
Public Function GetColorName(color As Color) As String
 For Each knownColor As KnownColor In [Enum].GetValues(GetType(KnownColor))
  ' only look for web colors, not system colors
  If (knownColor > KnownColor.Transparent) AndAlso (knownColor < KnownColor.ButtonFace)
   If color.ToArgb() = Color.FromKnownColor(knownColor).ToArgb() Then
    Return knownColor.ToString()
   End If
  End If
 Next
 Return KnownColor.Transparent.ToString()
End Function
```

[C#] Implementing the GetColorName Helper Method

```
public string GetColorName(Color color)
{
 foreach (KnownColor knownColor in Enum.GetValues(typeof(KnownColor)))
 {
   // only look for web colors, not system colors
   if ((knownColor > KnownColor.Transparent) && (knownColor < KnownColor.ButtonFace))
   {
     if (color.ToArgb() == Color.FromKnownColor(knownColor).ToArgb())
     {
       return knownColor.ToString();
```

```
        }
      }
    }
    return KnownColor.Transparent.ToString();
}
```

10. Override the ThemeEffectiveType in order to make the cell themable

[VB.NET] Override ThemeEffectiveType

```
Protected Overrides ReadOnly Property ThemeEffectiveType() As Type
        Get
                Return GetType(GridDataCellElement)
        End Get
End Property
```

[C#] Override ThemeEffectiveType

```
protected override Type ThemeEffectiveType
{
get
{
return typeof(GridDataCellElement);
}
}
```

11. Create a new class that inherits from GridViewDataColumn in order to create the custom column that we have mentioned in the begging of this example. In this class all we need is to override the GetCellType to make it use the custom cell that we have created:

[VB.NET] Create the custom column

```
Imports Telerik.WinControls.UI
Class ColorSwatchColumn
Inherits GridViewDataColumn
Public Sub New(ByVal fieldName As String)
MyBase.New(fieldName)
End Sub
Public Overrides Function GetCellType(ByVal row As GridViewRowInfo) As Type
If TypeOf row Is GridViewDataRowInfo Then
Return GetType(ColorSwatchCellElement)
End If
Return MyBase.GetCellType(row)
End Function
End Class
```

[VB.NET] Create the custom column

```
class ColorSwatchColumn : GridViewDataColumn
{
public ColorSwatchColumn(string fieldName)
: base(fieldName)
{
}
public override Type GetCellType(GridViewRowInfo row)
{
if (row is GridViewDataRowInfo)
{
return typeof(ColorSwatchCellElement);
}
```

```
return base.GetCellType(row);
}
}
```

12. Finally, go back to the custom cell that we have created and override its IsCompatible mehtod in order to make it compatible with the newly created custom column:

Example Title

```
Public Overrides Function IsCompatible(data As GridViewColumn, context As Object) As Boo
        Return TypeOf data Is ColorSwatchColumn AndAlso TypeOf context Is GridDataRowEle
End Function
```

Example Title

```
public override bool IsCompatible(GridViewColumn data, object context)
{
return data is ColorSwatchColumn && context is GridDataRowElement;
}
```

Using Your Custom Cell Type

1. In the code-behind for the default form of the project, in the Load event remove the existing Color column, create new instance of the ColorSwatchColumn with **Name** = "Color" in order to make it load the data from the data source and this column to the Columns collection of RadGridView:

[VB] Handling the CreateCell Event

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs) Handles MyBase.Load
Me.productTableAdapter.Fill(Me.adventureWorksDataSet1.Product)
radGridView1.MasterTemplate.BestFitColumns()
radGridView1.Columns.Remove("Color")
Dim col As New ColorSwatchColumn("Color")
radGridView1.Columns.Add(col)
End Sub
```

[C#] Handling the CreateCell Event

```
private void Form1_Load(object sender, EventArgs e)
{
this.productTableAdapter.Fill(this.adventureWorksDataSet1.Product);
radGridView1.MasterTemplate.BestFitColumns();
radGridView1.Columns.Remove("Color");
ColorSwatchColumn col = new ColorSwatchColumn("Color");
radGridView1.Columns.Add(col);
}
```

2. Run and test the application.

21.9 Grouping, Sorting and Filtering

It is often useful to organize data for a clearer presentation. The RadGridView control allows you to perform this organization either at run-time, through options available to the user, or beforehand from inside your project. Grouping allows data to be organized according to commonalities between records, filtering shows only data meeting certain criteria, and sorting changes the order based on a particular field. The following labs will demonstrate how these functions can be performed by the user at run-time, from inside the designer, or programmatically using the API.

Grouping

User Grouping at Runtime

Grouping is supported at runtime by dragging and dropping column names into the Grouping Panel, using the Property Builder to perform the drag and drop grouping ahead of time, or programmatically from the code-behind. We will use the same project to perform these grouping functions, but demonstrate each type separately.

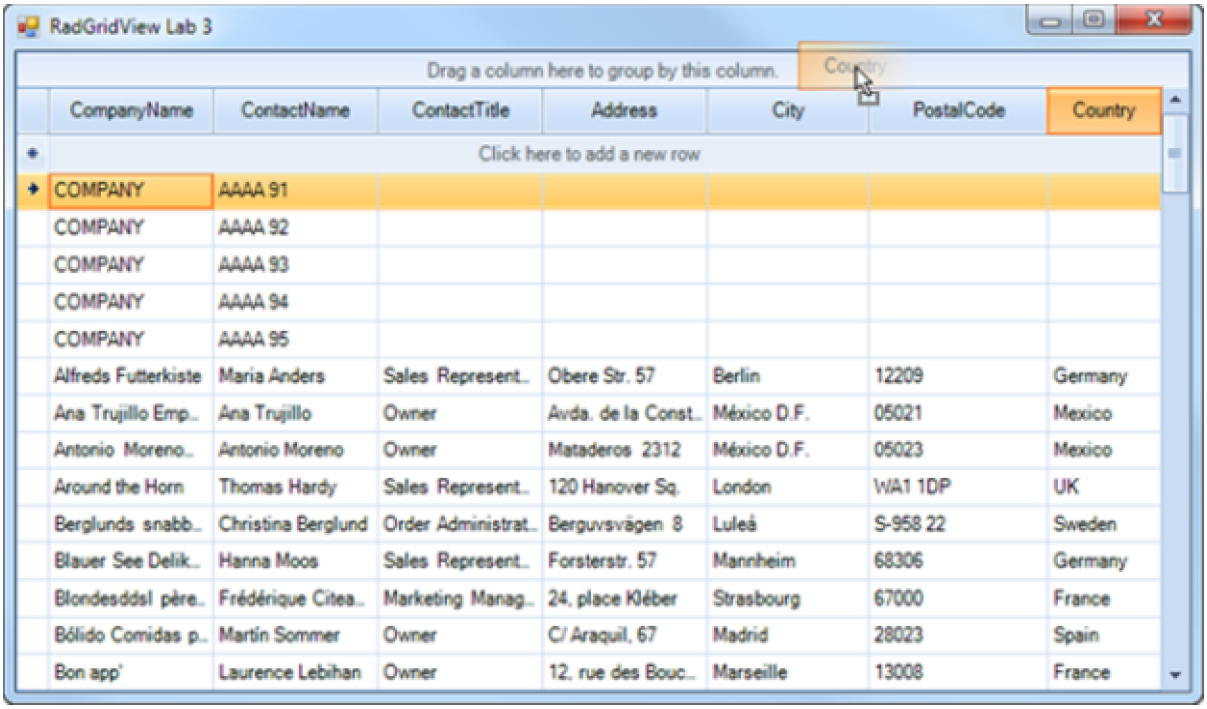


You can find the complete source for this project at:

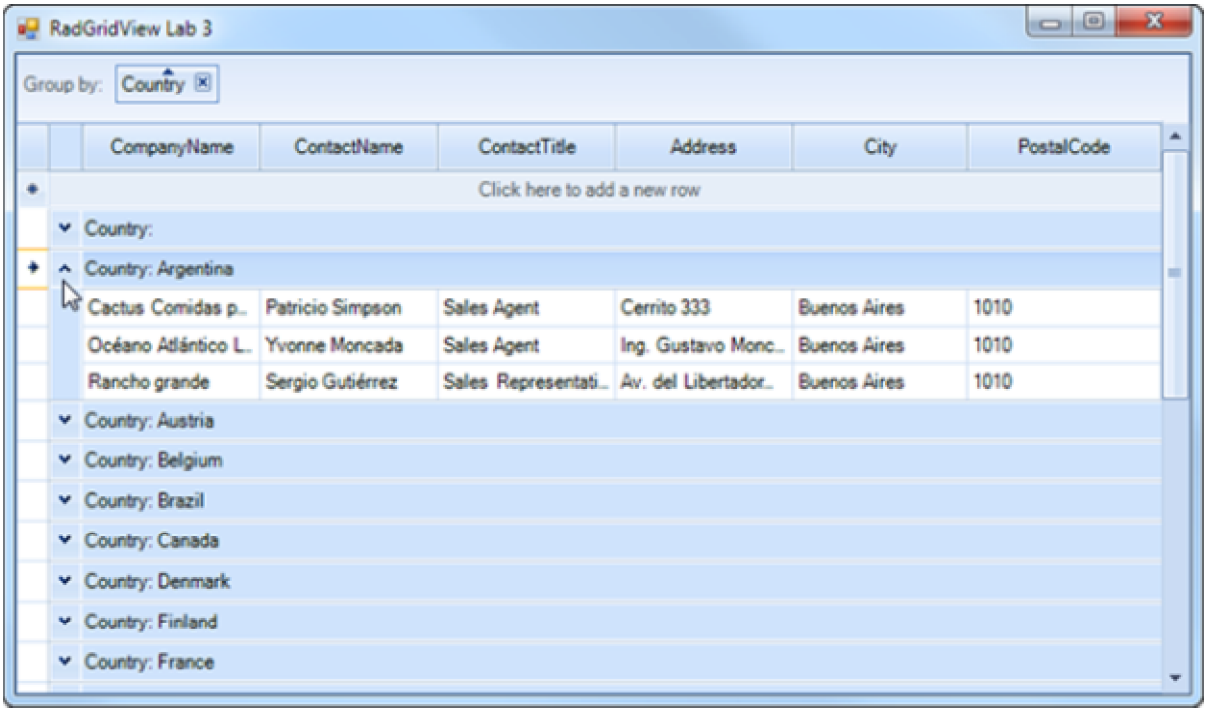
\Grid\<VB|CS>\RadGridView\Grouping

This example requires that you have pre installed NorthWind data base on your SQL server.

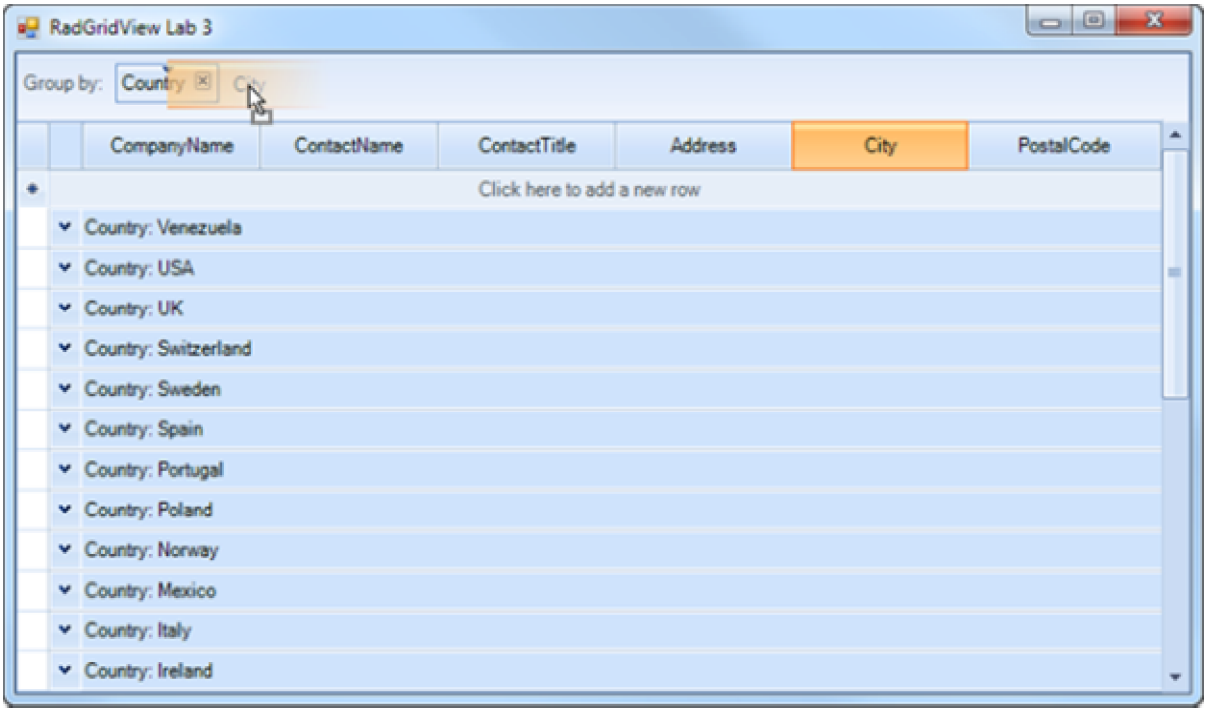
1. Create a new Windows Forms Project
2. Drag a RadGridView control onto the form, and connect its DataSource to the Customers table of the NorthWind sample database.
3. Use the RadGridView's Smart Tag to open the Property Builder, and un-select the following fields: "CustomerID", "Region", "Phone", and "Fax", then close the Property Builder.
4. Run the project to view the data in the grid. To group the data by Country, grab the Country column header, and drag it to the space above the grid. Or, right-click on the column header, and choose "Group By Column" from the context menu.

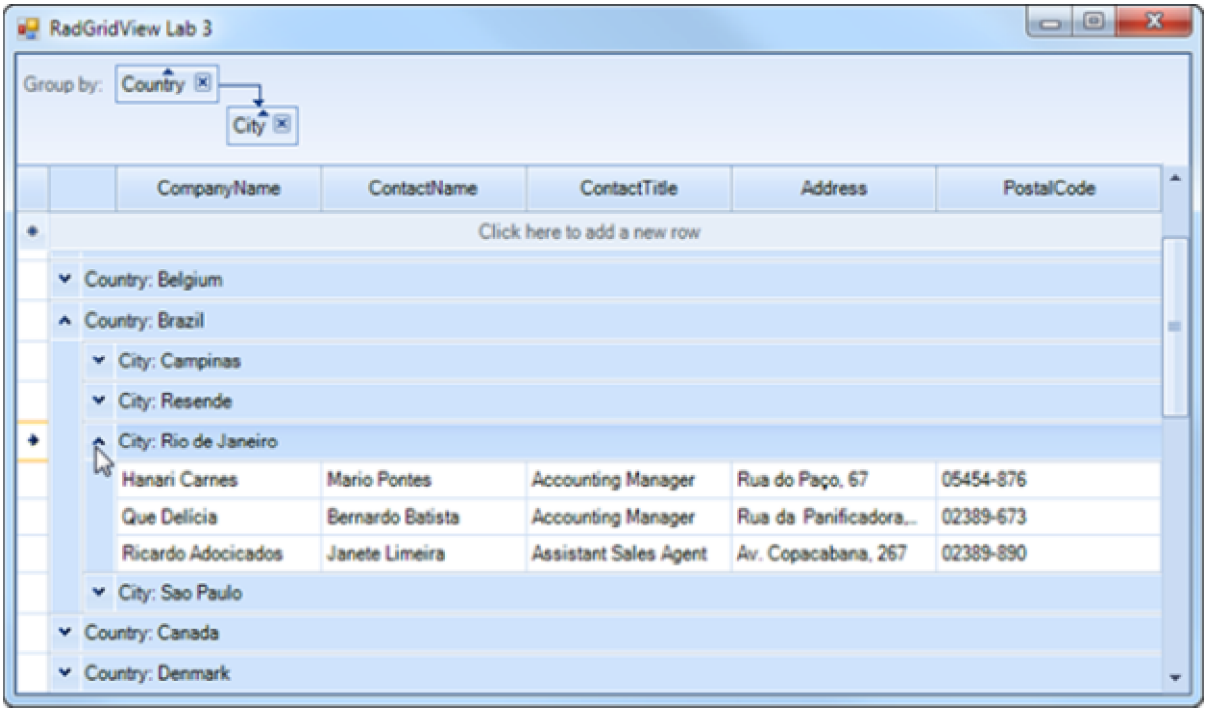


5. Now, you can expand a Country group to view the data in that group. Also, the order of the grouping can be changed by clicking the group header in the Grouping Panel.



6. To add an additional subgroup by City, drag the City column header to the right-hand side of the Country grouping block.

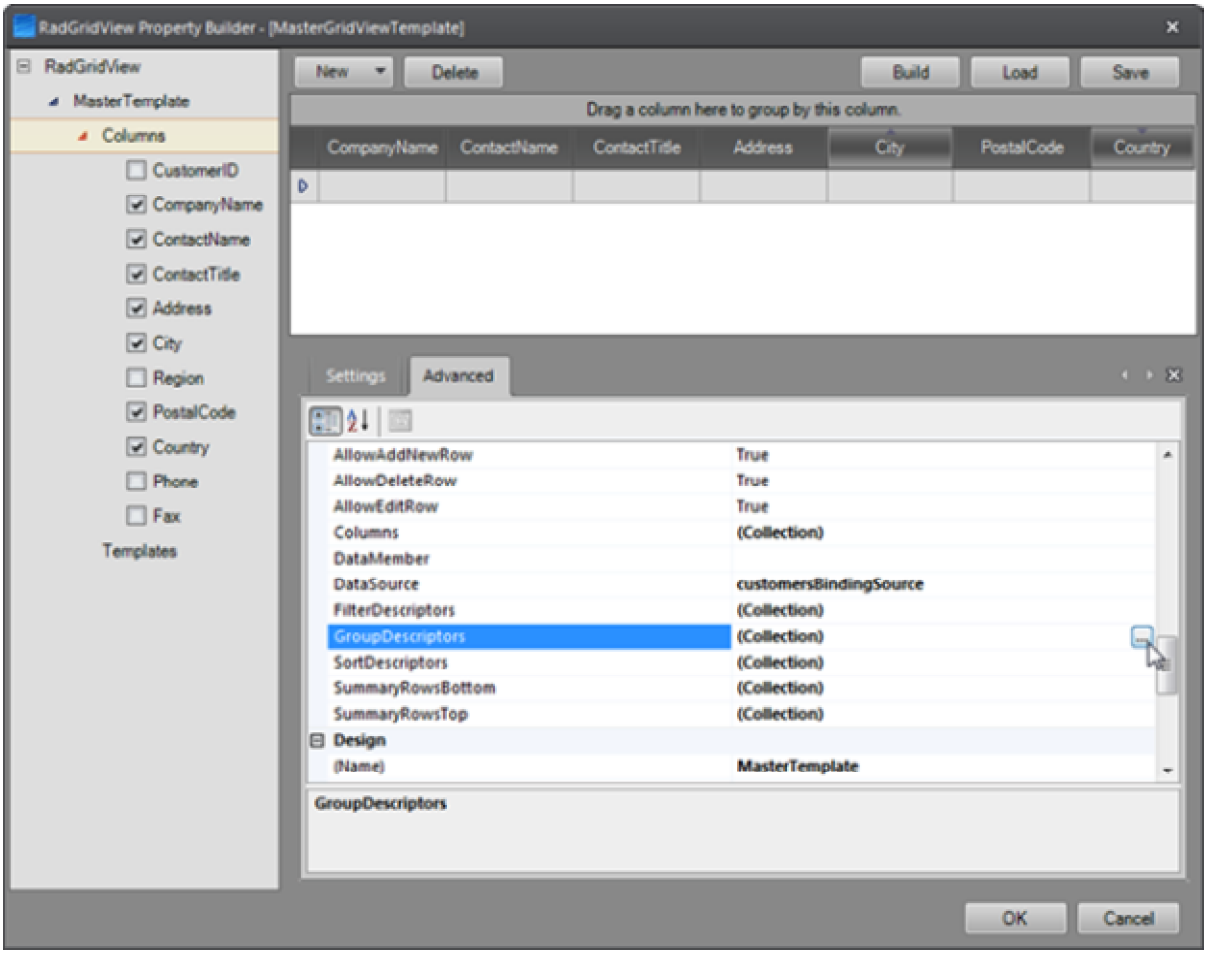




The hierarchy of the grouping can be re-arranged by changing the header blocks within the Grouping Panel. To Ungroup by a column, simply drag the column header back to the header row or use the close button in the group header.

Grouping Using the Property Builder at Design-Time

1. Use the Smart Tag to open the Property Builder on the RadGridView control.
2. In the Advanced tab of the Column, open the **GroupDescriptors** collection.

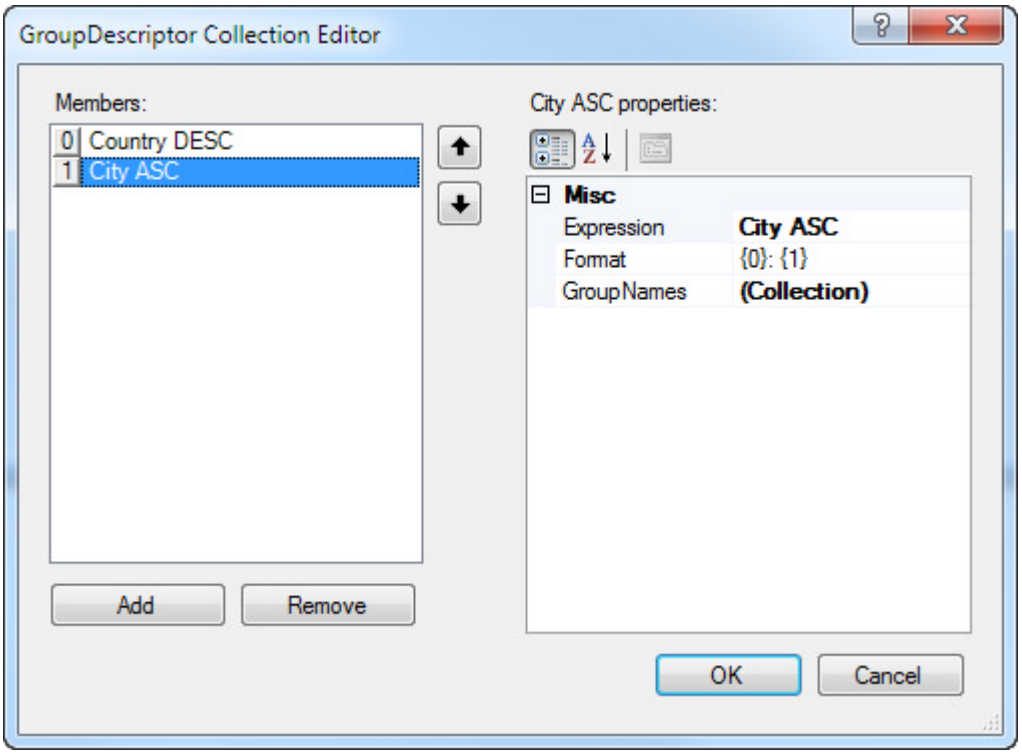


3. In the Collection Editor, add a new GroupDescriptor and set the **Expression** property to:

   Country DESC

4. Add a second GroupDescriptor and set its Expression to

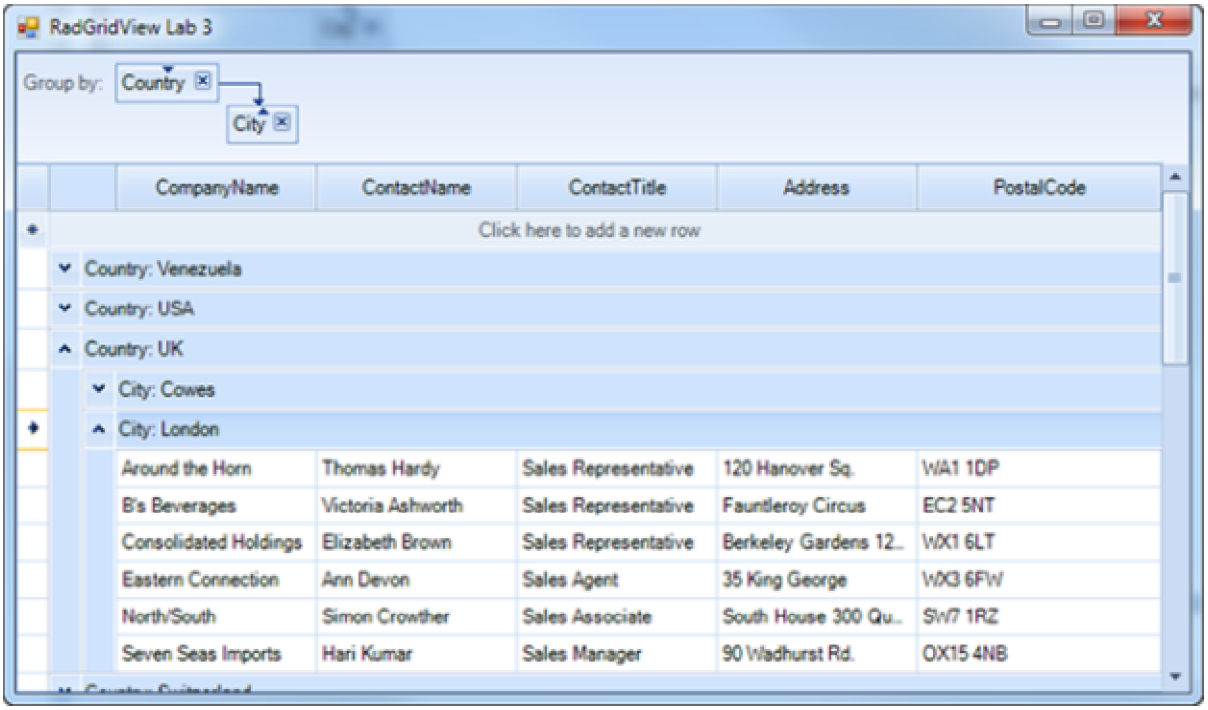
   City ASC



5. Click **OK** to close the Collection Editor
6. Click **OK** to close the Property Builder.

   You should now see your new groupings in the Grouping Panel in design view, just as earlier when they were created by the user at run-time.
7. Run the project to view the grouped data, this time in descending order for country, and ascending for city.

> Although the groupings set up in the designer will be applied on start-up, at runtime, the user can still drag the grouped header to re-arrange, add, or delete groupings from the grid, as long as the **EnableGrouping** property is set to True.



Programmatic Grouping at Run-Time

1. Add the following code to the form's Load event to remove the city grouping added in the designer, and add a grouping by Contact Title.

[VB] Removing and Adding Groupings

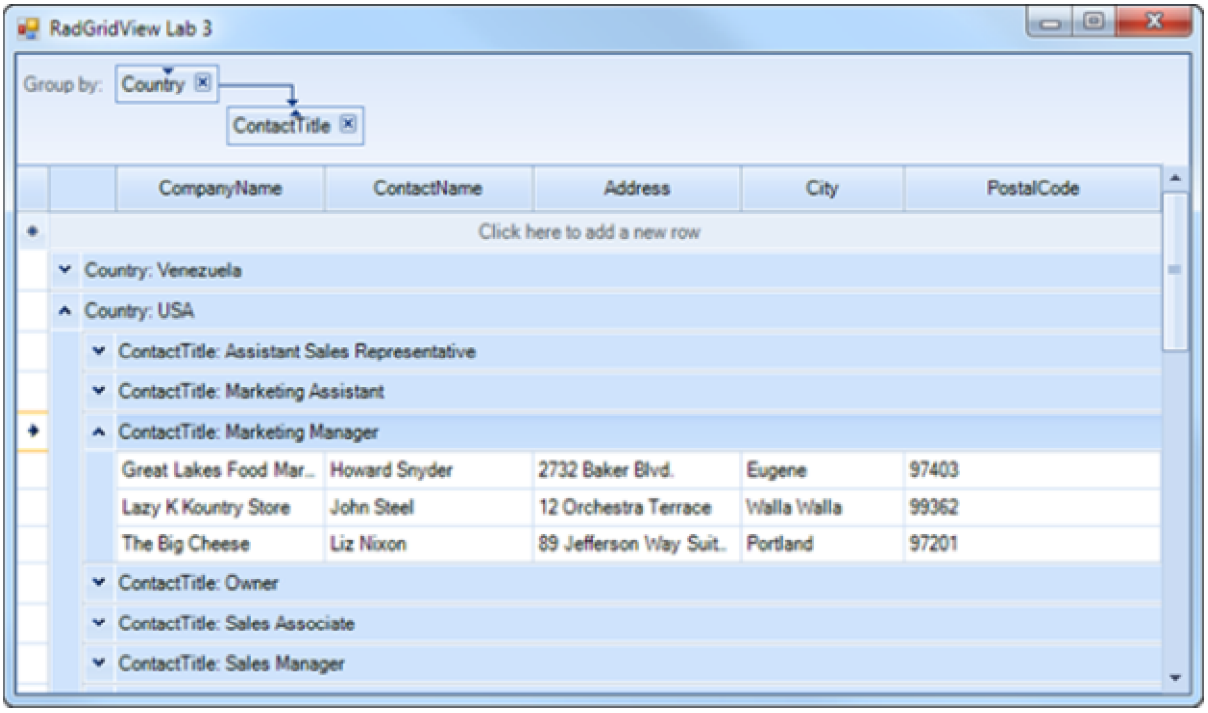
```
Private Sub RadGridViewLab3_Load(ByVal sender As Object, ByVal e As EventArgs)
 Me.customersTableAdapter.Fill(Me.dataSet1.Customers)
 'remove the "City" grouping added in the designer
radGridView1.MasterTemplate.GroupDescriptors.RemoveAt(1)
'add a new grouping by "Contact Title"
radGridView1.MasterTemplate.GroupDescriptors.Add(New GroupDescriptor("ContactTitle ASC"))
End Sub
```

[C#] Removing and Adding Groupings

```
private void RadGridViewLab3_Load(object sender, EventArgs e)
{
 this.customersTableAdapter.Fill(this.dataSet1.Customers);

// remove the "City" grouping added in the designer
radGridView1.MasterTemplate.GroupDescriptors.RemoveAt(1);
// add a new grouping by "Contact Title"
radGridView1.MasterTemplate.GroupDescriptors.Add(new GroupDescriptor("ContactTitle ASC"));
}
```

2. Run the project, and view the data, now grouped by country, and then sub-grouped by Contact Type within the country groups.



Sorting and Filtering



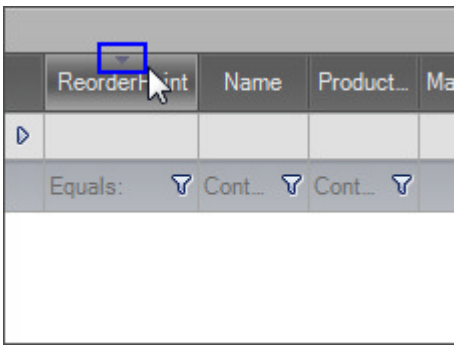
You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\SortingandFiltering

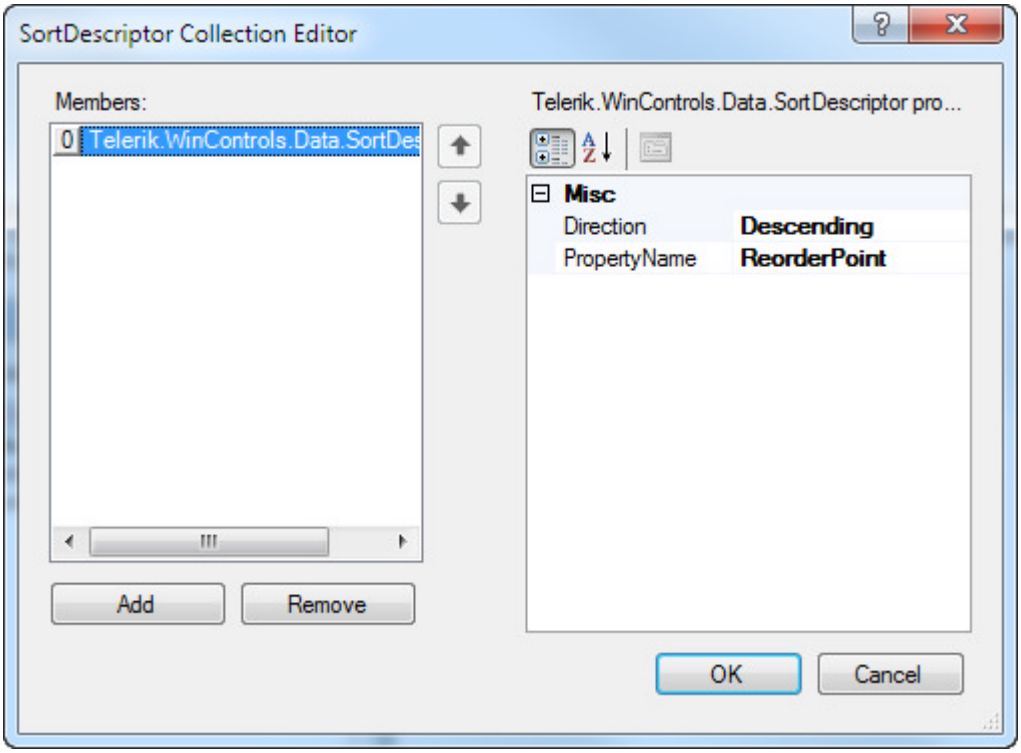


This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**

1. In a new Windows Forms Project, drag a RadGridView control onto the form.
2. Connect its DataSource to the Production.Product table of the AdventureWorks sample database.
3. In the Property Window for the RadGridView, set the MasterTemplate's **EnableFiltering** property to true.
4. Use the RadGridView's Smart Tag to open the Property Builder and un-select the following fields: ProductID, Size, SizeUnitMeasure, rowguid, Weight, WeightUnitMeasure, Style, ProductSubcategoryID, ProductModelID, SellStartDate, SellEndDate, DiscontinuedDate, and ModifiedDate.
5. Still in the Property Builder, drag the ReorderPoint column to the furthest left position, then click the header block itself until it shows a downward arrow, indicating a descending sort on the column.



6. To view the sorting applied to the grid, navigate to the Advanced Settings tab, find the Data Group and click the SortDescriptors property ellipses to open the SortDescriptor Collection Editor. Here, you can also add or delete sorting descriptors on the grid. You should now see the descending sort on the Reorder Point column.



7. Add namespace references to the "Imports" (VB) or "uses" (C#) clause of your code:

[VB] Adding References

```
Imports Telerik.WinControls.UI
Imports Telerik.WinControls.Data
```

[C#] Adding References

```
using Telerik.WinControls.UI;
using Telerik.WinControls.Data;
```

8. To add an additional expression to sort by Color programmatically, close the Property Builder and add the following code to the Load handler of the form:

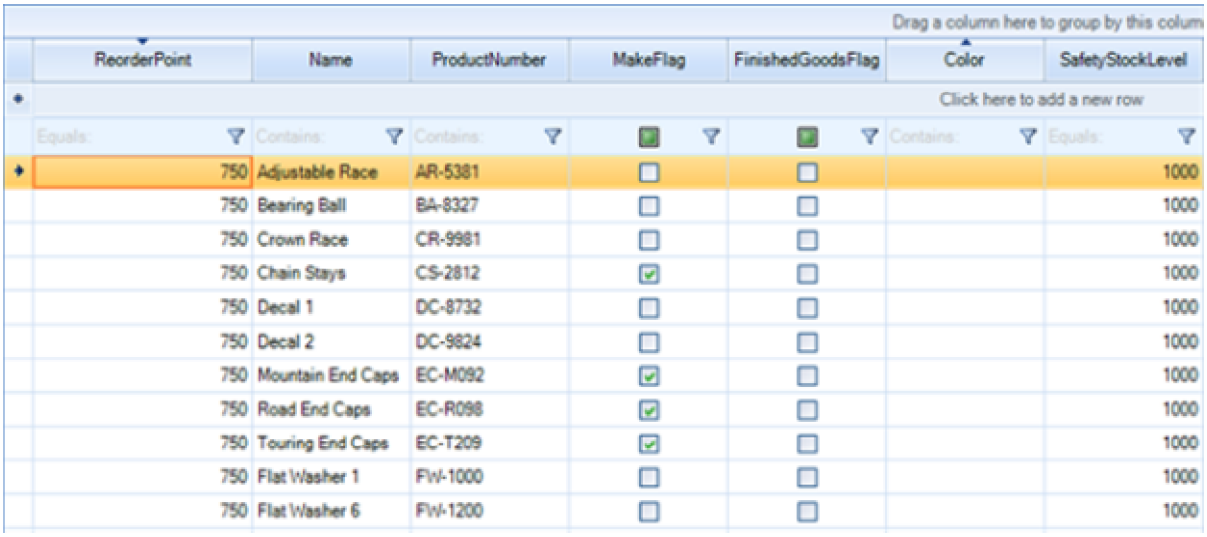
[VB] Adding a Sort Descriptor

```
radGridView1.MasterTemplate.SortDescriptors.Add(new SortDescriptor("Color",
System.ComponentModel.ListSortDirection.Ascending))
```

[C#] Adding a Sort Descriptor

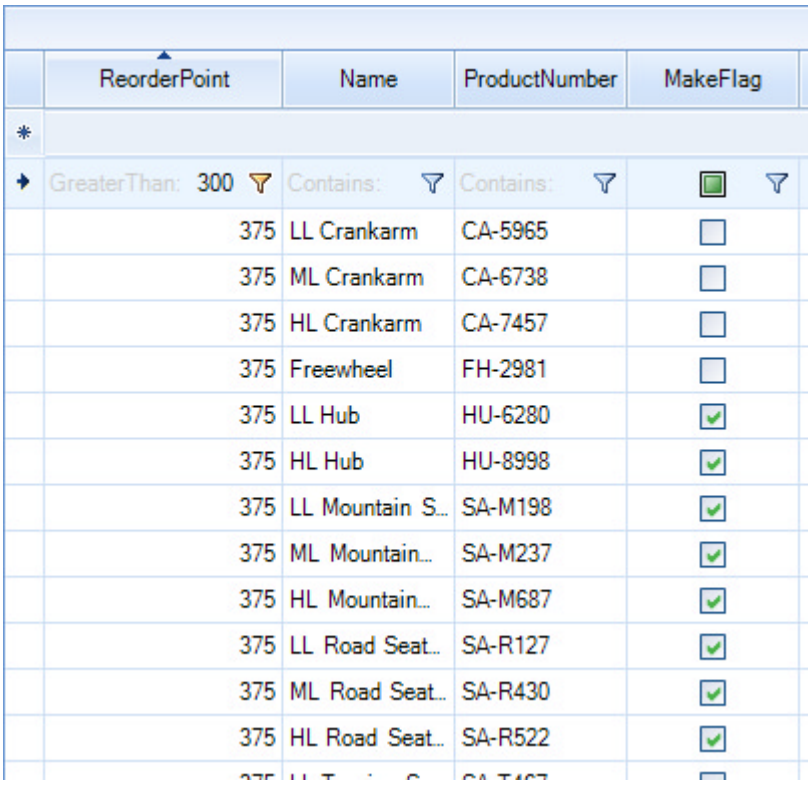
```
radGridView1.MasterTemplate.SortDescriptors.Add(new SortDescriptor("Color",
System.ComponentModel.ListSortDirection.Ascending));
```

9. Run the project to view your sorted data. Notice that the data is first sorted by Reorder Point, then within that sorting, is sorted by Color.



If EnableSorting is set to true, users will still be able to change your default sorting at runtime using the column headers. Setting EnableSorting to false freezes the sort order to the configuration set at either design time or programmatically.

10. To add run-time filtering to the table, click the filter icon on the Reorder Point column, and choose "Greater than" from the context menu. Now enter "300" into the filter row textbox to only show products with a Reorder Point greater than 300.



You can click the filter icon and choose "No filter" to remove the filtering criteria.

11. Stop the project and return to the Form's Load handler. Now programmatically add a filtering descriptor to

only show products whose "MakeFlag" is true. Add the code below to the Load handler.

This step creates a FilterDescriptor object and adds it to the Column's FilterDescriptors collection. The FilterDescriptor constructor has several overloads, but the parameters for this example are:

- A FilterOperator enumeration member that represent one of the filtering criteria in the drop down list, e.g. "GreaterThan", "EqualTo", etc.
- A criteria value, i.e. the value that the column value is being compared to. In the example below the value is "True".

[VB] Adding a Filter Expression

```
radGridView1.MasterTemplate.FilterDescriptors.Add(new FilterDescriptor("MakeFlag",
FilterOperator.IsEqualTo, true))
```

[C#] Adding a Filter Expression

```
radGridView1.MasterTemplate.FilterDescriptors.Add(new FilterDescriptor("MakeFlag",
FilterOperator.IsEqualTo, true));
```

12. To additionally filter results to only those with ProductNumbers starting with the letters "C" or "R" you should use CompositeFilterDescriptors.

    Here we will create a composite filter descriptor which will contain two filter descriptors in it - to filter the results by ProductNumbe starting with "C" and starting with "R". To have both or them working, the LogicalOperator of the CompositeFilterDescriptor instance should be set to "OR". After the composite descriptor is created, we are adding it to the filter descriptor collection of the MasterTemplates (where the descriptor for the MakeFlag column is already added):

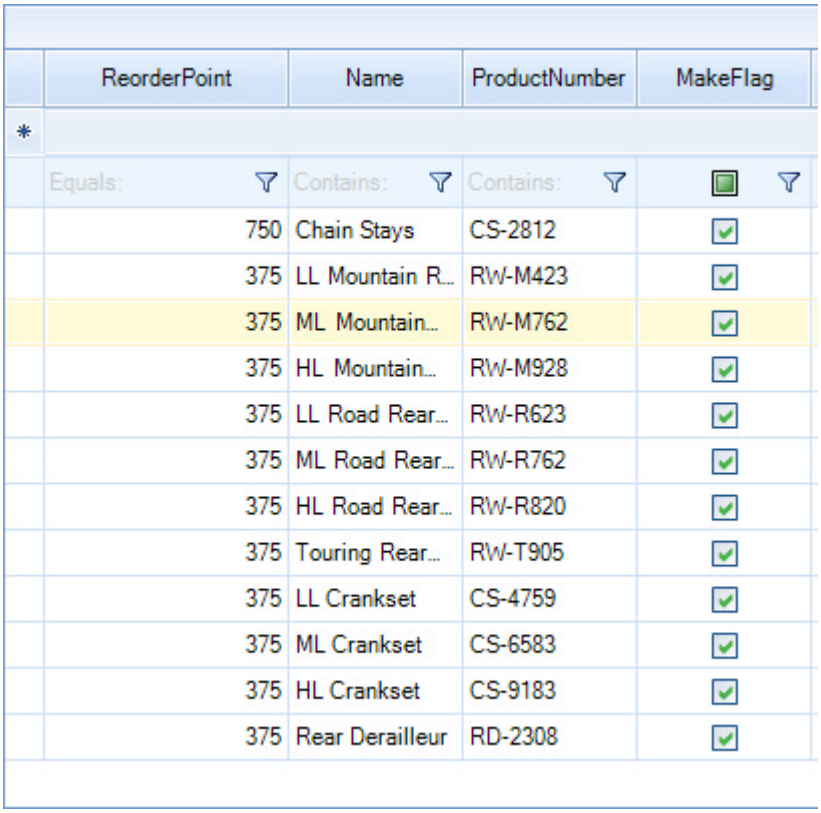
    [VB] Creating CompositeFilterDescriptor

    ```
    Dim compositeDescriptor As New CompositeFilterDescriptor()
    compositeDescriptor.LogicalOperator = FilterLogicalOperator.[Or]
    compositeDescriptor.FilterDescriptors.Add(New FilterDescriptor("ProductNumber",
    FilterOperator.StartsWith, "C"))
    compositeDescriptor.FilterDescriptors.Add(New FilterDescriptor("ProductNumber",
    FilterOperator.StartsWith, "R"))
    radGridView1.MasterTemplate.FilterDescriptors.Add(compositeDescriptor)
    ```

    [C#] Creating CompositeFilterDescriptor

    ```
    CompositeFilterDescriptor compositeDescriptor = new CompositeFilterDescriptor();
    compositeDescriptor.LogicalOperator = FilterLogicalOperator.Or;
    compositeDescriptor.FilterDescriptors.Add(new FilterDescriptor("ProductNumber",
    FilterOperator.StartsWith, "C"));
    compositeDescriptor.FilterDescriptors.Add(new FilterDescriptor("ProductNumber",
    FilterOperator.StartsWith, "R"));
    radGridView1.MasterTemplate.FilterDescriptors.Add(compositeDescriptor);
    ```

13. Run the project and notice the effect of our filter expressions. Only Products with "True" MakeFlags whose number starts with "C" or "R" are shown.



	ReorderPoint	Name	ProductNumber	MakeFlag
*				
	Equals:	Contains:	Contains:	
	750	Chain Stays	CS-2812	☑
	375	LL Mountain R...	RW-M423	☑
	375	ML Mountain...	RW-M762	☑
	375	HL Mountain...	RW-M928	☑
	375	LL Road Rear...	RW-R623	☑
	375	ML Road Rear...	RW-R762	☑
	375	HL Road Rear...	RW-R820	☑
	375	Touring Rear...	RW-T905	☑
	375	LL Crankset	CS-4759	☑
	375	ML Crankset	CS-6583	☑
	375	HL Crankset	CS-9183	☑
	375	Rear Derailleur	RD-2308	☑

There is a more convenient way to sort a column. Instead of adding a new SortDescriptor, set the **SortOrder** property:

[VB] Assigning the SortOrder

```
Me.radGridView1.Columns("Value").SortOrder = RadSortOrder.Descending
```

[C#] Assigning the SortOrder

```
this.radGridView1.Columns["Value"].SortOrder = RadSortOrder.Descending;
```

21.10 Hierarchy Support

ith some data, it is desirable to show tables within tables, also known as a Hierarchical view. Multiple tables can be related through data keys and RadGridView allows almost effortless setup to display such relationships. Hierarchical tables can be set up through either the designer manually, automatically, or programmatically in the code-behind. The following labs will demonstrate all implementation methods. Additionally, Object-relational hierarchy, Self-referencing hierarchy and Load on demand hierarchy will be reviewed.

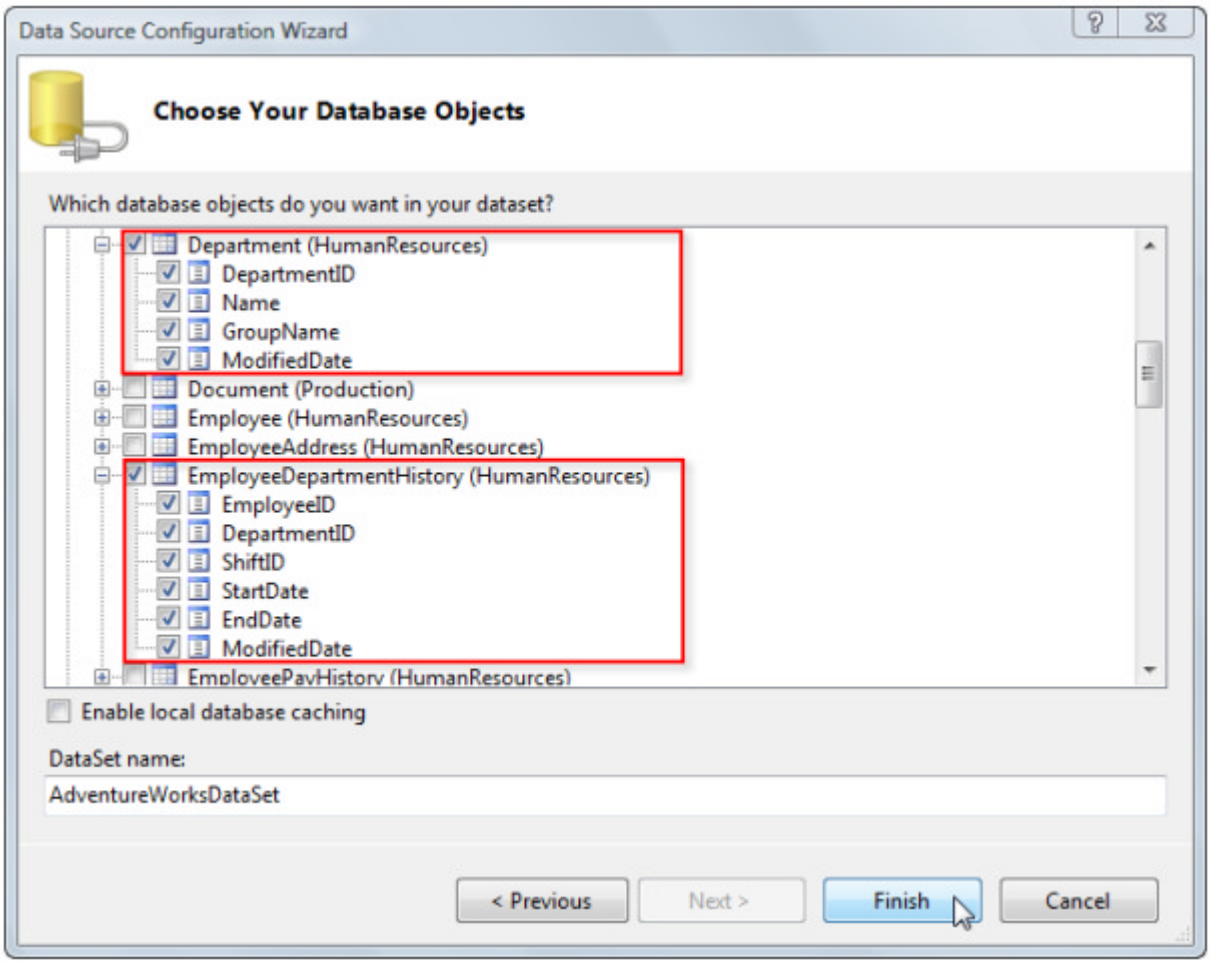
Hierarchy Table Setup in the Designer

You can find the complete source for this project at:

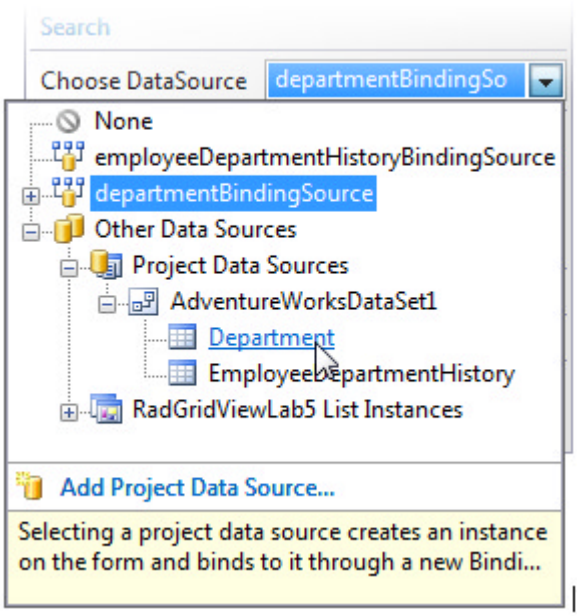
\Grid\<VB|CS>\RadGridView\HierarchyDesigner

This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**

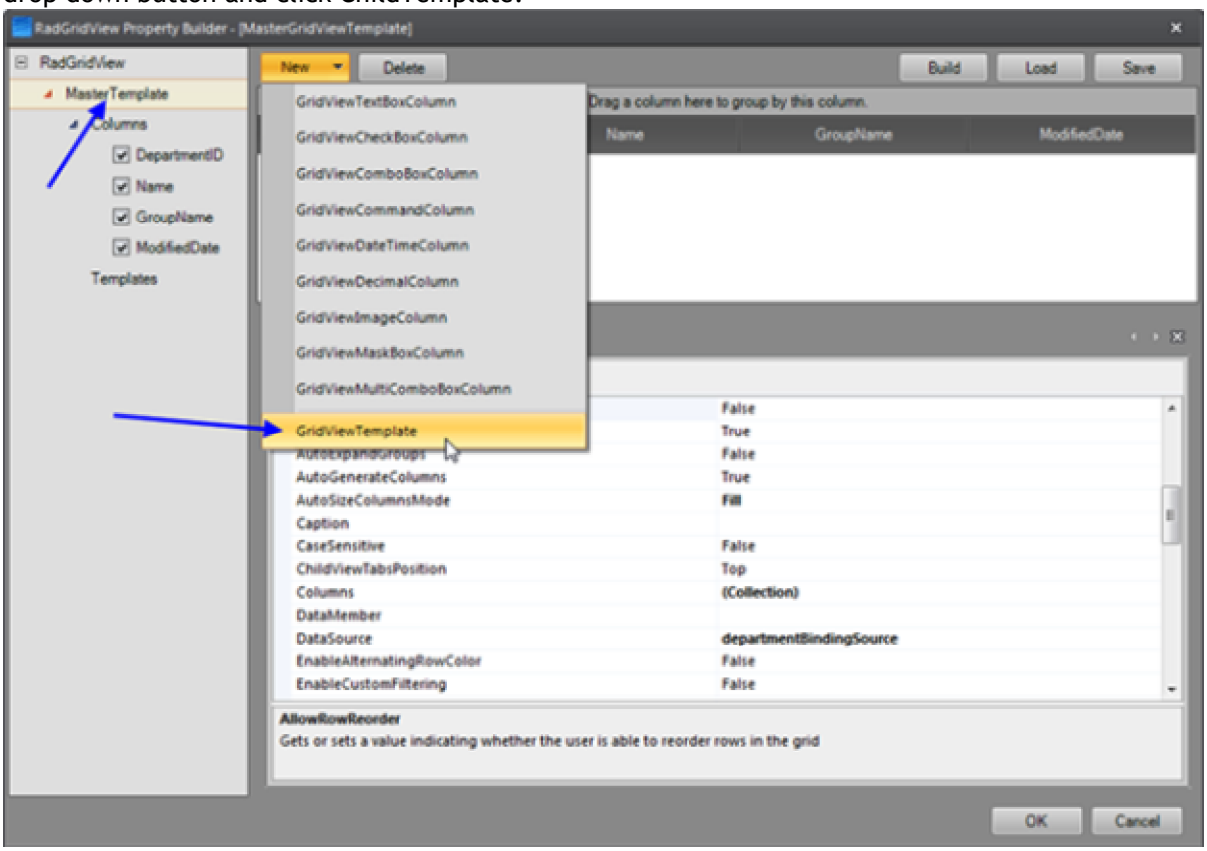
1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form, and use the Smart Tag to create a new DataSource. Use the AdventureWorks database, and select both the HumanResources.Department and HumanResources.EmployeeDepartmentHistory tables.



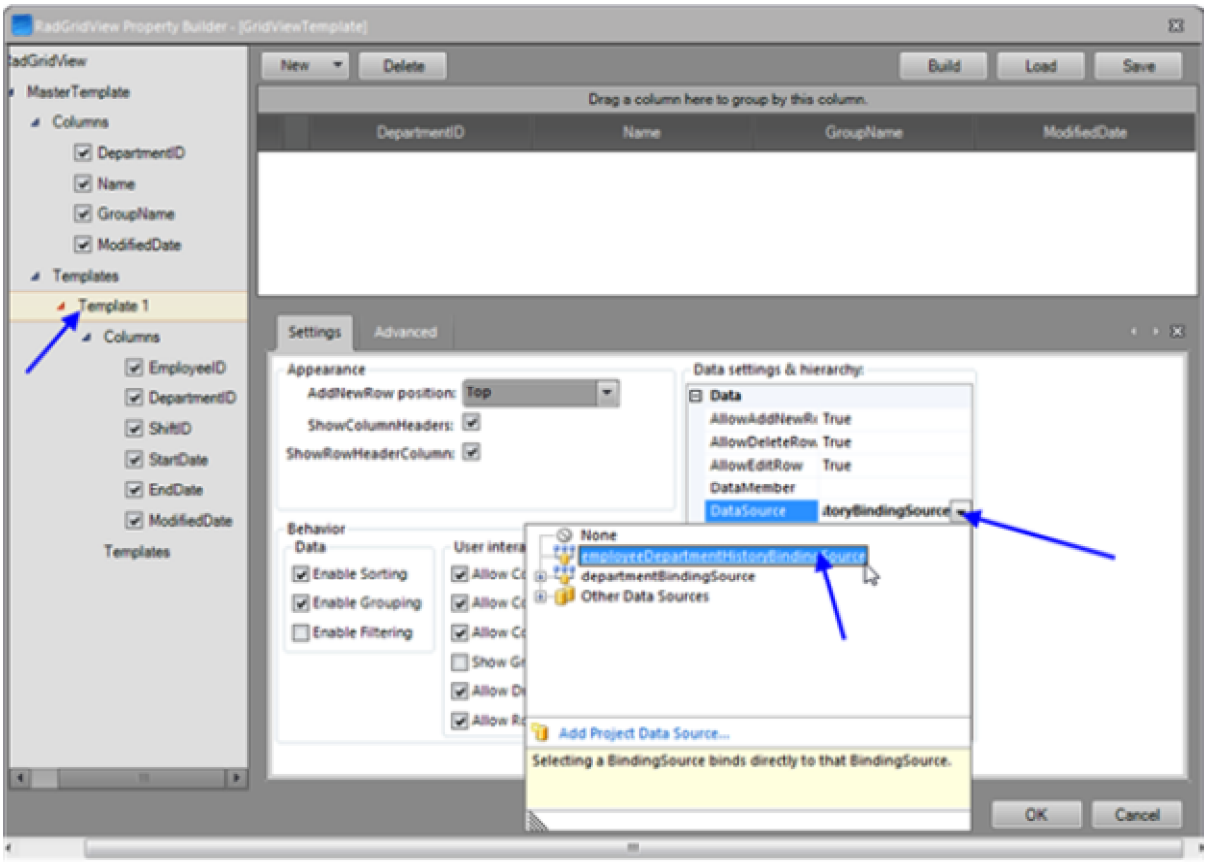
3. Again using the Smart Tag, select the Department Table as the source for the main grid.



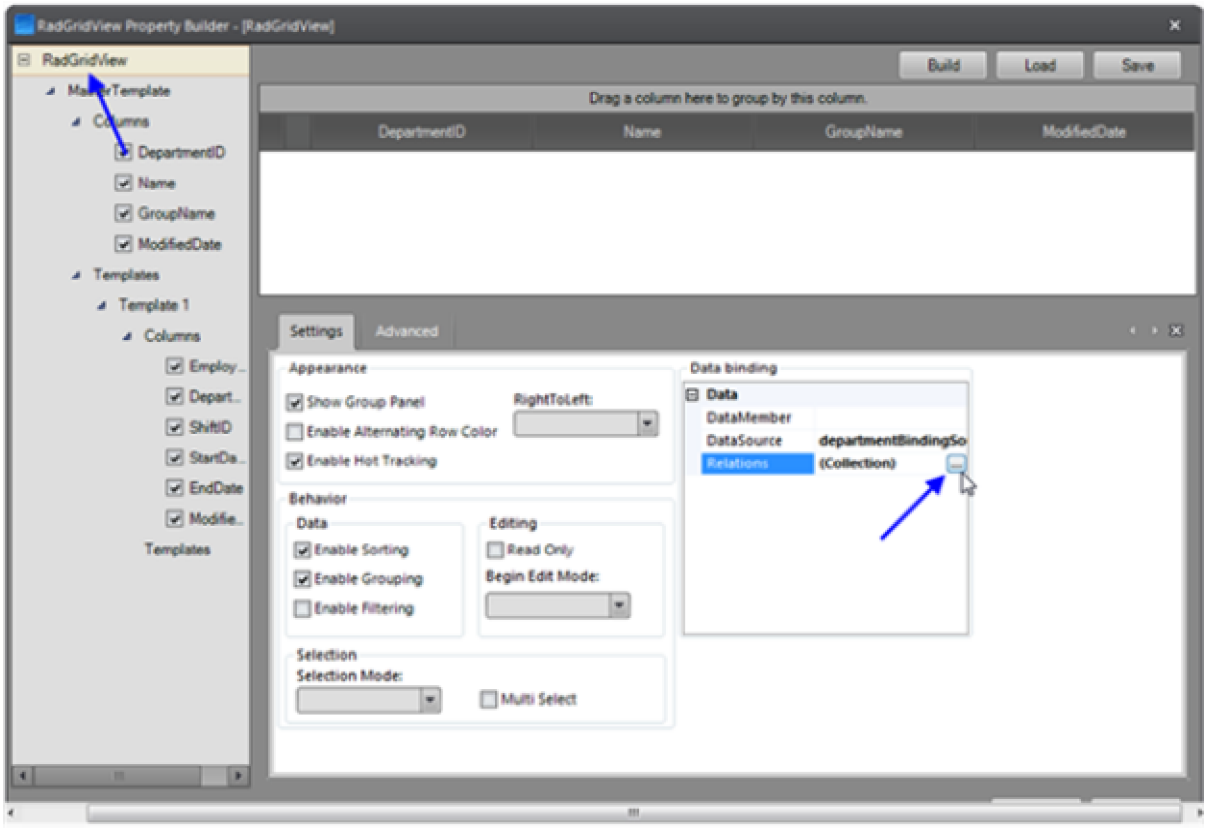
4. Now, open the Property Builder for the grid. Select the MasterTemplate node, and then expand the New drop down button and click ChildTemplate.



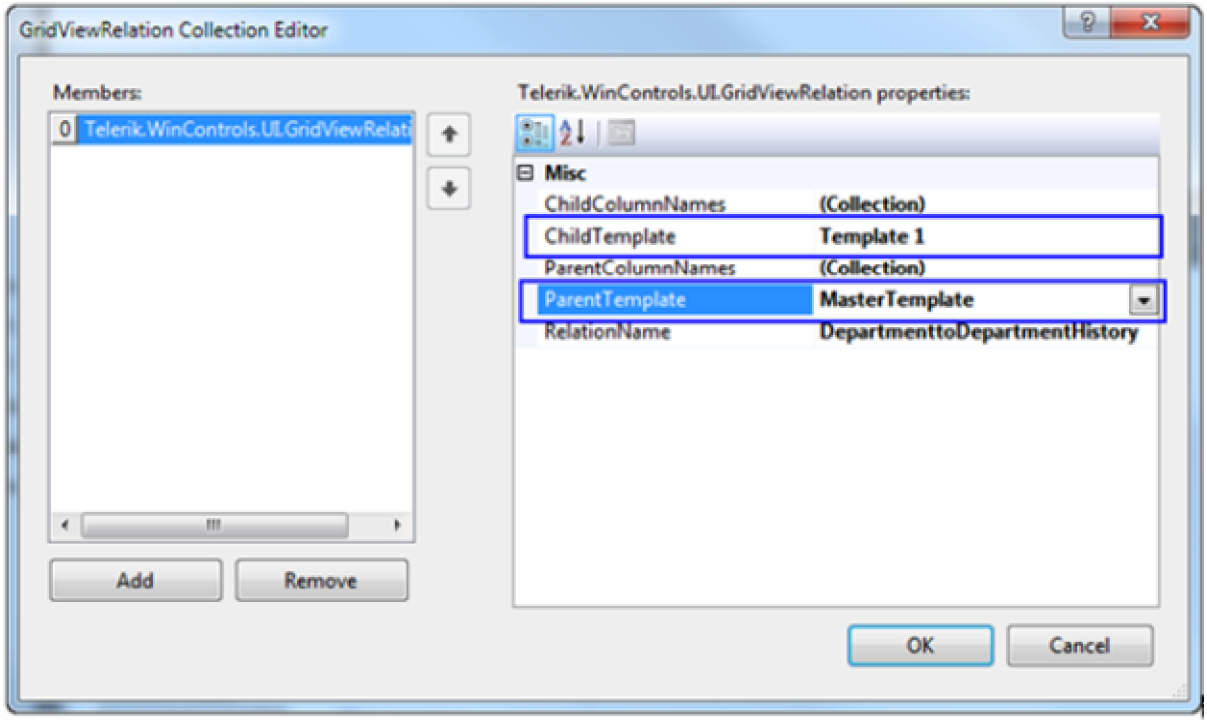
5. After the template is added, select it and navigate to the Data pane. Locate the DataSource property and from the drop down menu select the EmployeeDepartmentHistory table. This will be the child table within the grid. Afterwards, click OK to save the Templates collection.



6. Select the RadGridView node and open the Collections editor for the Relations collection.

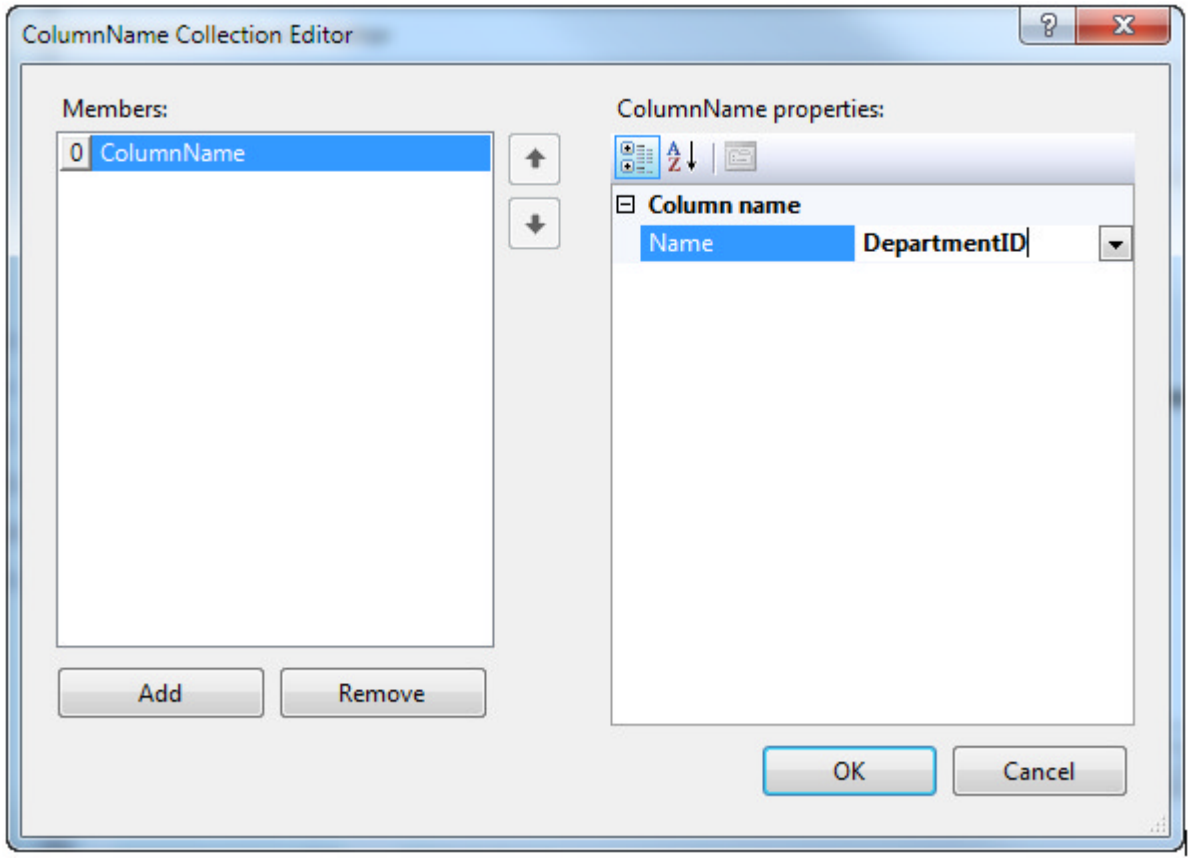


7. In the GridViewRelation Collection Editor, add a new relation by clicking the "Add" button. Then, name your relation in the properties pane on the right side of the Collection Editor. Here, the relation is named "DepartmenttoDepartmentHistory". Set the templates of the relation to your newly created Child template for the ChildTemplate property, and the MasterTemplate for the ParentTemplate.

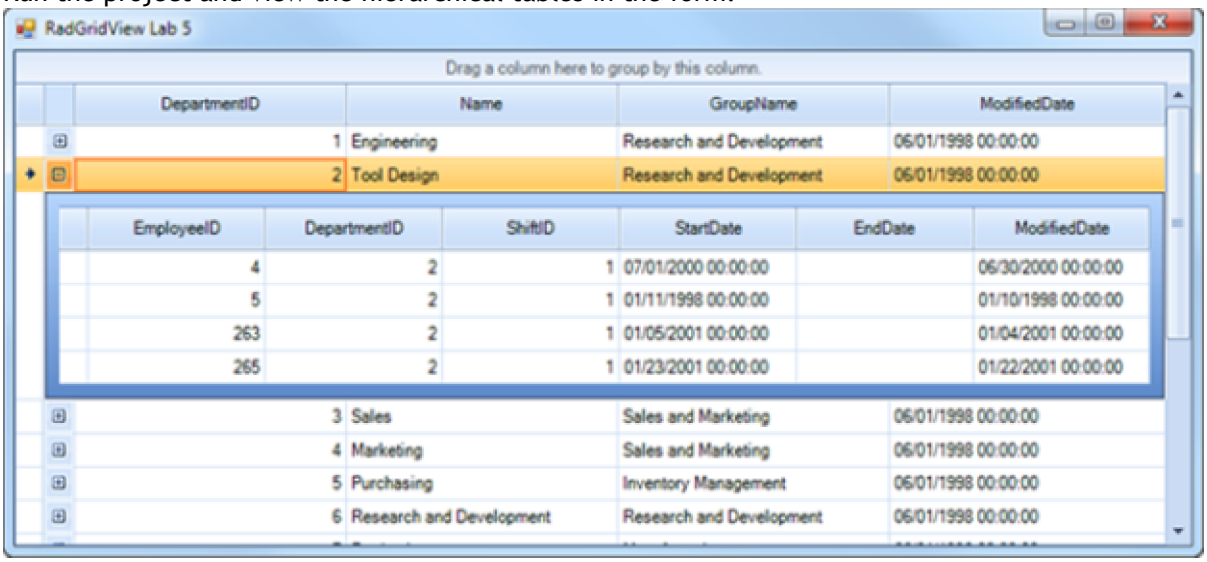


8. Finally, set the related column by adding the Column Name “DepartmentID” to the collections of the

ChildColumnNames and ParentColumnNames properties of the relation. Both collections should look like the figure below. Afterwards, click **OK** to close both dialogs and the Property Editor.



9. Run the project and view the hierarchical tables in the form:



Notice that the Main Grid shows the data from the Departments database table, and by clicking the plus sign to the left of a Department record, you can view the Child table, which displays the Employees who have the same DepartmentID in the EmployeeDepartmentHistory table.

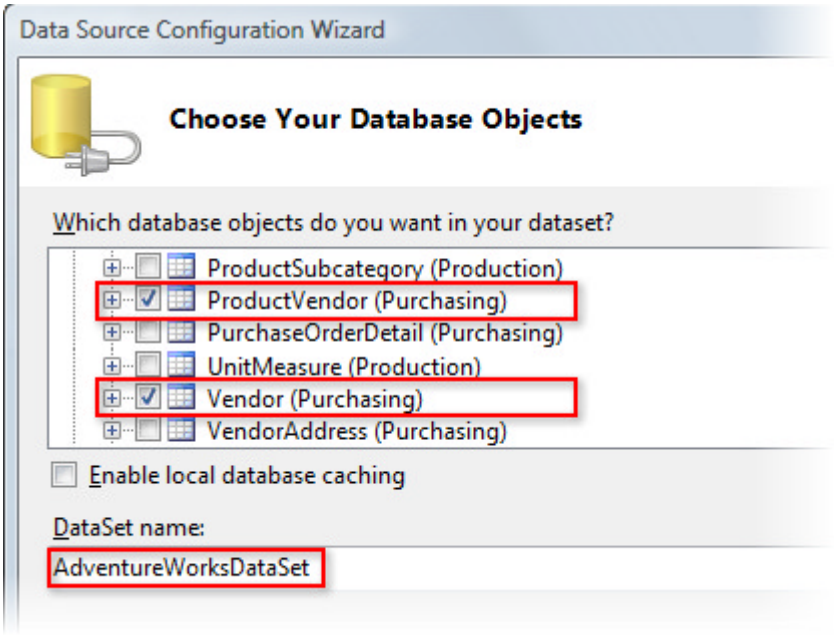
Automatic Hierarchy Table Setup

You can find the complete source for this project at:

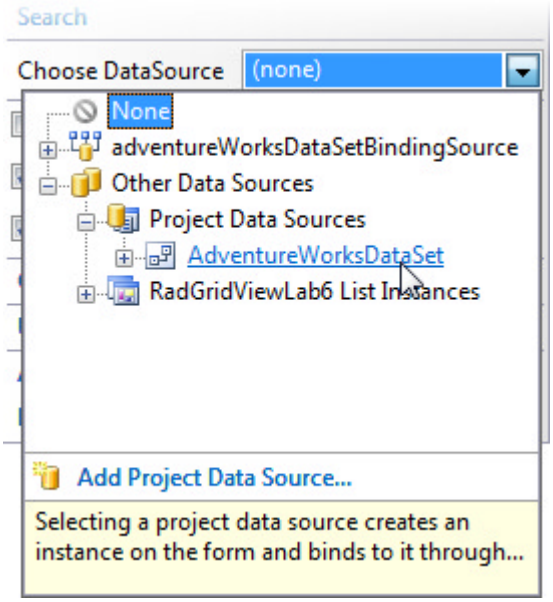
\Grid\<VB|CS>\RadGridView\HierarcyAuto

Besides creating the table relationships yourself, the RadGridView control can automatically generate hierarchical tables for use based upon Dataset relationships.

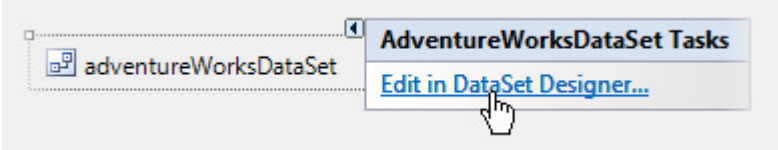
1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form, and use the Smart Tag to create a new DataSource. Use the AdventureWorks database, and select both the Purchasing.Vendor and Purchasing.ProductVendor tables. This will create a DataSet which includes both of the tables. Note the name of your new Dataset. In this example the name of the dataset is "AdventureWorksDataSet".



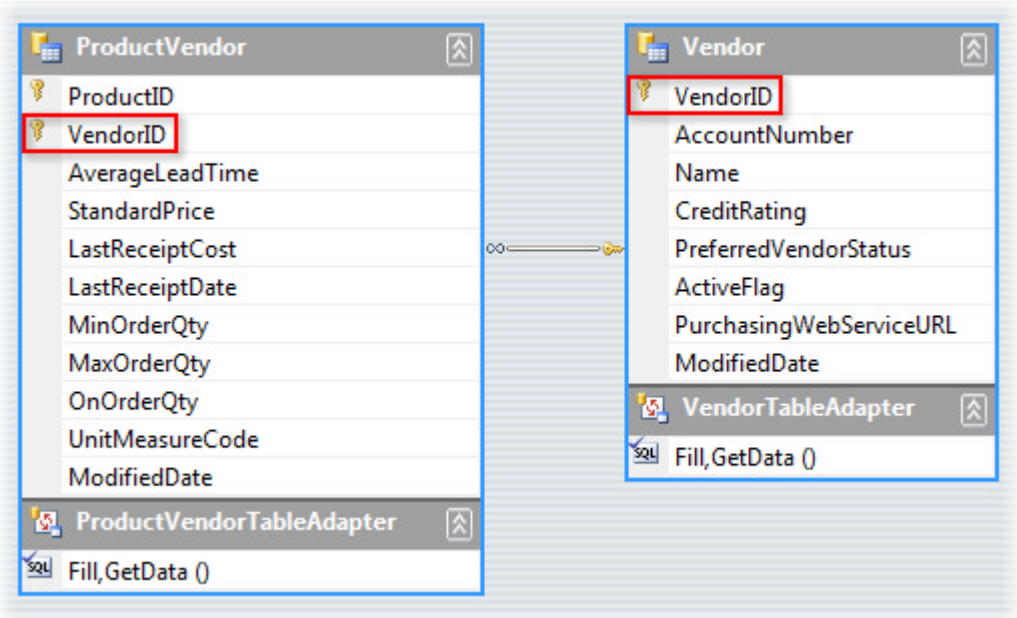
3. Use the Smart Tag to set the DataSource property to the DataSet itself. This will give RadGridView access to both tables and the relationship between them.



4. To view the relationships that are automatically created in the new Dataset, close the Property Builder and use the SmartTag on the Dataset object placeholder in the Form's design panel to choose "Edit in DataSet Designer".



5. In the DataSet Designer, notice that the two tables are linked by a relationship line. The relationship between the tables is the VendorID field, which is the Primary Key for the Vendor table, and a Foreign Key for the ProductVendor table. This relationship will be the basis for our auto-generated hierarchical table view.



6. Close the DataSet Designer and return to the design view of the main form.
7. Since we used the DataSet as the source for our grid, we must add TableAdapters to load data from the database. Navigate to the code-behind for the Form, and add the following declarations in the form's Load event handler. The code instantiates table adapters for both tables and fills each table. Finally, we must set the **DataMember** property to "Vendor" and also the **AutoGenerateHierarchy** property to true.

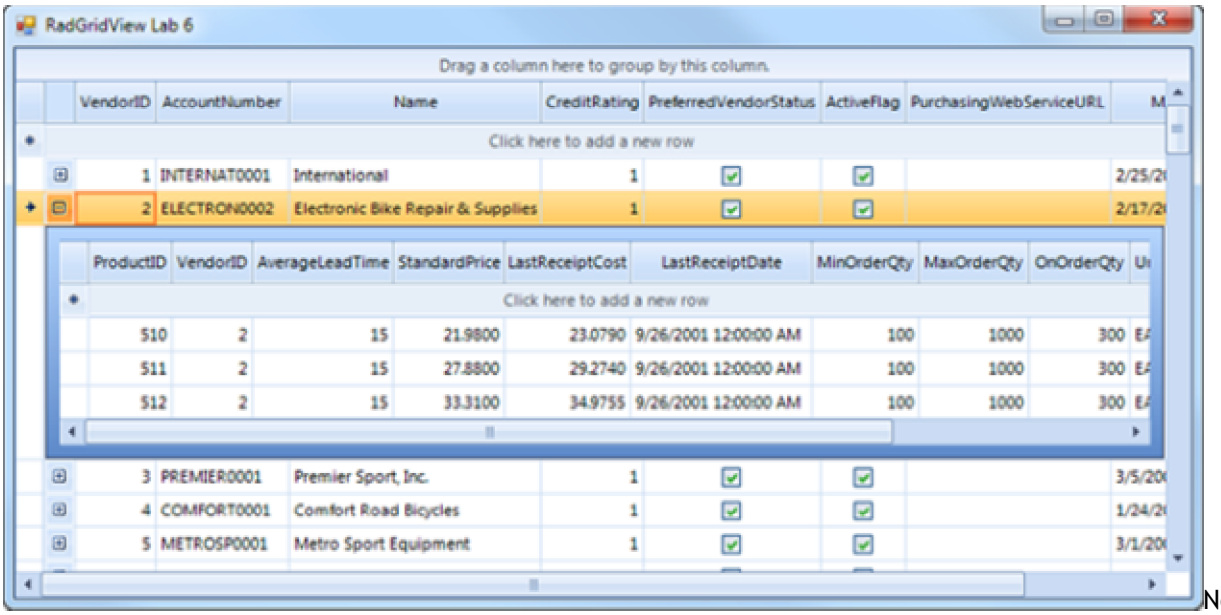
[VB] Loading the DataSet

```
Private Sub RadGridViewLab6_Load(ByVal sender As Object, ByVal e As EventArgs)
 Dim vendorTA As New AdventureWorksDataSetTableAdapters.VendorTableAdapter()
 Dim productvendorTA As New AdventureWorksDataSetTableAdapters.ProductVendorTableAdapter
 vendorTA.Fill(adventureWorksDataSet.Vendor)
 productvendorTA.Fill(adventureWorksDataSet.ProductVendor)
radGridView1.AutoGenerateHierarchy = True
radGridView1.DataMember = "Vendor"
End Sub
```

[C#] Loading the DataSet

```
private void RadGridViewLab6_Load(object sender, EventArgs e)
{
 AdventureWorksDataSetTableAdapters.VendorTableAdapter vendorTA =
   new AdventureWorksDataSetTableAdapters.VendorTableAdapter();
 AdventureWorksDataSetTableAdapters.ProductVendorTableAdapter productvendorTA =
   new AdventureWorksDataSetTableAdapters.ProductVendorTableAdapter();
 vendorTA.Fill(adventureWorksDataSet.Vendor);
 productvendorTA.Fill(adventureWorksDataSet.ProductVendor);
radGridView1.AutoGenerateHierarchy = true;
radGridView1.DataMember = "Vendor";
}
```

8. Run the project and test the hierarchy functionality:



Notice that the table of Vendors is the parent table, and by expanding a Vendor Record, we can view a child table displaying the Products associated with that Vendor from the ProductsVendor table.

For columns that expand to fill the form, set the AutoSizeColumnsMode property of both the parent and child tables to “Fill”.

Programmatic Hierarchical Table Setup



You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\HierarchyCode

In this lab on hierarchical tables, we will construct the hierarchy programmatically in the codebehind for the form itself.

1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form.
3. We will be doing all the setup for the RadGridView control itself programmatically, but first we still have to create a Dataset in our project to connect. Again, use the RadGridView’s Smart Tag to create a new project DataSource.
4. Use the AdventureWorks database, and this time select both the Sales.SalesTerritory and Sales. SalesPerson tables. Note the name of your new Dataset: again, ours is named AdventureWorksDataSet. **Note**: After creating the Dataset, be sure to leave the RadGridView’s DataSource set to “none”, as we will be setting this in code.
5. Add a reference to the Telerik.Winforms.UI namespace in the "Imports" (VB) or "using" (C#) clause of the code.
6. Add the code below to the form's Load event handler. This will create the DataSet and table adapter object instances, then load the table data:

[VB] Initializing the Data Access

```
Private Sub RadGridViewLab7_Load(ByVal sender As Object, ByVal e As EventArgs)
 Dim adventureWorksDS As New AdventureWorksDataSet()
 Dim salesPersonTA As New AdventureWorksDataSetTableAdapters.SalesPersonTableAdapter()
 Dim salesTerritoryTA As New AdventureWorksDataSetTableAdapters.SalesTerritoryTableAdapte
 salesTerritoryTA.Fill(adventureWorksDS.SalesTerritory)
 salesPersonTA.Fill(adventureWorksDS.SalesPerson)
'. . .
End Sub
```

[C#] Initializing the Data Access

```
private void RadGridViewLab7_Load(object sender, EventArgs e)
{
 AdventureWorksDataSet adventureWorksDS = new AdventureWorksDataSet();
 AdventureWorksDataSetTableAdapters.SalesPersonTableAdapter salesPersonTA =
   new AdventureWorksDataSetTableAdapters.SalesPersonTableAdapter();
 AdventureWorksDataSetTableAdapters.SalesTerritoryTableAdapter salesTerritoryTA =
   new AdventureWorksDataSetTableAdapters.SalesTerritoryTableAdapter();
 salesTerritoryTA.Fill(adventureWorksDS.SalesTerritory);
 salesPersonTA.Fill(adventureWorksDS.SalesPerson);
 //. . .
}
```

7. Add code to the end of the form's Load event handler to configure the master template.

   This code assigns the DataSource for the master template and also performs some miscellaneous housekeeping to configure how columns are sized and to disallow adding new rows.

[VB] Configuring the Master Grid View Template

```
Private Sub RadGridViewLab7_Load(ByVal sender As Object, ByVal e As EventArgs)
'. . .
 radGridView1.DataSource = adventureWorksDS.SalesTerritory
 radGridView1.MasterGridViewTemplate.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.F
 radGridView1.MasterGridViewTemplate.AllowAddNewRow = False
'. . .
End Sub
```

[C#] Configuring the Master Grid View Template

```
private void RadGridViewLab7_Load(object sender, EventArgs e)
{
 //. . .
 radGridView1.DataSource = adventureWorksDS.SalesTerritory;
 radGridView1.MasterGridViewTemplate.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.F
 radGridView1.MasterGridViewTemplate.AllowAddNewRow = false;
 //. . .
}
```

8. Add code to the end of the form's Load event handler to configure the child grid view.

   This code creates a GridViewTemplate object instance to represent the child view, assigns the SalesPerson table as the DataSource and adds the template to the master grid view's Templates collection.

   [VB] Configuring the Child Grid View Template

```
Private Sub RadGridViewLab7_Load(ByVal sender As Object, ByVal e As EventArgs)
'. . .
 Dim childTmpt As New GridViewTemplate()
 childTmpt.DataSource = adventureWorksDS.SalesPerson
 childTmpt.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill
 childTmpt.AllowAddNewRow = False
 radGridView1.MasterGridViewTemplate.ChildGridViewTemplates.Add(childTmpt)
'. . .
End Sub
```

   [C#] Configuring the Child Grid View Template

```
private void RadGridViewLab7_Load(object sender, EventArgs e)
{
 //. . .
 GridViewTemplate childTmpt = new GridViewTemplate();
 childTmpt.DataSource = adventureWorksDS.SalesPerson;
 childTmpt.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill;
 childTmpt.AllowAddNewRow = false;
 radGridView1.MasterGridViewTemplate.ChildGridViewTemplates.Add(childTmpt);
 //. . .
}
```

9. Now that our parent and child templates are in place, we need to set a relationship between them. Add the following code, which relates the tables using the TerritoryID field, to the end of the Load handler.

   [VB] Configuring the Child Grid View Template

```
Private Sub RadGridViewLab7_Load(ByVal sender As Object, ByVal e As EventArgs)
'. . .
 Dim relation As New GridViewRelation(radGridView1.MasterGridViewTemplate)
 relation.ChildTemplate = childTmpt
 relation.RelationName = "SalesTerritoryPerson"
```

```
 relation.ParentColumnNames.Add("TerritoryID")
 relation.ChildColumnNames.Add("TerritoryID")
 radGridView1.Relations.Add(relation)

'. . .
End Sub
```

[C#] Configuring the Child Grid View Template

```
private void RadGridViewLab7_Load(object sender, EventArgs e)
{
 //. . .
 GridViewRelation relation = new GridViewRelation(radGridView1.MasterGridViewTemplate);
 relation.ChildTemplate = childTmpt;
 relation.RelationName = "SalesTerritoryPerson";
 relation.ParentColumnNames.Add("TerritoryID");
 relation.ChildColumnNames.Add("TerritoryID");
 radGridView1.Relations.Add(relation);
 //. . .
}
```

10. Finally, add code to the end of the form's Load handler to hide the columns containing GUID's and set the theme name for the grid:

[VB] Configuring the Child Grid View Template

```
Private Sub RadGridViewLab7_Load(ByVal sender As Object, ByVal e As EventArgs)
'. . .
 radGridView1.Columns("rowguid").IsVisible = False
 childTmpt.Columns("rowguid").IsVisible = False

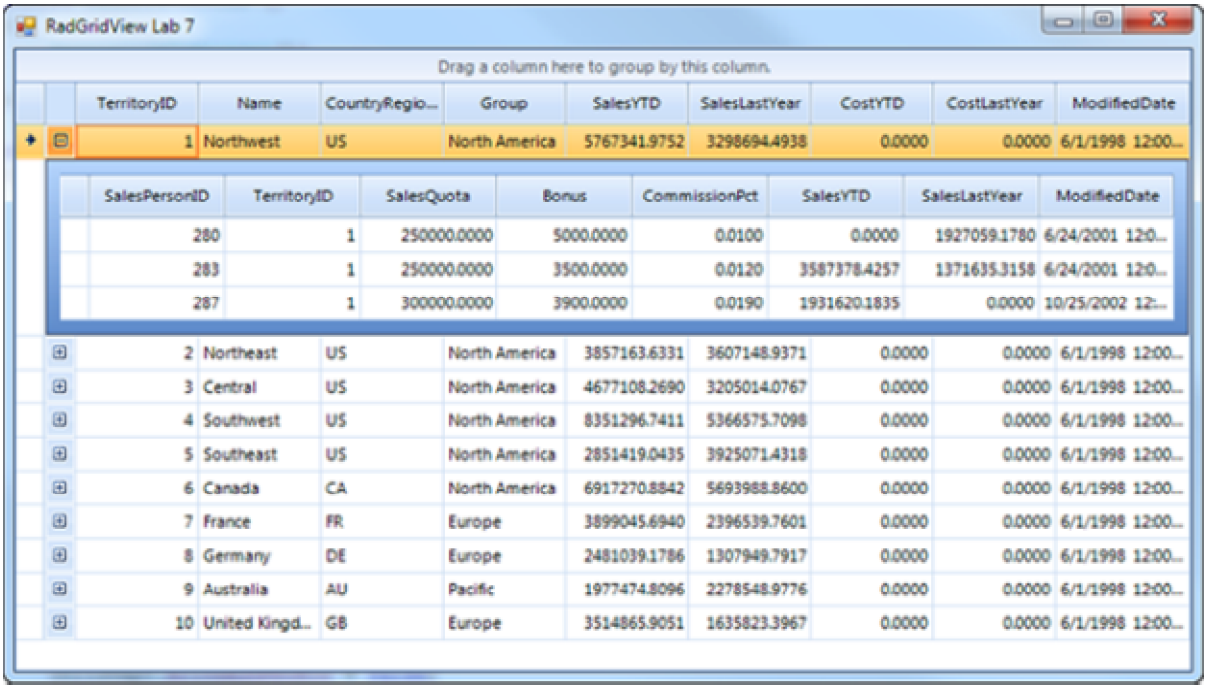
End Sub
```

[C#] Configuring the Child Grid View Template

```
private void RadGridViewLab7_Load(object sender, EventArgs e)
{
 //. . .
 radGridView1.Columns["rowguid"].IsVisible = false;
 childTmpt.Columns["rowguid"].IsVisible = false;

}
```

11. Run the project and expand a Territory record to view the SalesPerson records with matching TerritoriyID values.



Object-relational hierarchy

The Object-Relational Hierarchy mode is used to show hierarchy based on a complex IList (IEnumarable) object that contains inner ILits (IEnumerable) properties.
In order to create an Object-Relational Hierarchy automatically in this scenario, you must set only the DataSource and the AutoGenerateHierarachy properties of RadGridView.

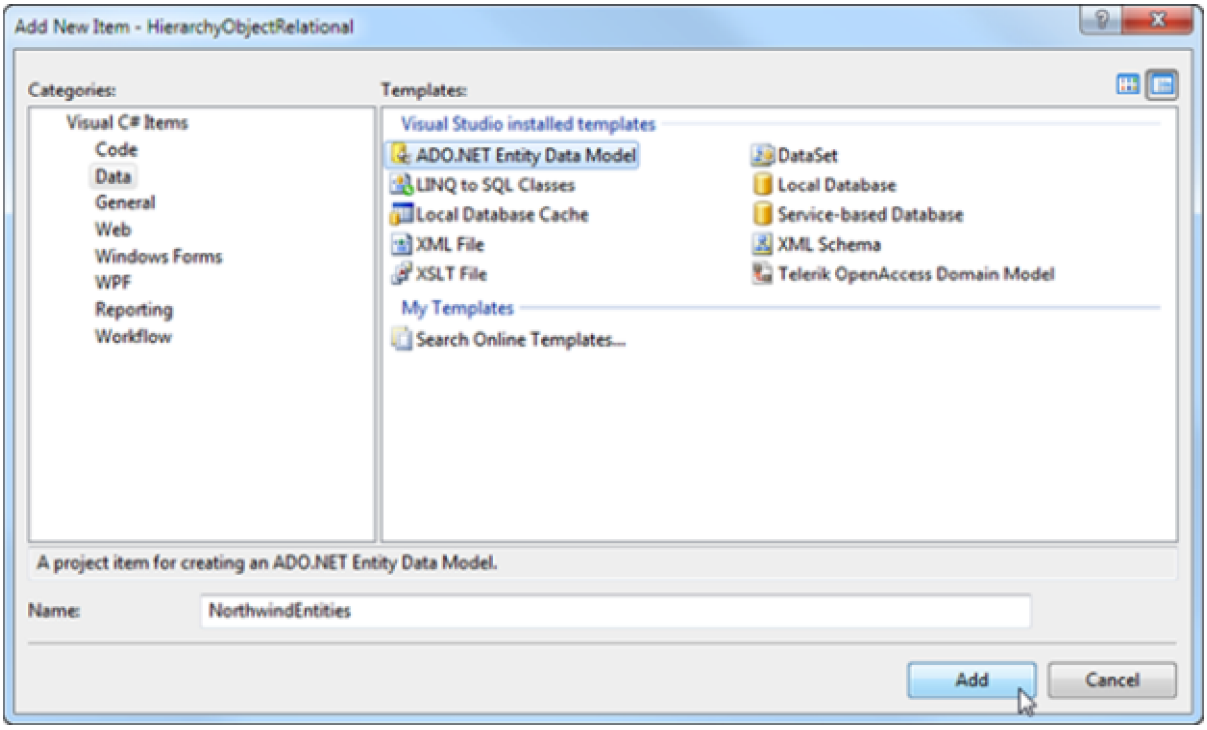


You can find the complete source for this project at:

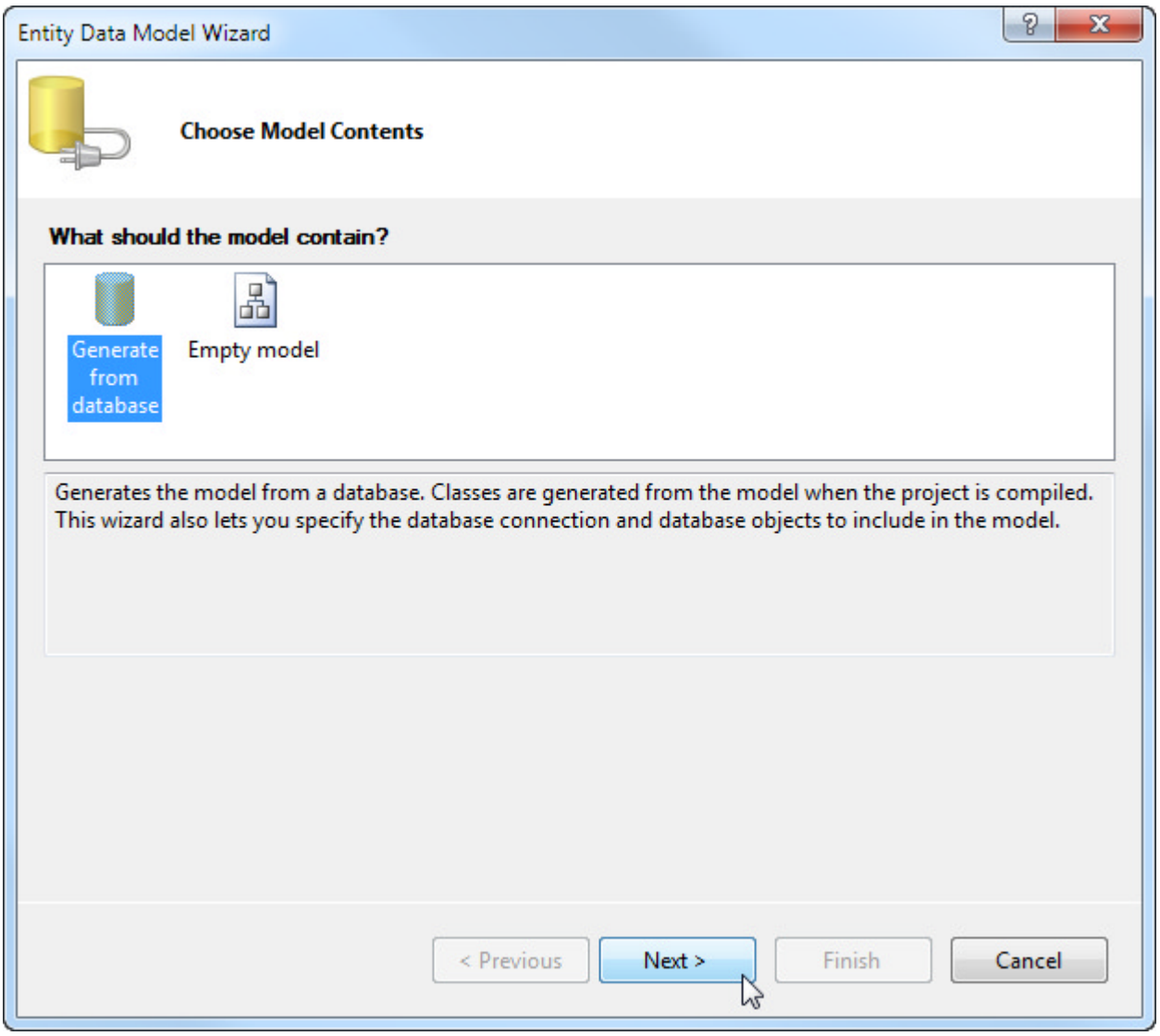
\Grid\<VB|CS>\RadGridView\ObjectRelationalHierarchy

In the final lab on hierarchical tables, we will construct the hierarchy programmatically in the codebehind for the form itself.

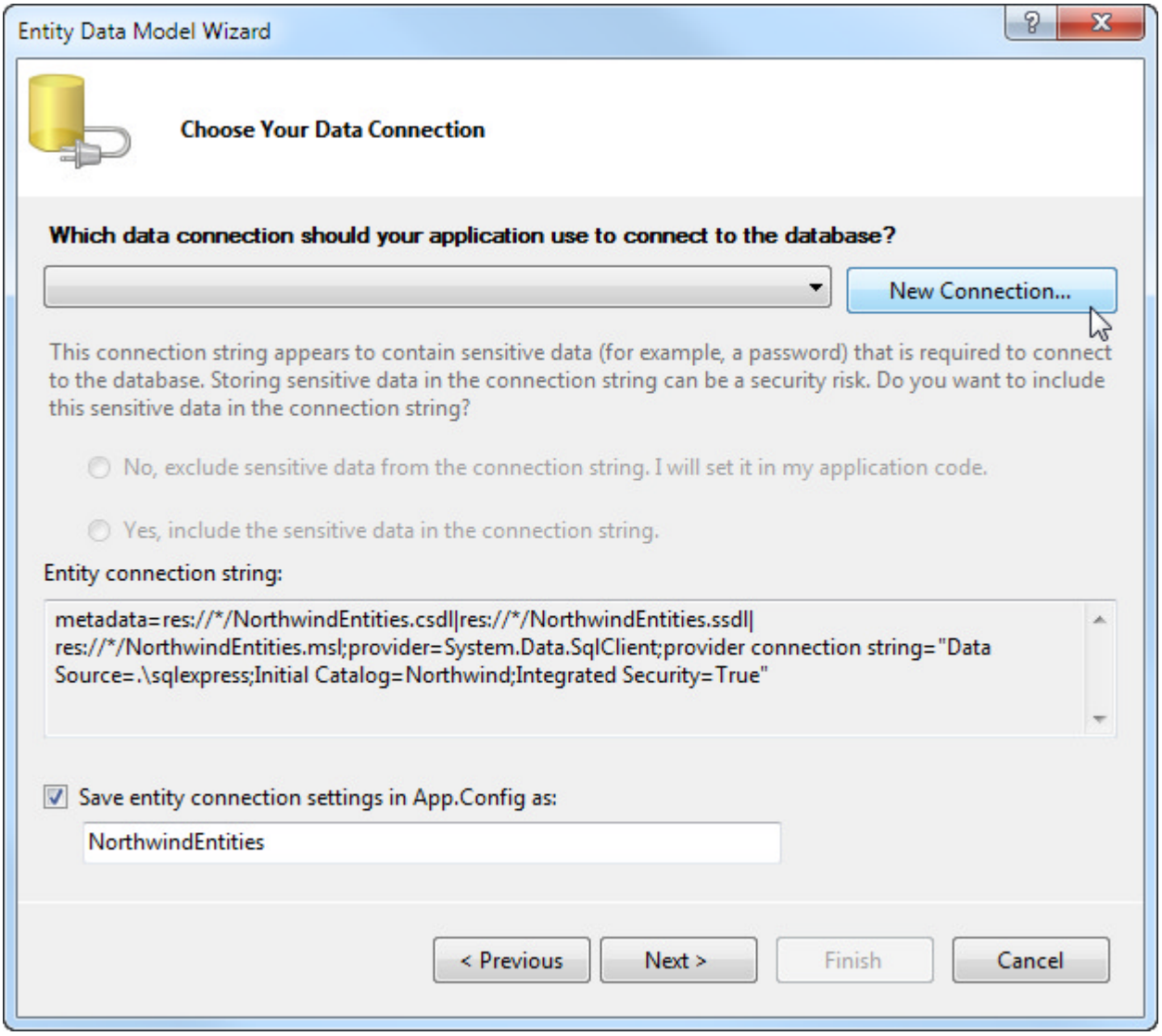
1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form.
3. Right click the project file and select Add > New Item.
4. Select ADO.NET Entity Data Model and change the Name to NorthwindEntities and click Add.



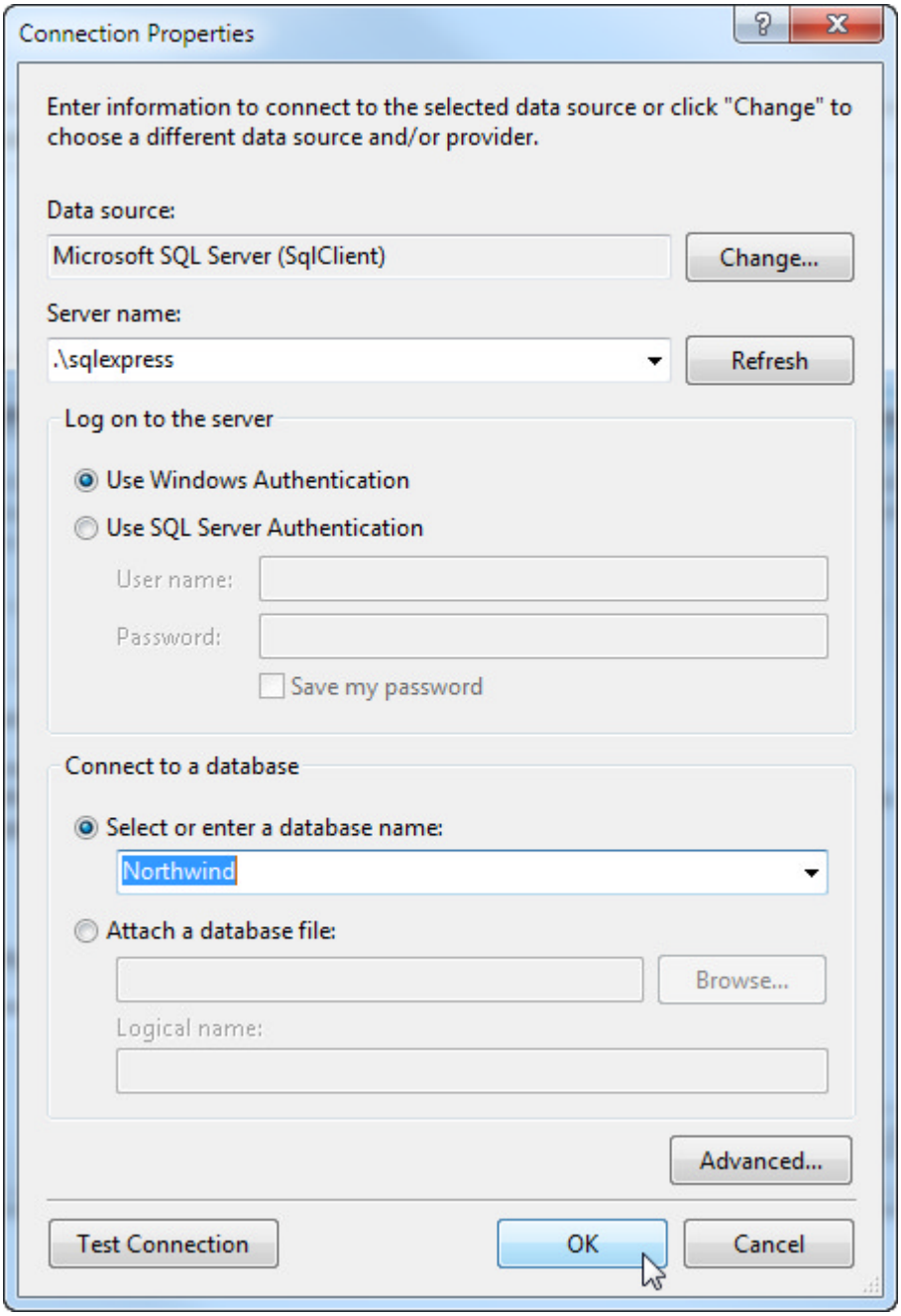
5. Select Generate from database option and click Next.



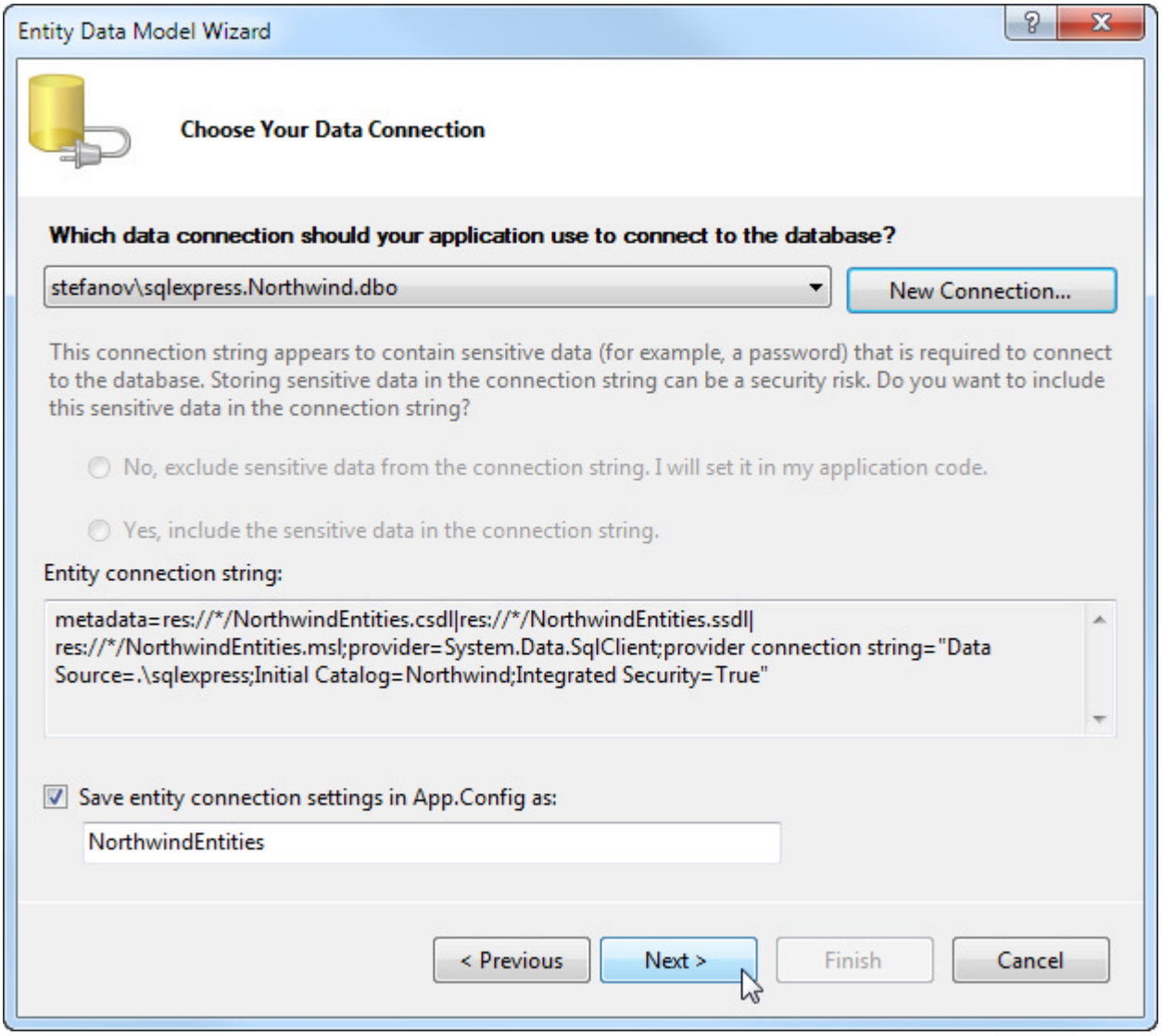
6. In the new window select New Connection ...



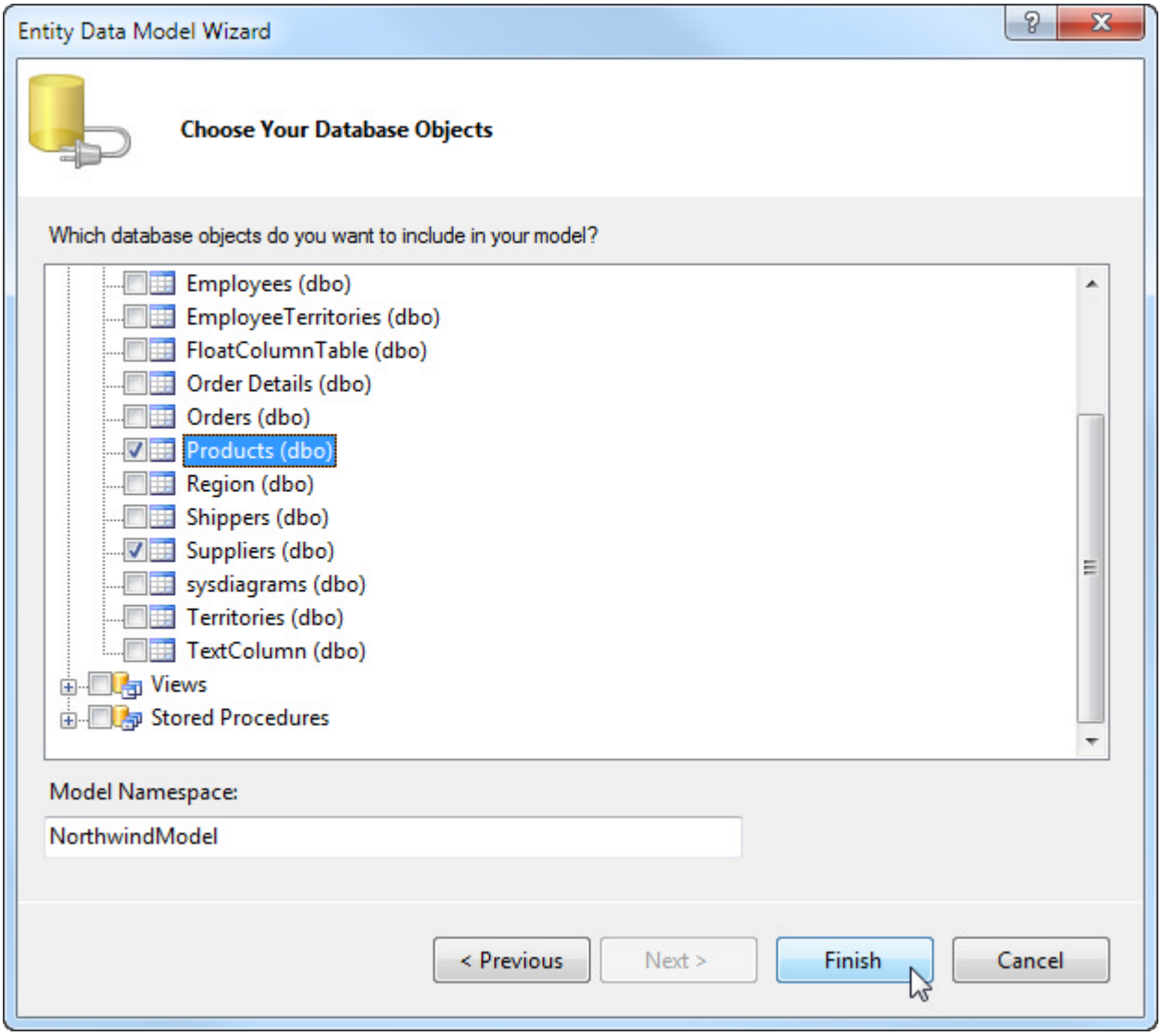
7. Type your server name, select the Northwind database and click OK.



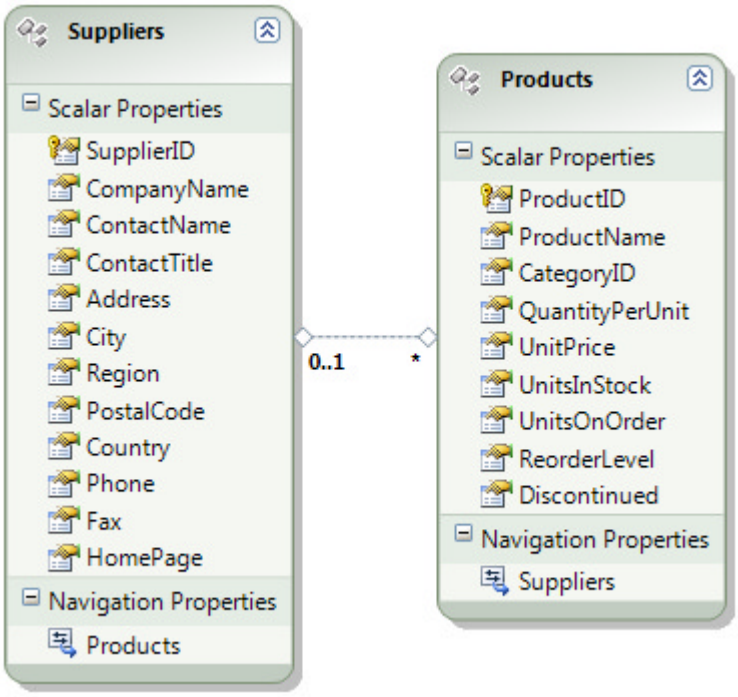
8. Click Next in the Entity Data Model Wizard window.



9. Expand the tables node and select the Products and Suppliers tables.



10. The entity model will be generated. It will look like this.



11. Double click the form to create Load event handler. Create an instance of the NorthwindEntities, create a query that will get the suppliers and include the Products table. Set the grid DataSource to the query to list and set the AutoGenerateHierarchy to true.

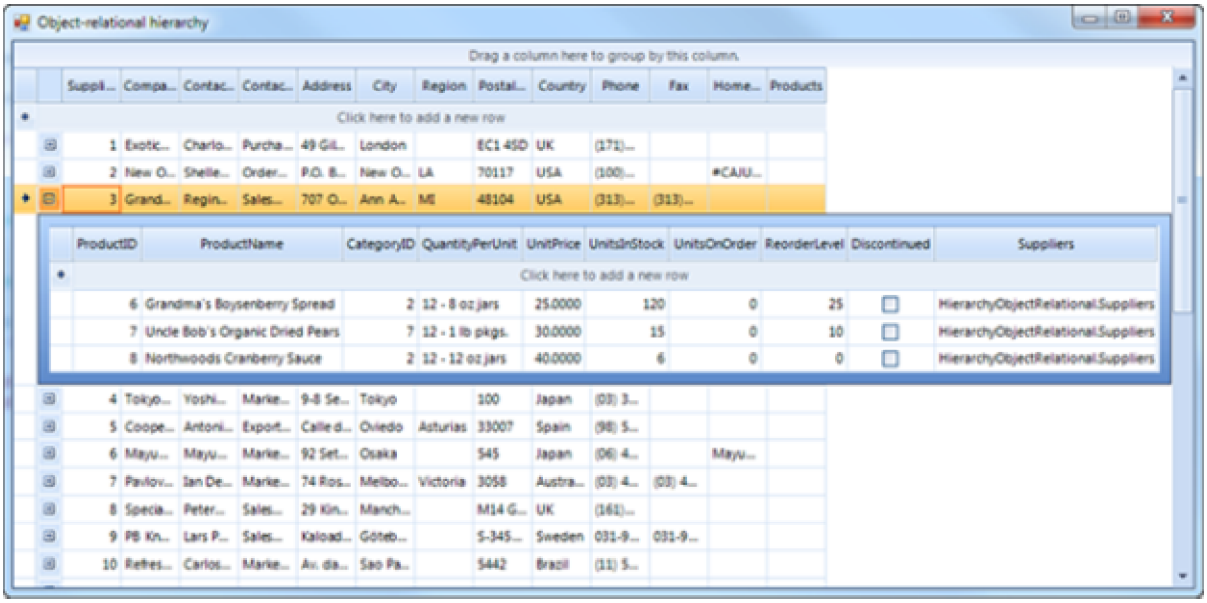
[VB.NET] Object-relational hierarchy

```
Dim entities As New NorthwindEntities()
Dim query = From suppliers In entities.Suppliers.Include("Products") Select suppliers
Me.RadGridView1.DataSource = query.ToList()
Me.RadGridView1.AutoGenerateHierarchy = True
```

[C#] Object-relational hierarchy

```
NorthwindEntities entities = new NorthwindEntities();
var query = from suppliers in entities.Suppliers.Include("Products") select suppliers;
this.radGridView1.DataSource = query.ToList();
this.radGridView1.AutoGenerateHierarchy = true;
```

12. Run the project to view the created hierarchy



Load-On-Demand hierarchy

In many cases you may need to load your data not when RadGridView is being initialized, but at a later moment, when you interact with RadGridView or with your application. For example, a child template can be loaded on demand to delay the initialization of a resource-demanding feature of the application until it is required. In order to load a GridViewTemplate on demand, you should follow these high-level steps:

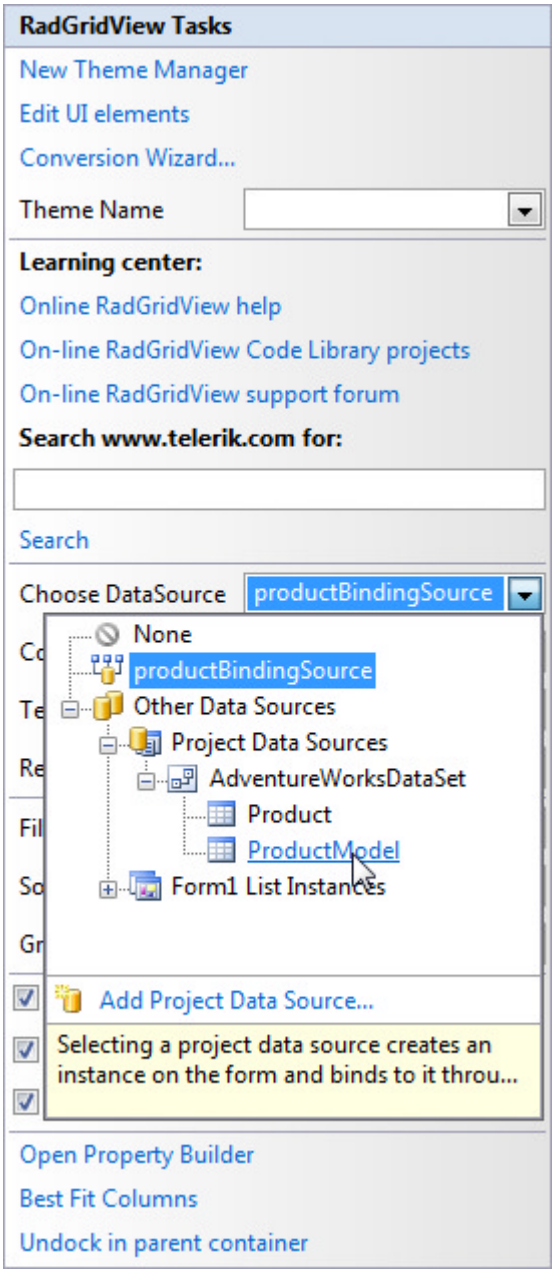
- Create and define a columns schema for the presented data at the first level of the hierarchy.
- Create and define a columns schema for the presented data and a child GridViewTemplate.
- Create and associate a GridViewEventDataProvider with the child GridViewTemplate.
- Handle the RowSourceNeeded event to populate the data for each parent row.



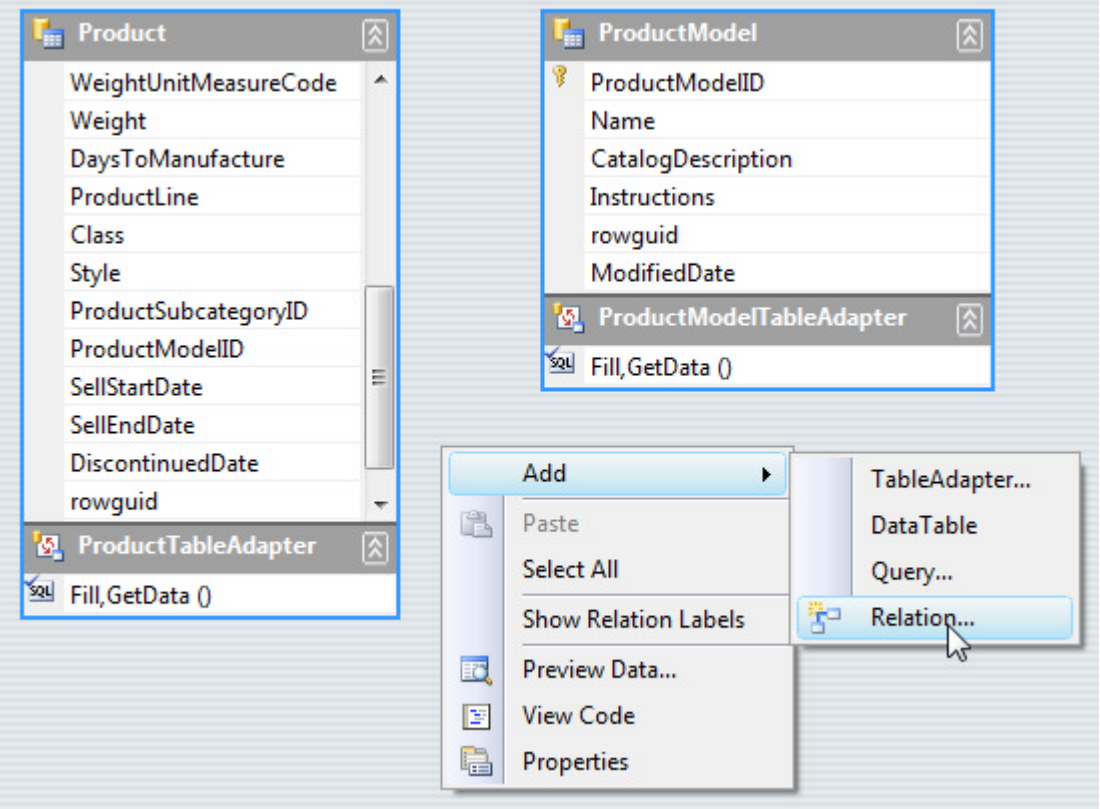
You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\HierarchyLoadOnDemand

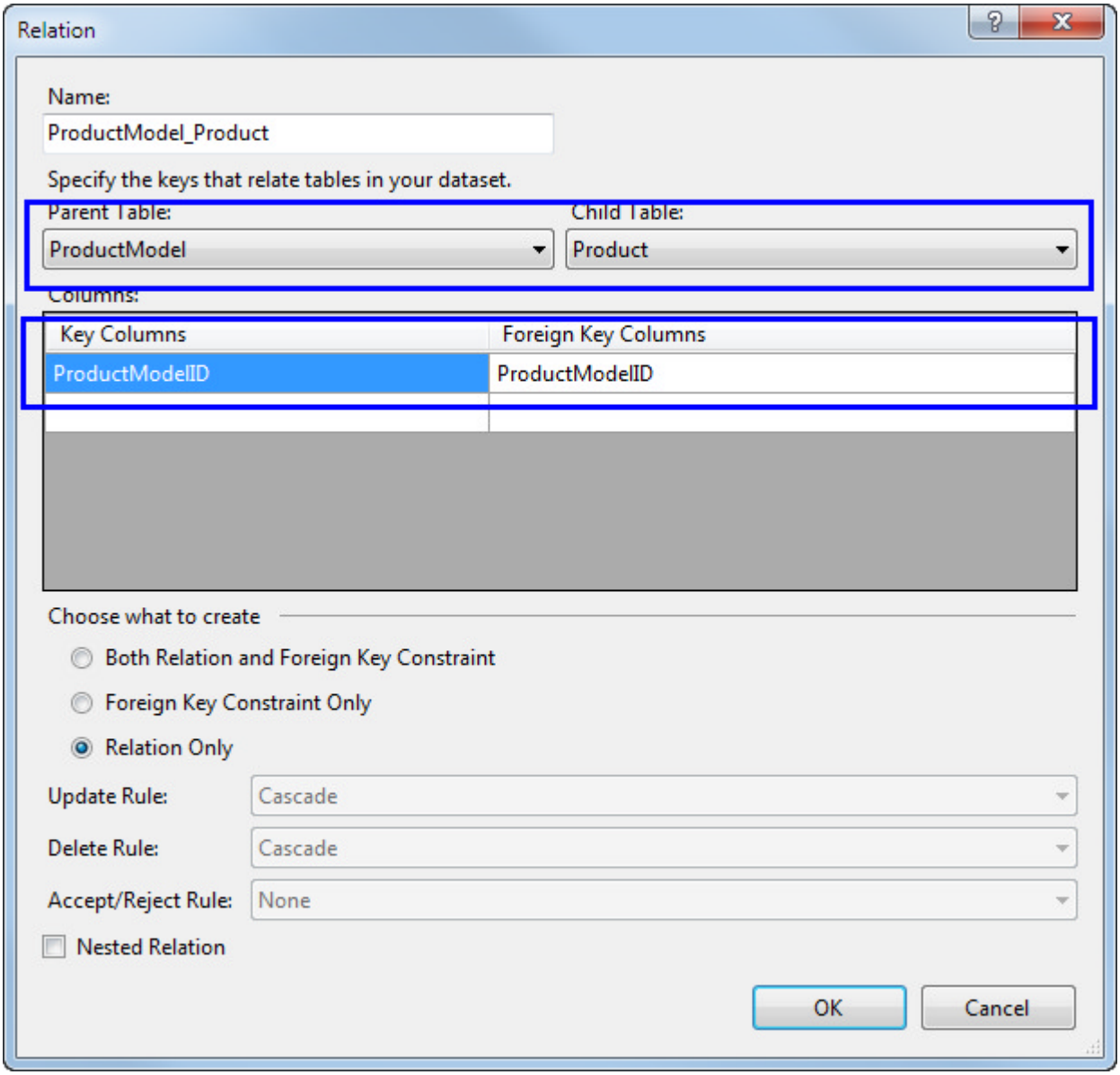
1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form, and use the Smart Tag to create a new DataSource. Use the AdventureWorks database, and select both the Product and ProductModel tables.
3. Again using the Smart Tag, first select the Product, then the ProductModel tables to create the table adapters and then set the grid datasource to None, since we are going to assisgn the data source in the code behind.



4. Double click the created AdventureWorksDataSet. Right click the designer and select Add > Relation



5. Set the Parent Table to Product and the Child Table to ProduceModel. Both column settings should point to ProductModelID.



6. First, create a columns schema for the first (parent) level of the hierarchy. If RadGridView is in a data-bound mode and we do not need to set a custom schema, we can just set the DataSource property of RadGridView which will set the schema and will populate the parent level with data.

[VB.NET] Defining the child template

```
Private Function CreateChildTemplate() As GridViewTemplate
    Dim template As New GridViewTemplate()
    template.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill

    Dim namecolumn As New GridViewTextBoxColumn("Name")
    Dim productNumberColumn As New GridViewTextBoxColumn("ProductNumber")
    Dim colorColumn As New GridViewTextBoxColumn("Color")
    Dim listPriceColumn As New GridViewDecimalColumn("ListPrice")
    Dim sizeColumn As New GridViewTextBoxColumn("Size")
    Dim weightColumn As New GridViewDecimalColumn("Weight")
    Dim discontinuedColumn As New GridViewDateTimeColumn("DiscontinuedDate")
    template.Columns.AddRange(namecolumn, productNumberColumn, colorColumn, listPriceColu
```

```
sizeColumn, weightColumn, discontinuedColumn)

    Return template
End Function
```

[C#] Defining the child template

```
private GridViewTemplate CreateChildTemplate()
{
   GridViewTemplate template = new GridViewTemplate();
   template.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill;

   GridViewTextBoxColumn namecolumn = new GridViewTextBoxColumn("Name");
   GridViewTextBoxColumn productNumberColumn = new GridViewTextBoxColumn("ProductNumber"
   GridViewTextBoxColumn colorColumn = new GridViewTextBoxColumn("Color");
   GridViewDecimalColumn listPriceColumn = new GridViewDecimalColumn("ListPrice");
   GridViewTextBoxColumn sizeColumn = new GridViewTextBoxColumn("Size");
   GridViewDecimalColumn weightColumn = new GridViewDecimalColumn("Weight");
   GridViewDateTimeColumn discontinuedColumn = new GridViewDateTimeColumn
("DiscontinuedDate");
   template.Columns.AddRange(namecolumn,
       productNumberColumn,
       colorColumn,
       listPriceColumn,
       sizeColumn,
       weightColumn,
       discontinuedColumn);

   return template;
}
```

7. Setup the load-on-demand mode by using GridViewEventDataProvider and RowSourceNeeded event:

Example Title

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs) Handle
MyBase.Load
    Me.ProductModelTableAdapter.Fill(Me.AdventureLT2008DataSet.ProductModel)
    Me.ProductTableAdapter.Fill(Me.AdventureLT2008DataSet.Product)



Me.radGridView1.DataSource = productModelBindingSource;
    Dim childTemplate As GridViewTemplate = CreateChildTemplate()
    Me.RadGridView1.Templates.Add(childTemplate)

    childTemplate.HierarchyDataProvider = New GridViewEventDataProvider(childTemplate)
    AddHandler RadGridView1.RowSourceNeeded, AddressOf radGridView1_RowSourceNeeded
End Sub
```

Example Title

```
void Form1_Load(object sender, EventArgs e)
{
   this.productModelTableAdapter.Fill(this.adventureLT2008DataSet.ProductModel);
   this.productTableAdapter.Fill(this.adventureLT2008DataSet.Product);
```

```
this.radGridView1.DataSource = productModelBindingSource;
   GridViewTemplate childTemplate = CreateChildTemplate();
   this.radGridView1.Templates.Add(childTemplate);

   childTemplate.HierarchyDataProvider = new GridViewEventDataProvider(childTemplate);
   this.radGridView1.RowSourceNeeded += new GridViewRowSourceNeededEventHandler
(radGridView1_RowSourceNeeded);
}
```

8. Load the data on demand for an expanded parent row by using the RowSourceNeeded event:

Example Title

```
Private Sub radGridView1_RowSourceNeeded(ByVal sender As Object, ByVal e As
GridViewRowSourceNeededEventArgs)
    Dim rowView As DataRowView = TryCast(e.ParentRow.DataBoundItem, DataRowView)
    Dim rows() As DataRow = rowView.Row.GetChildRows("ProductModel_Product")

    For Each dataRow As DataRow In rows
        Dim row As GridViewRowInfo = e.Template.Rows.NewRow()
        row.Cells("Name").Value = dataRow("Name")
        row.Cells("ProductNumber").Value = dataRow("ProductNumber")
        row.Cells("Color").Value = dataRow("Color")
        row.Cells("ListPrice").Value = dataRow("ListPrice")
        row.Cells("Size").Value = dataRow("Size")
        row.Cells("Weight").Value = dataRow("Weight")
        row.Cells("DiscontinuedDate").Value = dataRow("DiscontinuedDate")

        e.SourceCollection.Add(row)
    Next dataRow
End Sub
```

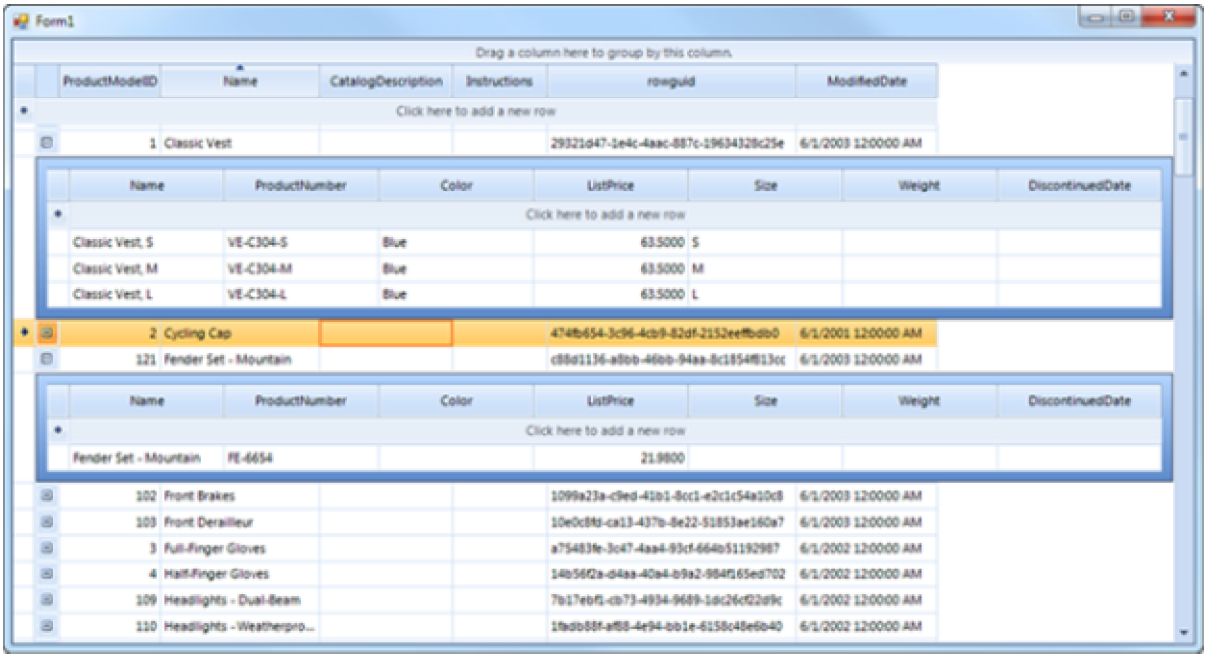
Example Title

```
void radGridView1_RowSourceNeeded(object sender, GridViewRowSourceNeededEventArgs e)
{
   DataRowView rowView = e.ParentRow.DataBoundItem as DataRowView;
   DataRow[] rows = rowView.Row.GetChildRows("ProductModel_Product");

   foreach (DataRow dataRow in rows)
   {
       GridViewRowInfo row = e.Template.Rows.NewRow();
       row.Cells["Name"].Value = dataRow["Name"];
       row.Cells["ProductNumber"].Value = dataRow["ProductNumber"];
       row.Cells["Color"].Value = dataRow["Color"];
       row.Cells["ListPrice"].Value = dataRow["ListPrice"];
       row.Cells["Size"].Value = dataRow["Size"];
       row.Cells["Weight"].Value = dataRow["Weight"];
       row.Cells["DiscontinuedDate"].Value = dataRow["DiscontinuedDate"];

       e.SourceCollection.Add(row);
   }
}
```

9. Run the project to view the created hierarchy.

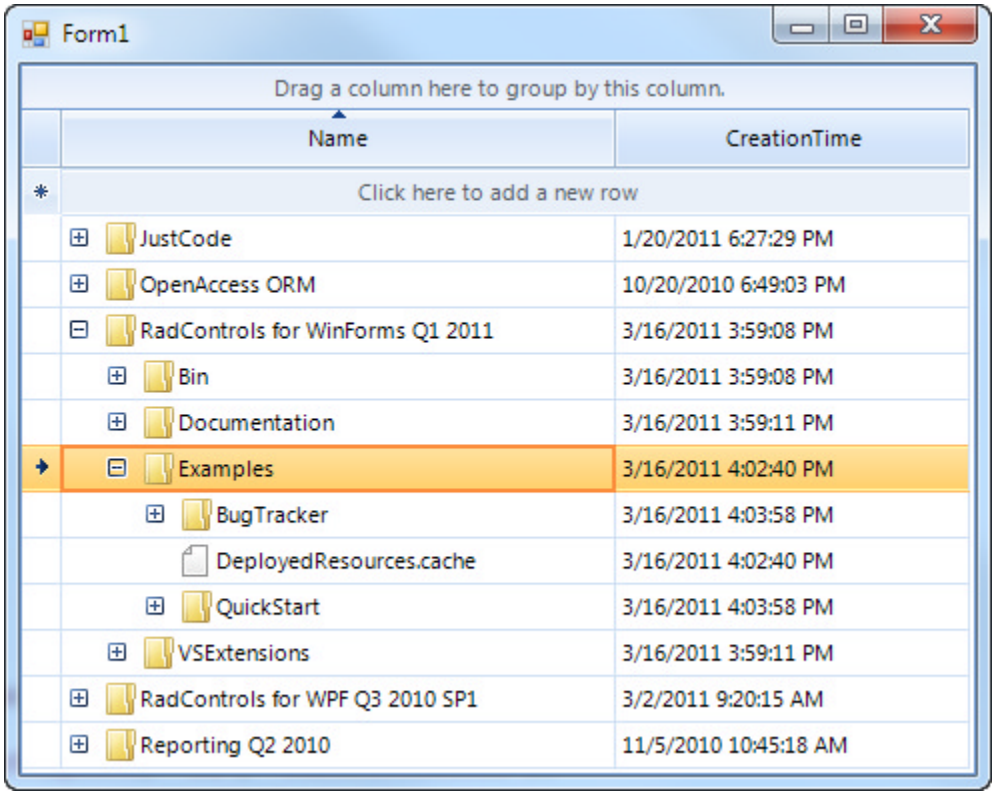


This new event based hierarchy mode can be used in different lazy loading scenarios including ORM frameworks, WCF services or complex business objects.

Self-referencing hierarchy

Except displaying hierarchical data and applying some custom criteria about when the subset of data to be displayed, the RadGridView allows you to define a relation that points back to the same table. In the cases when the hierarchical data is build from one type of items you can use a self referencing RadGridView to display the data.

The following example demonstrates how RadGridView displays a hierarchy based on the data provided by the file system where one folder can have files and folders that can have other files and folders, etc.





You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\HierarchySelfReferencing

1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form, and set its Dock property toi "Fill".
3. Create a datasource with an appropriate structure where each data record contains Id that serves as a unique identifier for the record and ParentId that determines the parent of the data record. In our case this is a business object of type FileSystemItem that can serve as a folder and as a document.

   [VB.NET] Create data source structure

```
Public Class FileSystemItem
    Private id_Renamed As Integer
    Private name_Renamed As String
    Private creationTime_Renamed As Date
    Private parentFolderId_Renamed As Integer

    Private type As String

    Public Property Id() As Integer
        Get
            Return id_Renamed
        End Get
        Set(ByVal value As Integer)
            id_Renamed = value
        End Set
    End Property
```

```
    Public Property FileSystemInfoType() As String
        Get
            Return type
        End Get
        Set(ByVal value As String)
            type = value
        End Set
    End Property

    Public Property Name() As String
        Get
            Return name_Renamed
        End Get
        Set(ByVal value As String)
            name_Renamed = value
        End Set
    End Property

    Public Property CreationTime() As Date
        Get
            Return creationTime_Renamed
        End Get
        Set(ByVal value As Date)
            creationTime_Renamed = value
        End Set
    End Property

    Public Property ParentFolderId() As Integer
        Get
            Return parentFolderId_Renamed
        End Get
        Set(ByVal value As Integer)
            parentFolderId_Renamed = value
        End Set
    End Property

    Public Sub New(ByVal id As Integer, ByVal type As String, ByVal name As String, ByVa
creationTime As Date, ByVal parentFolderId As Integer)
        Me.id_Renamed = id
        Me.type = type
        Me.name_Renamed = name
        Me.creationTime_Renamed = creationTime
        Me.parentFolderId_Renamed = parentFolderId
    End Sub
End Class
```

[C#] Create data source structure

```
public class FileSystemItem
{
   int id;
   string name;
   DateTime creationTime;
   int parentFolderId;
   private string type;
```

```
public int Id
{
    get
    {
        return id;
    }
    set
    {
        id = value;
    }
}

public string FileSystemInfoType
{
    get
    {
        return type;
    }
    set
    {
        type = value;
    }
}

public string Name
{
    get
    {
        return name;
    }
    set
    {
        name = value;
    }
}

public DateTime CreationTime
{
    get
    {
        return creationTime;
    }
    set
    {
        creationTime = value;
    }
}

public int ParentFolderId
{
    get
    {
        return parentFolderId;
    }
```

```
        set
        {
            parentFolderId = value;
        }
    }

    public FileSystemItem(int id, string type, string name, DateTime creationTime, int
parentFolderId)
    {
        this.id = id;
        this.type = type;
        this.name = name;
        this.creationTime = creationTime;
        this.parentFolderId = parentFolderId;
    }
}
```

4. Fill a BindingList with objects of type FileSystemItem. The content of the list will depend on the content of the "C:\Program Files (x86)\Telerik" folder:

[VB.NET] Fill the BindingList

```
Private list As New BindingList(Of FileSystemItem)()
Private fileFolderIndex As Integer = 0

Public Sub GetFilesAndFolders(ByVal dir As String, ByVal parentId As Integer)
    Dim di As New DirectoryInfo(dir)
    Dim rgFiles() As FileInfo = di.GetFiles()
    For Each fi As FileInfo In rgFiles
        fileFolderIndex += 1
        list.Add(New FileSystemItem(fileFolderIndex, "File", fi.Name, fi.CreationTime,
parentId))
    Next fi

    Dim dirs() As DirectoryInfo = di.GetDirectories()
    For Each d As DirectoryInfo In dirs
        fileFolderIndex += 1
        list.Add(New FileSystemItem(fileFolderIndex, "Folder", d.Name, d.CreationTime,
parentId))
        GetFilesAndFolders(d.FullName, fileFolderIndex)
    Next d
End Sub
```

[C#] Fill the BindingList

```
BindingList<FileSystemItem> list = new BindingList<FileSystemItem>();
int fileFolderIndex = 0;

public void GetFilesAndFolders(string dir, int parentId)
{
    DirectoryInfo di = new DirectoryInfo(dir);
    FileInfo[] rgFiles = di.GetFiles();
    foreach (FileInfo fi in rgFiles)
    {
        fileFolderIndex++;
        list.Add(new FileSystemItem(fileFolderIndex, "File", fi.Name, fi.CreationTime,
```

```
parentId));
   }

   DirectoryInfo[] dirs = di.GetDirectories();
   foreach (DirectoryInfo d in dirs)
   {
       fileFolderIndex++;
       list.Add(new FileSystemItem(fileFolderIndex, "Folder", d.Name, d.CreationTime,
parentId));
       GetFilesAndFolders(d.FullName, fileFolderIndex);
   }
}
```

5. The most imporant step of this example - setting up the Self Referencing mode of RadGridView. To do this, you should call the **AddSelfReference** method of the **Relations** collection passing the template that should reflect the structure of the business object and the properties that should determine the parent-child relation:

[VB.NET] Setting the self referencing mode

```
Me.RadGridView1.Relations.AddSelfReference(Me.RadGridView1.MasterTemplate, "Id",
"ParentFolderId")
Me.RadGridView1.DataSource = list
```

[C#] Setting the self referencing mode

```
this.radGridView1.Relations.AddSelfReference(this.radGridView1.MasterTemplate, "Id",
"ParentFolderId");
this.radGridView1.DataSource = list;
```

6. Hide the columns that are not useful to your clients. These columns in our case are Id, ParentFolderId and FileSystemInfoType:

[VB.NET] Hinding unnecessary columns

```
Me.RadGridView1.Columns("Id").IsVisible = False
Me.RadGridView1.Columns("ParentFolderId").IsVisible = False
Me.RadGridView1.Columns("FileSystemInfoType").IsVisible = False
```

[C#] Hinding unnecessary columns

```
this.radGridView1.Columns["Id"].IsVisible = false;
this.radGridView1.Columns["ParentFolderId"].IsVisible = false;
this.radGridView1.Columns["FileSystemInfoType"].IsVisible = false;
```

7. Last, but not least, you may want to show different images depending on the actual file system types of the data records. To do this, you should handle the **CellFormatting** event, check if the **FileSystemInfoType** is **File** or **Folder** and set the appropriate image accordingly:

Example Title

```
' Getting the images from the resources of the project
Private documentImage As Image = SamplesVB.My.Resources.Document
Private folderImage As Image = SamplesVB.My.Resources.Folder

Private Sub radGridView1_CellFormatting(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.CellFormattingEventArgs)
    Dim dataCell As GridDataCellElement = TryCast(e.CellElement, GridDataCellElement)

    If dataCell.ColumnInfo.Name = "Name" Then
```

```
        Dim dataRow As GridViewDataRowInfo = TryCast(dataCell.RowInfo, GridViewDataRowIn
        If dataRow IsNot Nothing Then
            dataCell.ImageAlignment = ContentAlignment.MiddleLeft

            Dim valueType As String = Convert.ToString(dataRow.Cells
("FileSystemInfoType").Value).ToUpperInvariant()

            If valueType.Contains("FILE") Then
                dataCell.Image = documentImage
            Else
                dataCell.Image = folderImage
            End If

            dataCell.TextImageRelation = TextImageRelation.ImageBeforeText
        End If
    Else
        dataCell.ResetValue(LightVisualElement.ImageProperty,
Telerik.WinControls.ValueResetFlags.Local)
        dataCell.ResetValue(LightVisualElement.ImageAlignmentProperty,
Telerik.WinControls.ValueResetFlags.Local)
        dataCell.ResetValue(LightVisualElement.TextImageRelationProperty,
Telerik.WinControls.ValueResetFlags.Local)
        dataCell.ResetValue(LightVisualElement.ImageLayoutProperty,
Telerik.WinControls.ValueResetFlags.Local)
    End If
End Sub
```

Example Title

```
// Getting the images from the resources of the project
Image documentImage = SamplesCS.Properties.Resources.Document;
Image folderImage = SamplesCS.Properties.Resources.Folder;

void radGridView1_CellFormatting(object sender,
Telerik.WinControls.UI.CellFormattingEventArgs e)
{
   GridDataCellElement dataCell = e.CellElement as GridDataCellElement;

   if (dataCell.ColumnInfo.Name == "Name")
   {
       GridViewDataRowInfo dataRow = dataCell.RowInfo as GridViewDataRowInfo;
       if (dataRow != null)
       {
           dataCell.ImageAlignment = ContentAlignment.MiddleLeft;

           string valueType = Convert.ToString(dataRow.Cells
["FileSystemInfoType"].Value).ToUpperInvariant();

           if (valueType.Contains("FILE"))
           {
               dataCell.Image = documentImage;
           }
           else
           {
               dataCell.Image = folderImage;
```

```
            }

            dataCell.TextImageRelation = TextImageRelation.ImageBeforeText;
        }
    }
    else
    {
        dataCell.ResetValue(LightVisualElement.ImageProperty,
Telerik.WinControls.ValueResetFlags.Local);
        dataCell.ResetValue(LightVisualElement.ImageAlignmentProperty,
Telerik.WinControls.ValueResetFlags.Local);
        dataCell.ResetValue(LightVisualElement.TextImageRelationProperty,
Telerik.WinControls.ValueResetFlags.Local);
        dataCell.ResetValue(LightVisualElement.ImageLayoutProperty,
Telerik.WinControls.ValueResetFlags.Local);
    }
}
```

This is it! In just a few steps we created a good looking self-referencing multi-column tree.

21.11 Virtual Mode

"Virtual Mode" provides a way to explicitly implement the data management of your RadGridView control. This is especially useful when binding to large groups of data, because it can let you only load the data currently being used, thus improving the performance of the grid. Virtual mode is also necessary when bound and unbound columns are used together, but sorted by the bound column's values.

In the following simplistic example of Virtual Mode, the RadGridView receives its data by calling the CellValueNeeded event handler. To allow us to see the updates, we will use a table of constantly changing randomly selected employee records as the RadGridView's contents.



You can find the complete source for this project at:

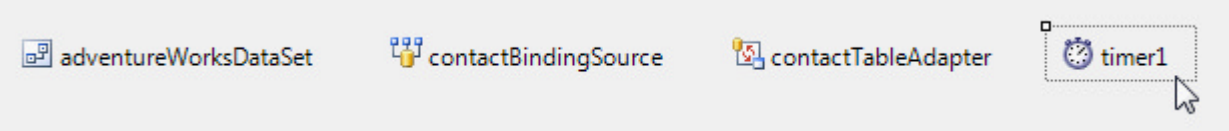
\Grid\<VB|CS>\RadGridView\VirtualMode



This example requires that you have pre installed AdventureWorks data base on your SQL server. You can download all AdventureWorks data bases from **this link at CodePlex (http://msftdbprodsamples.codeplex.com/releases/view/55926)**

1. Create a new Windows Forms Project.
2. Drag a RadGridView control onto the form, as well as a Timer object that will be used to control our data updates. Set the Timer **Interval** property to "100".
3. We will be doing all the setup for the RadGridView control itself programmatically, but first we still have to create a Dataset in our project to connect. Again, use the RadGridView's Smart Tag to create a new project DataSource.
4. Use the AdventureWorks database, and select the Person.Contact table only. We will again be using the default name AdventureWorksDataSet. **Note**: After creating the Dataset, be sure to leave the RadGridView's DataSource set to "none", as we will be using Virtual Mode to provide the data contents.

   Notice that now you have an adventureWorksDataSet object, along with a corresponding Binding Source and Table Adapter in the component tray. The Timer component will also appear in the tray.



Code to populate the DataSet using the Table Adapter is automatically placed in the Form's Load handler. Do not remove this code!

5. Add the following declarations to the Form class, directly above the constructor method. These lines declare the numbers of columns and rows for our data, along with the List of string Lists that will hold the current data.

[VB] Declaring Private Variables

```
Private ContactTable As New List(Of List(Of String))()
Private NumberOfRows As Integer = 20
Private NumberOfCols As Integer = 4
```

[C#] Declaring Private Variables

```
private List<List<string>> ContactTable = new List<List<string>>();
private int NumberOfRows = 20;
private int NumberOfCols = 4;
```

6. We also need to create a method which will update the data in our ContactTable. Add the following method to the Form's class.

   This method uses the current time's tick value as the seed to generate a new random number from 0 to 1000; then uses that index value to pull a Contact record from the dataset and add it to the table. The method will be called each time our Timer object's Tick event is triggered.

[VB] Refresh the Contact Table

```
Private Sub RefreshContactData()
 Dim random As New Random(DirectCast(DateTime.Now.Ticks, Integer))
 Dim i As Integer = 0
 While i < NumberOfRows
  Dim index As Integer = random.[Next](1000)
  Dim cr As AdventureWorksDataSet.ContactRow = adventureWorksDataSet.Contact(index)
  ContactTable(i)(0) = cr.FirstName
  ContactTable(i)(1) = cr.LastName
  ContactTable(i)(2) = cr.EmailAddress
  ContactTable(i)(3) = cr.Phone
  System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
 End While
End Sub
```

[C#] Refresh the Contact Table

```
private void RefreshContactData()
{
 Random random = new Random((int)DateTime.Now.Ticks);
 for (int i = 0; i < NumberOfRows; i++)
 {
   int index = random.Next(1000);
   AdventureWorksDataSet.ContactRow cr =
     adventureWorksDataSet.Contact[index];
   ContactTable[i][0] = cr.FirstName;
   ContactTable[i][1] = cr.LastName;
```

```
    ContactTable[i][2] = cr.EmailAddress;
    ContactTable[i][3] = cr.Phone;
  }
}
```

7. In the form's Load event handler we will set some properties of the RadGridView control. To simplify our example we will not allow editing, sorting, or filtering on the grid. If they were allowed, we would simply need to implement more functionality for our grid in Virtual Mode. Add the following code to the start of the Load event handler, just below the automatically generated statement that loads the Contact table.

[VB] Initialize Grid, Master Grid View

```
Private Sub RadGridViewLab8_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
  Me.contactTableAdapter.Fill(Me.adventureWorksDataSet.Contact)
  ' set grid properties
  radGridView1.MasterGridViewTemplate.AllowAddNewRow = False
  radGridView1.MasterGridViewTemplate.AllowCellContextMenu = False
  radGridView1.MasterGridViewTemplate.AllowDeleteRow = False
  radGridView1.MasterGridViewTemplate.AllowEditRow = False
  radGridView1.EnableSorting = False
  radGridView1.EnableFiltering = False
  radGridView1.EnableGrouping = False
  '. . .
}
```

[C#] Initialize Grid and Master Grid View

```
private void RadGridViewLab8_Load(object sender, EventArgs e)
{
 this.contactTableAdapter.Fill(this.adventureWorksDataSet.Contact);
 // set grid properties
 radGridView1.MasterGridViewTemplate.AllowAddNewRow = false;
 radGridView1.MasterGridViewTemplate.AllowCellContextMenu = false;
 radGridView1.MasterGridViewTemplate.AllowDeleteRow = false;
 radGridView1.MasterGridViewTemplate.AllowEditRow = false;
 radGridView1.EnableSorting = false;
 radGridView1.EnableFiltering = false;
 radGridView1.EnableGrouping = false;
 //. . .
}
```



The grid and master table view properties could also easily be set within the Property Builder or the Properties Window.

8. We need to initialize our ContactTable variable, by adding the following lines of code to the form's Load event handler.

The "ContactTable" generic list is loaded with yet other generic lists that each contain four empty strings. The four empty strings will be loaded later with the four columns worth of contact data.

[VB] Initialize "ContactTable"

```
Private Sub RadGridViewLab8_Load(ByVal sender As Object, ByVal e As EventArgs) Handles
MyBase.Load
  ' . . .
  For i As Integer = 0 To NumberOfRows - 1
    Dim list As List(Of String) = New List(Of String)(New String() {String.Empty,
String.Empty, String.Empty, String.Empty})
```

```
    ContactTable.Add(list)
  Next i
  '. . .
}
```

[C#] Initialize "ContactTable"

```
private void RadGridViewLab8_Load(object sender, EventArgs e)
{
 this.contactTableAdapter.Fill(this.adventureWorksDataSet.Contact);
 // . . .
 for (int i = 0; i < NumberOfRows; i++)
   ContactTable.Add(new List<string> { string.Empty, string.Empty, string.Empty,
string.Empty });
 //. . .
}
```

9. Add the code to set up the data columns and rows, size the columns, and start the timer. Add the following lines to the end of the Form's Load handler to finish setting up the grid for Virtual Mode and start the timer.

[VB] Setup Columns/Rows and Start Timer

```
Private Sub RadGridViewLab8_Load(ByVal sender As Object, ByVal e As EventArgs)
 '. . .
 radGridView1.VirtualMode = True
 radGridView1.ColumnCount = NumberOfCols
 radGridView1.Columns(0).HeaderText = "First Name"
 radGridView1.Columns(1).HeaderText = "Last Name"
 radGridView1.Columns(2).HeaderText = "Email"
 radGridView1.Columns(3).HeaderText = "Phone Number"
 radGridView1.RowCount = NumberOfRows
 radGridView1.MasterGridViewTemplate.AutoSizeColumnsMode =
Telerik.WinControls.UI.GridViewAutoSizeColumnsMode.Fill
 timer1.Start()
End Sub
```

[C#] Setup Columns/Rows and Start Timer

```
private void RadGridViewLab8_Load(object sender, EventArgs e)
{
 //. . .
 radGridView1.VirtualMode = true;
 radGridView1.ColumnCount = NumberOfCols;
 radGridView1.Columns[0].HeaderText = "First Name";
 radGridView1.Columns[1].HeaderText = "Last Name";
 radGridView1.Columns[2].HeaderText = "Email";
 radGridView1.Columns[3].HeaderText = "Phone Number";
 radGridView1.RowCount = NumberOfRows;
 radGridView1.MasterGridViewTemplate.AutoSizeColumnsMode =
Telerik.WinControls.UI.GridViewAutoSizeColumnsMode.Fill;
 timer1.Start();
}
```



The number of rows and columns must be explicitly set when using Virtual Mode so that the control can request the correct cells from the CellValueNeeded handler.

10. Back in the design view of the form, double-click the Timer component to create a Tick event handler. Add the code below to refresh the table and trigger the grid update. The call to Update() notifies the grid that

the data has changed by passing the DataChanged value.

[VB] Handling the Tick Event

```
Private Sub timer1_Tick(ByVal sender As Object, ByVal e As EventArgs)
 ' reload the contact table
 RefreshContactData()
 ' signal that the grid should be updated
radGridView1.TableElement.Update(Telerik.WinControls.UI.GridUINotifyAction.DataChanged)
End Sub
```

[C#] Handling the Tick Event

```
private void timer1_Tick(object sender, EventArgs e)
{
 // reload the contact table
 RefreshContactData();
 // signal that the grid should be updated
radGridView1.TableElement.Update(Telerik.WinControls.UI.GridUINotifyAction.DataChanged);
}
```

11. The call to TableElement.Update() in the previous step will precipitate a CellValueNeeded event. Back in the design view of the form, select the grid and in the Events tab () of the Properties window, double-click the CellValueNeeded event to create an event handler and add the code below.

    The code supplies the cell value from the ContactTable using the row and column index passed in the GridViewCellValueEventArgs parameter.

    [VB] Handling the CellValueNeeded Event

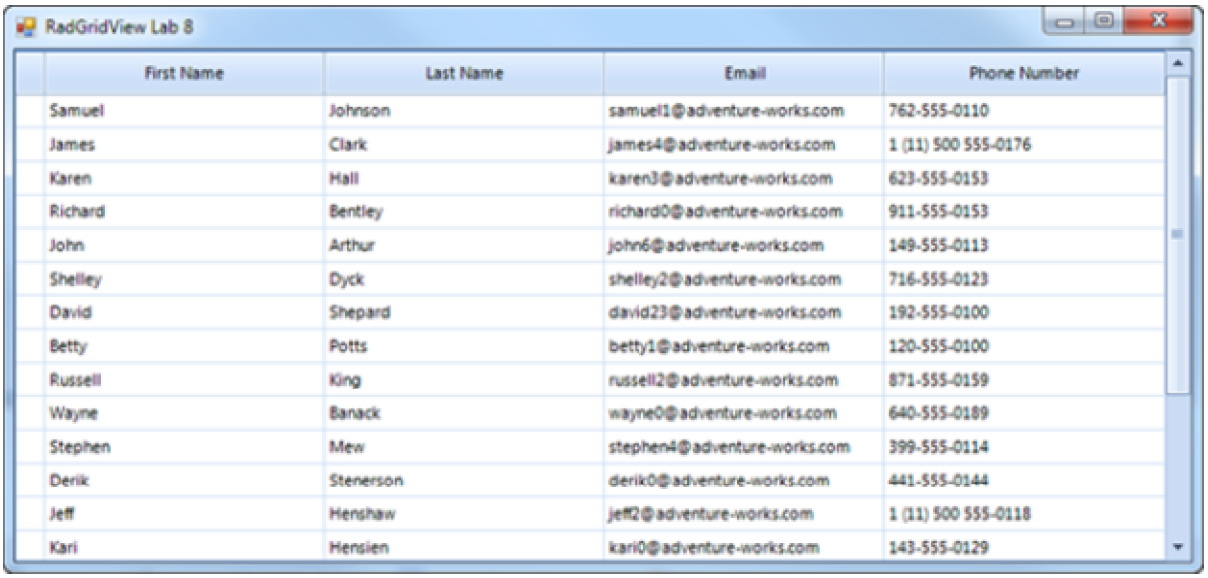
    ```
    Private Sub radGridView1_CellValueNeeded_1(ByVal sender As Object, ByVal e As
    Telerik.WinControls.UI.GridViewCellValueEventArgs)
     e.Value = ContactTable(e.RowIndex)(e.ColumnIndex)
    End Sub
    ```

    [C#] Handling the CellValueNeeded Event

    ```
    private void radGridView1_CellValueNeeded_1(object sender,
    Telerik.WinControls.UI.GridViewCellValueEventArgs e)
    {
     e.Value = ContactTable[e.RowIndex][e.ColumnIndex];
    }
    ```

    > ⚠ If you have allowed column re-ordering at runtime, changes to the column order will need to be compensated for in this handler.

12. Run the project to see the data updating automatically as the grid runs in Virtual Mode.



21.12 Using RadGridView with Dynamic LINQ Query Library

Using the Dynamic LINQ Query Library, the RadGridView control can implement dynamic paging, sorting, and filtering to millions of records, allowing unmatched performance because the record manipulation is done by dynamically created LINQ queries performed on the database itself.

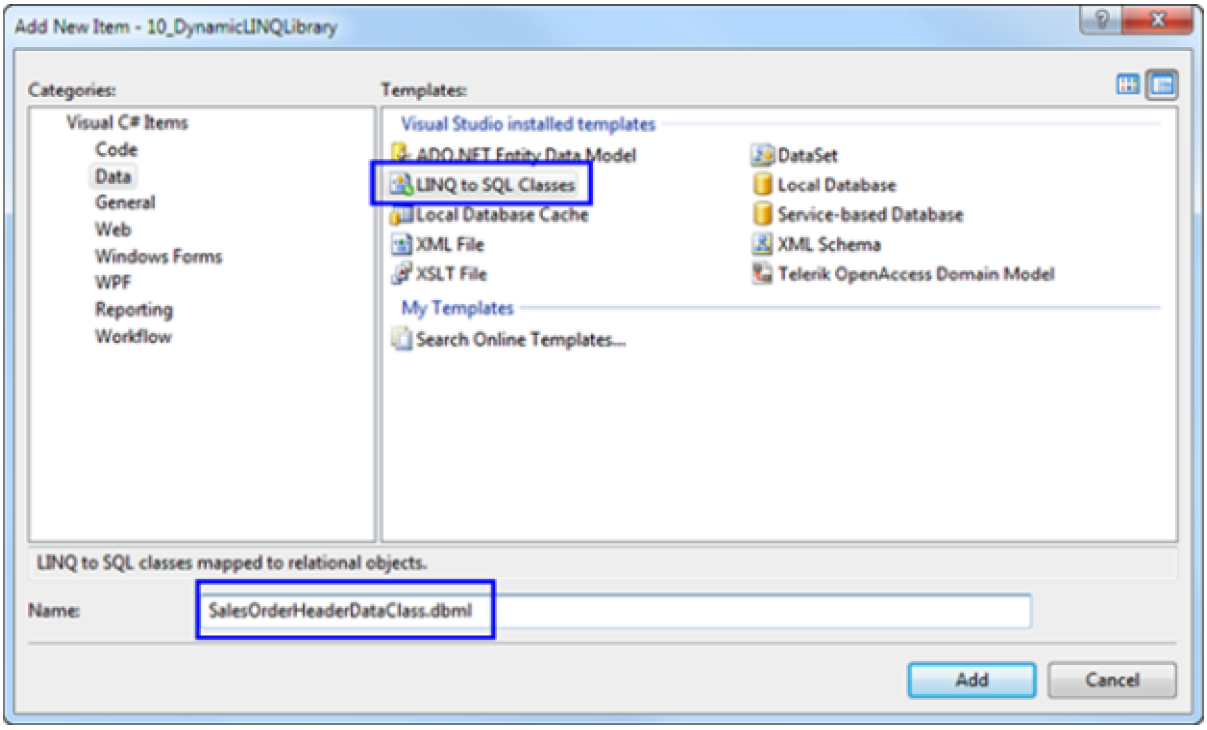
For this lab, we will be using the Dynamic.cs class, which can be downloaded along with the CS code samples here: **Visual Studio 2008 Samples (http://msdn.microsoft.com/en-us/bb330936.aspx)**



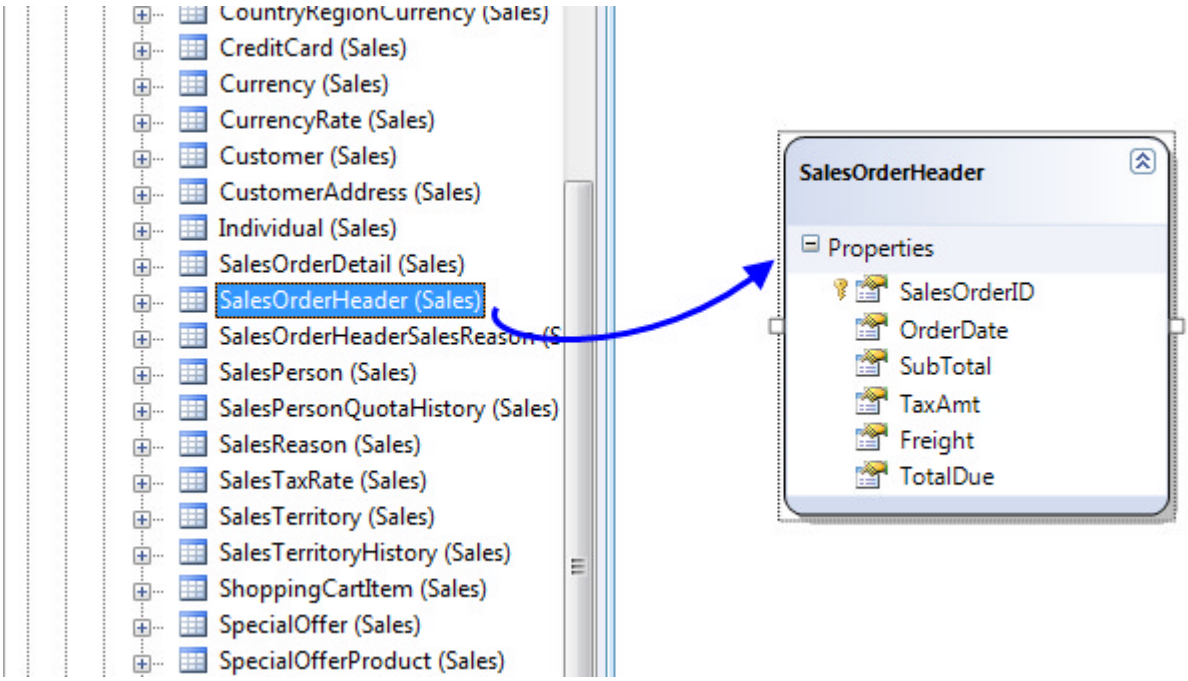
You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\DynamicLINQLibrary

1. Create a new Windows Forms Project.
2. Add the Dynamic.cs file to the project directory, and use the **Project** | **Add** | **Add Existing Item** option to add to the project as well. This class will add the LINQ extensions we will be using to build our dynamic LINQ queries.
3. Use the **Project** | **Add** | **Add New Item** option to add a LINQ to SQL Data Class to the project. Name the class SalesOrderHeaderDataClass.dbml. This class will be the basis for our LINQ dynamic queries.



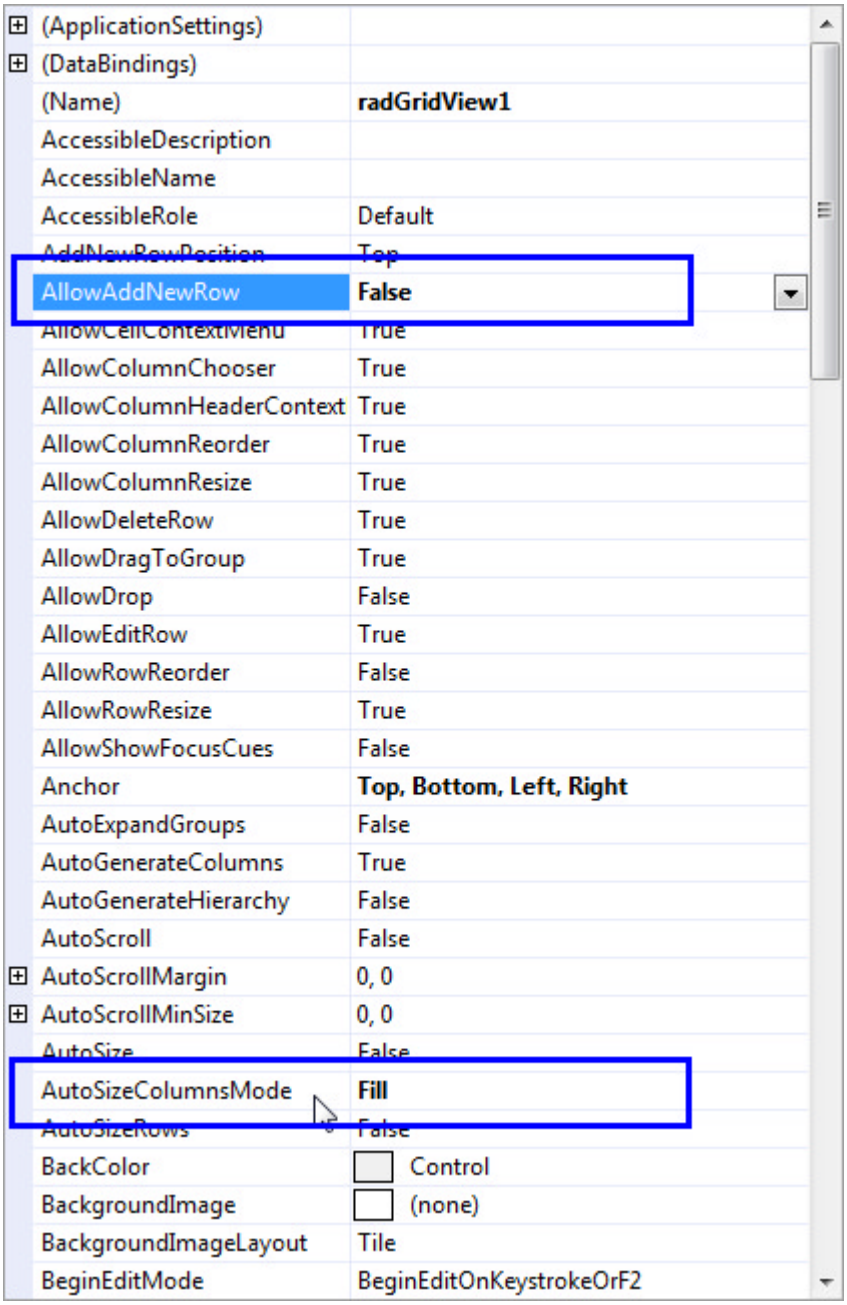
4. Next we will add the data connection to our LINQ to SQL class. While in the design view for the SalesOrderHeaderDataClass, expand the Server Explorer window of Visual Studio to the Sales.SalesOrderHeader table of the AdventureWorks database. You may need to open a connection to the database to view the tables. Then, drag the SalesOrderHeader table onto the design surface to add the table to the class.



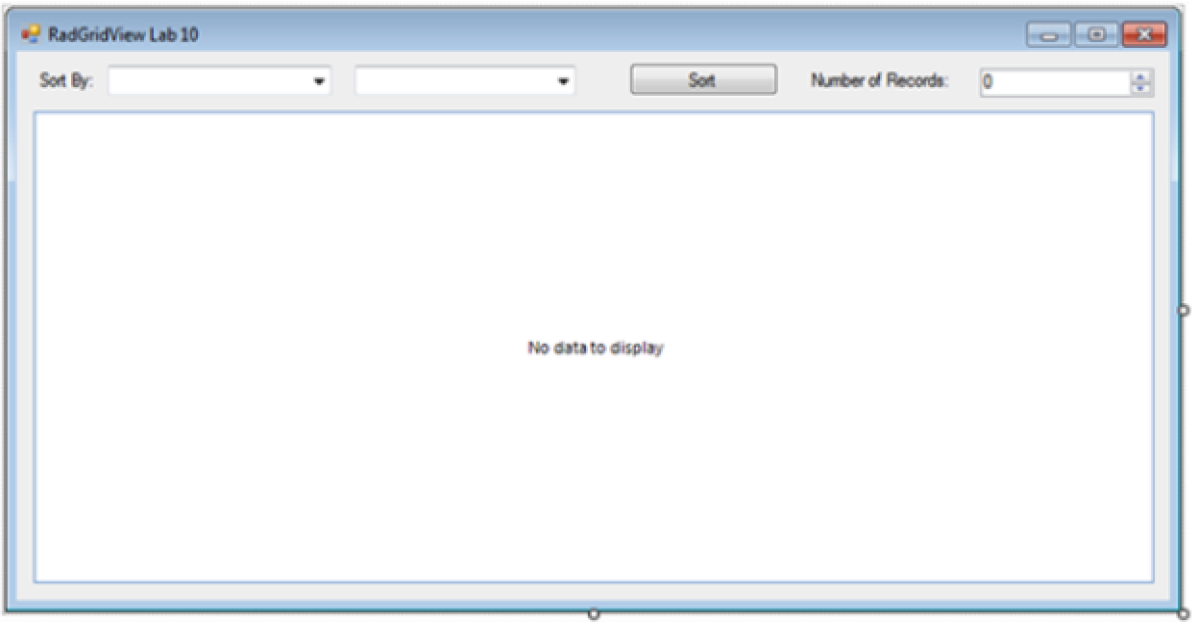
Remove all but the following fields from the Table object: SalesOrderID, OrderDate, SubTotal, TaxAmt, Freight, and TotalDue.

5. Now, to set up the user interface of our example, drag a RadGridView control onto the Form, and set the

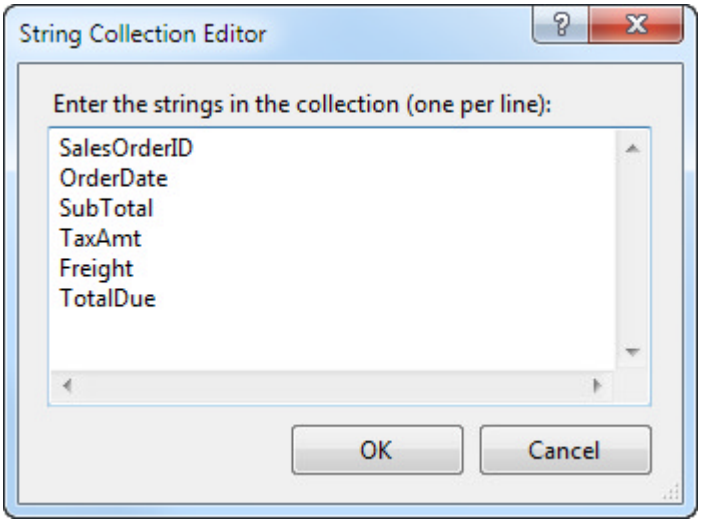
following properties of the MasterTemplate using the Property Builder.



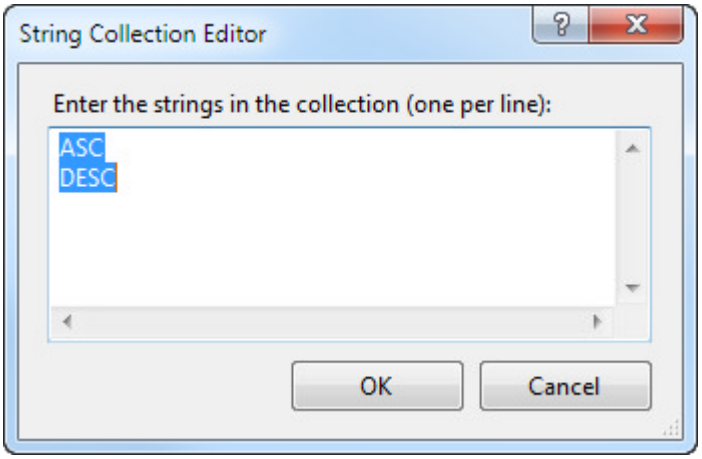
6. Also drag onto the Form the following controls: two ComboBoxes, a NumberUpDown control, a Button, and two Labels. Arrange the controls similar to the layout shown below, and name the controls as follows, from left to right: cbField, cbSortType, btnSort, and numRecords.



7. To populate the Field combobox, use the Smart Tag to open the Items list editor, and add the following Fields:



8. Populate the SortType combobox as well, adding the following items:



9. To add a Click event handler for the Button control, double click the button in design view, and add the following code to the new event handler:

[VB] Loading Data Via LINQ

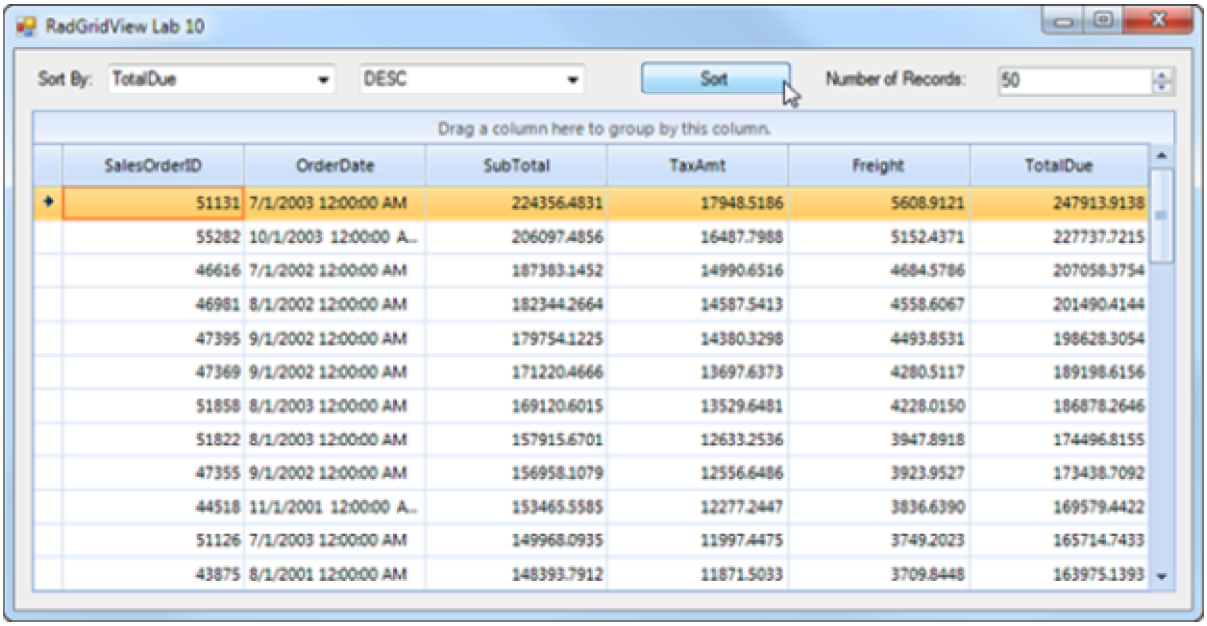
```
Private Sub btnSort_Click(ByVal sender As Object, ByVal e As EventArgs)
 Me.radGridView1.TableElement.BeginUpdate()
 Dim queryable As IQueryable = New SalesOrderHeaderDataClassDataContext
().SalesOrderHeaders.AsQueryable()
 queryable = queryable.OrderBy([String].Format("{0} {1}", cbField.Text, cbSortType.Text))
 radGridView1.DataSource = queryable.Take(Convert.ToInt32(numRecords.Value))
 Me.radGridView1.TableElement.EndUpdate(True)
End Sub
```

[C#] Loading Data Via LINQ

```
private void btnSort_Click(object sender, EventArgs e)
{
 this.radGridView1.TableElement.BeginUpdate();
 IQueryable queryable = new SalesOrderHeaderDataClassDataContext
().SalesOrderHeaders.AsQueryable();
 queryable = queryable.OrderBy(String.Format("{0} {1}",
 cbField.Text, cbSortType.Text));
 radGridView1.DataSource =
 queryable.Take(Convert.ToInt32(numRecords.Value));
 this.radGridView1.TableElement.EndUpdate(true);
}
```

This code temporarily disables updating on the grid, then assembles the new query from the options selected by the user, and uses the query to retrieve data from the table using the LINQ to SQL data class.

10. Add a reference to System.Linq.Dynamic in the "Imports" (VB) or "uses" (C#) section of the code.
11. Run the project to view the form. Select a Field and Sort direction from the comboboxes, and set the number of records to return. Click the Sort button to view the results, which are retrieved from the database using the dynamic query.



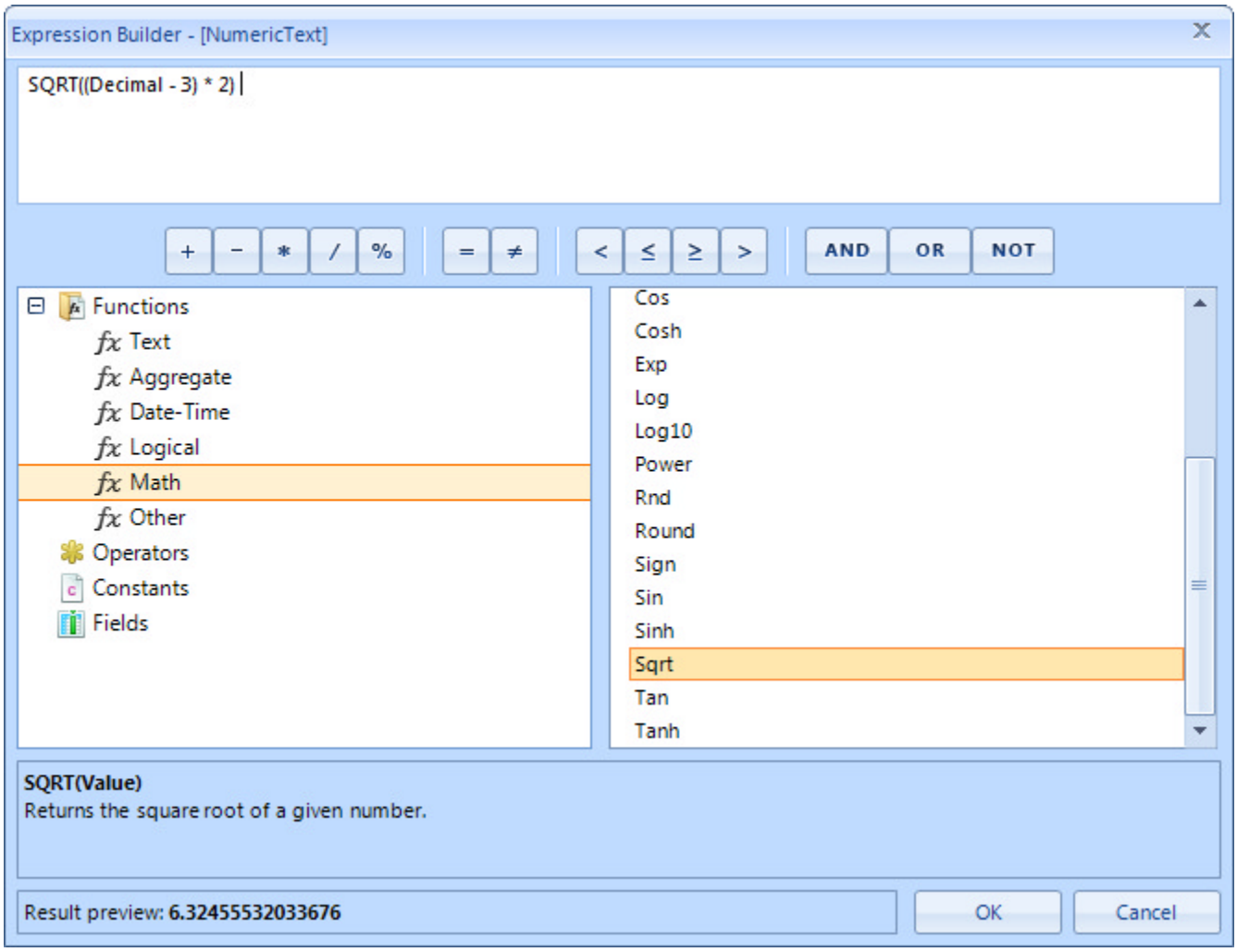
Even though the actual data table is over 30,000 records, the grid is only loading the values retrieved by the query, and so maintains high performance even with very large record sets. This same technique can be used to implement custom paging and filtering on the RadGridView control.

21.13 Expression Editor

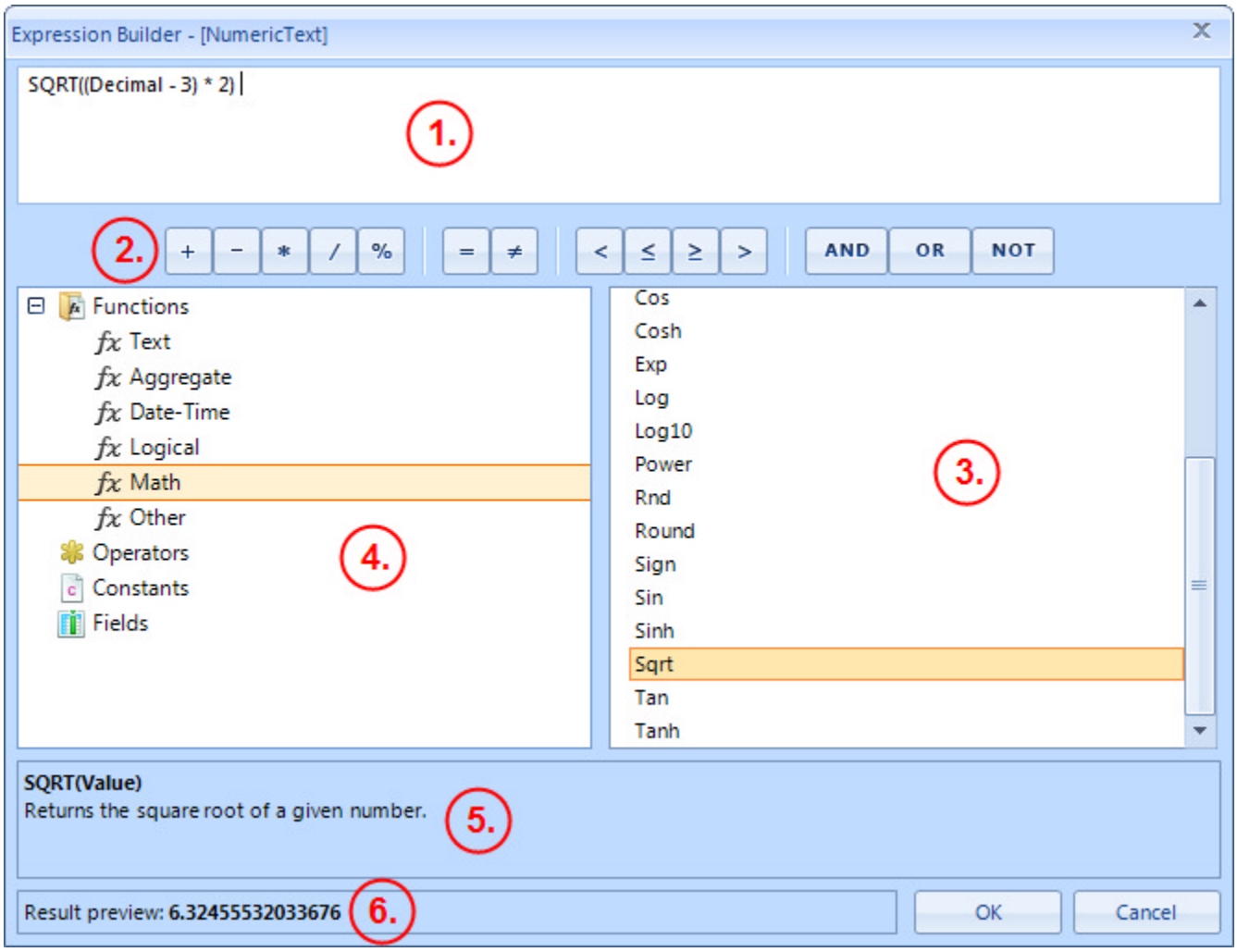
RadExpressionEditor is a powerful editor that allows you to build complex expressions using a simple Domain Specific Language, reminiscent of formulas in many popular spreadsheet applications. It provides easy access to a set of predefined functions, operators, constants. You can also access all RadGridView fields.

Expressions consist of functions, operators, constants, and identifiers (for example, the names of fields, tables, forms, and queries). The Expression Builder enables you to easily look up and insert these components, and thereby enter expressions more quickly and accurately. Expression editor can be started from the context menu of the particular column, or it can be initialized and shown from code. RadExpressionEditor also can be used in design time.

Once shown, RadExpressionEditor automatically loads all available functions, operators and current grid's fields (columns). In addition, every entered expression is parsed and evaluated. A preview of the result is shown (for the current row of the grid) and the confirmation button is enabled only if there is a valid expression. Please note that the preview is shown only if there is at least one row at run time.



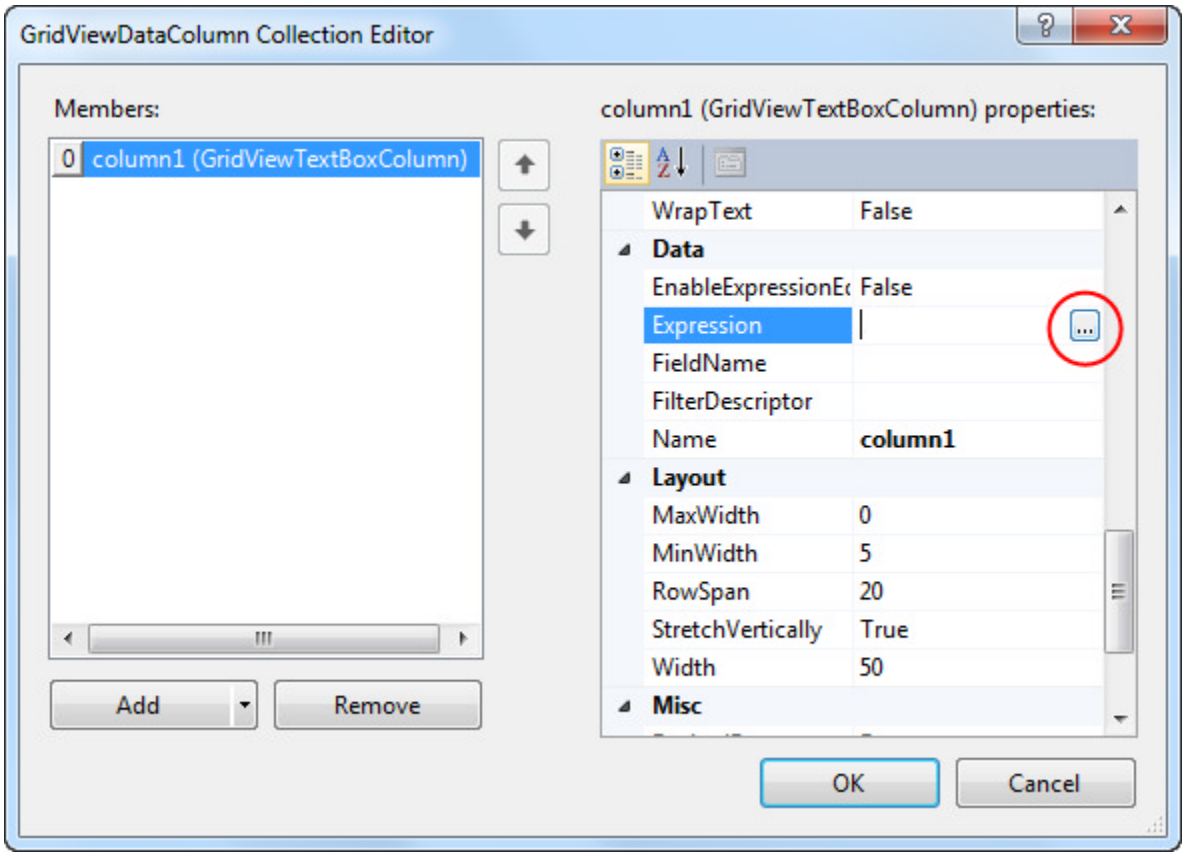
Anatomy of RadExpressionEditor



1. **Expression Box**. Type your expression here, or add expression elements by double-clicking or dragging items in the element lists below;
2. **Common Expression Operators**. Use buttons as fast shortcuts to add the required operator;
3. **Expression Elements Tree**. Navigate through the available categories of expression elements;
4. **Expression Values List**. Scroll through the available expression functions, operators, constants or fields. Double-Click or drag-and-drop to add a chosen value into the expression box;
5. **Help and information about the selected expression value**. If available, here you will see the description and the syntax of the chosen expression value;
6. **Result Preview**. A preview of the calculated result of the entered expression will be shown. The preview is evaluated as you type, and is shown only if there is a valid expression. The result is calculated for the current row in the DataView.

Design-time

At design time you can open RadExpressionEditor by clicking the ellipsis button of the columns' **Expression** property:



At design time the "Result Preview" functionality is not working, because there is no real data to evaluate. In addition, the expression is checked for validity. However, it cannot be verified according to actual data and value types. Because of this, it is possible to save an invalid expression at design time. If an invalid expression is created, an exception message will be shown when the project is ran.

Customizing RadExpressionEditor

Creating custom expression functions

Although Telerik provides a great number of predefined functions, there could be scenarios, which require additional functionality. We provide a mechanism, which allows the developer to create a custom expression method, which can be used in RadExpressionEditor or as a value of the **Expression** property. In order to do that, **ExpressionContext** class has to derive from the **ExpressionContext** class

[C#]

```
public class CustomExpressionContext : Telerik.Data.Expressions.ExpressionContext
{
   /// <summary>
   /// My custom function, which returns Pi constant.
   /// </summary>
   /// <returns></returns>
   public double Pi()
   {
       return Math.PI;
```

```
    }
}
```

[VB.NET]

```
Public Class CustomExpressionContext
    Inherits Telerik.Data.Expressions.ExpressionContext
    ''' <summary>
    ''' My custom function, which returns Pi constant.
    ''' </summary>
    ''' <returns></returns>
    Public Function Pi() As Double
        Return Math.PI
    End Function
End Class
```

Once implemented, the new custom **ExpressionContext** class, has to be set for the static **Context** property:

[C#]

```
Telerik.Data.Expressions.ExpressionContext.Context = new CustomExpressionContext();
```

[VB.NET]

```
Telerik.Data.Expressions.ExpressionContext.Context = New CustomExpressionContext()
```

Than the new expression method can be used:

[C#]

```
this.radGridView1.Columns["expression"].Expression = "PI()";
```

[VB.NET]

```
Me.RadGridView1.Columns("expression").Expression = "PI()"
```

Customizing the functions list

RadExpressionEditor supports loading functions, operators and constants from an xml file. This allows for customizing the available elements, localizing functions descriptions and adding custom functions. All available values should be described in the source xml file according to the following xml structure:

[XML]

```
<?xml version="1.0" encoding="utf-8" ?>
<ExpressionItemsList>
 <!-- Custom functions -->
 <ExpressionItem Name="GetPi (Custom Function)" Value="PI()" Syntax="PI()" Type="OtherFunc">
   <Description>
     Get the value of Pi. This is a custom added function.
   </Description>
 </ExpressionItem>

 <ExpressionItem Name="Sum" Value="SUM()" Syntax="SUM([List(Of Number)])"
Type="AggregateFunc">
```

```
   <Description>
     Computes the sum of a sequence of values.
   </Description>
 </ExpressionItem>
</ExpressionItemsList>
```

To load prepared xml source file LoadFromXML method should be used as shown bellow:

[C#]

```
string path =
"Telerik.Examples.WinControls.GridView.Expressions.ExpressionItemsListData.xml";
Stream stream = Assembly.GetExecutingAssembly().GetManifestResourceStream(path);
RadExpressionEditorForm.ExpressionItemsList.LoadFromXML(stream);
```

[VB.NET]

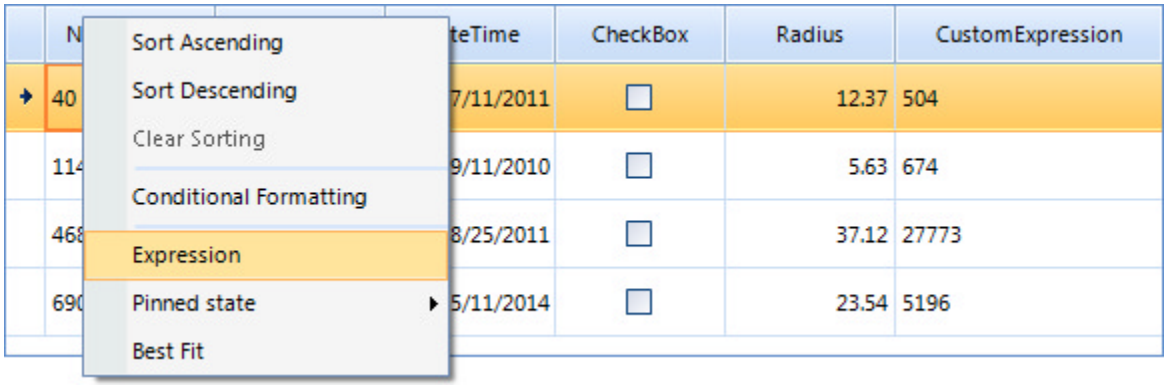
```
Dim path As String =
"Telerik.Examples.WinControls.GridView.Expressions.ExpressionItemsListData.xml"
Dim stream_ As Stream = Assembly.GetExecutingAssembly().GetManifestResourceStream(path)
RadExpressionEditorForm.ExpressionItemsList.LoadFromXML(stream_)
```

End-user Support

End-users have two options for getting the RadExpressionEditor shown at run time:

Showing RadExpressionEditor from RadGridView User Interface

End-users can open the editor by navigating to a built-in menu item from the context menu of RadGridView. In order to enable this menu item, the developer has to set the**EnableExpressionEditor** property *true*. Please note that the expression column should be an unbound column:



[C#]

```
GridViewTextBoxColumn col = new GridViewTextBoxColumn();
col.Name = "expression";
col.HeaderText = "My Expression";
col.Width = 150;
col.EnableExpressionEditor = true;
this.radGridView1.Columns.Add(col);
```

[VB.NET]

```
Dim col As GridViewTextBoxColumn = New GridViewTextBoxColumn()
col.Name = "expression"
col.HeaderText = "My Expression"
col.Width = 150
col.EnableExpressionEditor = True
Me.RadGridView1.Columns.Add(col)
```

Showing RadExpressionEditor on a custom user action

The developer can show RadExpressionEditor on a custom user action, for example on a button click, without using the standard UI of RadGridView. To do so, the developer should just call the static Show method of the RadExpressionEditorForm class:

[C#]

```
RadExpressionEditorForm.Show(this.radGridView1, this.radGridView1.Columns["expression"]);
```

[VB.NET]

```
RadExpressionEditorForm.Show(Me.RadGridView1, Me.RadGridView1.Columns("expression"))
```

21.14 Exporting RadGridView Data

Exporting Data

Telerik's RadGridView control makes for simple data exports to multiple formats including Microsoft Excel, CSV, HTML, PDF. In this lab we are going to create an application that exports RadGridView in all four formats.



All exporting functionalities are location in the **TelerikData.dll.** You need to include the following namespaces to access the types contained in TelerikData:

- Telerik.WinControls.Data

- Telerik.WinControls.UI.Export



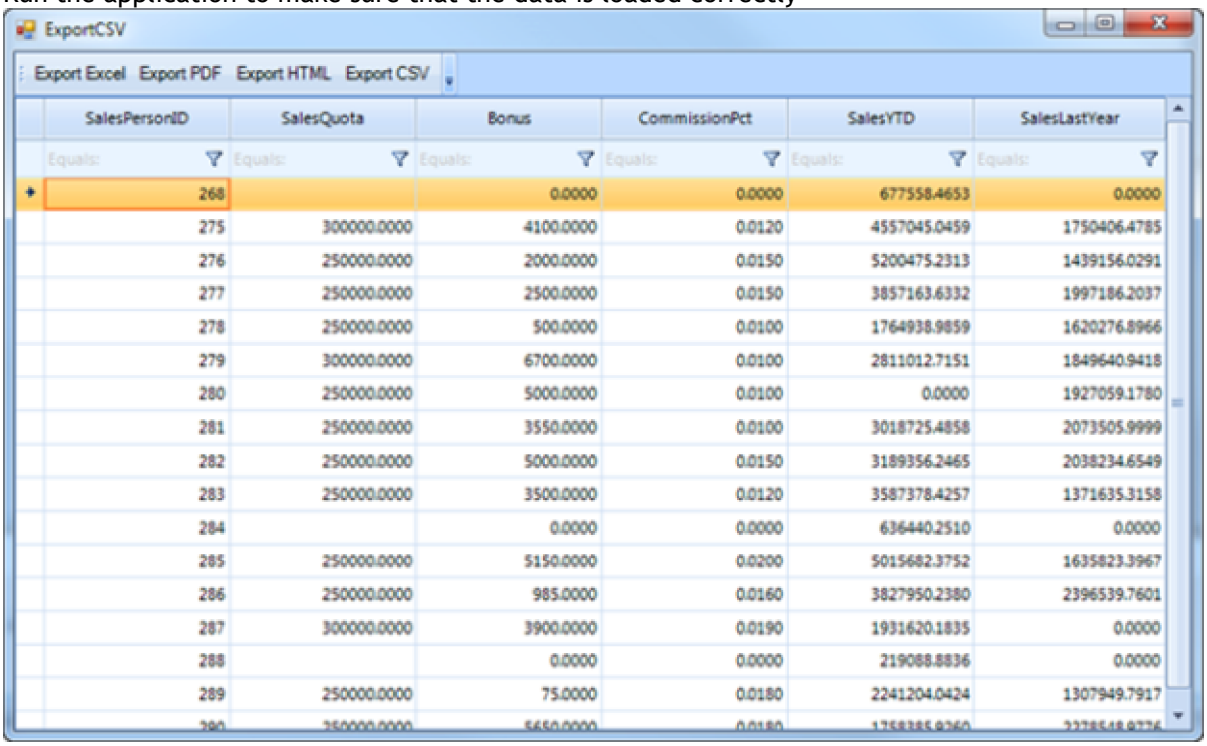
You can find the complete source for this project at:

\Grid\<VB|CS>\RadGridView\Export

1. Start a new Windows Forms Project.
2. Drag **RadCommandBar** to the form and add four **CommandBarButtons** on it. For each button set the following properties
   - **DrawText** = true
   - **Image** = ""
   - **Text** = Export Excel, Export PDF, Export HTML and Export CSV
   - **Name** = btnExportExcel, btnExportPDF, btnExportHTML and btnExportCSV
3. Drag a RadGridView grid onto the form, and set the **Dock** property to "Fill".
4. Open the RadGridView's Smart Tag, and under "Choose DataSource" select "Add Project DataSource".
5. Use the DataSource wizard to set up a DataBase DataSource to the AdventureWorks database, and the Sales.SalesPerson table. Select all the columns in the table, and save the BindingSource to the project.

   The RadGridView should now be configured to show all of the columns in the table.

6. Use the Properties window for the RadGridView to set the **AutoSizeColumnsMode** property to "Fill", **EnableGrouping** = false and **AllowAddNewRow** = false.
7. Use the Smart Tag to open the Property Builder and uncheck the following columns: "TerritoryID", "rowguid", and "ModifiedDate" from the columns list.
8. Close the Property Builder.
9. Double click each command bar button to create an event handler for it. In the following lines you can see how to populate each button Click event handler to get each exporting functionality.
10. Run the application to make sure that the data is loaded correctly



Exporting to Excel

This method offers excellent export performance and does not require MS Office installation on users' machines. The ExcelML format can be read from MS Excel 2002 (MS Office XP) and above.

1. In code view, navigate to btnExportExcel_Click event handler and create an instance of SaveFileDialog and the exporter. When instantiating the exporter pass in the parameters the RadGridView instance which you want to export.

[VB.NET] Create an instance of Excel exporter

```
Dim sfd As New SaveFileDialog()
sfd.Filter = [String].Format("{0} (*{1})|*{1}", "Excel Files", ".xls")
If sfd.ShowDialog() = DialogResult.OK Then
'create instance of the exporter
Dim exporter As New ExportToExcelML(Me.RadGridView1)
End If
```

[C#] Create an instance of Excel exporter

```
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = String.Format("{0} (*{1})|*{1}",
"Excel Files",
".xls");
if (sfd.ShowDialog() == DialogResult.OK)
```

```
{
//create instance of the exporter
ExportToExcelML exporter = new ExportToExcelML(this.radGridView1);
}
```

2. Make the desired modifications of the exporter:
   - **FileExtension -** defines the file extension for the exported file
   - **HiddenColumnOption** and **HiddenRowOption** properties to determine how hidden rows and columns should be handled by the exporter.
   - **ExportVisualSettings** property defines wether to export the visual settings (themes) too.
   - **SheetMaxRows** - RadGridView splits the data on separate sheets if the number of rows is greater than the Excel maximum. The maximum rows for Excel 2007 is 1048574 and for previous versions it is 65536 (which is the default setting)
   - **SheetName** - sets the sheet name
   - **SummariesExportOption** - defines how summary rows should be handled

   [VB.NET] Exporter settings

   ```
   'define the file extension
   exporter.FileExtension = "xls"
   'define what action should be taken for the hidden columns
   exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport
   'should it export formatting
   exporter.ExportVisualSettings = True
   'prevent exporting of summaries
   exporter.SummariesExportOption = SummariesOption.DoNotExport
   'set the sheet max rows
   exporter.SheetMaxRows = ExcelMaxRows._1048576
   'set the sheet name
   exporter.SheetName = "Sheet"
   ```

   [C#] Exporter settings

   ```
   //define the file extension (set this to xml if you want to prevent Excel warnings)
   exporter.FileExtension = "xls";
   //define what action should be taken for the hidden columns
   exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport;
   //should it export formatting
   exporter.ExportVisualSettings = true;
   //prevent exporting of summaries
   exporter.SummariesExportOption = SummariesOption.DoNotExport;
   //set the sheet max rows
   exporter.SheetMaxRows = ExcelMaxRows._1048576;
   //set the sheet name
   exporter.SheetName = "Sheet";
   ```

3. The exporter offers also two events **ExcelCellFormatting** and **ExcelTableCreated**. The first one allows you to customize the appearance of each cell related to the exported RadGridView. The second one can be used with the **AddCustomExcelRow** method, which allows adding and formatting of new custom rows on the top of every sheet. Subscribe for this events in the button click event handler. The following code creates a custom row above the exported data and customizes that row appearance. Additionally, the cells in the SalesLastYear with value above 1 500 000 will be in yellow color.

   [VB.NET] Modify cell appearance and add custom row

   ```
   Sub exporter_ExcelTableCreated(ByVal sender As System.Object, ByVal e As
   Telerik.WinControls.UI.Export.ExcelML.ExcelTableCreatedEventArgs)
   ```

```
Dim headerText As String = "Custom added header text."
Dim style As SingleStyleElement = DirectCast(sender, ExportToExcelML).AddCustomExcelRow
(e.ExcelTableElement, 50, headerText)
style.FontStyle.Bold = True
style.FontStyle.Size = 18
style.FontStyle.Color = Color.White
style.InteriorStyle.Color = Color.Red
style.InteriorStyle.Pattern = InteriorPatternType.Solid
style.AlignmentElement.HorizontalAlignment = HorizontalAlignmentType.Center
style.AlignmentElement.VerticalAlignment = VerticalAlignmentType.Center
End Sub
Sub exporter_ExcelCellFormatting(ByVal sender As System.Object, ByVal e As
Telerik.WinControls.UI.Export.ExcelML.ExcelCellFormattingEventArgs)
If e.GridRowInfoType Is GetType(GridViewTableHeaderRowInfo) Then
Dim headerBorder As New BorderStyles()
headerBorder.Color = Color.Black
headerBorder.Weight = 2
headerBorder.LineStyle = LineStyle.Continuous
headerBorder.PositionType = PositionType.Bottom
e.ExcelStyleElement.Borders.Add(headerBorder)
ElseIf e.GridColumnIndex = RadGridView1.Columns("SalesLastYear").Index Then
Dim value As Integer = Convert.ToInt32(RadGridView1.Rows(e.GridRowIndex).Cells
(e.GridColumnIndex).Value)
If value > 1500000 Then
e.ExcelStyleElement.InteriorStyle.Color = Color.Yellow
e.ExcelStyleElement.AlignmentElement.WrapText = True
End If
End If
End Sub
```

[C#] Modify cell appearance and add custom row

```
void exporter_ExcelTableCreated(object sender,
Telerik.WinControls.UI.Export.ExcelML.ExcelTableCreatedEventArgs e)
{
string headerText = "Custom added header text.";
SingleStyleElement style = ((ExportToExcelML)sender).AddCustomExcelRow(e.ExcelTableEleme
50, headerText);
style.FontStyle.Bold = true;
style.FontStyle.Size = 18;
style.FontStyle.Color = Color.White;
style.InteriorStyle.Color = Color.Red;
style.InteriorStyle.Pattern = InteriorPatternType.Solid;
style.AlignmentElement.HorizontalAlignment = HorizontalAlignmentType.Center;
style.AlignmentElement.VerticalAlignment = VerticalAlignmentType.Center;
}

void exporter_ExcelCellFormatting(object sender,
Telerik.WinControls.UI.Export.ExcelML.ExcelCellFormattingEventArgs e)
{
if (e.GridRowInfoType == typeof(GridViewTableHeaderRowInfo))
{
BorderStyles headerBorder = new BorderStyles();
headerBorder.Color = Color.Black;
headerBorder.Weight = 2;
headerBorder.LineStyle = LineStyle.Continuous;
```

```
headerBorder.PositionType = PositionType.Bottom;
e.ExcelStyleElement.Borders.Add(headerBorder);
}
else if (e.GridColumnIndex == radGridView1.Columns["SalesLastYear"].Index)
{
int value = Convert.ToInt32(radGridView1.Rows[e.GridRowIndex].Cells
[e.GridColumnIndex].Value);
if (value > 1500000)
{
e.ExcelStyleElement.InteriorStyle.Color = Color.Yellow;
e.ExcelStyleElement.AlignmentElement.WrapText = true;
}
}
}
```

4. Finally, go back to the button event handler and run the exporter. Also add code to show you a MessageBox when the export process is completed.

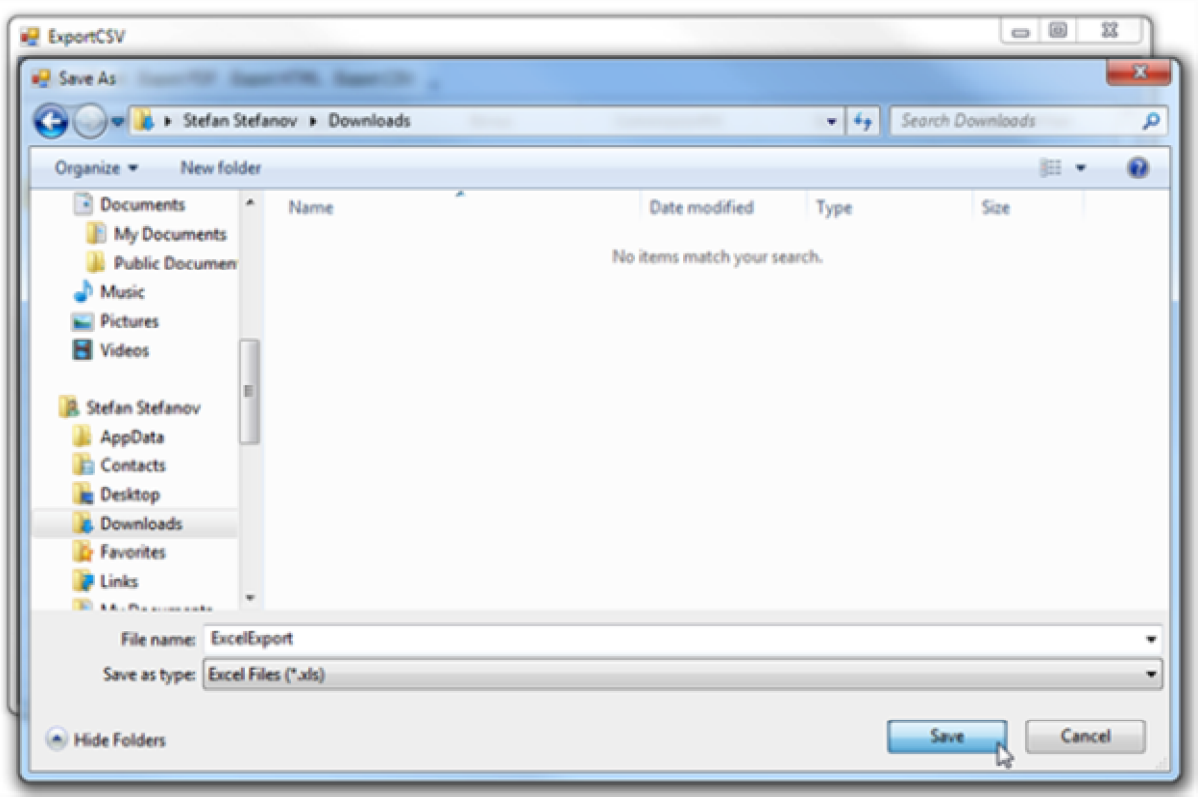
[VB.NET] Run the exporter

```
'run the export
exporter.RunExport(sfd.FileName)
MessageBox.Show("Export completed")
```

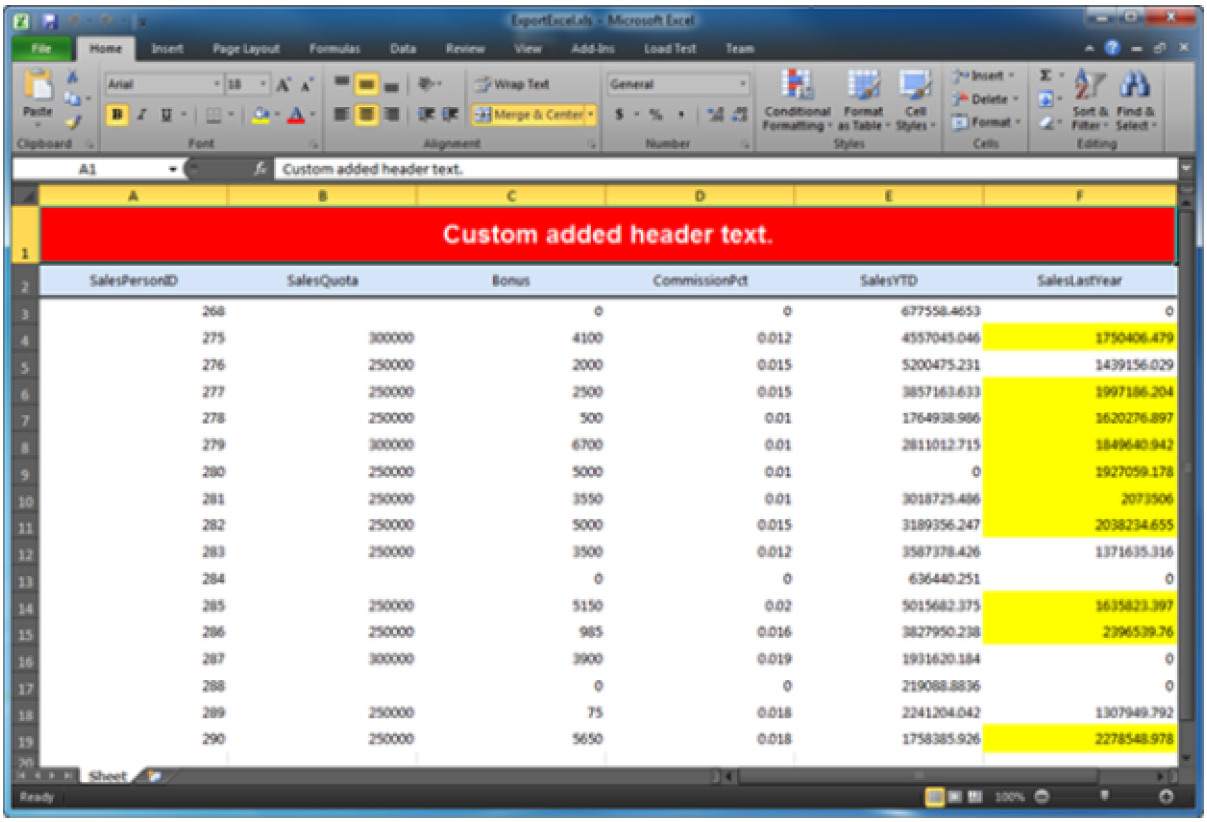
[C#] Run the exporter

```
//run the export
exporter.RunExport(sfd.FileName);
MessageBox.Show("Export completed");
```

5. Run the application and click the Export Excel button. Specify the file where you want to export the data and click Save



6. When the export is completed, a MessageBox will notify you. This is how the exported file will look like



Exporting to PDF

Export to PDF offers various rendering settings. RadGridView will be first rendered as an XHTML table and the export process will convert that table to a PDF document. That said, Export to PDF supports all of the ExportToHTML settings, but it also adds some PDF specific ones.

Here is a list of the available properties:

- **FileExtension -** defines the file extension for the exported file
- **HiddenColumnOption** and **HiddenRowOption** properties to determine how hidden rows and columns should be handled by the exporter.
- **ExportVisualSettings** property defines whether to export the visual settings (themes) too.
- **SummariesExportOption** - defines how summary rows should be handled
- **PageTitle** - sets the page title for every page in the exported file.
- **FitToPageWidth -** fits the grid to the PDF page width
- **Scale -** changes the grid size
- **TableBorderTickness -** control the tickness of the table border
- **PdfExportSettings** - these property supports various settings on PDF file level:
    - Author
    - Creator
    - EnableAdd
    - EnableCopy

- EnableModify
- EnablePrinting
- FontType
- Keywords
- Page Margins: PageBottomMargin, PageTopMargin, PageLeftMargin, PageRightMargin, PageFooterMargin
- Page Size: PageHeight and PageWidth
- Producer
- Subject
- Title

- **HTMLCellFormatting event**- since the export process first renders RadGridView data in XHTML you can use the event which comes from ExportToHTML class to make additional formatting on every cell related to the exported RadGridView

The export process follows the same logic. Create and instance of the **ExportToPDF** class and introduce the desired property modifications. Appearance formatting can be introduced in the HTMLCellFormatting event handler. Finally, run the exporter.

[VB.NET] Export to PDF

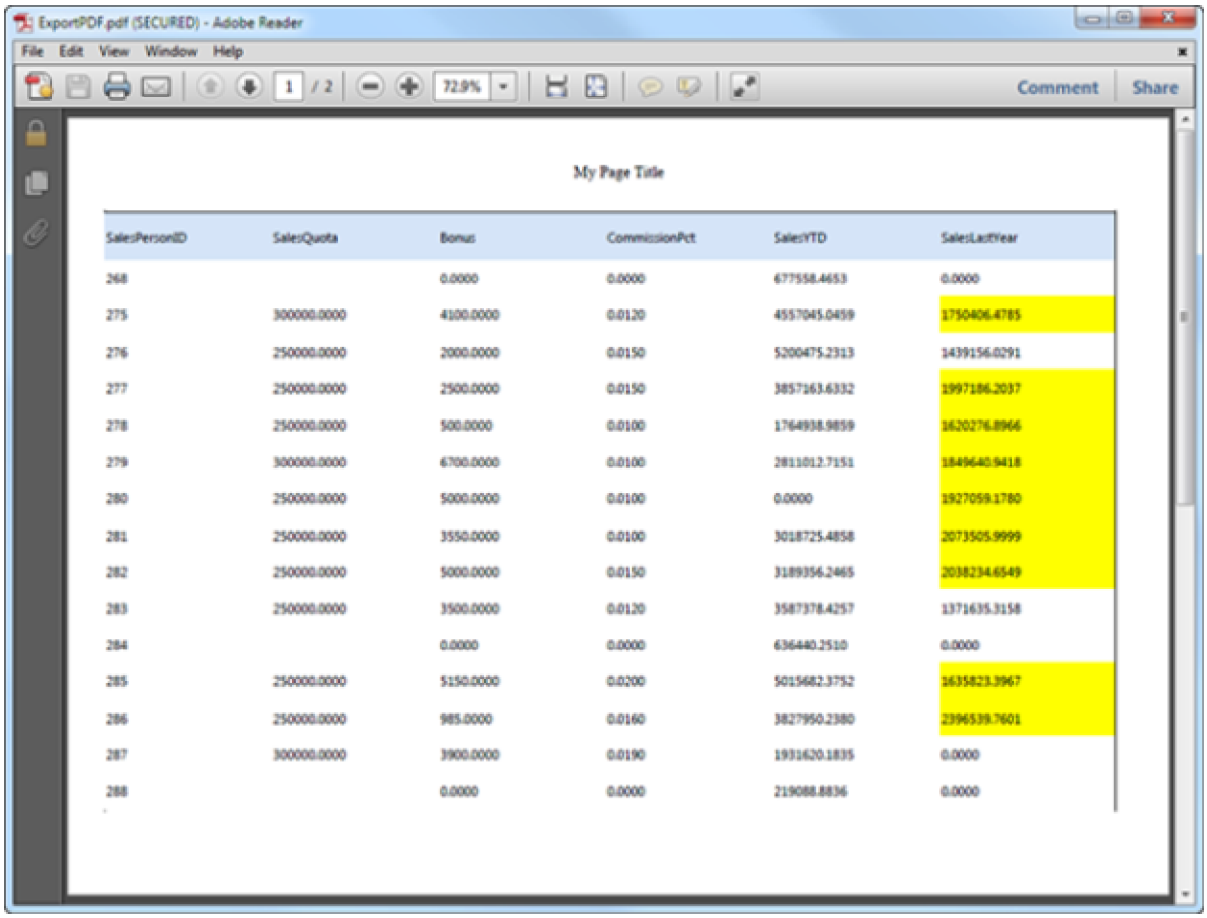
```
Private Sub btnExportPDF_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExportPDF.Click
Dim sfd As New SaveFileDialog()
sfd.Filter = [String].Format("{0} (*{1})|*{1}", "PDF Files", ".pdf")
If sfd.ShowDialog() = DialogResult.OK Then
Dim exporter As New ExportToPDF(Me.RadGridView1)
exporter.FileExtension = "pdf"
exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport
exporter.ExportVisualSettings = True
exporter.PageTitle = "My Page Title"
exporter.SummariesExportOption = SummariesOption.DoNotExport
exporter.FitToPageWidth = True
exporter.Scale = 1.2F
exporter.TableBorderThickness = 1
exporter.PdfExportSettings.PageHeight = 210
exporter.PdfExportSettings.PageWidth = 297
exporter.PdfExportSettings.Author = "Telerik"
AddHandler exporter.HTMLCellFormatting, AddressOf exporter_HTMLCellFormatting
exporter.RunExport(sfd.FileName)
MessageBox.Show("Export completed")
End If
End Sub
Sub exporter_HTMLCellFormatting(ByVal sender As System.Object, ByVal e As
HTMLCellFormattingEventArgs)
If e.GridRowIndex > -1 AndAlso e.GridColumnIndex = RadGridView1.Columns
("SalesLastYear").Index Then
Dim value As Integer = Convert.ToInt32(RadGridView1.Rows(e.GridRowIndex).Cells
(e.GridColumnIndex).Value)
If value > 1500000 Then
'cells with value over 1500000 will be yellow
e.HTMLCellElement.Styles.Add("background-color", ColorTranslator.ToHtml(Color.Yellow))
End If
```

```
End If
End Sub
```

[C#] Export to PDF

```
private void btnExportPDF_Click(object sender, EventArgs e)
{
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = String.Format("{0} (*{1})|*{1}",
"PDF Files",
".pdf");
if (sfd.ShowDialog() == DialogResult.OK)
{
ExportToPDF exporter = new ExportToPDF(this.radGridView1);
exporter.FileExtension = "pdf";
exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport;
exporter.ExportVisualSettings = true;
exporter.PageTitle = "My Page Title";
exporter.SummariesExportOption = SummariesOption.DoNotExport;
exporter.FitToPageWidth = true;
exporter.Scale = 1.2f;
exporter.TableBorderThickness = 1;
exporter.PdfExportSettings.PageHeight = 210;
exporter.PdfExportSettings.PageWidth = 297;
exporter.PdfExportSettings.Author = "Telerik";
exporter.HTMLCellFormatting += new
Telerik.WinControls.UI.Export.HTML.HTMLCellFormattingEventHandler
(exporter_HTMLCellFormatting);
exporter.RunExport(sfd.FileName);
MessageBox.Show("Export completed");
}
}
void exporter_HTMLCellFormatting(object sender,
Telerik.WinControls.UI.Export.HTML.HTMLCellFormattingEventArgs e)
{
if (e.GridRowIndex > -1
&& e.GridColumnIndex == radGridView1.Columns["SalesLastYear"].Index)
{
int value = Convert.ToInt32(radGridView1.Rows[e.GridRowIndex].Cells
[e.GridColumnIndex].Value);
if (value > 1500000)
{
//cells with value over 1500000 will be yellow
e.HTMLCellElement.Styles.Add("background-color", ColorTranslator.ToHtml(Color.Yellow));
}
}
}
```

Here is the exported file.



Exporting to HTML

This method offers excellent export performance and creates an html formatted file, which can be opened in a browser or MS Word.

- **FileExtension -** defines the file extension for the exported file
- **HiddenColumnOption** and **HiddenRowOption** properties to determine how hidden rows and columns should be handled by the exporter.
- **ExportVisualSettings** property defines wether to export the visual settings (themes) too.
- **TableCaption**- specifies the table caption
- **SummariesExportOption** - defines how summary rows should be handled
- **HTMLCellFormatting event** - gives access to a single cell's html element that allows you to make additional formatting for every html cell related to the exported RadGridView
- **HTMLTableCaptionFormatting** - can be used to make an additional formatting on the html caption table elemen

The following code creates a sample html exporter in the btnExportHTML_Click event handler:

[VB.NET] Export to HTML

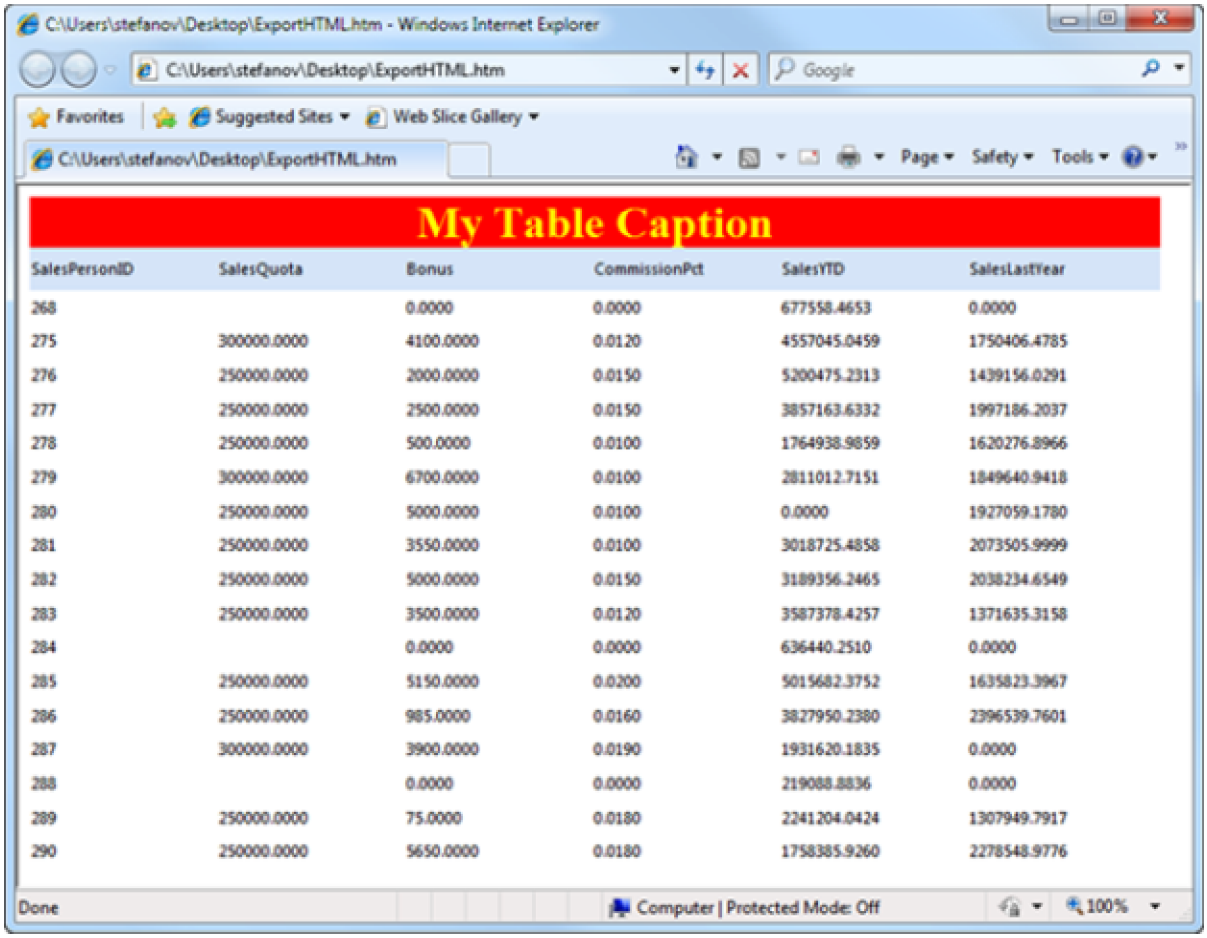
```
Private Sub btnExportHTML_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExportHTML.Click
Dim sfd As New SaveFileDialog()
sfd.Filter = [String].Format("{0} (*{1})|*{1}", "HTML Files", ".html")
```

```
If sfd.ShowDialog() = DialogResult.OK Then
Dim exporter As New ExportToHTML(Me.RadGridView1)
exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport
exporter.ExportVisualSettings = True
exporter.SummariesExportOption = SummariesOption.DoNotExport
exporter.TableCaption = "My Table"
AddHandler exporter.HTMLTableCaptionFormatting, AddressOf
exporter_HTMLTableCaptionFormatting
exporter.RunExport(sfd.FileName)
MessageBox.Show("Export completed")
End If
End Sub
Sub exporter_HTMLTableCaptionFormatting(ByVal sender As System.Object, ByVal e As
HTMLTableCaptionFormattingEventArgs)
e.TableCaptionElement.Styles.Add("background-color", ColorTranslator.ToHtml(Color.Red))
e.TableCaptionElement.Styles.Add("font-size", "200%")
e.TableCaptionElement.Styles.Add("color", ColorTranslator.ToHtml(Color.Yellow))
e.TableCaptionElement.Styles.Add("font-weight", "bold")
e.CaptionText = "My Table Caption"
End Sub
```

[C#] Export to HTML

```
private void btnExportHTML_Click(object sender, EventArgs e)
{
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = String.Format("{0} (*{1})|*{1}",
"HTML Files",
".html");
if (sfd.ShowDialog() == DialogResult.OK)
{
ExportToHTML exporter = new ExportToHTML(this.radGridView1);
exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport;
exporter.ExportVisualSettings = true;
exporter.SummariesExportOption = SummariesOption.DoNotExport;
exporter.TableCaption = "My Table";
exporter.HTMLTableCaptionFormatting += new
Telerik.WinControls.UI.Export.HTML.HTMLTableCaptionFormattingEventHandler
(exporter_HTMLTableCaptionFormatting);
exporter.RunExport(sfd.FileName);
MessageBox.Show("Export completed");
}
}
void exporter_HTMLTableCaptionFormatting(object sender,
Telerik.WinControls.UI.Export.HTML.HTMLTableCaptionFormattingEventArgs e)
{
e.TableCaptionElement.Styles.Add("background-color", ColorTranslator.ToHtml(Color.Red));
e.TableCaptionElement.Styles.Add("font-size", "200%");
e.TableCaptionElement.Styles.Add("color", ColorTranslator.ToHtml(Color.Yellow));
e.TableCaptionElement.Styles.Add("font-weight", "bold");
e.CaptionText = "My Table Caption";
}
```

This is how the exported file looks like.



Exporting to CSV

This method offers excellent export performance. It creates a csv file and supports formatting events to allow customizing exported data.

The most interesting properties are:.

- **FileExtension -** defines the file extension for the exported file
- **HiddenColumnOption** and **HiddenRowOption** properties to determine how hidden rows and columns should be handled by the exporter.
- **SummariesExportOption** - defines how to handle the summary rows
- **CSVCellFormatting event** - gives access to a single cell's element that allows you to replace the actual value for every cell related to the exported RadGridView
- **CSVTableCaptionFormatting** - can be used together with the public method AddCustomCSVRow. It allows for adding and formatting new custom rows on the top of the csv file

Here is a sample code that creates export to CSV functionality in the btnExportCSV event handler

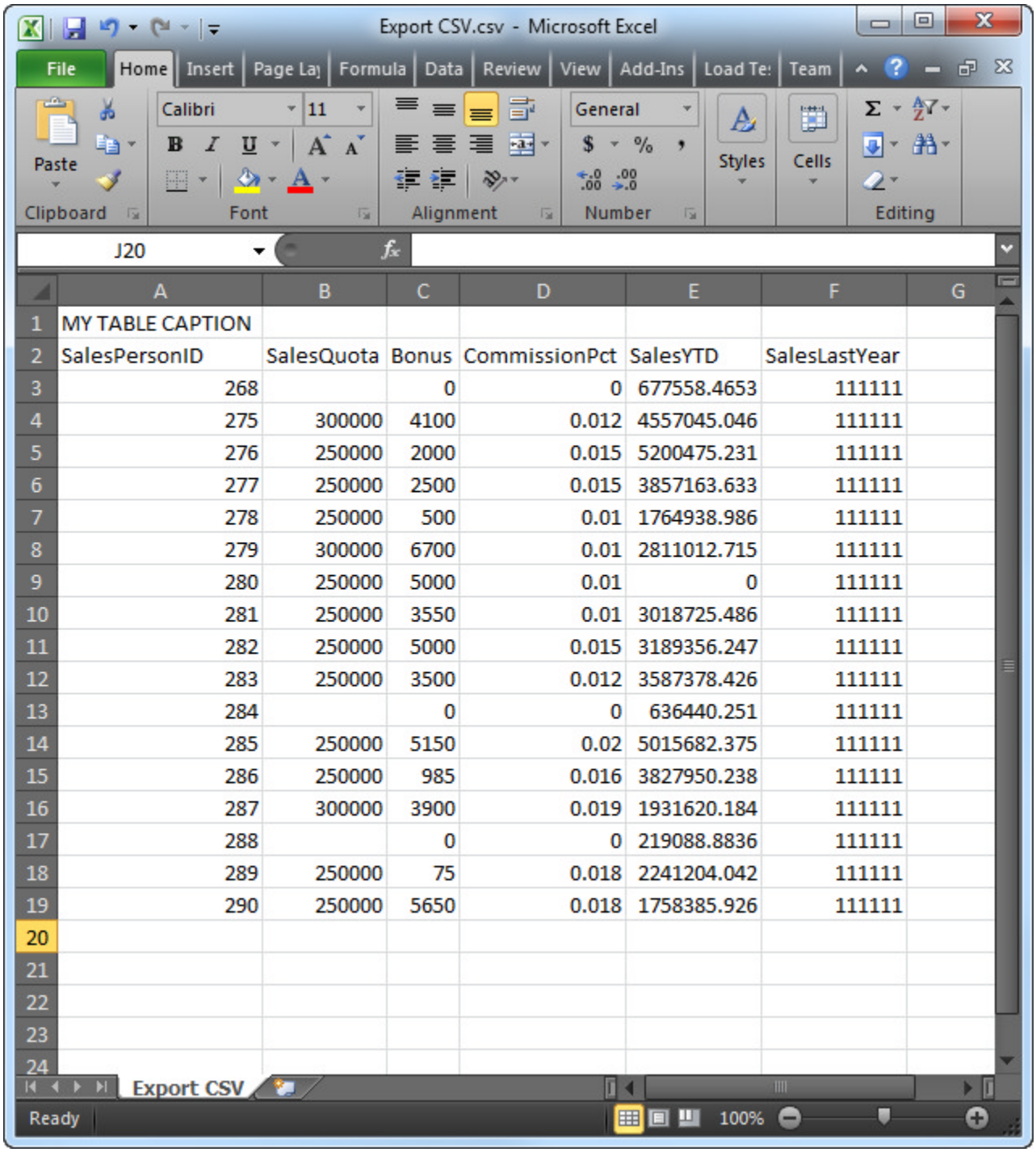
[VB.NET] Export to CSV

```
Private Sub btnExportCSV_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles btnExportCSV.Click
Dim sfd As New SaveFileDialog()
sfd.Filter = [String].Format("{0} (*{1})|*{1}", "CSV Files", ".csv")
If sfd.ShowDialog() = DialogResult.OK Then
```

```
Dim exporter As New ExportToCSV(Me.RadGridView1)
exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport
exporter.SummariesExportOption = SummariesOption.DoNotExport
AddHandler exporter.CSVCellFormatting, AddressOf exporter_CSVCellFormatting
AddHandler exporter.CSVTableCreated, AddressOf exporter_CSVTableCreated
exporter.RunExport(sfd.FileName)
MessageBox.Show("Export completed")
End If
End Sub
Private Sub exporter_CSVTableCreated(ByVal sender As Object, ByVal e As
CSVTableCreatedEventArgs)
DirectCast(sender, ExportToCSV).AddCustomCSVRow(e.CSVTableElement, "MY TABLE CAPTION")
End Sub
Private Sub exporter_CSVCellFormatting(ByVal sender As Object, ByVal e As
CSVCellFormattingEventArgs)
If e.GridColumnIndex = 8 AndAlso e.GridRowInfoType Is GetType(GridViewDataRowInfo) Then
e.CSVCellElement.Value = "111111"
End If
End Sub
```

[C#] Export to CSV

```
private void btnExportCSV_Click(object sender, EventArgs e)
{
SaveFileDialog sfd = new SaveFileDialog();
sfd.Filter = String.Format("{0} (*{1})|*{1}",
"CSV Files",
".csv");
if (sfd.ShowDialog() == DialogResult.OK)
{
ExportToCSV exporter = new ExportToCSV(this.radGridView1);
exporter.HiddenColumnOption = Telerik.WinControls.UI.Export.HiddenOption.DoNotExport;
exporter.SummariesExportOption = SummariesOption.DoNotExport;
exporter.CSVCellFormatting += new
Telerik.WinControls.UI.Export.CSV.CSVCellFormattingEventHandler(exporter_CSVCellFormatting);
exporter.CSVTableCreated += new
Telerik.WinControls.UI.Export.CSV.CSVTableCreatedEventHandler(exporter_CSVTableCreated);
exporter.RunExport(sfd.FileName);
MessageBox.Show("Export completed");
}
}
void exporter_CSVTableCreated(object sender,
Telerik.WinControls.UI.Export.CSV.CSVTableCreatedEventArgs e)
{
((ExportToCSV)sender).AddCustomCSVRow(e.CSVTableElement, "MY TABLE CAPTION");
}
void exporter_CSVCellFormatting(object sender,
Telerik.WinControls.UI.Export.CSV.CSVCellFormattingEventArgs e)
{
if (e.GridColumnIndex == 8 && e.GridRowInfoType == typeof(GridViewDataRowInfo))
{
e.CSVCellElement.Value = "111111";
}
}
```



	A	B	C	D	E	F	G
1	MY TABLE CAPTION						
2	SalesPersonID	SalesQuota	Bonus	CommissionPct	SalesYTD	SalesLastYear	
3	268		0	0	677558.4653	111111	
4	275	300000	4100	0.012	4557045.046	111111	
5	276	250000	2000	0.015	5200475.231	111111	
6	277	250000	2500	0.015	3857163.633	111111	
7	278	250000	500	0.01	1764938.986	111111	
8	279	300000	6700	0.01	2811012.715	111111	
9	280	250000	5000	0.01	0	111111	
10	281	250000	3550	0.01	3018725.486	111111	
11	282	250000	5000	0.015	3189356.247	111111	
12	283	250000	3500	0.012	3587378.426	111111	
13	284		0	0	636440.251	111111	
14	285	250000	5150	0.02	5015682.375	111111	
15	286	250000	985	0.016	3827950.238	111111	
16	287	300000	3900	0.019	1931620.184	111111	
17	288		0	0	219088.8836	111111	
18	289	250000	75	0.018	2241204.042	111111	
19	290	250000	5650	0.018	1758385.926	111111	
20							
21							
22							
23							
24							

21.15 Summary

In this chapter you became familiar with the RadGridView control, using the Smart Tag, Property Builder and Properties window to configure the grid and to bind the grid to data. You learned how to add special purpose column types at design-time and in code, how to group/filter/sort data based on user input at runtime, by hand at design-time and programmatically in code. You displayed hierarchical data from multiple related tables in the grid. You used the RadGridView Virtual Mode feature to take low-level control over the grid's refresh

process. You learned how to bind RadGridView to LINQ data sources. Finally you learned multiple methods of exporting grid data.

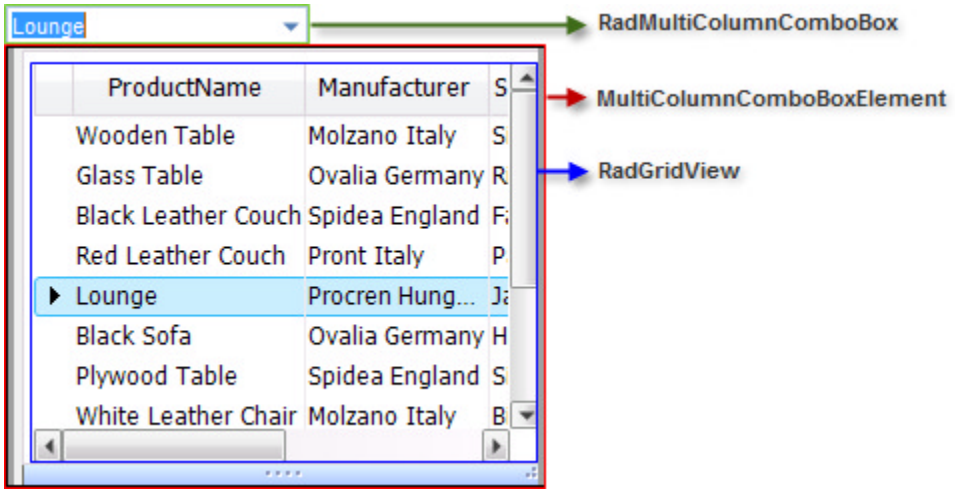
22 Multi-Column ComboBox

22.1 Objectives

- Learn how to access the underlying combo and grid in the RadMultiColumnComboBox.
- Learn how to bind data to the control.
- Filter the list based on user entry.
- React to user selections in the list.

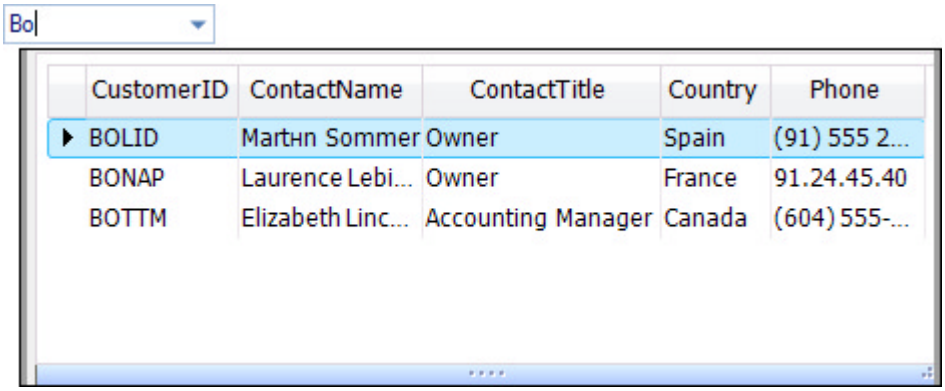
22.2 Introduction

The RadMultiColumnComboBox is a special case of combobox control with a RadGridView integrated in its drop-down. The drop down display has the rich capabilities of the grid view, but you can still use the control in a limited amount of space. The image below shows the structure of the control:



1. The drop-down element is represented by the **MultiColumnComboBoxElement** property. Use this element to set the drop-down style, animation and sizing as you would do for the regular combobox.

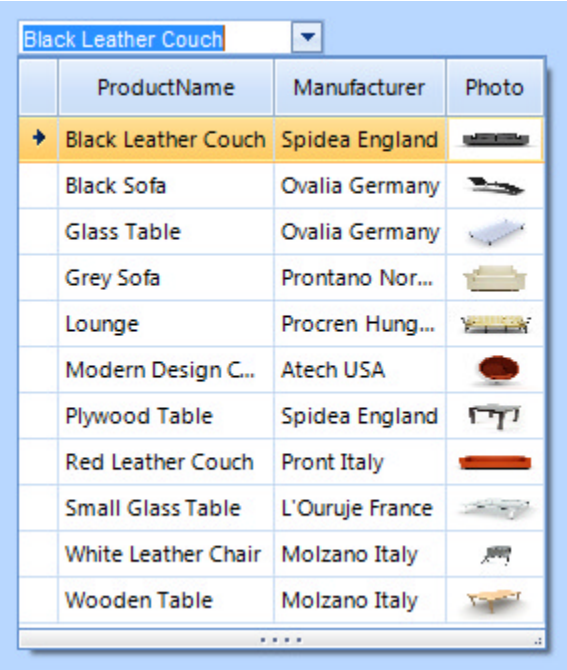
   The control can also automatically filter entries in the grid as characters are typed in.



2. The **EditorControl** property of the MultiColumnComboBoxElement gives you a reference to the RadGridView control. From there you can access grid properties, i.e. MasterGridViewTemplate, columns, cells, etc.

22.3 Getting Started

Working with RadMultiColumnComboBox is similar to working with RadComboBox if you don't need direct access to the underlying grid view. To get started you can simply bind to the DataSource property at design-time and leave it at that.



Preparing the Project

You can find the complete source for this project at:

\MultiColumnComboBox\<VB|CS>\GettingStarted

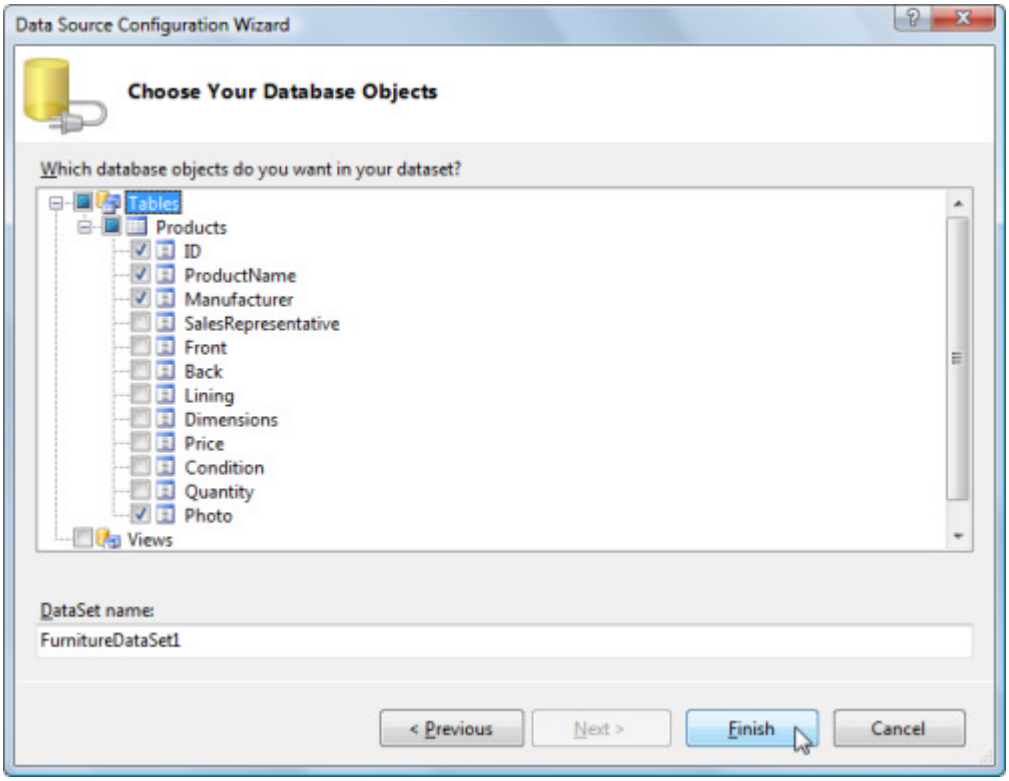
1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. Change the new RadForm1 to be the startup form.

RadMultiColumnComboBox Setup

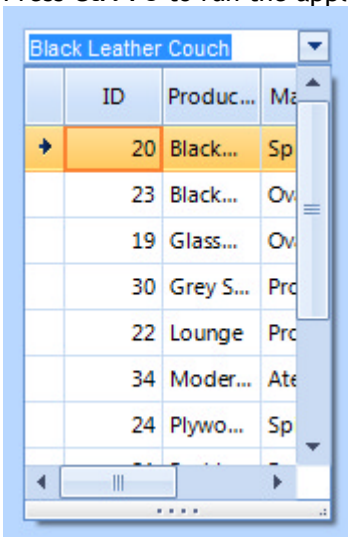
1. Add a **RadMultiColumnComboBox** control to the form.
2. In the "Choose DataSource" drop down choose "Add Project Data Source...". This step will display the Data Source Configuration Wizard dialog.
3. In the "Choose a Data Source Type" page of the wizard, select "Database" and click the **Next** button.
4. In the "Choose Your Data Connection" page of the wizard, click "New Connection...". This will display the Add Connection dialog.
5. In the Add Connection dialog, set the Data Source to "Microsoft Access Database File (OLE DB)", and the Database File Name to "Furniture.mdb". Click **OK** to close the dialog.

Furniture.mdb can be found in the RadControls for Winforms installation directory in the \Examples\QuickStart\Datasources directory.

6. Back at the "Choose Your Data Connection" page of the wizard, click the **Next** button to continue.
7. Click the **Next** button and take defaults until you arrive at the "Choose Your Database Objects" page of the wizard.
8. Select the "ID", "ProductName", "Manufacturer" and "Photo" columns of the "Products" table. Click the **Finish** button.



9. Press **Ctrl-F5** to run the application.



So far the drop down list is a little narrow and the drop down can't be resized. Except for the Photo column, the columns are a little narrow, and we don't really need to see the "ID" column. Let's fix these issues:

10. Set the RadMultiColumnComboBox **DropDownSizingMode** to **UpDownAndRightBottom** and the **Text** property to "".

11. In the Form's Load event handler, add code to size the columns and hide the "ID" column.

There should already be a line of code to fill the Furniture DataSet. Get a reference to the MultiColumnComboBoxElement and use it to access the Columns by name, get the "ID" column and set the IsVisible property of the column to false. Then call the BestFit() method of each column except the "Photo" column. Set the images layout for GridViewImageColumn.

[VB] Sizing Columns

```
Private Sub RadForm1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
Me.productsTableAdapter.Fill(Me.furnitureDataSet.Products)
' get a reference to the combo box element
Dim combo As RadMultiColumnComboBoxElement =
radMultiColumnComboBox1.MultiColumnComboBoxElement
' hide the ID column
combo.Columns("ID").IsVisible = False
' size all the columns except "Photo"
For Each column As GridViewDataColumn In combo.Columns
If (Not column.Name.Equals("Photo")) Then
column.BestFit()
End If
Next column
'set the image layout in the "Photo" column
DirectCast(combo.EditorControl.Columns("Photo"), GridViewImageColumn).ImageLayout =
ImageLayout.Stretch
' set initial drop down width
combo.DropDownWidth = 265
combo.DropDownHeight = 315
End Sub
```

[C#] Sizing Columns

```
private void RadForm1_Load(object sender, EventArgs e)
{
this.productsTableAdapter.Fill(this.furnitureDataSet.Products);
// get a reference to the combo box element
RadMultiColumnComboBoxElement combo = radMultiColumnComboBox1.MultiColumnComboBoxElement;
// hide the ID column
combo.Columns["ID"].IsVisible = false;
// size all the columns except "Photo"
foreach (GridViewDataColumn column in combo.Columns)
{
if (!column.Name.Equals("Photo"))
column.BestFit();
```

```
}
//set the image layout in the "Photo" column
((GridViewImageColumn)combo.EditorControl.Columns["Photo"]).ImageLayout =
ImageLayout.Stretch;
// set initial drop down width and height
combo.DropDownWidth = 265;
combo.DropDownHeight = 315;
}
```

12. Press **Ctrl-F5** to run the application. The drop down should be wider initially and allow vertical and horizontal sizing. The "ID" field should be hidden. The visible text columns should be wide enough to show most of the content.

22.4 Programming the Multi Column Combo Box

Using the Combo and Grid References

The important piece of information for RadMultiColumnComboBox is how to get references to the combo box element and the internal grid control. From there you can leverage what you already know about RadComboBox and RadGridView.

[VB] Getting Combo and Grid References

```
' get a reference to the combo box element
Dim combo As RadMultiColumnComboBoxElement =
radMultiColumnComboBox1.MultiColumnComboBoxElement
' get a reference to the grid view
Dim grid As RadGridView = combo.EditorControl
```

[C#] Getting Combo and Grid References

```
// get a reference to the combo box element
RadMultiColumnComboBoxElement combo = radMultiColumnComboBox1.MultiColumnComboBoxElement;
// get a reference to the grid view
RadGridView grid = combo.EditorControl;
```

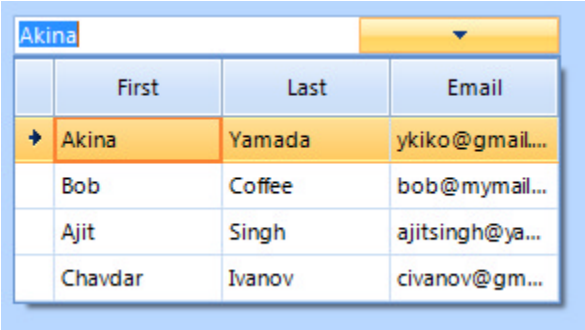


You can find the complete source for this project at:

\MultiColumnComboBox\<VB|CS>\Programming

The MultiColumnComboBoxElement reference gives you access to any properties not surfaced directly by the multi column combo box control.

The example below shows a generic list of custom "Contact" objects that each have a First, Last and Email fields. The list is bound to the RadMultiColumnComboBox. As with a RadComboBox, you set the **DisplayMember** and **ValueMember** properties; these settings won't alter the display, but ValueMember is used for data retrieval via the **SelectedValue** property and DisplayMember is used to support filtering. The MultiColumnComboBoxElement is used to set the drop down width and the **ArrowButtonMinWidth** to a wide 100 pixels.



[VB] Using the RadMultiColumnComboBoxElement Reference

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 Dim contacts As New List(Of Contact)()
 contacts.Add(New Contact("Akina", "Yamada", "ykiko@gmail.com"))
 contacts.Add(New Contact("Bob", "Coffee", "bob@mymail.com"))
 contacts.Add(New Contact("Ajit", "Singh", "ajitsingh@yahoo.com"))
 contacts.Add(New Contact("Chavdar", "Ivanov", "civanov@gmail.com"))
 radMultiColumnComboBox1.DataSource = contacts
 radMultiColumnComboBox1.DisplayMember = "First"
 radMultiColumnComboBox1.ValueMember = "Email"
 Dim combo As RadMultiColumnComboBoxElement =
radMultiColumnComboBox1.MultiColumnComboBoxElement
 combo.DropDownWidth = radMultiColumnComboBox1.Width
 combo.ArrowButtonMinWidth = 100
End Sub
```

[C#] Using the RadMultiColumnComboBoxElement Reference

```
private void RadForm1_Load(object sender, EventArgs e)
{
 List<Contact> contacts = new List<Contact>();
 contacts.Add(new Contact("Akina", "Yamada", "ykiko@gmail.com"));
 contacts.Add(new Contact("Bob", "Coffee", "bob@mymail.com"));
 contacts.Add(new Contact("Ajit", "Singh", "ajitsingh@yahoo.com"));
 contacts.Add(new Contact("Chavdar", "Ivanov", "civanov@gmail.com"));
 radMultiColumnComboBox1.DataSource = contacts;
 radMultiColumnComboBox1.DisplayMember = "First";
 radMultiColumnComboBox1.ValueMember = "Email";
 RadMultiColumnComboBoxElement combo = radMultiColumnComboBox1.MultiColumnComboBoxElement;
 combo.DropDownWidth = radMultiColumnComboBox1.Width;
 combo.ArrowButtonMinWidth = 100;
}
```



Gotcha! When binding to a list of custom objects, be sure to scope the properties of your object "public". Leaving them at their "private" defaults will not generate errors, but you still won't see data.

You can programmatically drop down the list using the MultiColumnComboBoxElement **ShowPopup()** method.

[VB] The ShowPopup Method

```
Private Sub btnShowPopup_Click(sender As Object, e As EventArgs)
 Dim combo As RadMultiColumnComboBoxElement =
radMultiColumnComboBox1.MultiColumnComboBoxElement
 combo.ShowPopup()
```

```
End Sub
```

[C#] The ShowPopup Method

```
private void btnShowPopup_Click(object sender, EventArgs e)
{
 RadMultiColumnComboBoxElement combo = radMultiColumnComboBox1.MultiColumnComboBoxElement;
 combo.ShowPopup();
}
```

The EditorControl property lets you access all the properties of RadGridView:

[VB] Accessing RadGridView Properties

```
combo.EditorControl.MasterTemplate.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill
```

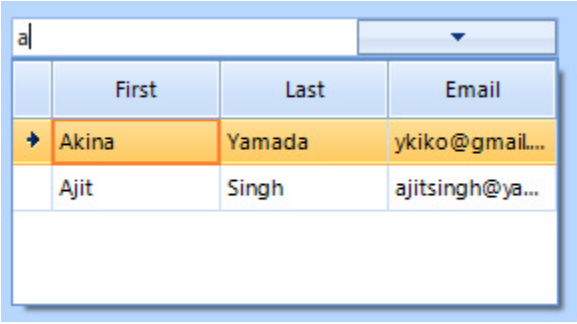
[C#] Accessing RadGridView Properties

```
combo.EditorControl.MasterTemplate.AutoSizeColumnsMode =
 GridViewAutoSizeColumnsMode.Fill;
```

Filtering

To filter the grid as the user types in the edit box of the combo:

- Set the **AutoFilter** property to true.
- Set the **PropertyName** to the column that you want to filter against (usually DisplayMember).
- Set the desired **Operator** to one of the FilterOperator enumerated values.
- Build a **FilterDescriptor** that describes the rules used to filter the data and add the expression to the grid's MasterTemplate FilterDescriptor collection.



[VB] Filtering the List

```
radMultiColumnComboBox1.AutoFilter = true;
FilterDescriptor filter = new FilterDescriptor();
filter.PropertyName = radMultiColumnComboBox1.DisplayMember;
filter.Operator = FilterOperator.StartsWith;
combo.EditorControl.MasterTemplate.FilterDescriptors.Add(filter);
```

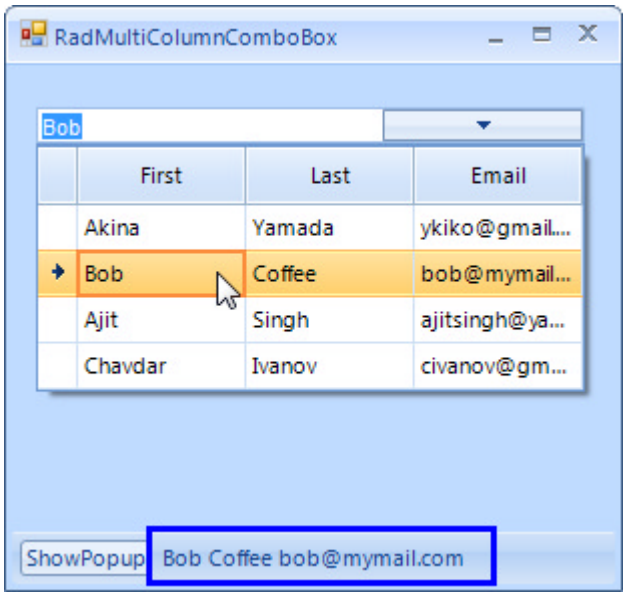
[C#] Filtering the List

```
radMultiColumnComboBox1.AutoFilter = True
Dim filter As FilterDescriptor = New FilterDescriptor()
filter.PropertyName = radMultiColumnComboBox1.DisplayMember
filter.Operator = FilterOperator.StartsWith
combo.EditorControl.MasterTemplate.FilterDescriptors.Add(filter)
```

This example used a generic list of Contact objects with fields "First", "Last" and "Email". DisplayMember in this case was "First".

Events

Respond to the SelectedIndexChanged event to react to user selections in the list.



"Sender" in this event handler is the RadMultiColumnComboBox itself. Retrieve the **SelectedItem** and cast it to a **GridViewDataRowInfo**. GridViewDataRowInfo has a **DataBoundItem** property that is the actual object that is being bound to a given row.

[VB] Handling the SelectedIndexChanged Event

```
Private Sub radMultiColumnComboBox1_SelectedIndexChanged(sender As Object, e As EventArgs)
 Dim rowInfo As GridViewDataRowInfo = TryCast((TryCast(sender,
RadMultiColumnComboBox)).SelectedItem, GridViewDataRowInfo)
 Dim contact As Contact = TryCast(rowInfo.DataBoundItem, Contact)
 If contact <> Nothing Then
  lblSelected.Text = contact.First + " " + contact.Last + " " + contact.Email
 End If
End Sub
```

[C#] Handling the SelectedIndexChanged Event

```
private void radMultiColumnComboBox1_SelectedIndexChanged(object sender, EventArgs e)
{
 GridViewDataRowInfo rowInfo = (sender as RadMultiColumnComboBox).SelectedItem as
GridViewDataRowInfo;
 Contact contact = rowInfo.DataBoundItem as Contact;
 if (contact != null)
 {
   lblSelected.Text = contact.First + " " + contact.Last + " " + contact.Email;
 }
}
```

22.5 Summary

In this chapter you learned how to access the RadMultiColumnComboBox underlying combo and grid controls. You learned how to bind data to the control, how to filter the data in the drop down based on user entry and how to react to user selections by handling the SelectedIndexChanged event.

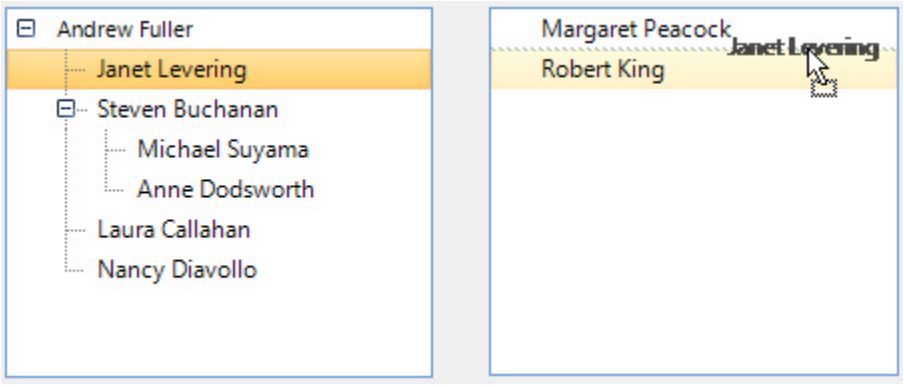
23 Tree

23.1 Objectives

- Display, navigate and manage hierarchical data using the RadTreeView control.
- Learn to add, move, remove, iterate and locate nodes programmatically.
- Learn how to edit nodes.
- Use the RadBreadCrumb control to help the user keep track of their location in deeply nested hierarchical structures.
- Implement drag-and-drop within the treeview, between treeviews and to other controls.
- Use context menus.
- Learn about RadTreeView specific databinding issues including binding to self referencing data, binding to related data, writing to and from XML and load-on-demand.

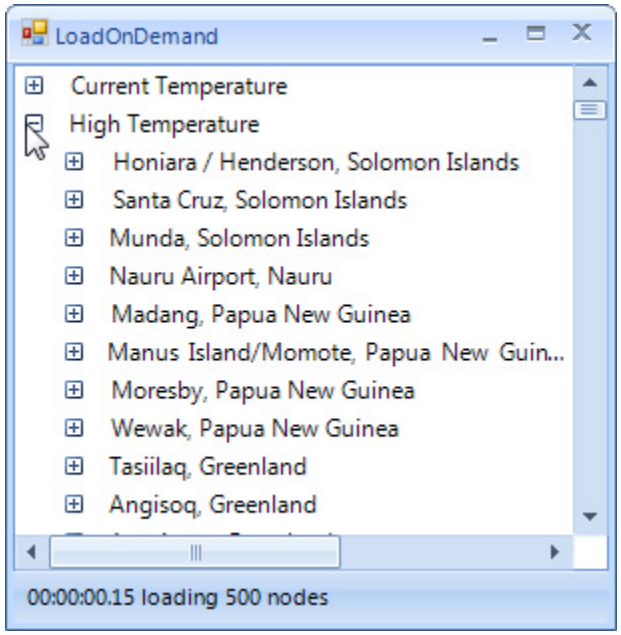
23.2 Introduction

RadTreeView is the supercharged tree view component for Windows Forms that displays, manages, and navigates hierarchical data structures. The product offers advanced features like drag-and-drop, load-on-demand, filtering, context menus and data binding. Here are just a few features:

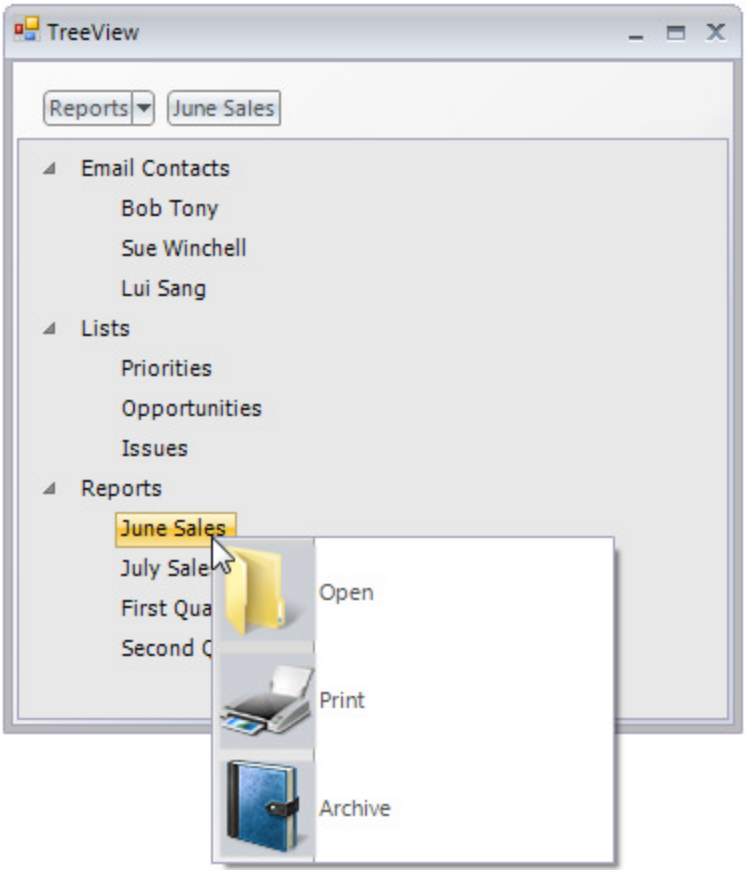


RadTreeView supports drag and drop within the same tree and between RadTreeView controls. The position indication cursor provides feedback to the user so that nodes can be dropped above, below or as a child of a given node. When dragging to a hidden or collapsed node the control will scroll up/down and automatically open the collapsed node. Drag and drop behavior is customizable at the treeview and node levels, allowing you to restrict interactions between source and target nodes.

RadTreeView binds to hierarchical data for most popular databases, custom business objects and XML. RadTreeView binds to any object that supports IList, IListSource or IBindingList. RadTreeView can also load and save XML directly to file

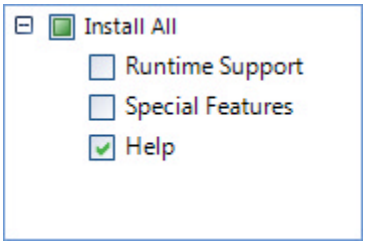


or string. For industrial strength applications with heavy data requirements, the Load On Demand feature reduces loading time and performance overhead.



Expand RadTreeView functionality by adding context menus either for the entire treeview or to each node. You have the capability to attach a different context menu to each tree node. Each context menu can be individually styled.

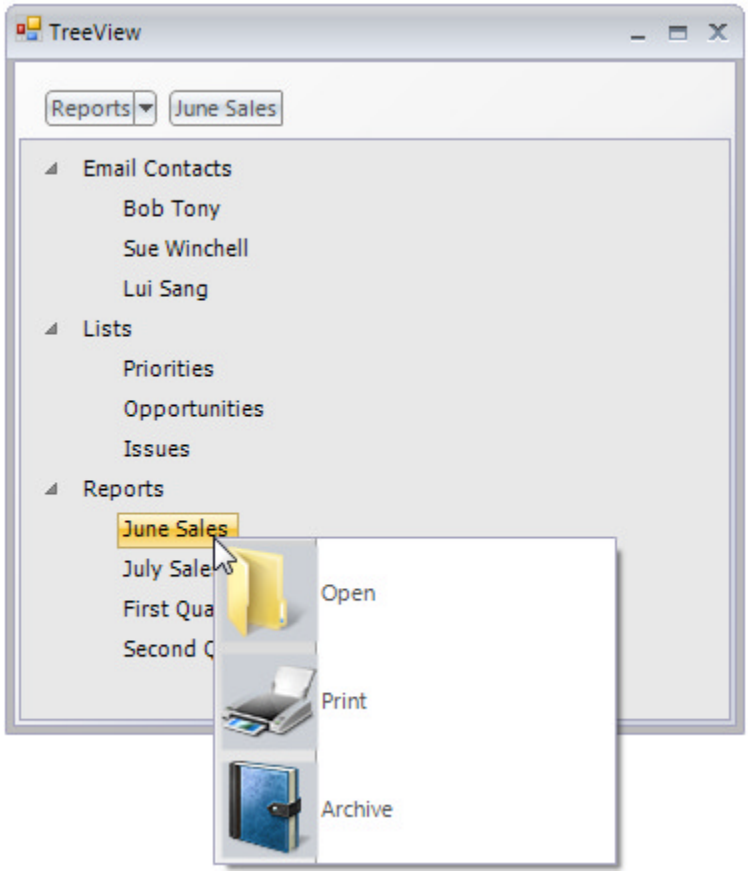
You can mix check boxes and radio buttons to form "option trees". Both can be



configured to display a third "indeterminate" state that provides enhanced feedback to the user.

23.3 Getting Started

In this tutorial you will add nodes at design time and programmatically, hook up a RadBreadCrumb to help navigate the tree and add context menus to all the "Reports" nodes.



Preparing the Project



You can find the complete source for this project at:

\TreeVIew\<VB|CS>\GettingStarted

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.

5. Change the new RadForm1 to be the startup form.
6. Add a **RadContextMenu** to the form. Using the Smart Tag "Edit Items" link, open the RadItem Collection Editor and add three RadMenuItems with Text "Open", "Print" and "Archive". Optionally you can add images for each item.
7. Drop a **RadPanel** on the form and set the **Dock** property to "Fill", set the **Margin** to (0, 0, 0, 0) and **Padding** to (3,3,3,3). Feel free to open the Edit UI Elements link from the panel Smart Tag and change the back color and gradient properties.
8. Add the **Office2007SilverTheme** from the ToolBox to the form.
9. Either set to each control individually its ThemeName property to "Office2007Silver" or add the following line of code in the form constructor

C# Setting the theme for the entire application

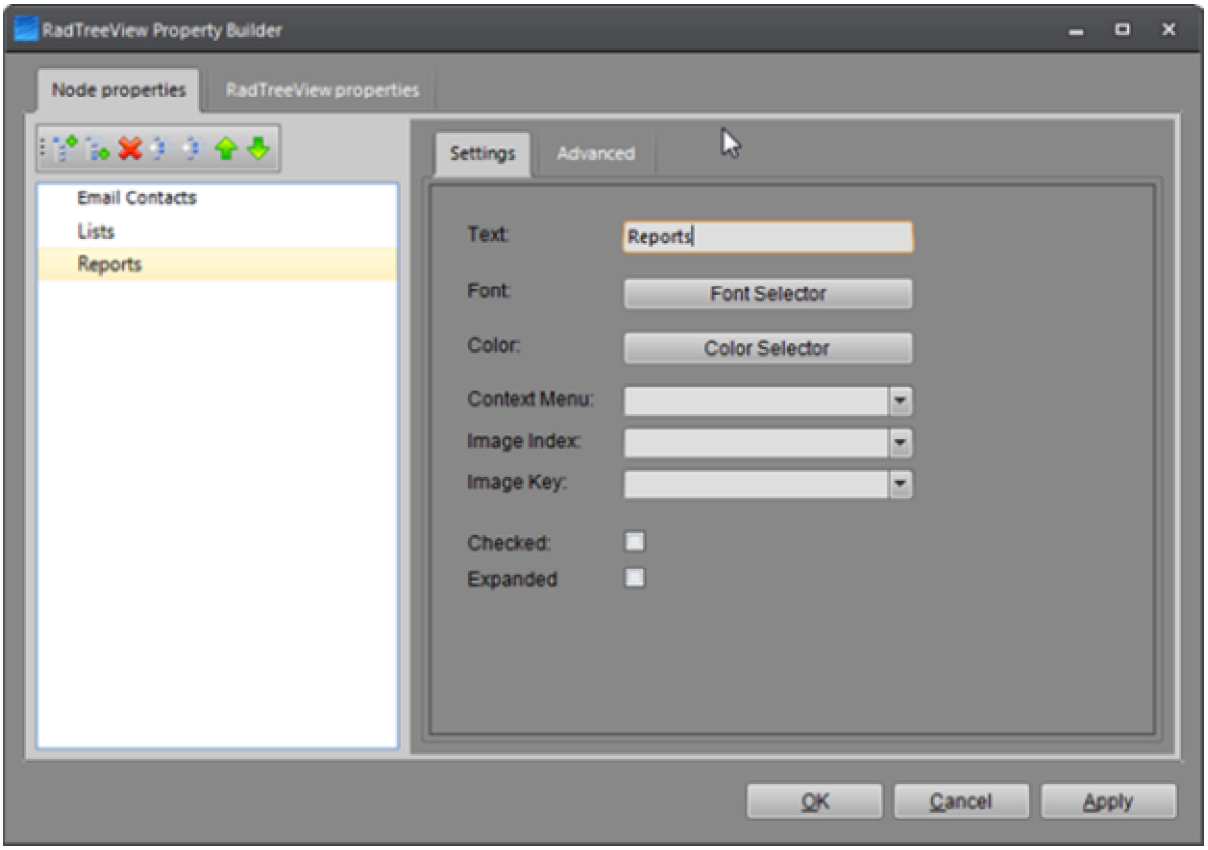
```
ThemeResolutionService.ApplicationThemeName = "Office2007Silver";
```

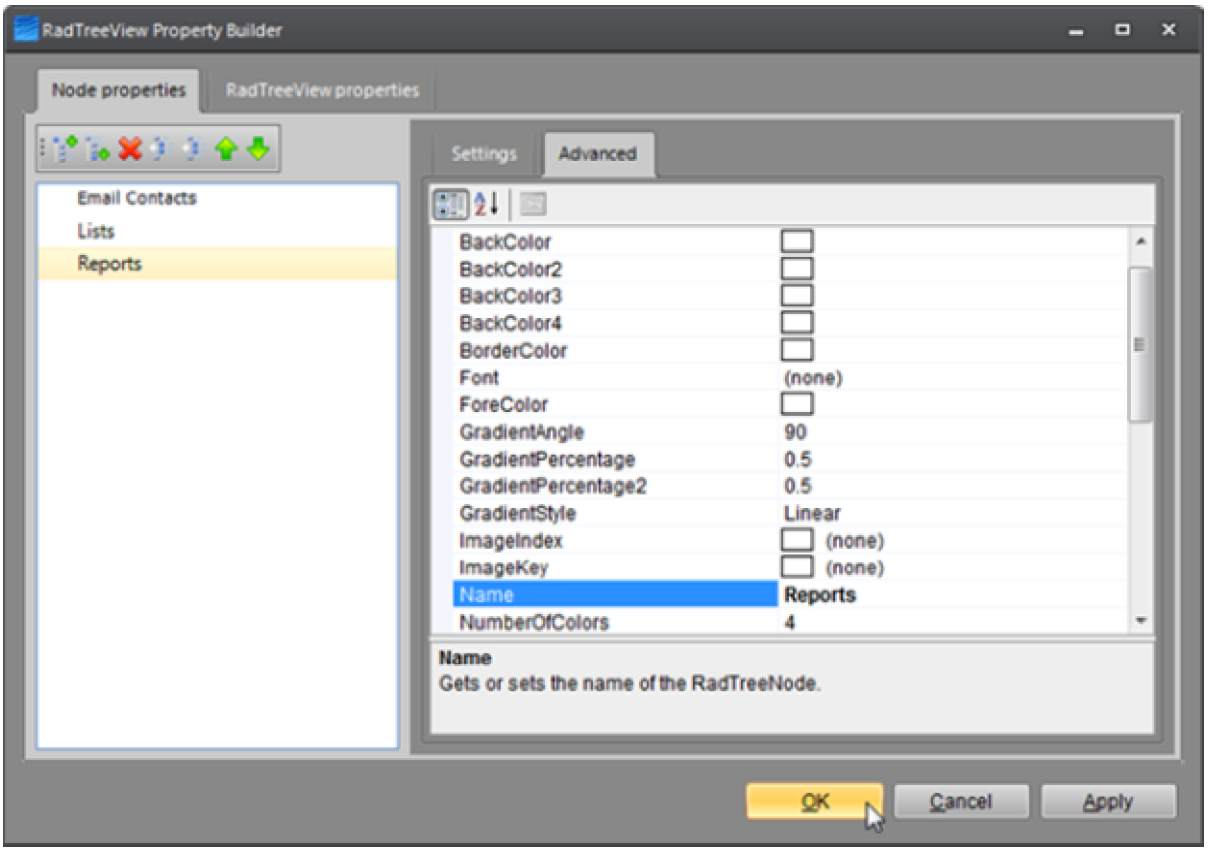
VB.NET Setting the theme for the entire application

```
ThemeResolutionService.ApplicationThemeName = "Office2007Silver"
```

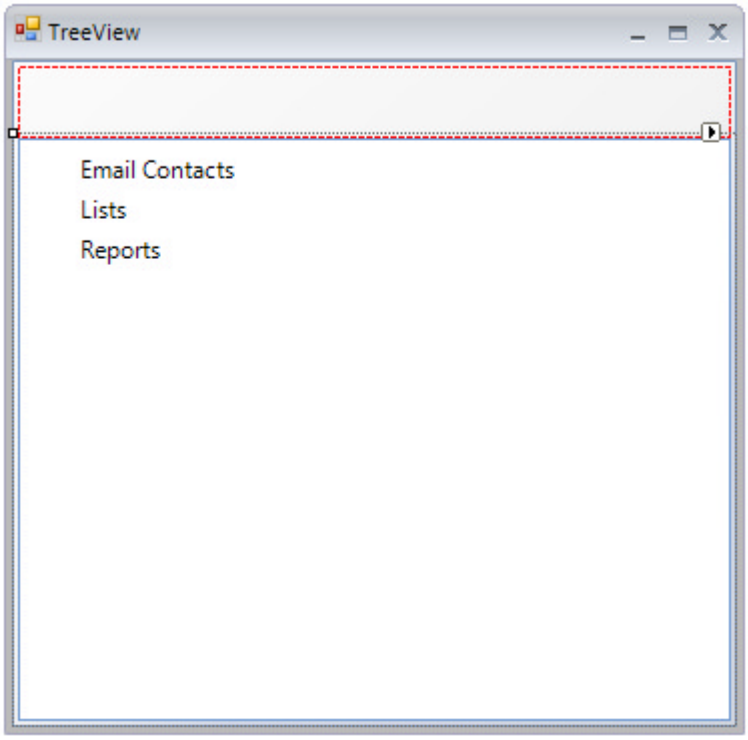
RadTreeView and Breadcrumb Setup

1. Add a **RadBreadCrumb** to the panel and set the **Dock** property to "Top".
2. Add a **RadTreeView** to the form below the RadBreadCrumb and set the **Dock** property to "Fill".
3. Set the RadBreadCrumb **DefaultTreeView** property to point at the RadTreeView.
4. Configure the root level nodes.
    - Right click the RadTreeView and select Open Property Builder from the context menu. You will see a set of buttons on the left side of the RadTreeView Property Builder and a preview of the treeview below the buttons.
    - Click the far left button (a green "Plus" button that adds nodes) three times to add three root level nodes.
    - Click the first node and enter "Email Contacts" in the Text entry. In the Advanced tab set its Name property to the same value.
    - Click the second node and enter "Lists" in the Text entry. In the Advanced tab set its Name property to the same value.
    - Click the third node and enter "Reports" in the Text entry. In the Advanced tab set its Name property to the same value.
    - Click **OK** to close the Property Builder.


RadTreeView Property Builder
Node properties
RadTreeView properties
Email Contacts
Lists
Reports
Settings
Advanced
Text:
Reports
Font:
Font Selector
Color:
Color Selector
Context Menu:
Image Index:
Image Key:
Checked:
Expanded
OK
Cancel
Apply


RadTreeView Property Builder
Node properties
RadTreeView properties
Email Contacts
Lists
Reports
Settings
Advanced
BackColor
BackColor2
BackColor3
BackColor4
BorderColor
Font
(none)
ForeColor
GradientAngle
90
GradientPercentage
0.5
GradientPercentage2
0.5
GradientStyle
Linear
ImageIndex
(none)
ImageKey
(none)
Name
Reports
NumberOfColors
4
Name
Gets or sets the name of the RadTreeNode.
OK
Cancel
Apply

The form layout should look something like the example below.



5. Add code to the form load event handler. The code here will add new nodes to the existing root nodes. Also notice that the "Reports" nodes **ContextMenu** properties are assigned the RadContextMenu **DropDown** property.

[VB] Handling the Load Event Handler

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
ThemeResolutionService.ApplicationThemeName = "Office2007Silver"
Dim node As RadTreeNode = radTreeView1.Nodes("Email Contacts").Nodes.Add("Bob Tony")
node.Selected = True
radTreeView1.Nodes("Email Contacts").Nodes.Add("Sue Winchell")
radTreeView1.Nodes("Email Contacts").Nodes.Add("Lui Sang")
radTreeView1.Nodes("Lists").Nodes.Add("Priorities")
radTreeView1.Nodes("Lists").Nodes.Add("Opportunities")
radTreeView1.Nodes("Lists").Nodes.Add("Issues")
node = radTreeView1.Nodes("Reports").Nodes.Add("June Sales")
node.ContextMenu = radContextMenu1
node = radTreeView1.Nodes("Reports").Nodes.Add("July Sales")
node.ContextMenu = radContextMenu1
node = radTreeView1.Nodes("Reports").Nodes.Add("First Quarter Summary")
node.ContextMenu = radContextMenu1
node = radTreeView1.Nodes("Reports").Nodes.Add("Second Quarter Summary")
node.ContextMenu = radContextMenu1
End Sub
```

[C#] Handling the Load Event Handler

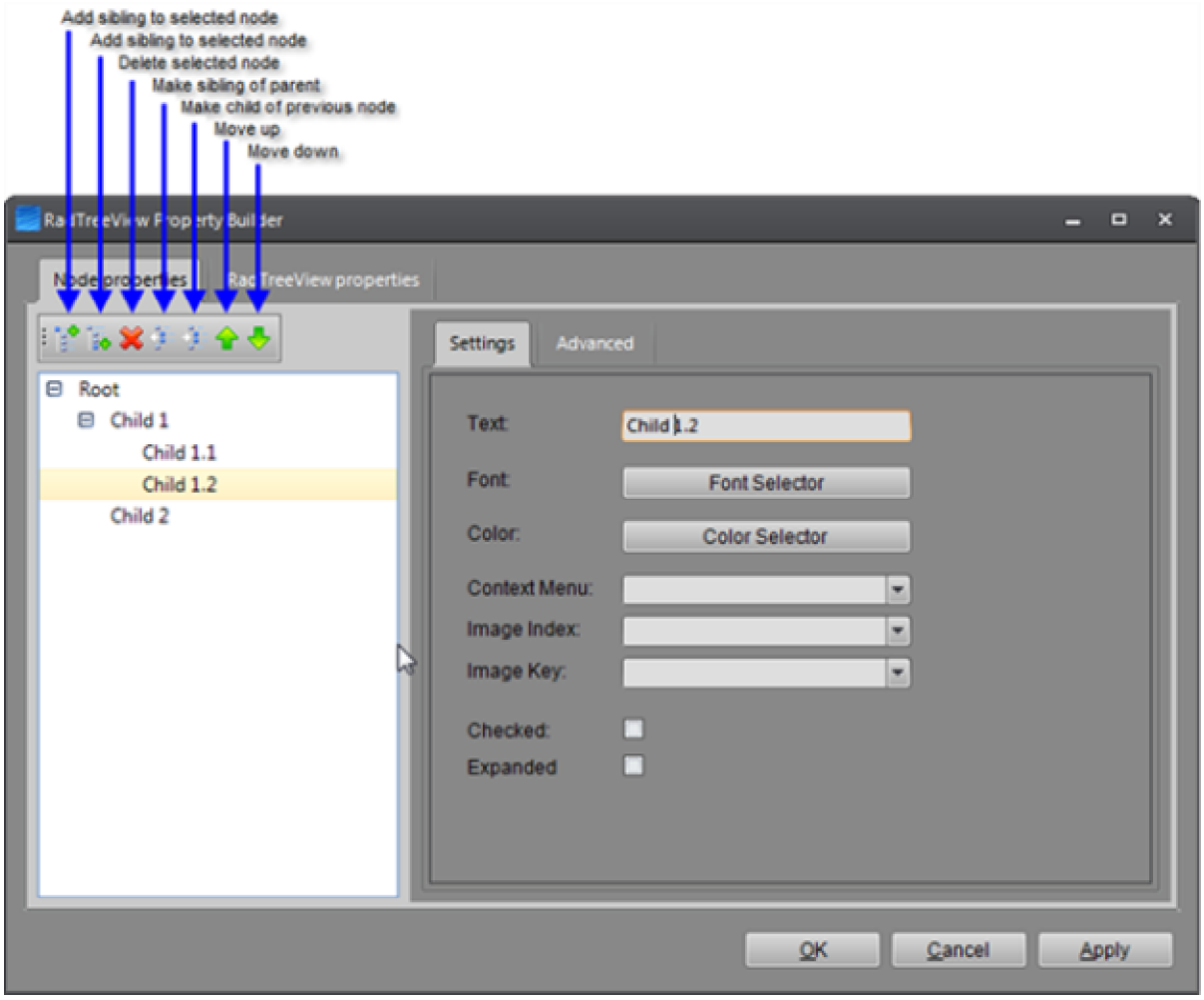
```
private void RadForm1_Load(object sender, EventArgs e)
{
ThemeResolutionService.ApplicationThemeName = "Office2007Silver";
```

```
RadTreeNode node = radTreeView1.Nodes["Email Contacts"].Nodes.Add("Bob Tony");
node.Selected = true;
radTreeView1.Nodes["Email Contacts"].Nodes.Add("Sue Winchell");
radTreeView1.Nodes["Email Contacts"].Nodes.Add("Lui Sang");
radTreeView1.Nodes["Lists"].Nodes.Add("Priorities");
radTreeView1.Nodes["Lists"].Nodes.Add("Opportunities");
radTreeView1.Nodes["Lists"].Nodes.Add("Issues");
node = radTreeView1.Nodes["Reports"].Nodes.Add("June Sales");
node.ContextMenu = radContextMenu1;
node = radTreeView1.Nodes["Reports"].Nodes.Add("July Sales");
node.ContextMenu = radContextMenu1;
node = radTreeView1.Nodes["Reports"].Nodes.Add("First Quarter Summary");
node.ContextMenu = radContextMenu1;
node = radTreeView1.Nodes["Reports"].Nodes.Add("Second Quarter Summary");
node.ContextMenu = radContextMenu1;
}
```

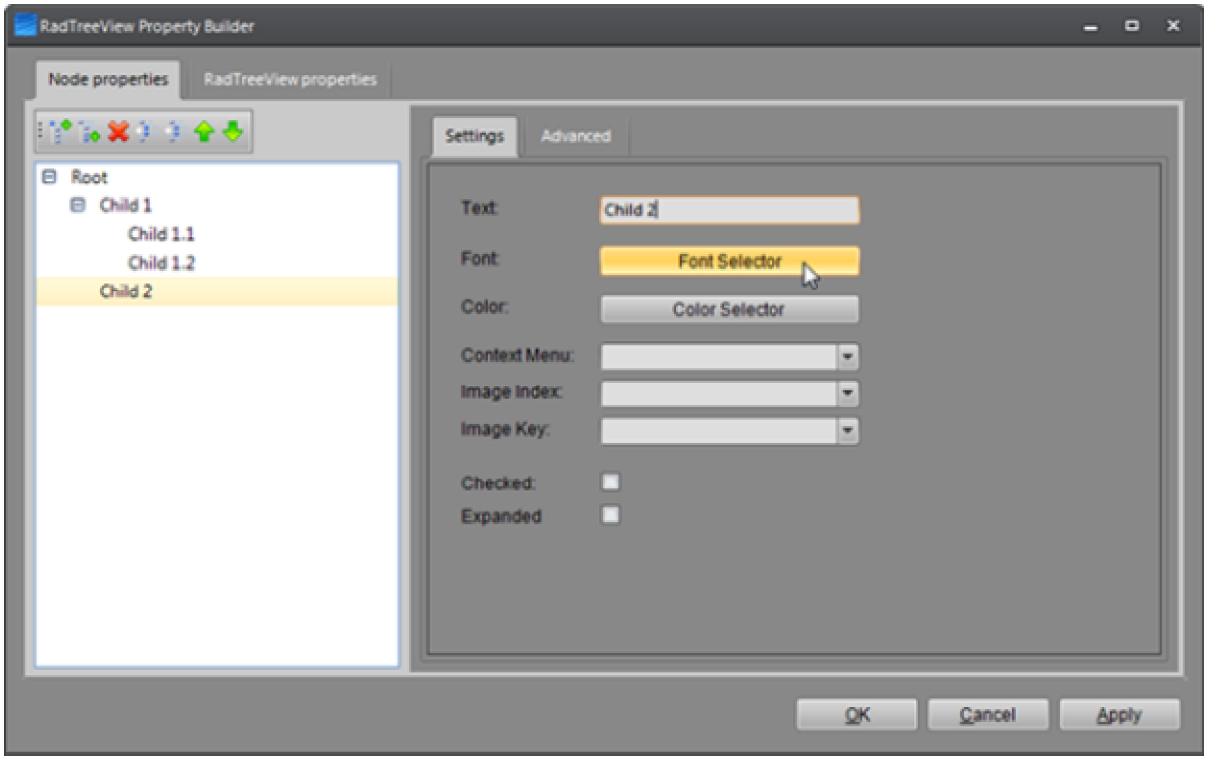
23.4 Using the Design Time Interface

The Property Builder is the main design tool for building tree views interactively. Invoke the dialog from the right-click context menu or from the Smart Tag.

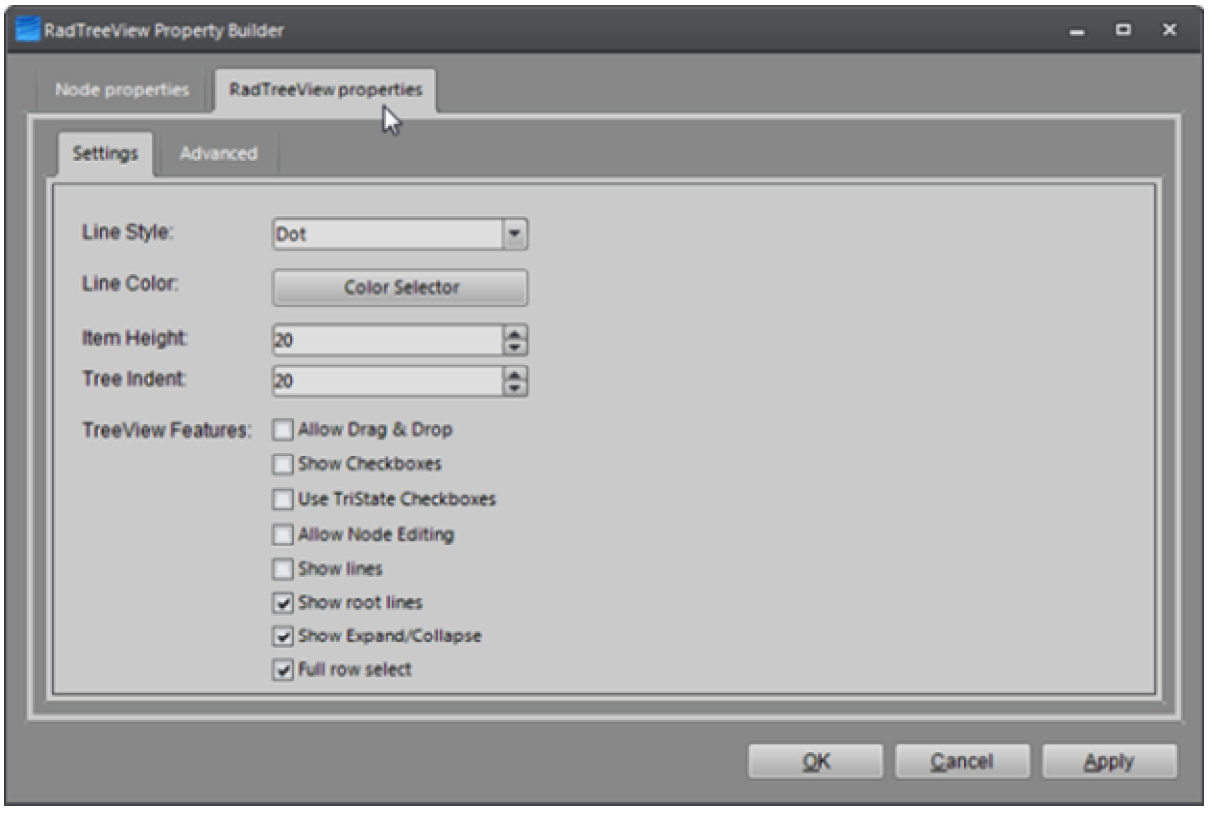
Use the left panel to build the tree structure. The set of buttons at the top handle adding, deleting and moving nodes.



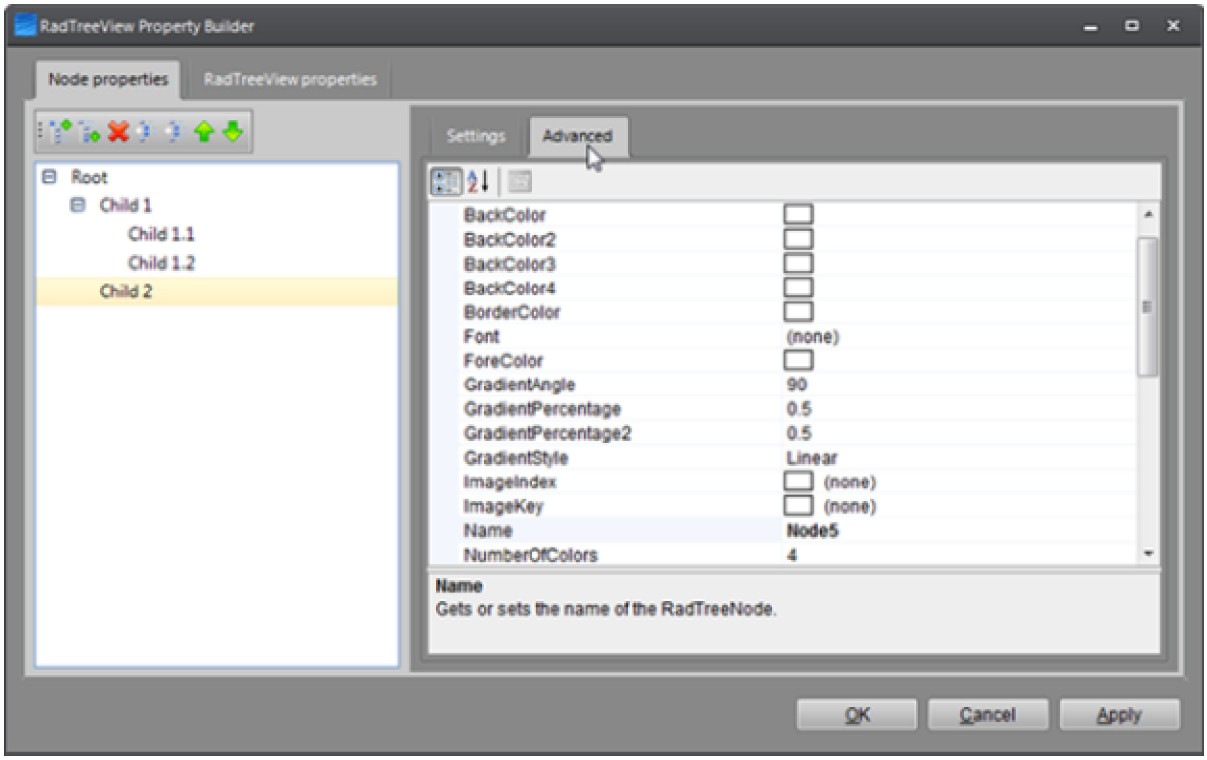
Below the strip buttons pane is a representation of the nodes as they will appear on the form. Click individual nodes to set node-level properties or use button to delete or move them.



Above the strip of buttons you'll see the "RadTreeView properties" tab. Select this tab to view tree view level properties in the right hand pane of the dialog.



Use the button found on the far right side of the dialog to toggle between "Settings" and "Advanced". Use Advanced to get a full list of properties for the RadTreeView or selected RadTreeNode.



23.5 Programming the Tree View

Working with Nodes

Add Nodes

Instead of an Items collection, RadTreeView has a **Nodes** collection. There are several options to adding new nodes. You can create a new **RadTreeNode** using one of several constructors and add it to the collection:

[VB] Adding a Root Node

```
Dim rootNode As New RadTreeNode("Root")
radTreeView1.Nodes.Add(rootNode)
```

[C#] Adding a Root Node

```
RadTreeNode rootNode = new RadTreeNode("Root");
radTreeView1.Nodes.Add(rootNode);
```

Each RadTreeNode in the collection has its own Nodes collection. You can use the Nodes.Add() method and pass the method a Text property string.

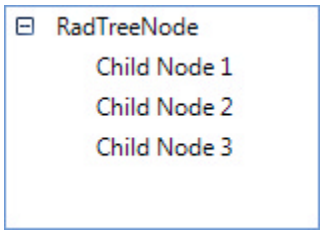
[VB] Adding Child Nodes

```
Dim rootNode As New RadTreeNode("Root")
radTreeView1.Nodes.Add(rootNode)
rootNode.Nodes.Add("Child Node 1")
rootNode.Nodes.Add("Child Node 2")
rootNode.Nodes.Add("Child Node 3")
```

[C#] Adding Child Nodes

```
RadTreeNode rootNode = new RadTreeNode("Root");
radTreeView1.Nodes.Add(rootNode);
rootNode.Nodes.Add("Child Node 1");
```

```
rootNode.Nodes.Add("Child Node 2");
rootNode.Nodes.Add("Child Node 3");
```



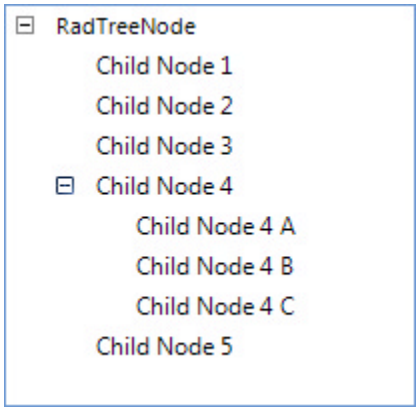
Here's a slightly more complicated example with three levels of nodes. Notice that the Nodes.Add() method returns the RadTreeNode that was added to the collection.

[VB] Adding Multiple Levels of Nodes

```
Dim rootNode As New RadTreeNode("Root")
radTreeView1.Nodes.Add(rootNode)
rootNode.Nodes.Add("Child Node 1")
rootNode.Nodes.Add("Child Node 2")
rootNode.Nodes.Add("Child Node 3")
Dim childNode4 As RadTreeNode = rootNode.Nodes.Add("Child Node 4")
rootNode.Nodes.Add("Child Node 5")
childNode4.Nodes.Add("Child Node 4 A")
childNode4.Nodes.Add("Child Node 4 B")
childNode4.Nodes.Add("Child Node 4 C")
radTreeView1.ExpandAll()
```

[C#] Adding Multiple Levels of Nodes

```
RadTreeNode rootNode = new RadTreeNode("Root");
radTreeView1.Nodes.Add(rootNode);
rootNode.Nodes.Add("Child Node 1");
rootNode.Nodes.Add("Child Node 2");
rootNode.Nodes.Add("Child Node 3");
RadTreeNode childNode4 = rootNode.Nodes.Add("Child Node 4");
rootNode.Nodes.Add("Child Node 5");
childNode4.Nodes.Add("Child Node 4 A");
childNode4.Nodes.Add("Child Node 4 B");
childNode4.Nodes.Add("Child Node 4 C");
radTreeView1.ExpandAll();
```



If you have a string that describes a hierarchy (a file path, or an organization chart for example) and is delimited with a specific character, the RadTreeView **AddNodeByPath()** method will automatically place nodes to the correct level of the hierarchy. Before calling the method, set the **PathSeparator** property to the

delimeter character.

[VB] Adding Nodes by Path

```
Dim places As String() = {"universe", "universe/planet", "universe/planet/continent",
"universe/planet/continent/region"}
Me.radTreeView1.PathSeparator = "/"
For Each place As String In places
 radTreeView1.AddNodeByPath(place)
Next
```

[C#] Adding Nodes by Path

```
string[] places = {
                    "universe",
                    "universe/planet",
                    "universe/planet/continent",
                    "universe/planet/continent/region"};
this.radTreeView1.PathSeparator = "/";
foreach (string place in places)
{
 radTreeView1.AddNodeByPath(place);
}
```



Selecting Nodes

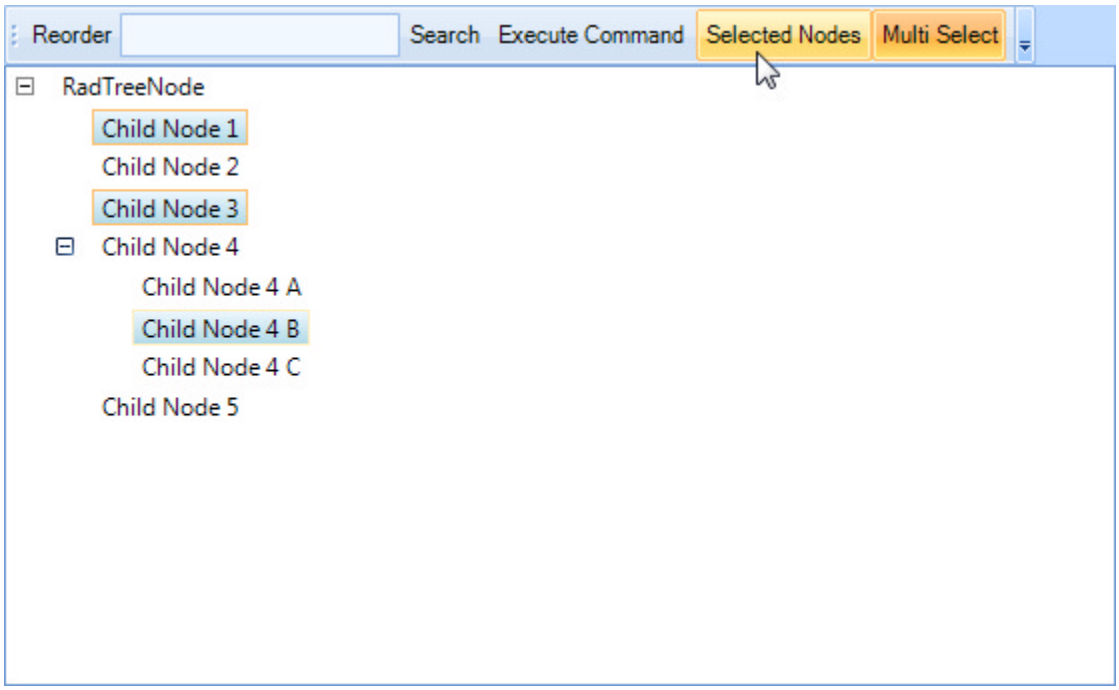
By default the **Multiselect** property is False and you can use the **SelectedNode** property to retrieve the single node. If you set Multiselect to True, use the **SelectedNodes** collection. This example iterates the selected nodes and changes the background colors for each node.

[VB] Iterating Selected Nodes

```
Private Sub btnSelectedNodes_Click(sender As Object, e As EventArgs)
 For Each node As RadTreeNode In radTreeView1.SelectedNodes
  node.BackColor = Color.AliceBlue
  node.BackColor2 = Color.LightBlue
  Next
End Sub
```

[C#] Iterating Selected nodes

```
private void btnSelectedNodes_Click(object sender, EventArgs e)
{
 foreach (RadTreeNode node in radTreeView1.SelectedNodes)
 {
   node.BackColor = Color.AliceBlue;
   node.BackColor2 = Color.LightBlue;
 }
}
```



Move Nodes

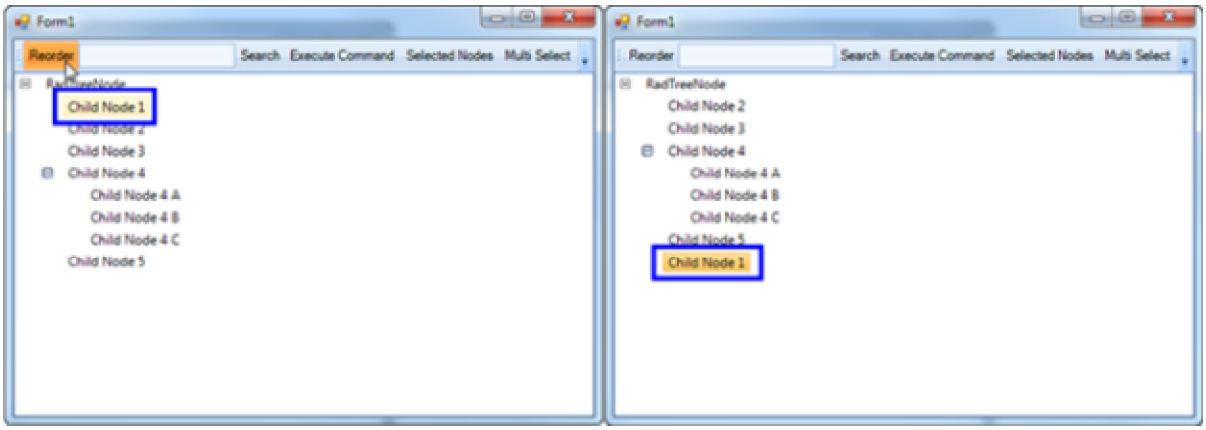
Some operations require nodes to be reordered within the collection using the Nodes collection Move() method. The method takes the index of the node to be moved and the index of its new location. Here's an example that moves the selected node to last place among its siblings.

[VB] Relocating Nodes

```
Private Sub btnReorder_Click(sender As Object, e As EventArgs)
 ' move selected node to the end of the list
 ' Note: Parent and Index properties are deprecated only for "getters"
 If (radTreeView1.SelectedNode <> Nothing) AndAlso (radTreeView1.SelectedNode.Parent <>
Nothing) Then
  Dim nodes As RadTreeNodeCollection = radTreeView1.SelectedNode.Parent.Nodes
  nodes.Move(radTreeView1.SelectedNode.Index, nodes.Count - 1)
 End If
End Sub
```

[C#] Relocating Nodes

```
private void btnReorder_Click(object sender, EventArgs e)
{
 // move selected node to the end of the list
 // Note: Parent and Index properties are deprecated only for "getters"
 if ((radTreeView1.SelectedNode != null) &&
     (radTreeView1.SelectedNode.Parent != null))
 {
   RadTreeNodeCollection nodes = radTreeView1.SelectedNode.Parent.Nodes;
   nodes.Move(radTreeView1.SelectedNode.Index, nodes.Count - 1);
 }
}
```





You can find the complete source for this project at:

\Treeview\<VB|CS>\WorkingWithNodes

Locating Nodes

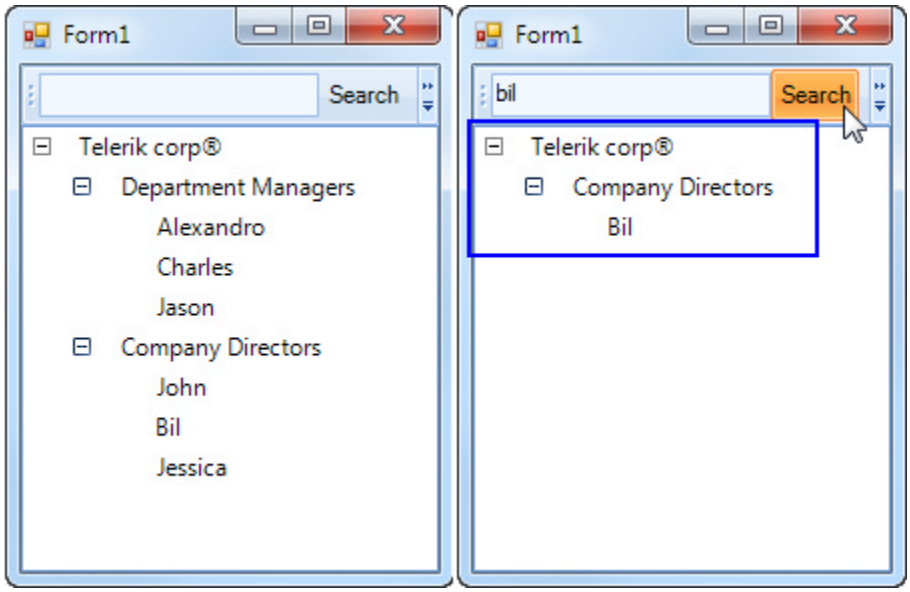
RadTreeView has a filtering functionality, which will return all nodes that contain the specified text and all the parent nodes results.

Example Title

```
Private Sub CommandBarButton1_Click(sender As System.Object, e As System.EventArgs) Handles
CommandBarButton1.Click
radTreeView1.Filter = commandBarTextBox1.Text
Dim filteredResults As RadTreeNode() = radTreeView1.Nodes.ToArray()

End Sub
```

The filteredResults array contains one node ("Telerik corp®") which has one node in its node collection ("Company Directors") which has the searched node in its nodes collection ("Bil").



The **FindNodes()** method compares the value of the **Text** property of nodes with the given search string (like the standard WinForms TreeView control). The function iteration through all the nodes of the nodes collection recursively.

[VB] Finding a Node by Name

```
Dim foundNodes As RadTreeNode() = radTreeView1.FindNodes("NodeText")
RadMessageBox.Show(foundNodes(0).Text)
```

[C#] Finding a Node by Name

```
RadTreeNode[] foundNodes = radTreeView1.Nodes.FindNodes("NodeText");
RadMessageBox.Show(foundNodes[0].Text);
```

You can also use the Nodes collection using node Name as the index to access a node.

[VB] Getting the Node by Text Value

```
radTreeView1.Nodes("Email Contacts")
```

[C#] Getting the Node by Text Value

```
radTreeView1.Nodes["Email Contacts"];
```

You can set the **Selected** property for each node. If the collection of Nodes scrolls completely off the viewable area, you can call the RadTreeView **BringIntoView()** method to scroll a particular node into view.

[VB] Selecting a node and bringing it into view

```
radTreeView1.Nodes("MyNodeName").Selected = True
radTreeView1.BringIntoView(radTreeView1.Nodes("MyNodeName"))
```

[C#] Selecting a node and bringing it into view

```
radTreeView1.Nodes["MyNodeName"].Selected = true;
radTreeView1.BringIntoView(radTreeView1.Nodes["MyNodeName"]);
```

Node Images

RadTreeView can react visually as nodes are selected, opened and closed using image properties. Each node has image properties with three ways to access:

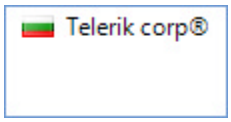
- **Image**: An Image class that can be assigned directly without using an ImageList.
- **ImageIndex**: The position of an image within an ImageList.
- **ImageKey**: The identifier for an image within an ImageList.

[VB] Setting Image and "Far" Image Index

```
Dim rootNode As New RadTreeNode("Telerik corp®")
rootNode.Image = Image.FromFile("C:\image.png")
rootNode.ImageKey = "imageName.png"
rootNode.ImageIndex = 0
radTreeView1.Nodes.Add(rootNode)
```

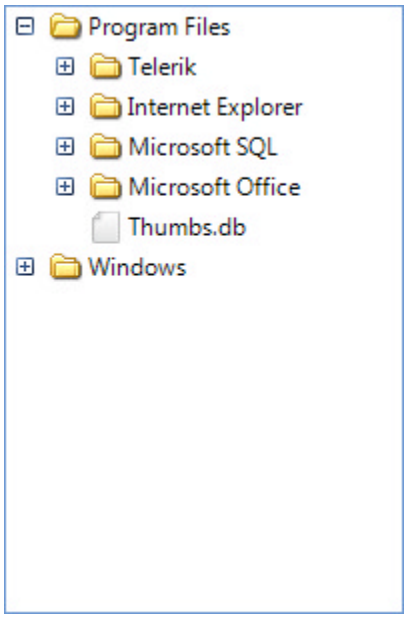
[C#] Setting Image and "Far" Image Index

```
RadTreeNode rootNode = new RadTreeNode("Telerik corp®");
rootNode.Image = Image.FromFile(@"C:\image.png");
rootNode.ImageKey = "imageName.png";
rootNode.ImageIndex = 0;
radTreeView1.Nodes.Add(rootNode);
```



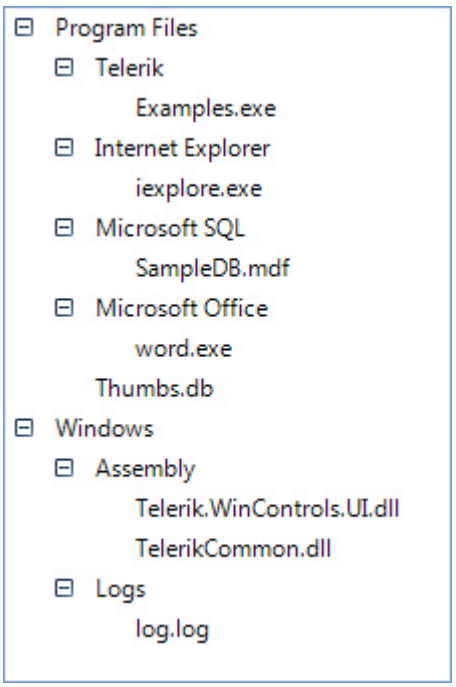
Node Formatting

Similar to RadGridView, RadTreeView is virtualized. Its visual nodes (TreeNodeElement) are reused by the data nodes (RadTreeNode) which bring increased performance and optimized memory footprint. You can easily format node element by handling the NodeFormatting event. The arguments of this event return the visual Node (and the data Node that is currently assigned to it) that is currently formatted and which you can additionally style.



The purpose of the example that follows is to demonstrate how you can show different images for the RadTreeView nodes depending on their logical state. We are going to simulate a tree of files and folders displaying an image of a closed folder when a node that contains other nodes is collapsed and displaying an image of an opended folder when a node that contains other nodes is expanded.

1. Let's fill RadTreeView with some sample nodes which represent files and folders:



2. Then, add three images as project resources. These images will indicate if a node is a file, an opened folder or a closed folder
3. Handle the NodeFormatting event implementing, setting the image to the Image property of the ImageElement. This element is responsible for containing the image of the visual NodeElement:

[VB.NET] Formatting nodes

```
Private folderOpen As Bitmap = My.Resources.folder_open
Private folderClose As Bitmap = My.Resources.folder_close
Private file As Bitmap = My.Resources.file

Private Sub radTreeView1_NodeFormatting(ByVal sender As Object, ByVal e As
TreeNodeFormattingEventArgs)
    If e.Node.Nodes.Count > 0 Then
        If e.Node.Expanded Then
            e.NodeElement.ImageElement.Image = folderOpen
        Else
            e.NodeElement.ImageElement.Image = folderClose
        End If
    Else
        e.NodeElement.ImageElement.Image = file
    End If
End Sub
```

[C#] Formatting nodes

```
Bitmap folderOpen = SamplesCS.Properties.Resources.folder_open;
Bitmap folderClose = SamplesCS.Properties.Resources.folder_close;
Bitmap file = SamplesCS.Properties.Resources.file;

void radTreeView1_NodeFormatting(object sender, TreeNodeFormattingEventArgs e)
{
   if (e.Node.Nodes.Count > 0)
   {
```

```
            if (e.Node.Expanded)
            {
                e.NodeElement.ImageElement.Image = folderOpen;
            }
            else
            {
                e.NodeElement.ImageElement.Image = folderClose;
            }
        }
        else
        {
            e.NodeElement.ImageElement.Image = file;
        }
    }
```



Please note that you should always provide an 'else' clause for each 'if' clause that you have in the implementation of the NodeFormatting event. Skipping this operation will lead to incorrect images applied to the inappropriate nodes.



It is also imporant that we are accessing the project resources outside the NodeFormatting event handler. Accessing project resources is a time consuming operation and since NodeFormatting is fired for every visible node, you may experience performance issues if you try to access such resources in the event handler.

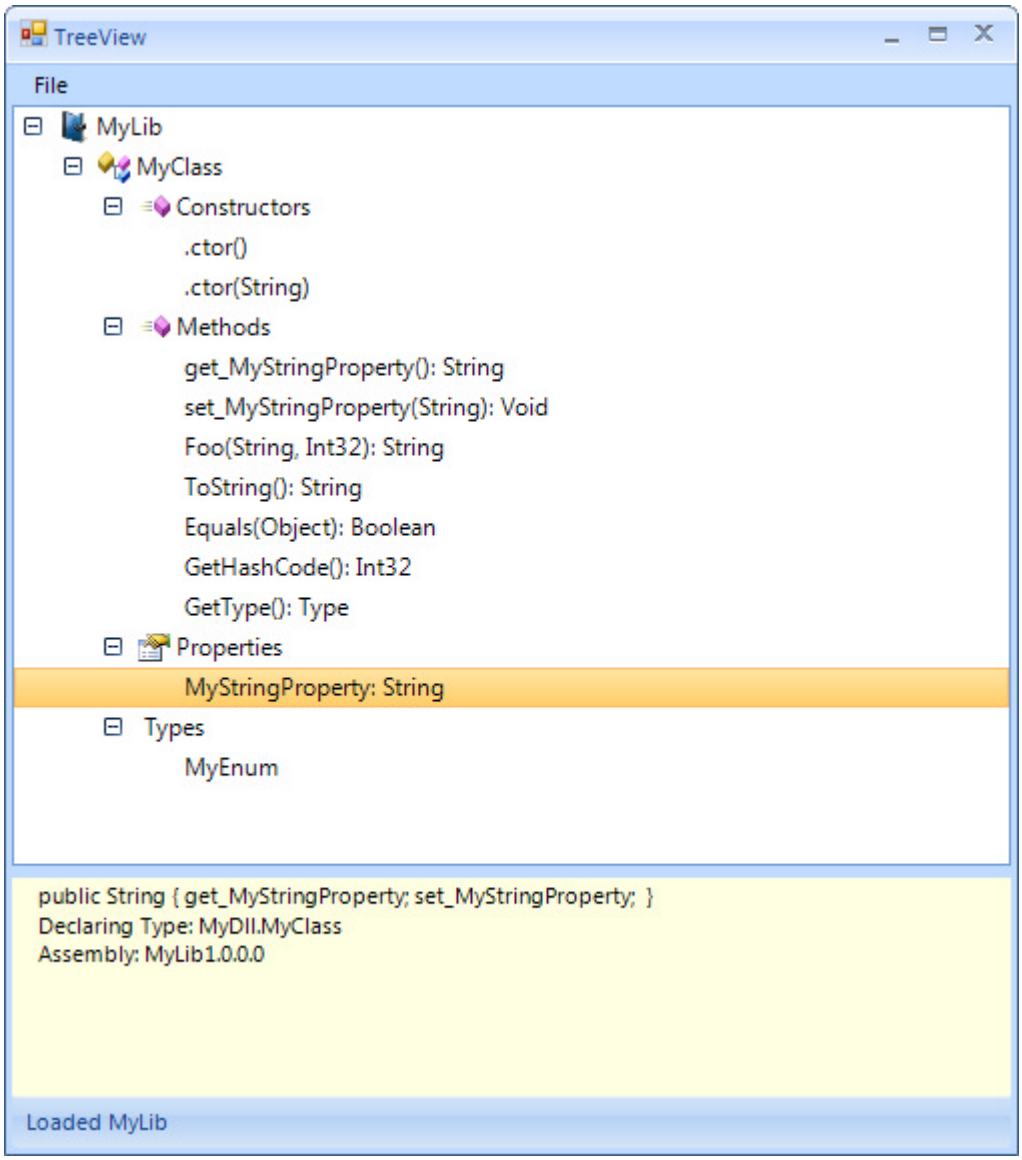
Working with NodesWalk-Through

This project walks you through building a multi-level tree using assembly reflection as the content data. The structure of the tree will be:

library
  class
    constructors
    methods
    properties
    types



The application will only show a limited amount of reflection information and may not work with all possible assemblies due to security and other restrictions. In this example we will write our own simple assemblay to reflect against.





You can find the complete source for this project at:

\Treeview\<VB|CS>\WorkingWithNodesWalkThrough

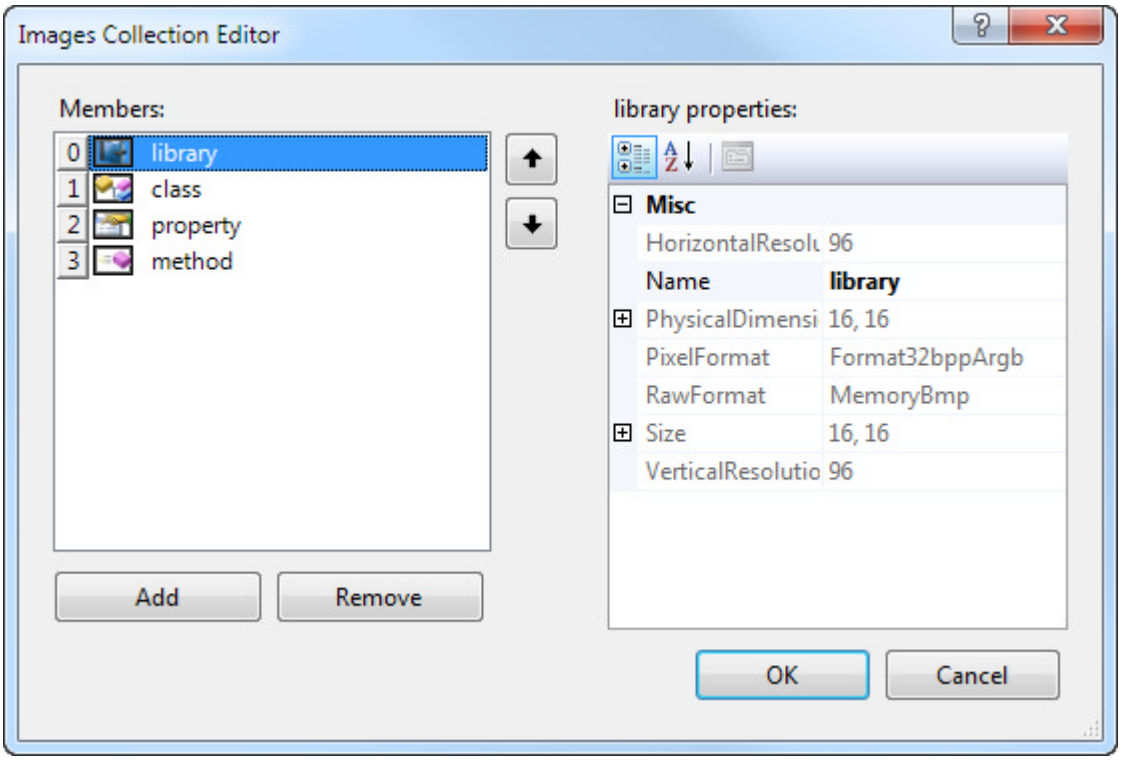
Preparing the Project

1. Create a new Windows Forms application.
2. In the Solution Explorer, delete the default form.
3. Also in the Solution Explorer, right-click the project and select **Add | New Item...** from the context menu.
4. Select the "Telerik RadForm" template and click the **Add** button to close the dialog.
5. Add a standard **ImageList** component to the form and name it "illcons". Use the ImageList Smart Tag to set the Image Bit Depth to "Depth32Bit". In the Properties window, set the ImageList **TransparentColor** to "Fuchsia". Click the ImageList Smart Tag "Choose Images" option and add images to represent the ideas below.

- Library
- Class
- Property
- Method



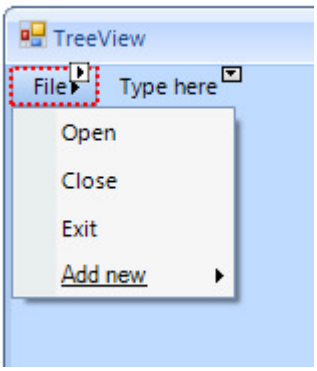
You can typically find images in the Visual Studio Common7\VS2008ImageLibrary\1033 directory in "VS2008ImageLibrary.zip". There are copies of the images used in this project at \Treeview\<VB|CS>\WorkingWithNodesWalkThrough\Images. You can add these to the ImageList and rename them to "library", "class", "property" and "method" for easier referencing in code.



6. Change the new RadForm1 to be the startup form.

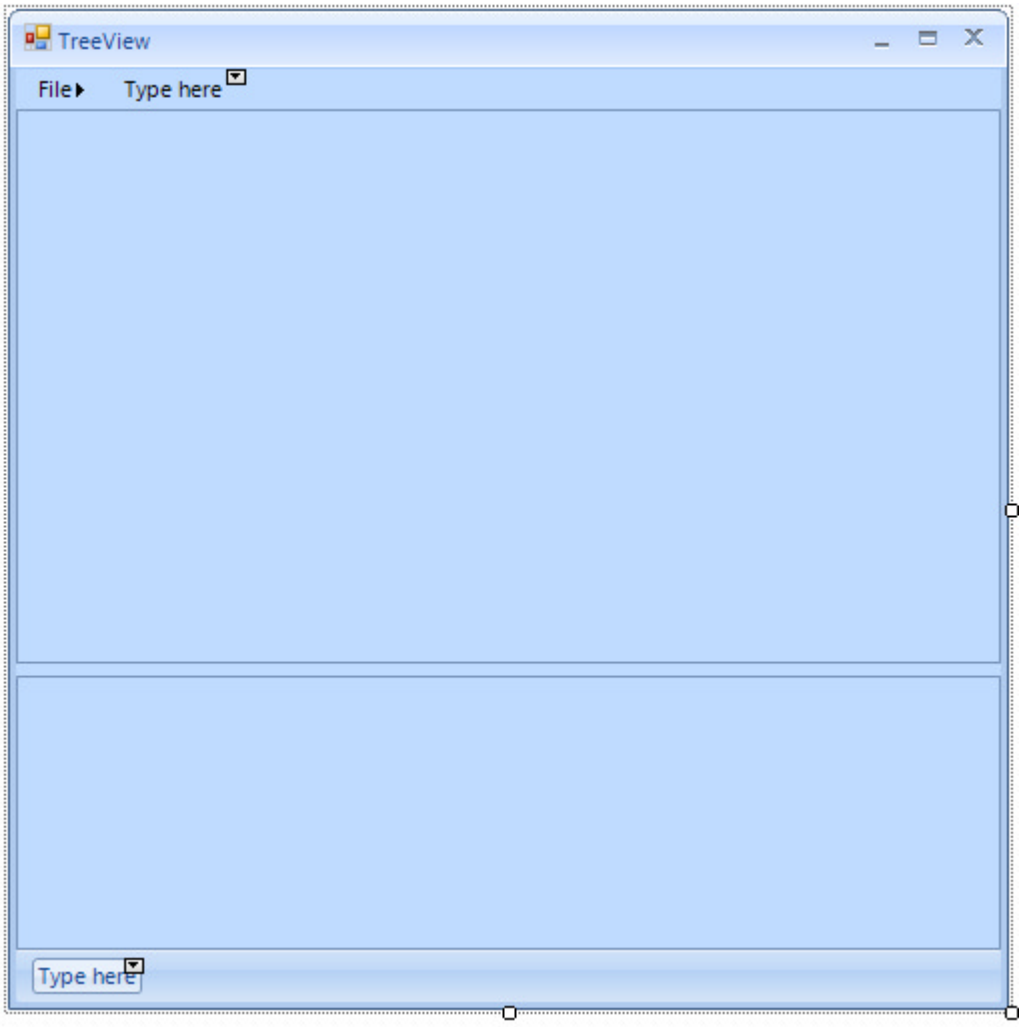
Form Setup

1. From the Toolbox, add a **RadMenu** to the form. In the designer, enter "File" to the "Type here" space on the menu. From the Smart tag click the "Add new" link of the "File" item, add three more menu Items and set properties:
   - **Name** = "miOpen", **Text** = "Open"
   - **Name** = "miClose", **Text** = "Close"
   - **Name** = "miExit", **Text** = "Exit"



2. Add a **RadStatusStrip** to the form. Add a **RadLabelElement** to the status strip, **Name** it "lblStatus" and set the **Text** = "".
3. Add two **RadPanels**, **Dock** the first one to "Top" and the second one to "Bottom". Change the top panel size to take approximately three quaters of the available space and the bottom panel size to take the rest of the space. Set both panels **Text** = "" and set the second panel **BackColor** to "Info".

   The form should look like the screenshot:



4. Add a **RadLabel** to the lower panel and set properties: **Name** = "lblDetail", **AutoSize** = False, **Dock** = "Fill", **Padding** = "10, 0, 0, 0".
5. Add a **RadTreeView** to the upper window and set properties: **Name** = "tvReflection", **Dock** = "Fill".

Adding Code

1. Make sure you are referencing the following assemblies in the "Imports" (VB) or "uses" (C#) section of code.

[VB] Adding References

```
Imports System
Imports System.IO
Imports System.Reflection
Imports System.Text
Imports System.Windows.Forms
Imports Telerik.WinControls
Imports Telerik.WinControls.UI
```

[C#] Adding References

```
using System;
using System.IO;
using System.Reflection;
using System.Text;
using System.Windows.Forms;
using Telerik.WinControls;
using Telerik.WinControls.UI;
```

2. Add a private helper method to return a comma-delimited list of parameter types, e.g. "(String, int, int, object)". It takes an array of ParameterInfo from the System.Reflection namespace. We will use this later to get information about constructors and methods.

[VB] Getting a Formatted List of Parameters

```
' return a string surrounded with parenthesis and containing
' a comma-delimited list of parameter types, e.g.
' "(String, String, Int32)"
Private Function GetParameterList(parameters As ParameterInfo()) As String
 Dim builder As New StringBuilder("(")
 Dim i As Integer = 0
 For Each parameter As ParameterInfo In parameters
  builder.Append(parameter.ParameterType.Name)
  If System.Threading.Interlocked.Increment(i) < parameters.Length Then
   builder.Append(", ")
  End If
 Next
 builder.Append(")")
 Return builder.ToString()
End Function
```

[C#] Getting a Formatted List of Parameters

```
// return a string surrounded with parenthesis and containing
// a comma-delimited list of parameter types, e.g.
// "(String, String, Int32)"
private string GetParameterList(ParameterInfo[] parameters)
{
 StringBuilder builder = new StringBuilder("(");
 int i = 0;
 foreach (ParameterInfo parameter in parameters)
```

```
 {
   builder.Append(parameter.ParameterType.Name);
   if (++i < parameters.Length)
     builder.Append(", ");
 }
 builder.Append(")");
 return builder.ToString();
}
```

3. A bit of house-keeping code: in the designer, double-click the "Close" and "Exit" menu items and add code to both event handlers.

[VB] Handling Close and Exit Menu Item Click Events

```
Private Sub miClose_Click(sender As Object, e As EventArgs)
 tvReflection.Nodes.Clear()
 lblStatus.Text = [String].Empty
 lblDetail.Text = [String].Empty
End Sub
Private Sub miExit_Click(sender As Object, e As EventArgs)
 Me.Close()
End Sub
```

[C#] Handling Close and Exit Menu Item Click Events

```
private void miClose_Click(object sender, EventArgs e)
{
 tvReflection.Nodes.Clear();
 lblStatus.Text = String.Empty;
 lblDetail.Text = String.Empty;
}
private void miExit_Click(object sender, EventArgs e)
{
 this.Close();
}
```

4. In the designer, double-click the "Open" menu item and add code to the click event. The event handler uses an OpenFileDialog to get the path of an assembly and then loads the assembly. Finally the method calls private method LoadNodes() that will traverse the reflection information of the assembly. Notice that we're using the static Assembly ReflectionOnlyLoadFrom() method to load the file.

[VB] Handling the Menu Item Click Event

```
' Get a path and load an Assembly.
' Call private method LoadNodes() to begin reflecting
' on the assembly contents
Private Sub miOpen_Click(sender As Object, e As EventArgs)
 Dim dialog As New OpenFileDialog()
 dialog.Filter = "Dynamic Link Libraries|*.dll|Executable|*.exe"
 If dialog.ShowDialog() = DialogResult.OK Then
  Dim assembly As Assembly = Assembly.ReflectionOnlyLoadFrom(dialog.FileName)
  LoadNodes(tvReflection.Nodes, assembly)
 End If
 lblStatus.Text = "Loaded " + Path.GetFileNameWithoutExtension(dialog.FileName)
 tvReflection.ExpandAll()
End Sub
```

[C#] Handling the Menu Item Click Event

```
// Get a path and load an Assembly.
// Call private method LoadNodes() to begin reflecting
```

```
// on the assembly contents
private void miOpen_Click(object sender, EventArgs e)
{
 OpenFileDialog dialog = new OpenFileDialog();
 dialog.Filter = "Dynamic Link Libraries|*.dll|Executable|*.exe";
 if (dialog.ShowDialog() == DialogResult.OK)
 {
   Assembly assembly = Assembly.ReflectionOnlyLoadFrom(dialog.FileName);
   LoadNodes(tvReflection.Nodes, assembly);
 }
 lblStatus.Text = "Loaded " + Path.GetFileNameWithoutExtension(dialog.FileName);
 tvReflection.ExpandAll();
}
```

5. Add a private method LoadNodes() to traverse the Assembly reflection information and create nodes to structure this information in the tree. The method adds a root node for the assembly itself, then iterates all the types in the assembly and calls the LoadTypeNode() method.

[VB] Traversing Assembly Reflection

```
Private Sub LoadNodes(ByVal nodes As RadTreeNodeCollection, ByVal [assembly] As
System.Reflection.Assembly)
  nodes.Clear()
  ' get the name, e.g. "MyDll" from the Assembly
  Dim assemblyName As String = Path.GetFileNameWithoutExtension([assembly].Location)
  ' add a root node with the assembly name and a "library" icon
  ' store the assembly reference itself in the tag for later retrieval
  Dim rootNode As RadTreeNode = nodes.Add(assemblyName)
  rootNode.Image = ilIcons.Images("library")
  rootNode.Tag = [assembly]
  ' reflect all the types in the assembly
  For Each type As Type In [assembly].GetTypes()
 ' only look at top level types
 If (Not type.IsNested) Then
   ' load a node for each top level type
   Dim typeNode As RadTreeNode = rootNode.Nodes.Add(type.Name)
   typeNode.Tag = type
   If type.IsClass Then
  typeNode.Image = ilIcons.Images("class")
   End If
   ' drill down for further reflection on the type
   LoadTypeNode(type, typeNode)
 End If
  Next type
End Sub
```

[C#] Traversing Assembly Reflection

```
private void LoadNodes(RadTreeNodeCollection nodes, Assembly assembly)
{
 nodes.Clear();
 // get the name, e.g. "MyDll" from the Assembly
 string assemblyName = Path.GetFileNameWithoutExtension(assembly.Location);
 // add a root node with the assembly name and a "library" icon
 // store the assembly reference itself in the tag for later retrieval
 RadTreeNode rootNode = nodes.Add(assemblyName);
 rootNode.Image = ilIcons.Images["library"];
 rootNode.Tag = assembly;
```

```
 // reflect all the types in the assembly
 foreach (Type type in assembly.GetTypes())
 {
   // only look at top level types
   if (!type.IsNested)
   {
     // load a node for each top level type
     RadTreeNode typeNode = rootNode.Nodes.Add(type.Name);
     typeNode.Tag = type;
     if (type.IsClass)
       typeNode.Image = ilIcons.Images["class"];
     // drill down for further reflection on the type
     LoadTypeNode(type, typeNode);
   }
 }
}
```

6. Add a private method LoadTypeNode() that creates nodes for the constructors, methods, properties and sub-types of a given type.

[VB] Adding Reflection Information for a Type

```
Private Sub LoadTypeNode(type As Type, typeNode As RadTreeNode)
 ' add nodes for different categories of reflection
 Dim constructorNode As RadTreeNode = typeNode.Nodes.Add("Constructors")
 constructorNode.Image = ilIcons.Images("method")
 Dim methodNode As RadTreeNode = typeNode.Nodes.Add("Methods")
 methodNode.Image = ilIcons.Images("method")
 Dim propertyNode As RadTreeNode = typeNode.Nodes.Add("Properties")
 propertyNode.Image = ilIcons.Images("property")
 Dim nestedTypeNode As RadTreeNode = typeNode.Nodes.Add("Types")
 ' add nodes for each type...
 For Each constructor As ConstructorInfo In type.GetConstructors()
  Dim tempNode As RadTreeNode = constructorNode.Nodes.Add(constructor.Name +
GetParameterList(constructor.GetParameters()))
  tempNode.Tag = constructor
 Next
 For Each method As MethodInfo In type.GetMethods()
  Dim methodText As String = method.Name + GetParameterList(method.GetParameters()) + ":
method.ReturnParameter.ParameterType.Name
  Dim tempNode As RadTreeNode = methodNode.Nodes.Add(methodText)
  tempNode.Tag = method
 Next
 For Each [property] As PropertyInfo In type.GetProperties()
  Dim tempNode As RadTreeNode = propertyNode.Nodes.Add([property].Name + ": " +
[property].PropertyType.Name)
  tempNode.Tag = [property]
 Next
 For Each nestedType As Type In type.GetNestedTypes()
  Dim tempNode As RadTreeNode = nestedTypeNode.Nodes.Add(nestedType.Name)
  tempNode.Tag = nestedType
 Next
End Sub
```

[C#] Adding Reflection Information for a Type

```
private void LoadTypeNode(Type type, RadTreeNode typeNode)
{
```

```
// add nodes for different categories of reflection
RadTreeNode constructorNode = typeNode.Nodes.Add("Constructors");
constructorNode.Image = ilIcons.Images["method"];
RadTreeNode methodNode = typeNode.Nodes.Add("Methods");
methodNode.Image = ilIcons.Images["method"];
RadTreeNode propertyNode = typeNode.Nodes.Add("Properties");
propertyNode.Image = ilIcons.Images["property"];
RadTreeNode nestedTypeNode = typeNode.Nodes.Add("Types");
// add nodes for each type...
foreach (ConstructorInfo constructor in type.GetConstructors())
{
  RadTreeNode tempNode = constructorNode.Nodes.Add(constructor.Name +
    GetParameterList(constructor.GetParameters()));
  tempNode.Tag = constructor;
}
foreach (MethodInfo method in type.GetMethods())
{
  string methodText = method.Name +
    GetParameterList(method.GetParameters()) + ": " +
    method.ReturnParameter.ParameterType.Name;
  RadTreeNode tempNode = methodNode.Nodes.Add(methodText);
  tempNode.Tag = method;
}
foreach (PropertyInfo property in type.GetProperties())
{
  RadTreeNode tempNode =
    propertyNode.Nodes.Add(property.Name + ": " + property.PropertyType.Name);
  tempNode.Tag = property;
}
foreach (Type nestedType in type.GetNestedTypes())
{
  RadTreeNode tempNode = nestedTypeNode.Nodes.Add(nestedType.Name);
  tempNode.Tag = nestedType;
}
}
```

7. Now we need an assembly to test against. In the Solution Explorer, right-click the solution and select **Add | New Project...**. Select "Class Library" and name it MyLib. Open the Class1.cs file that gets generated automatically and replace the code. This will give us some simple assembly reflection to work with.

[VB] Declaring a Class to Test Reflection

```
Namespace MyDll
 Public Class [MyClass]
  Public Enum MyEnum
   Up
   Down
   Sideways
  End Enum

  Public Sub New()
  End Sub
  Public Sub New(myStringProperty As String)
   Me.MyStringProperty = myStringProperty
  End Sub
  Private _myStringProperty As String
  Public Property MyStringProperty() As String
```

```
    Get
     Return _myStringProperty
    End Get
    Set
     _myStringProperty = value
    End Set
   End Property
   Public Function Foo(message As String, count As Integer) As String
    Dim result As String = ""
    Dim i As Integer = 0
    While i < count
     result += message
     System.Math.Max(System.Threading.Interlocked.Increment(i),i - 1)
    End While
    Return result
   End Function
   Private Sub Bar()
   End Sub
 End Class
End Namespace
```

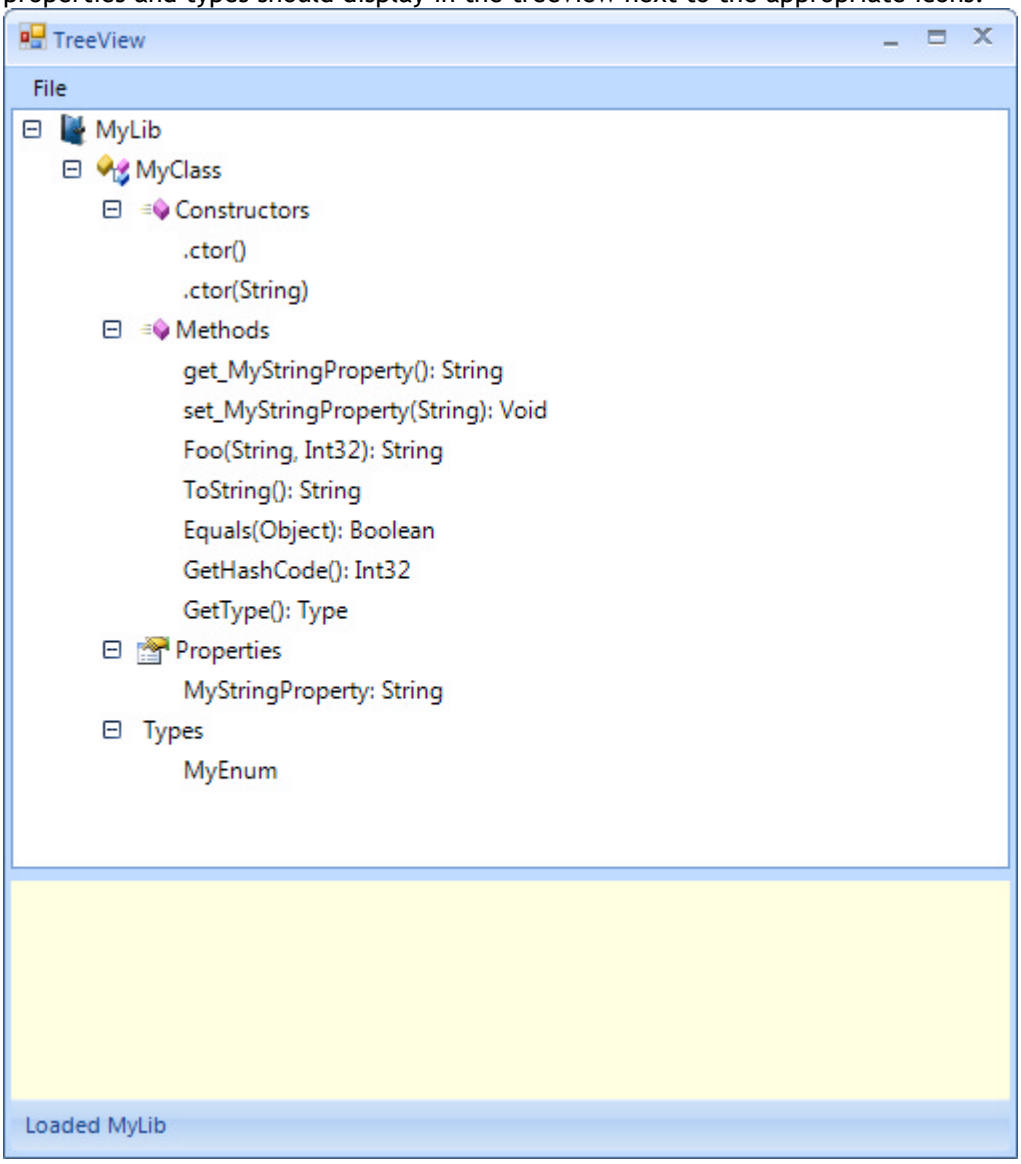
[C#] Declaring a Class to Test Reflection

```
namespace MyDll
{
 public class MyClass
 {
   public enum MyEnum { Up, Down, Sideways };
   public MyClass()
   {
   }
   public MyClass(string myStringProperty)
   {
     this.MyStringProperty = myStringProperty;
   }
   private string _myStringProperty;
   public string MyStringProperty
   {
     get
     {
       return _myStringProperty;
     }
     set
     {
       _myStringProperty = value;
     }
   }
   public string Foo(string message, int count)
   {
     string result = "";
     for (int i = 0; i < count; i++)
     {
       result += message;
     }
     return result;
   }
```

```
        private void Bar()
        {
        }
    }
}
```

8. In the Solution Explorer locate the **References** node of the forms project, right-click and select **Add Reference** from the context menu. Select "MyLib" from the list in the Projects tab and click **OK** to add the reference.
9. Press **Ctrl-F5** to run the application.

   Click the "Open" file menu. Locate the "MyLib.dll" file and open it. "MyClass", its constructors, methods, properties and types should display in the treeview next to the appropriate icons.



Responding to Events

Lets improve the example by showing detail in the lower panel for selected nodes. We will be responding to the treeview SelectedNodeChanged event and retrieving the Tag value if there is one. The Tag may hold an Assembly, ConstructorInfo, MethodInfo, PropertyInfo or TypeInfo object. The information in these classes will

be displayed in the lower "detail" panel.

1. Add the following helper methods that dump object details from the node Tag into strings.

[VB] Extracting Reflection Information

```
Private Function GetAssemblyDescription(ByVal [assembly] As System.Reflection.Assembly) A
String
  Dim builder As New StringBuilder([assembly].GetName().Name)
  Dim version As Version = [assembly].GetName().Version
  builder.AppendFormat("{0}.{1}.{2}.{3}", version.Major, version.MajorRevision,
version.Minor, version.MinorRevision)
  Return builder.ToString()
End Function
Private Function GetTypeDescription(ByVal type As Type) As String
  Dim builder As New StringBuilder()
  If type.IsPublic Then
 builder.Append("public ")
  End If
  If type.IsClass Then
 builder.Append("class ")
  ElseIf type.IsEnum Then
 builder.Append("enum ")
  End If
  builder.Append(type.Name)
  builder.Append(Environment.NewLine)
  builder.Append("Name: ")
  builder.Append(type.FullName)
  builder.Append(Environment.NewLine)
  builder.Append("Assembly: ")
  builder.Append(GetAssemblyDescription(type.Assembly))
  Return builder.ToString()
End Function
Private Function GetMethodDescription(ByVal method As MethodBase) As String
  Dim builder As New StringBuilder()
  If method.IsPublic Then
 builder.Append("public ")
  End If
  If TypeOf method Is MethodInfo Then
 builder.Append(method.Name)
  Else
 builder.Append(method.DeclaringType.Name)
  End If
  builder.Append(GetParameterList(method.GetParameters()))
  If TypeOf method Is MethodInfo Then
 builder.Append(": ")
 builder.Append((TryCast(method, MethodInfo)).ReturnType.Name)
  End If
  builder.Append(Environment.NewLine)
  builder.Append("Declaring Type: ")
  builder.Append(method.DeclaringType.FullName)
  builder.Append(Environment.NewLine)
  builder.Append("Assembly: ")
  builder.Append(GetAssemblyDescription(method.DeclaringType.Assembly))
  Return builder.ToString()
End Function
Private Function GetPropertyDescription(ByVal [property] As PropertyInfo) As String
```

```
  Dim builder As New StringBuilder()
  If [property].PropertyType.IsPublic Then
 builder.Append("public ")
  End If
  builder.Append([property].PropertyType.Name)
  builder.Append(" { ")
  For Each method As MethodInfo In [property].GetAccessors()
 builder.AppendFormat("{0}; ", method.Name)
  Next method
  builder.Append(" } ")
  builder.Append(Environment.NewLine)
  builder.Append("Declaring Type: ")
  builder.Append([property].DeclaringType.FullName)
  builder.Append(Environment.NewLine)
  builder.Append("Assembly: ")
  builder.Append(GetAssemblyDescription([property].DeclaringType.Assembly))
  Return builder.ToString()
End Function
```

[C#] Extracting Reflection Information

```
private string GetAssemblyDescription(Assembly assembly)
{
 StringBuilder builder = new StringBuilder(assembly.GetName().Name);
 Version version = assembly.GetName().Version;
 builder.AppendFormat("{0}.{1}.{2}.{3}", version.Major,
   version.MajorRevision, version.Minor, version.MinorRevision);
 return builder.ToString();
}
private string GetTypeDescription(Type type)
{
 StringBuilder builder = new StringBuilder();
 if (type.IsPublic)
   builder.Append("public ");
 if (type.IsClass)
   builder.Append("class ");
 else if (type.IsEnum)
   builder.Append("enum ");
 builder.Append(type.Name);
 builder.Append(Environment.NewLine);
 builder.Append("Name: ");
 builder.Append(type.FullName);
 builder.Append(Environment.NewLine);
 builder.Append("Assembly: ");
 builder.Append(GetAssemblyDescription(type.Assembly));
 return builder.ToString();
}
private string GetMethodDescription(MethodBase method)
{
 StringBuilder builder = new StringBuilder();
 if (method.IsPublic)
   builder.Append("public ");
 if (method is MethodInfo)
   builder.Append(method.Name);
 else
   builder.Append(method.DeclaringType.Name);
```

```
 builder.Append(GetParameterList(method.GetParameters()));
 if (method is MethodInfo)
 {
   builder.Append(": ");
   builder.Append((method as MethodInfo).ReturnType.Name);
 }
 builder.Append(Environment.NewLine);
 builder.Append("Declaring Type: ");
 builder.Append(method.DeclaringType.FullName);
 builder.Append(Environment.NewLine);
 builder.Append("Assembly: ");
 builder.Append(GetAssemblyDescription(method.DeclaringType.Assembly));
 return builder.ToString();
}
private string GetPropertyDescription(PropertyInfo property)
{
 StringBuilder builder = new StringBuilder();
 if (property.PropertyType.IsPublic)
   builder.Append("public ");
 builder.Append(property.PropertyType.Name);
 builder.Append(" { ");
 foreach (MethodInfo method in property.GetAccessors())
 {
   builder.AppendFormat("{0}; ", method.Name);
 }
 builder.Append(" } ");
 builder.Append(Environment.NewLine);
 builder.Append("Declaring Type: ");
 builder.Append(property.DeclaringType.FullName);
 builder.Append(Environment.NewLine);
 builder.Append("Assembly: ");
 builder.Append(GetAssemblyDescription(property.DeclaringType.Assembly));
 return builder.ToString();
}
```

2. Create a **SelectedNoteChanged** event handler that assigns description strings to the "lblDetail" label in the lower panel.

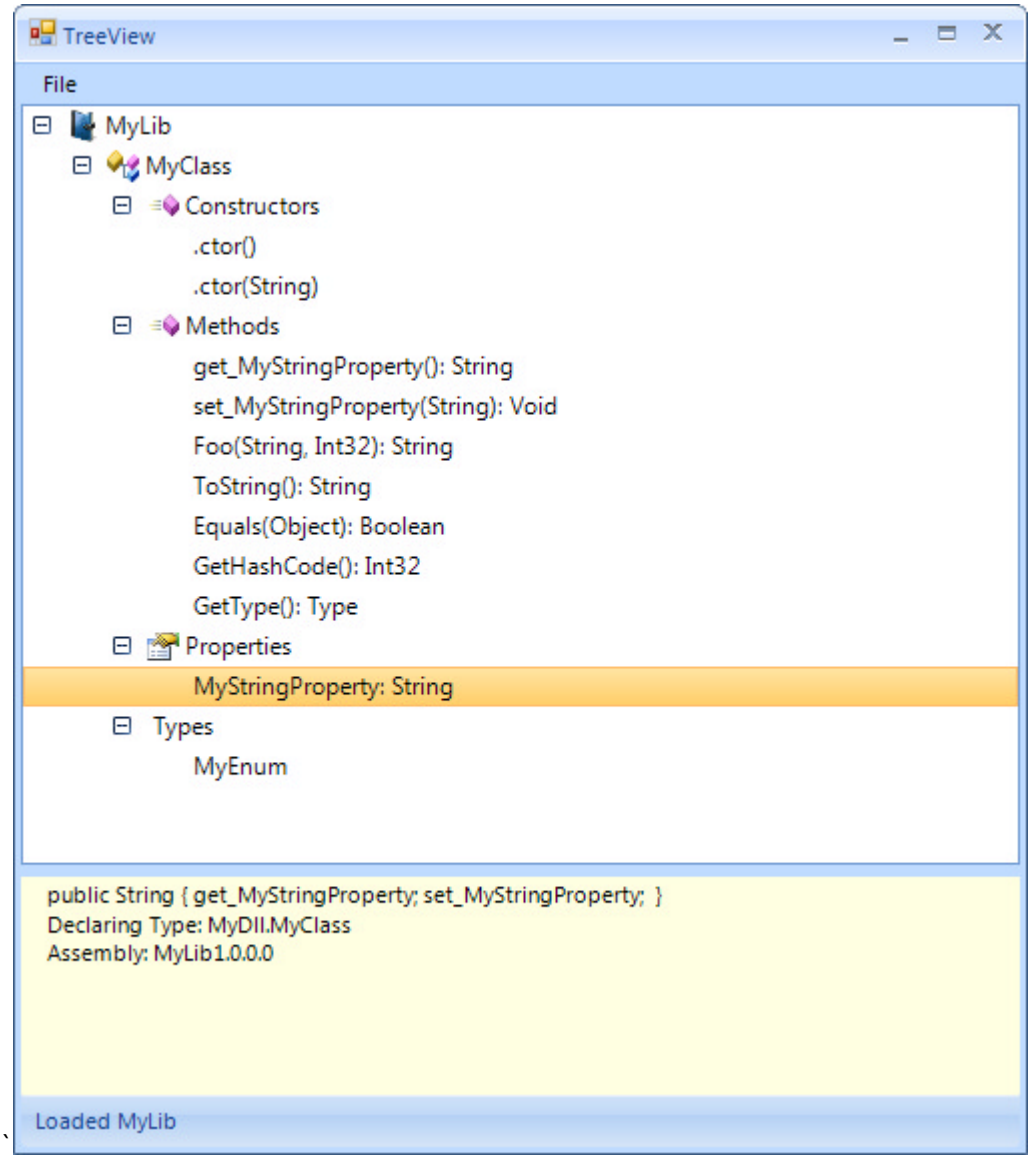
[VB] Handling the SelectedNoteChanged Event

```
Private Sub tvReflection_SelectedNodeChanged(sender As Object, e As RadTreeViewEventArgs)
 Dim node As RadTreeNode = TryCast(sender, RadTreeNode)
 lblDetail.Text = [String].Empty
 If node.Tag <> Nothing Then
  If TypeOf node.Tag Is Assembly Then
   lblDetail.Text = GetAssemblyDescription(TryCast(node.Tag, Assembly))
  ElseIf TypeOf node.Tag Is Type Then
   lblDetail.Text = GetTypeDescription(TryCast(node.Tag, Type))
  ElseIf TypeOf node.Tag Is ConstructorInfo Then
   lblDetail.Text = GetMethodDescription(TryCast(node.Tag, ConstructorInfo))
  ElseIf TypeOf node.Tag Is MethodInfo Then
   lblDetail.Text = GetMethodDescription(TryCast(node.Tag, MethodInfo))
  ElseIf TypeOf node.Tag Is PropertyInfo Then
   lblDetail.Text = GetPropertyDescription(TryCast(node.Tag, PropertyInfo))
  End If
 End If
End Sub
```

[C#] Handling the SelectedNoteChanged Event

```
private void tvReflection_SelectedNodeChanged(object sender, RadTreeViewEventArgs e)
{
 RadTreeNode node = sender as RadTreeNode;
 lblDetail.Text = String.Empty;
 if (node.Tag != null)
 {
   if (node.Tag is Assembly)
     lblDetail.Text = GetAssemblyDescription(node.Tag as Assembly);
   else if (node.Tag is Type)
     lblDetail.Text = GetTypeDescription(node.Tag as Type);
   else if (node.Tag is ConstructorInfo)
     lblDetail.Text = GetMethodDescription(node.Tag as ConstructorInfo);
   else if (node.Tag is MethodInfo)
     lblDetail.Text = GetMethodDescription(node.Tag as MethodInfo);
   else if (node.Tag is PropertyInfo)
     lblDetail.Text = GetPropertyDescription(node.Tag as PropertyInfo);
 }
}
```

3. Press **Ctrl-F5** to run the application. Click the assembly, constructors, methods, types and properties and notice the results in the lower details panel.



Data Binding

Simple Data Binding

Before we bind to hierarchical data, lets use the treeview as a simple list to see the most minimal databinding possible.



You can find the complete source for this project at:

\Treeview\<VB|CS>\DatabindingTreeView

Set Program.cs to start with the form SimpleData.

Start with a **RadTreeView** on a form or RadForm. Using the RadTreeView DataSource property, add a project data source that points at the NWind.mdb file found in the RadControls for Winforms installation directory under \Examples\QuickStart\DataSources (see the Data Binding chapter for more information on how to do this). We only need three lines of code and the first line is generated automatically when you configure the DataSource.

[VB] Minimal Binding Example

```
Private Sub SimpleData_Load(sender As Object, e As EventArgs)
 Me.employeesTableAdapter.Fill(Me.nwindDataSet.Employees)

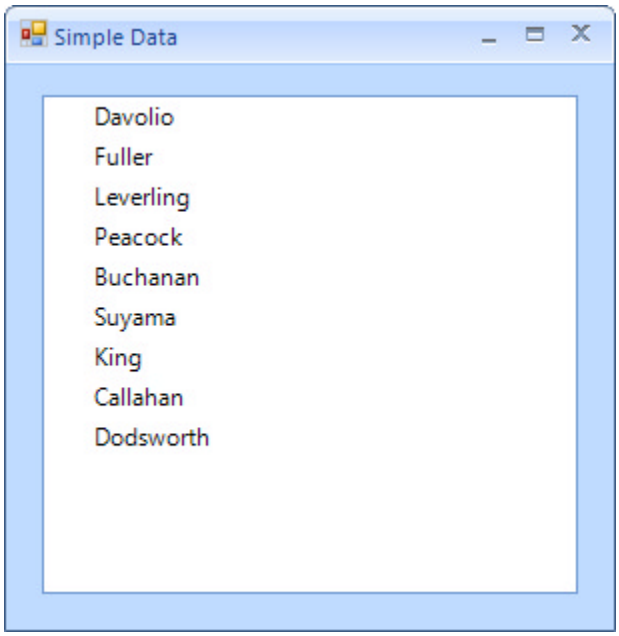
radTreeView1.DataSource = this.nwindDataSet.Employees
radTreeView1.DisplayMember = "LastName"
End Sub
```

[C#] Minimal Binding Example

```
private void SimpleData_Load(object sender, EventArgs e)
{
 this.employeesTableAdapter.Fill(this.nwindDataSet.Employees);

radTreeView1.DataSource = this.nwindDataSet.Employees;
radTreeView1.DisplayMember = "LastName";
}
```

This produces a flat list. The **DisplayName** property is set in order to specify which column records from the Employees table to be listed in RadTreeView.



Binding Hierarchical Data Overview

To get best use of the treeview, databinding has to take data hierarchy into account. There are a couple of ways to approach this:

- Bind a single table that references to itself. For example the data might include an "ID" and a "ParentID" column. Or as in the case of the NWind.mdb Employees table there is a "EmployeeID" that uniquely identifies each record and a "ReportsTo" column that points to a parent record's EmployeeID. The root level node will have no parent record as is the case for the "Vice President, Sales" that has a null "ReportsTo" value.

EmployeeID	ReportsTo	Title
1	2	Sales Representative
2	*NULL*	Vice President, Sales
3	2	Sales Representative
4	2	Sales Representative
5	2	Sales Manager
6	5	Sales Representative
7	5	Sales Representative
8	2	Inside Sales Coordinator
9	5	Sales Representative

- Bind the treeview to multiple tables and add RelationBindings that describe how one table relates to another. For example, use the MusicCollection.mdb database that has three tables Artists, Albums for the artists and Songs that belong to the Albums.

Binding Self Referencing Data



You can find the complete source for this project at:

\Treeview\<VB|CS>\Databinding

Set Program.cs to start with the form "SelfReferencing".

Using the previous simple binding example we only need to specify a couple of properties to implement self referencing that will produce a true hierarchy display of the data. You can see in the code below that we only need to define:

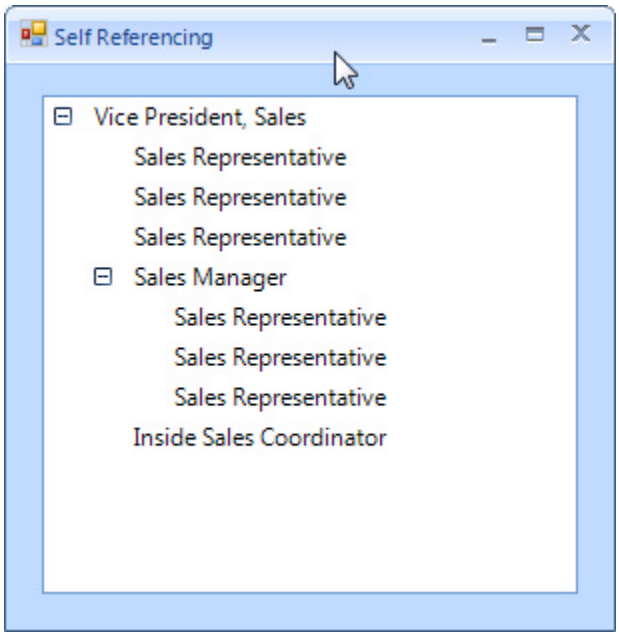
- **DisplayMember** - the name of the field that populates the **Text** property of each child node.
- **ChildMember** - the name of the child data filend which takes part in the relation.
- **ParentMember** - the name of the parent data field which takes part in the relation.

[VB] Binding Self Referencing Data

```
Private Sub RadForm1_Load(sender As Object, e As EventArgs)
 Me.employeesTableAdapter.Fill(Me.nwindDataSet.Employees)
 radTreeView1.DataSource = Me.nwindDataSet.Employees
 radTreeView1.DisplayMember = "Title"
 radTreeView1.ChildMember = "EmployeeID"
 radTreeView1.ParentMember = "ReportsTo"
End Sub
```

[C#] Binding Self Referencing Data

```
private void RadForm1_Load(object sender, EventArgs e)
{
 this.employeesTableAdapter.Fill(this.nwindDataSet.Employees);
 radTreeView1.DataSource = this.nwindDataSet.Employees;
 radTreeView1.DisplayMember = "Title";
 radTreeView1.ChildMember = "EmployeeID";
 radTreeView1.ParentMember = "ReportsTo";
}
```



Binding Related Data



You can find the complete source for this project at:

\Treeview\<VB|CS>\Databinding

Set Program.cs to start with the form "RelatedData".

Related data can take the form of:

- DataTables in a DataSet participating in relations represented by a DataRelation.
- Business Objects that have nested collections. In this case items in nested collections are considered to be child records of the business object that contains the collections.

The configuration of relational data is described by adding to the **RelationBindings** collection of RadTreeView. Minimally you need to set the data source and the display member for the control and datasource, display member, parent member and child member (or the parentChildMember when their value is equal) for the relation. For example:

[VB] Adding a RelationBinding

```
Me.radTreeView1.DataSource = this.artistsBindingSource
Me.radTreeView1.DisplayMember = "ArtistName"
radTreeView1.RelationBindings.Add(albumsBindingSource, "AlbumName", "ArtistID");
```

[C#] Adding a RelationBinding

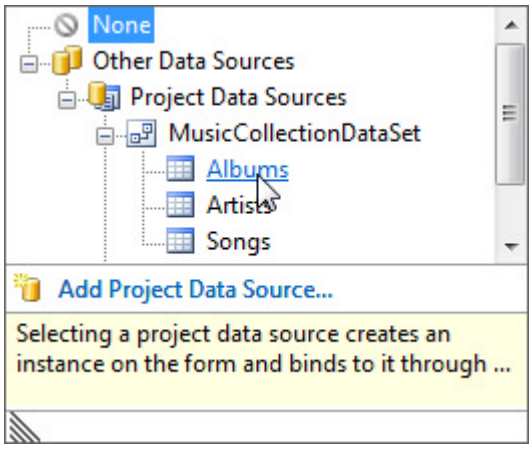
```
this.radTreeView1.DataSource = this.artistsBindingSource;
this.radTreeView1.DisplayMember = "ArtistName";
radTreeView1.RelationBindings.Add(albumsBindingSource, "AlbumName", "ArtistID");
```

Lets walk through binding the treeview to the Artists/Albums/Songs trilogy of tables.

1. Start with a **RadTreeView** on a Form or RadForm.
2. Find the treeview **DataSource** property in the Properties Window, drop down the list and select **Add Project Data Source**. This will display the Data Source Configuration Wizard.
3. In the "Choose a Data Source Type" page of the wizard select the **Database** icon, then click the **Next**

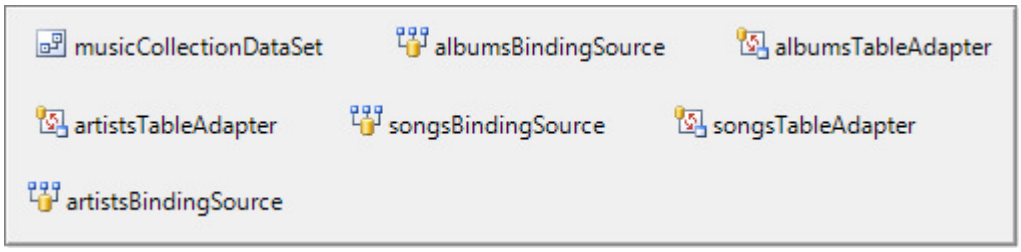
button.

4. In the "Choose Your Data Connection" page of the wizard add a new connection:
   - Click the **New Connection** button.
   - In the "Add Connection" dialog click the **Change** button, select "Microsoft Access Database File" and click the **OK** button.
   - Next to the Database file name entry, click the **Browse** button and locate the MusicCollection.mdb file in the installation directory under \Examples\QuickStart\DataSources.
   - Click the **OK** button to close the "Add Connection" dialog.
5. When prompted if you would like to copy the local data file to your project click the **No** button.
6. In the "Save the Connection String to the Application Configuration File" page of the wizard click the **Next** button.
7. In the "Choose Your Database Objects" page of the wizard click the Tables checkbox (this will automatically check the Albums, Artists and Songs tables).
8. Click the **Finish** button to complete and close the wizard dialog.
9. Select each table to create components
   - Back in the treeview DataSource property, drop down the list and locate the "Albums" table from under the "MusicCollectionDataSet" dataset.



   - Select the "Artists" table.
   - Select the "Songs" table.

   You should now have the DataSet for the music collection and pairs of BindingSource and TableAdapter components for each table in the component tray.



10. Add the following statements to the form Load event handler
    - Add a "Imports" (VB) or "uses" (C#) statement referencing **Telerik.WinControls.UI** to support the RelationBinding object.

- Include RelationBinding.Add() methods for both "ArtistsAlbums" and "AlbumSongs".
- Define the RadTreeView RootRelationDisplayname property.
- Set the DataSource to use the "artistsBindingSource" component.

[VB] Loading Data and Defining Relation Bindings

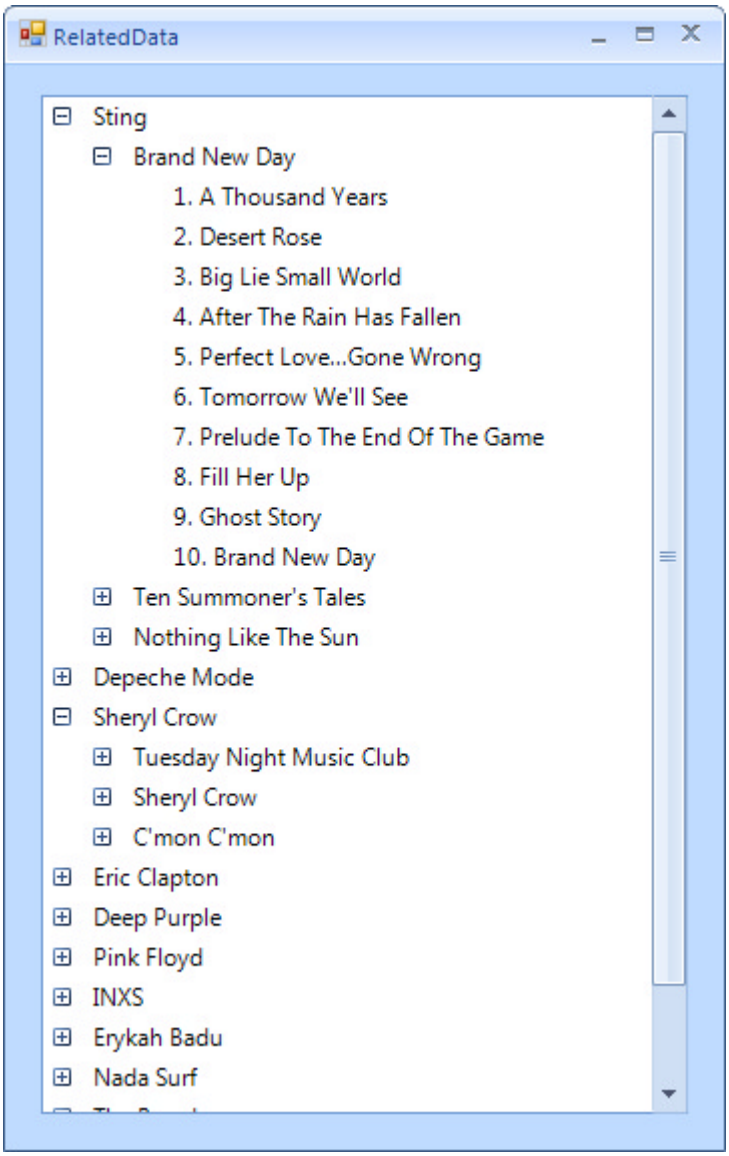
```
Private Sub RelatedData_Load(sender As Object, e As EventArgs)
 ' generated automatically by configuration
 Me.songsTableAdapter.Fill(Me.musicCollectionDataSet.Songs)
 Me.artistsTableAdapter.Fill(Me.musicCollectionDataSet.Artists)
 Me.albumsTableAdapter.Fill(Me.musicCollectionDataSet.Albums)
 ' added by hand to define the relation bindings

Me.radTreeView1.DataSource = this.artistsBindingSource
Me.radTreeView1.DisplayMember = "ArtistName"
radTreeView1.RelationBindings.Add(albumsBindingSource, "AlbumName", "ArtistID")
radTreeView1.RelationBindings.Add(songsBindingSource, "SongName", "AlbumID")
End Sub
```

[C#] Loading Data and Defining Relation Bindings

```
private void RelatedData_Load(object sender, EventArgs e)
{
 // generated automatically by configuration
 this.songsTableAdapter.Fill(this.musicCollectionDataSet.Songs);
 this.artistsTableAdapter.Fill(this.musicCollectionDataSet.Artists);
 this.albumsTableAdapter.Fill(this.musicCollectionDataSet.Albums);
 // added by hand to define the relation bindings
this.radTreeView1.DataSource = this.artistsBindingSource;
this.radTreeView1.DisplayMember = "ArtistName";
radTreeView1.RelationBindings.Add(albumsBindingSource, "AlbumName", "ArtistID");
radTreeView1.RelationBindings.Add(songsBindingSource, "SongName", "AlbumID");
}
```

11. Press **Ctrl-F5** to run the application.



Using XML Data

There are two approaches to consuming XML data:

- Bind to a DataSet and reading the XML into the DataSet.
- Use special methods of RadTreeView to save and load the XML directly, i.e. serialize/deserialize the XML.

Binding to XML Data



You can find the complete source for this project at:

\Treeview\<VB|CS>\Databinding

Set Program.cs to start with the form "XMLBinding".

The XML should be structured so that it can be read by the standard DataSet ReadXml() method, something like the example below. The naming for each XML element is arbitrary, although the names are case sensitive. Notice that the "parentId" for the "Main title" node is null. Parent ID nodes with invalid data or that don't point at another valid ID node are ignored.

[XML] Sample Data to be Consumed

```
<toc>
 <item>
   <title>Main title</title>
   <id>1</id>
   <parentId></parentId>
 </item>
 <item>
   <title>Child title</title>
   <id>2</id>
   <parentId>1</parentId>
 </item>
</toc>
```

Assuming that "toc.xml" is in the same directory with the executable, the code below can be run from the form Load event handler.



If the XML file is in your project you should set its "Copy to Output Directory" property to "Copy Always" or "Copy if Newer" so that the file ends up in the \bin directory with the executable.

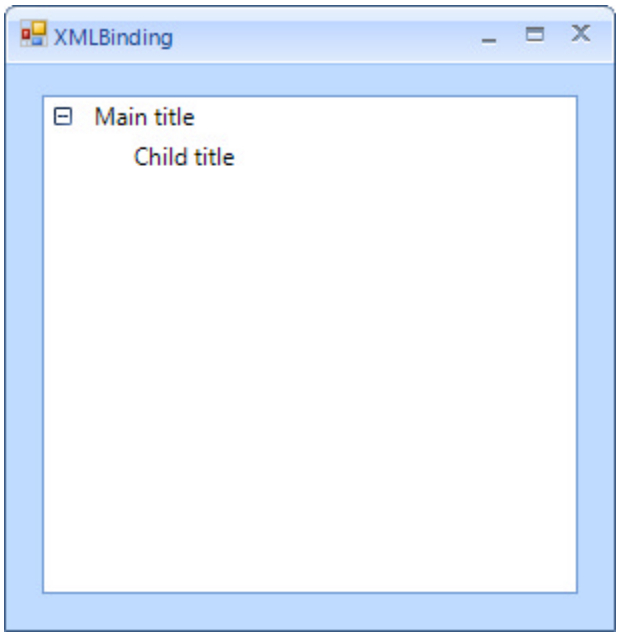
After the DataSet ReadXml() method consumes the XML, the **DataMember** is set to the top level XML element for each item, in this case "<item>". The **Text** property for each tree node is supplied by the **DisplayMember.** The **ParentMember** property is used to make a node the child of a node with a matching **ChildMember**.

[VB] Reading XML into the DataSet

```
Private Sub XMLBinding_Load(sender As Object, e As EventArgs)
 Dim dataSet As New DataSet()
 dataSet.ReadXml("toc.xml")
 Me.radTreeView1.DataMember = "item"
 Me.radTreeView1.DisplayMember = "title"
 Me.radTreeView1.ChildMember = "id"
 Me.radTreeView1.ParentMember = "parentId"
 Me.radTreeView1.DataSource = dataSet
End Sub
```

[C#] Reading XML into the DataSet

```
private void XMLBinding_Load(object sender, EventArgs e)
{
 DataSet dataSet = new DataSet();
 dataSet.ReadXml("toc.xml");
 this.radTreeView1.DataMember = "item";
 this.radTreeView1.DisplayMember = "title";
 this.radTreeView1.ChildMember = "id";
 this.radTreeView1.ParentMember = "parentId";
 this.radTreeView1.DataSource = dataSet;
}
```



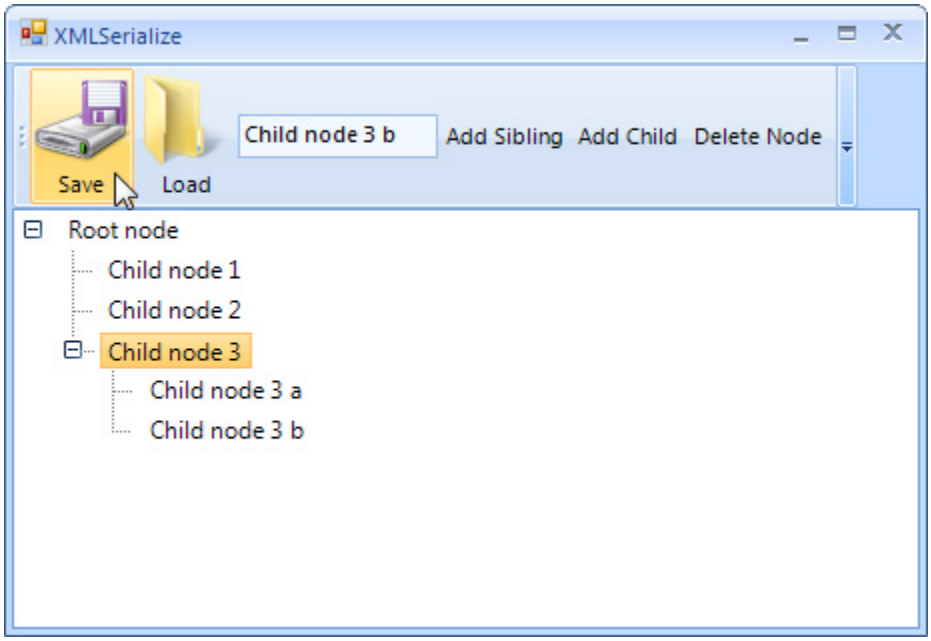
Saving and Loading XML

You can find the complete source for this project at:

\Treeview\<VB|CS>\Databinding

Set Program.cs to start with the form "XMLSerialize".

The contents of a RadTreeView can be serialized as XML directly to a file. In the sample project, nodes can be added to create an arbitrary tree. The Save button calls the RadTreeView **SaveXML()** method. SaveXML() takes either a file path to save your XML directly to disk or a stream that you could use to save the XML in a database.



The saved XML looks something like the example below. Notice that it even retains the current node selection.

```
<?xml version="1.0" encoding="utf-8"?>
<TreeView xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        ShowLines="true" FullRowSelect="false" CausesValidation="true" ThemeName="ControlDefault" BackColor="Transparent">
  <Nodes Name="Root node" Expanded="true" Text="Root node">
    <Nodes Name="Child node 1" Text="Child node 1" />
    <Nodes Name="Child node 2" Text="Child node 2" />
    <Nodes Name="Child node 3" Expanded="true" Text="Child node 3">
      <Nodes Name="Child node 3 a" Text="Child node 3 a" />
      <Nodes Name="Child node 3 b" Text="Child node 3 b" />
    </Nodes>
  </Nodes>
</TreeView>
```

The code to perform the save is using OpenSaveDialog and OpenLoadDialog, but is otherwise a one-liner.

[VB] Load and Save TreeView Structure as XML

```
Private Sub btnSave_Click(sender As Object, e As EventArgs)
 Dim dialog As New SaveFileDialog()
 dialog.Filter = "XML Files|*.xml"
 If dialog.ShowDialog() = DialogResult.OK Then
  radTreeView1.SaveXML(dialog.FileName)
 End If
End Sub
Private Sub btnLoad_Click(sender As Object, e As EventArgs)
 Dim dialog As New OpenFileDialog()
 dialog.Filter = "XML Files|*.xml"
 If dialog.ShowDialog() = DialogResult.OK Then
  radTreeView1.LoadXML(dialog.FileName)
 End If
End Sub
```

[C#] Load and Save TreeView Structure as XML

```
private void btnSave_Click(object sender, EventArgs e)
{
 SaveFileDialog dialog = new SaveFileDialog();
 dialog.Filter = "XML Files|*.xml";
 if (dialog.ShowDialog() == DialogResult.OK)
 {
   radTreeView1.SaveXML(dialog.FileName);
 }
}
private void btnLoad_Click(object sender, EventArgs e)
{
 OpenFileDialog dialog = new OpenFileDialog();
 dialog.Filter = "XML Files|*.xml";
 if (dialog.ShowDialog() == DialogResult.OK)
 {
   radTreeView1.LoadXML(dialog.FileName);
 }
}
```

Saving to a stream requires a stream descendant instance such as MemoryStream. From there you can use the array of bytes from GetBuffer() and send the bytes to a database or other persistence medium.

[VB] Using SaveXML() with a Stream

```
Dim stream As New MemoryStream()
radTreeView1.SaveXML(stream)
RadMessageBox.Show(stream.GetBuffer().Length.ToString())
```

[C#] Using SaveXML() with a Stream

```
MemoryStream stream = new MemoryStream();
radTreeView1.SaveXML(stream);
RadMessageBox.Show(stream.GetBuffer().Length.ToString());
```

You can achieve rapid loading time using two RadTreeView features:

- "Load-on-demand" where large or potentially unknown amounts of data are loaded only when needed.
- Surround batch changes to the tree with BeginUpdate() and EndUpdate() calls to avoid constant repainting.

The Load On Demand feature helps reduce performance and memory costs incurred when all nodes are loaded at one time. To benefit from this feature, you just need to handle the **NodesNeeded** event. This event is fired when you try to expand a node. The event arguments of this event return contain the **Parent** node that you are trying to expand and the **Nodes** collection that you should fill with subnodes. The code snippet below demonstrates how you can handle the NodesNeeded event and load 500 nodes to any expanded node:

[VB] Load on Demand

```
Private Sub radTreeView1_NodesNeeded(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.NodesNeededEventArgs) Handles radTreeView1.NodesNeeded
' measure how long this takes
Dim stopwatch As New Stopwatch()
stopwatch.Start()
Dim count As Integer = 0
' get the data for all child nodes
Dim weatherStations As String() = My.Resources.WeatherStations.Split(New String()
{Constants.vbCrLf}, StringSplitOptions.RemoveEmptyEntries)
For Each station As String In weatherStations
e.Nodes.Add(New RadTreeNode(station))
count += 1
Next
' show performance results
stopwatch.[Stop]()
Dim ts As TimeSpan = stopwatch.Elapsed
lblStatus.Text = [String].Format("{0:00}:{1:00}:{2:00}.{3:00} loading {4} nodes", ts.Hours,
ts.Minutes, ts.Seconds, ts.Milliseconds / 10, count)
Me.radTreeView1.EndUpdate()
End Sub
```

[C#] Load on Demand

```
void radTreeView1_NodesNeeded(object sender, NodesNeededEventArgs e)
{
// measure how long this takes
Stopwatch stopwatch = new Stopwatch();
stopwatch.Start();
int count = 0;
// get the data for all child nodes
string[] weatherStations =
Properties.Resources.WeatherStations.Split(
new string[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
foreach (string station in weatherStations)
{
e.Nodes.Add(new RadTreeNode(station));
count++;
}
// show performance results
stopwatch.Stop();
TimeSpan ts = stopwatch.Elapsed;
```

```
lblStatus.Text = String.Format("{0:00}:{1:00}:{2:00}.{3:00} loading {4} nodes",
ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10, count);
}
```



You can find the complete source for this project at:

\Treeview\<VB|CS>\Databinding

Set Program.cs to start with the form "LoadOnDemand".

The basic structure of the NodesNeeded event handler is the same as the previous example except:

- We are only loading children to the root level nodes and no further.
- We're using a System.Diagnostics StopWatch component to measure the amount of time it takes to build the tree.

[VB] Load On Demand

```
Private Sub radTreeView1_NodeExpandedChanged(sender As Object, e As
Telerik.WinControls.UI.RadTreeViewEventArgs)
 ' measure how long this takes
 Dim stopwatch As New Stopwatch()
 stopwatch.Start()
 Dim count As Integer = 0
 ' the node is expanded, hasn't been loaded yet, and only load the first level
 If e.Node.Expanded AndAlso Not e.Node.LoadedOnDemand AndAlso e.Node.Level = 0 Then
  ' don't repaint till we're done

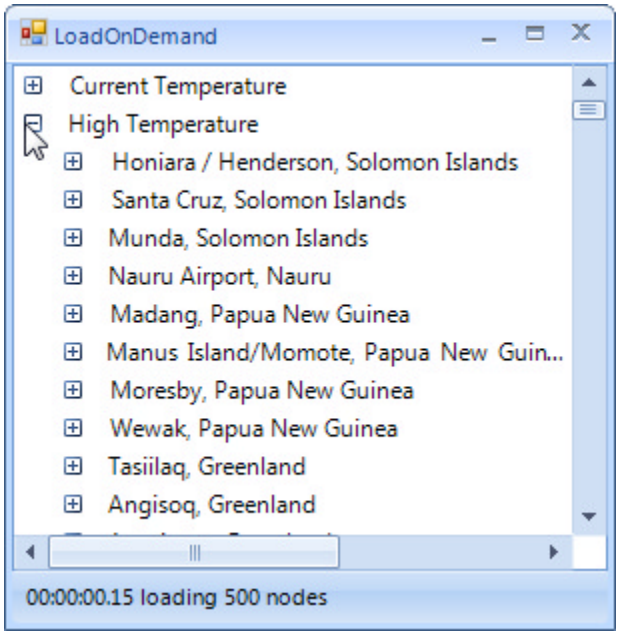
  ' get the data for all child nodes
  Dim weatherStations As String() = Properties.Resources.WeatherStations.Split(New String()
{vbCr & vbLf}, StringSplitOptions.RemoveEmptyEntries)
  For Each station As String In weatherStations
   e.Node.Nodes.Add(station)
   System.Math.Max(System.Threading.Interlocked.Increment(count),count - 1)
  Next
  ' flag the node as having been loaded
  e.Node.LoadedOnDemand = True
   End If
 End If
 ' show performance results
 stopwatch.[Stop]()
 Dim ts As TimeSpan = stopwatch.Elapsed
 lblStatus.Text = [String].Format("{0:00}:{1:00}:{2:00}.{3:00} loading {4} nodes", ts.Hours,
ts.Minutes, ts.Seconds, ts.Milliseconds / 10, count)
End Sub
```

[C#] Load On Demand

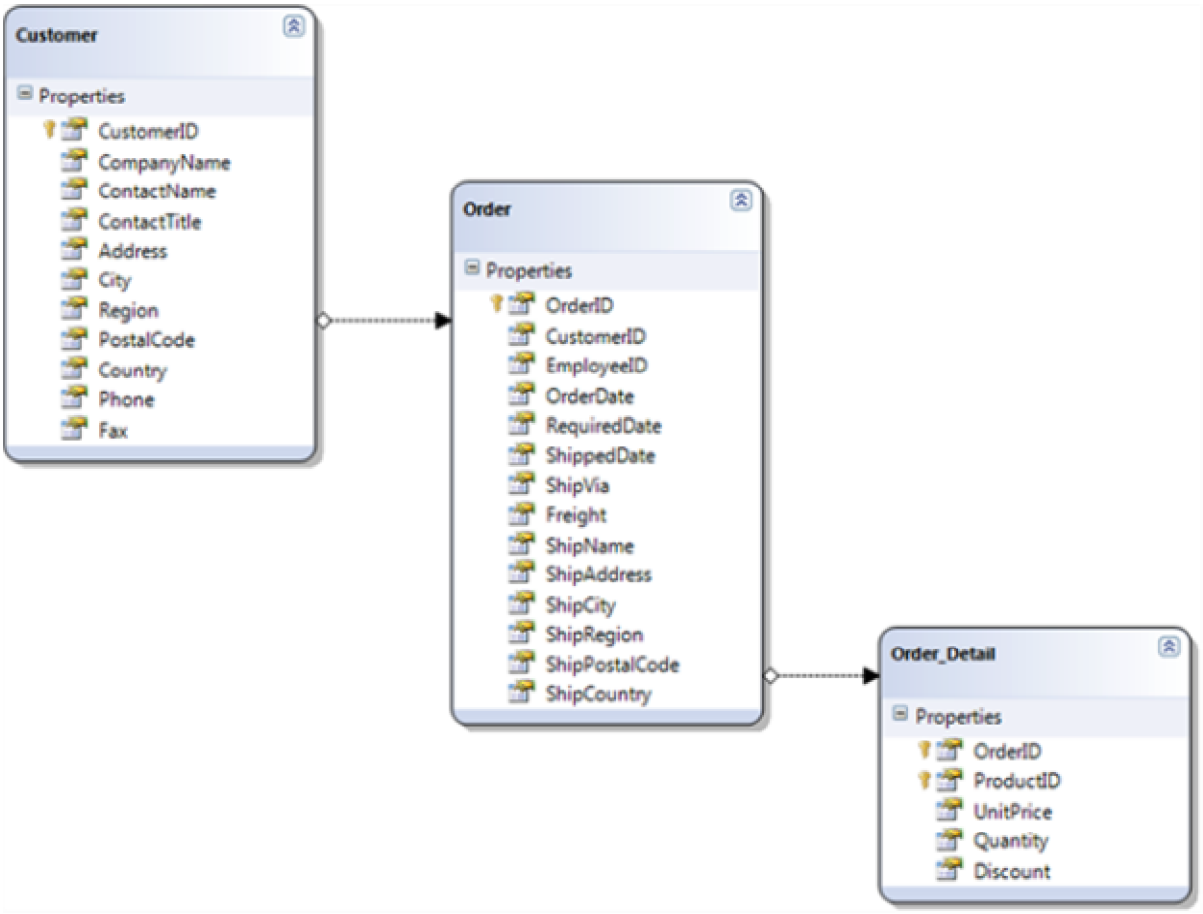
```
private void radTreeView1_NodeExpandedChanged(object sender,
Telerik.WinControls.UI.RadTreeViewEventArgs e)
{
 // measure how long this takes
 Stopwatch stopwatch = new Stopwatch();
 stopwatch.Start();
 int count = 0;
 // the node is expanded, hasn't been loaded yet, and only load the first level
 if (e.Node.Expanded && !e.Node.LoadedOnDemand && e.Node.Level == 0)
```

```
 {
   // don't repaint till we're done
   if (cbUseBeginEndUpdate.ToggleState == ToggleState.On)
     radTreeView1.BeginUpdate();
   // get the data for all child nodes
   string[] weatherStations =
     Properties.Resources.WeatherStations.Split(
     new string[] { "\r\n" }, StringSplitOptions.RemoveEmptyEntries);
   foreach (string station in weatherStations)
   {
     e.Node.Nodes.Add(station);
     count++;
   }
   // flag the node as having been loaded
   e.Node.LoadedOnDemand = true;
   // ok, repaint now
   if (cbUseBeginEndUpdate.ToggleState == ToggleState.On)
     radTreeView1.EndUpdate();
 }
 // show performance results
 stopwatch.Stop();
 TimeSpan ts = stopwatch.Elapsed;
 lblStatus.Text = String.Format("{0:00}:{1:00}:{2:00}.{3:00} loading {4} nodes",
     ts.Hours, ts.Minutes, ts.Seconds, ts.Milliseconds / 10, count);
}
```

When you run the project, also notice that expanding a previously loaded node has no appreciable loading cost.



RadTreeView has the ability to bind to related data of custom objects which have member collections of related objects (ORM generated classes for example).
Consider the following class diagram:



There are three classes named **Customer**, **Order** and **Order_Detail**.

The data relations in the case of custom objects are represented by properties which provide references to collections of related objects. In this case the **Customer** class has a property named **Orders** which is a collection of **Order** objects. This is the first relation. The second relation is the **Order_Details** property in the **Order** class, it offers a reference to a collection of **Order_Details**. Once you have such class composition, it is trivial for RadTreeView to represent it visually in your application.

The two steps that must be done are these:

1. Set the DataSource of RadTreeView to a collection of your root objects (a collection of Customer object in this case):

   [VB] Set RadTreeView data source

   ```
   Dim customers As IEnumerable(Of Customer) = context.GetTable(Of Customer)().ToList()
   Me.RadTreeView1.DataSource = customers
   ```

   [C#] Set RadTreeView data source

   ```
   IEnumerable<Customer> customers = context.GetTable<Customer>().ToList();
   this.radTreeView1.DataSource = customers;
   ```

2. Set DisplayMember corresponding to the DisplayMembers of the different types of objects and set the ChildMember corresponding to the names of the properties that represent the collections of subobjects.

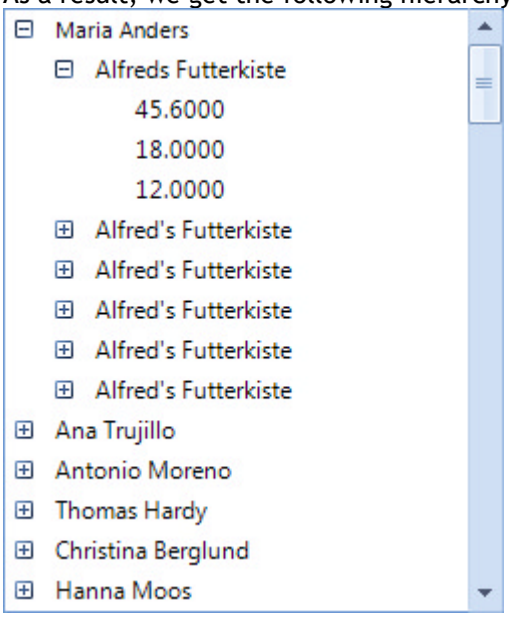
   [VB.NET] Set he Display and Child members

```
Me.RadTreeView1.DisplayMember = "ContactName\ShipName\UnitPrice"
Me.RadTreeView1.ChildMember = "Customers\Orders\Order_Details"
```

[C#] Set he Display and Child members

```
this.radTreeView1.DisplayMember = "ContactName\\ShipName\\UnitPrice";
this.radTreeView1.ChildMember = "Customers\\Orders\\Order_Details";
```

3. As a result, we get the following hierarchy in RadTreeView:



Binding RadTreeView to self referencing data differs form binding to related data in that RadTreeView is bound to a single list instead of multiple related lists.
In order to set the parent-child relation between the records of the data source, we should set the ParentMember and ChildMember properties to the respective fields this data source. If the parent ID for a record does not have a respective value in the child ID field of the records, then that record is considered to have no parents.

The following example demonstrates how to bind RadTreeView to a self referencing DataTable.

[VB.NET] Bind RadTreeView to self referencing data

```
Public Sub New()
    InitializeComponent()


    Me.RadTreeView1.DisplayMember = "name"
    Me.RadTreeView1.ValueMember = "id"
    Me.RadTreeView1.ParentMember = "pid"
    Me.RadTreeView1.DataSource = Me.GetSampleData()
    Me.RadTreeView1.DataMember = "Nodes"

End Sub

Private Function GetSampleData() As DataTable
    Dim dt As New DataTable()

    Dim dc As New DataColumn()
    dc.ColumnName = "id"
```

```
dc.DataType = GetType(Integer)
dt.Columns.Add(dc)

Dim dc1 As New DataColumn()
dc1.ColumnName = "name"
dc1.DataType = GetType(String)
dt.Columns.Add(dc1)

Dim dc2 As New DataColumn()
dc2.ColumnName = "pid"
dc2.DataType = GetType(Integer)
dt.Columns.Add(dc2)

Dim dr As DataRow = dt.NewRow()
dr(0) = 0
dr(1) = "My Computer"
dr(2) = DBNull.Value
dt.Rows.Add(dr)

dr = dt.NewRow()
dr(0) = 1
dr(1) = "C:\"
dr(2) = 0
dt.Rows.Add(dr)

dr = dt.NewRow()
dr(0) = 2
dr(1) = "D:\"
dr(2) = 0
dt.Rows.Add(dr)

dr = dt.NewRow()
dr(0) = 3
dr(1) = "Program Files"
dr(2) = 1
dt.Rows.Add(dr)

dr = dt.NewRow()
dr(0) = 4
dr(1) = "Microsoft"
dr(2) = 3
dt.Rows.Add(dr)

dr = dt.NewRow()
dr(0) = 5
dr(1) = "Telerik"
dr(2) = 3
dt.Rows.Add(dr)

dr = dt.NewRow()
dr(0) = 6
dr(1) = "WINDOWS"
dr(2) = 1
dt.Rows.Add(dr)
```

```
    Return dt
End Function
```

[C#] Bind RadTreeView to self referencing data

```
public BindingToSelfRefData()
{
   InitializeComponent();

   this.radTreeView1.DisplayMember = "name";
   this.radTreeView1.ParentMember = "pid";
   this.radTreeView1.ChildMember = "id";
   this.radTreeView1.DataSource = this.GetSampleData();
}

private DataTable GetSampleData()
{
   DataTable dt = new DataTable();

   DataColumn dc = new DataColumn();
   dc.ColumnName = "id";
   dc.DataType = typeof(int);
   dt.Columns.Add(dc);

   DataColumn dc1 = new DataColumn();
   dc1.ColumnName = "name";
   dc1.DataType = typeof(string);
   dt.Columns.Add(dc1);

   DataColumn dc2 = new DataColumn();
   dc2.ColumnName = "pid";
   dc2.DataType = typeof(int);
   dt.Columns.Add(dc2);

   DataRow dr = dt.NewRow();
   dr[0] = 0;
   dr[1] = "My Computer";
   dr[2] = DBNull.Value;
   dt.Rows.Add(dr);

   dr = dt.NewRow();
   dr[0] = 1;
   dr[1] = @"C:\";
   dr[2] = 0;
   dt.Rows.Add(dr);

   dr = dt.NewRow();
   dr[0] = 2;
   dr[1] = @"D:\";
   dr[2] = 0;
   dt.Rows.Add(dr);

   dr = dt.NewRow();
   dr[0] = 3;
   dr[1] = "Program Files";
   dr[2] = 1;
```

```
    dt.Rows.Add(dr);

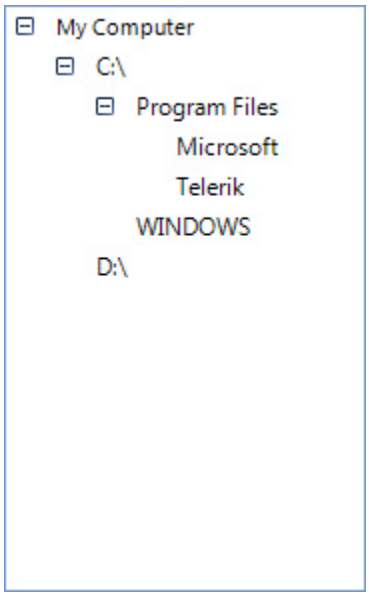
    dr = dt.NewRow();
    dr[0] = 4;
    dr[1] = "Microsoft";
    dr[2] = 3;
    dt.Rows.Add(dr);

    dr = dt.NewRow();
    dr[0] = 5;
    dr[1] = "Telerik";
    dr[2] = 3;
    dt.Rows.Add(dr);

    dr = dt.NewRow();
    dr[0] = 6;
    dr[1] = "WINDOWS";
    dr[2] = 1;
    dt.Rows.Add(dr);

    return dt;
}
```

As a result we get the hierarchy of nodes shown below:



Editing Nodes

By default RadTreeView does not allow node editing. If the **AllowEditing** property is set to true, the user may select a node and either press **F2** to initiate editing or simply double click the node. By default a text editor is invoked and allows the editing of the node label. When the edit process ends the entered value is assigned to the node **Text** property. If the user cancels editing by pressing **Escape** the value is not persisted.

To edit progrmatically, call **BeginEdit()** to initiate editing the selected node and **EndEdit()** to conclude editing. EndEdit() takes a single "cancelEdit" parameter; when cancelEdit is true, changes are abandoned.

You can find the complete source for this project at:



\Treeview\<VB|CS>\Editing

The Editing Lifecycle

Here is an overview of the stages that occur during editing a single node.

A node enters edit +mode

- A node that is being displayed by the RadTreeView control is selected and the user presses the F2 key to bring the node into edit mode.
- The RadTreeView control calls the **BeginEdit()** method and a new editor instance is initialized. It is available publicly through the **ActiveEditor** property in RadTreeView and is associated with the node that is about to be edited.
- The editor fires **EditorInitialized** event which you can handle according to your requirements. This event also provides you with the ability to change the input editor.
- A text box based editor appears for input.

A node is brought out of edit mode

- The Enter key is pressed while the node is in edit mode.
- The editor determines if it wants to handle the keystroke.
- The editor instance performs the action it has defined for the Enter key. Typically this indicates that edit mode should be exited and any changes made during the edit session should be applied to the node Text property.
- In response to the action described in the previous step the **EndEdit()** method is called and the **ValueChanged** event is fired.
- The RadTreeView control sets the node Text property to the string representation of the editor Value property.
- The RadTreeView **Edited** event fires.

The code below shows these properties and events in play. Also notice the MouseDoubleClick event handler that also initiates editing without the user having to press the F2 key.



In some cases you may want to compare new and old values for validation or display purposes. Save off the old value in the **Editing** event handler and get the new value during **ValueValidating**.

[VB] Handling Editing Events

```
Private Sub EditNodes_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)
Handles MyBase.Load
radTreeView1.AllowEdit = True
End Sub
Private _oldNodeText As String
Private Sub radTreeView1_Editing(ByVal sender As Object, ByVal e As
TreeNodeEditingEventArgs) Handles radTreeView1.Editing
_oldNodeText = e.Node.Text
' disallow editing root nodes.
If e.Node.Level = 0 Then
e.Cancel = True
lblStatus.Text = "You cannot edit a root node"
End If
End Sub
```

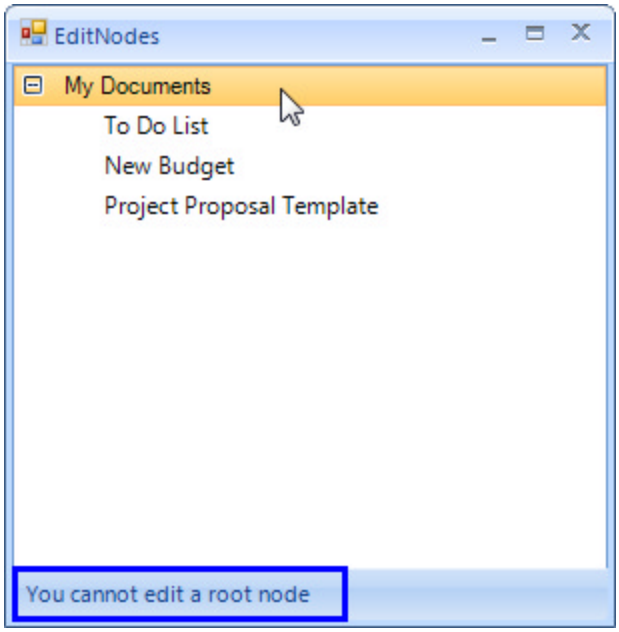
```
Private Sub radTreeView1_ValueValidating(ByVal sender As Object, ByVal e As
TreeNodeValidatingEventArgs) Handles radTreeView1.ValueValidating
Dim newNodeText As String = TryCast(radTreeView1.ActiveEditor.Value, String)
' disallow blank entries
If newNodeText.Equals(String.Empty) Then
e.Cancel = True
lblStatus.Text = "Cannot be blank - Enter a new value"
End If
End Sub
Private Sub radTreeView1_Edited(ByVal sender As Object, ByVal e As TreeNodeEditedEventArgs)
Handles radTreeView1.Edited
' display text before and after editing
Dim newNodeText As String = e.Node.Text
If (Not _oldNodeText.Equals(newNodeText)) Then
lblStatus.Text = String.Format("Node text has changed from {0} to {1}", _oldNodeText,
newNodeText)
End If
End Sub
Private Sub radTreeView1_MouseDoubleClick(ByVal sender As Object, ByVal e As MouseEventArgs)
Handles radTreeView1.MouseDoubleClick
' edit double-clicked node
Dim node As RadTreeNode = radTreeView1.GetNodeAt(e.X, e.Y)
If Not node Is Nothing Then
node.BeginEdit()
End If
End Sub
```

[C#] Handling Editing Events

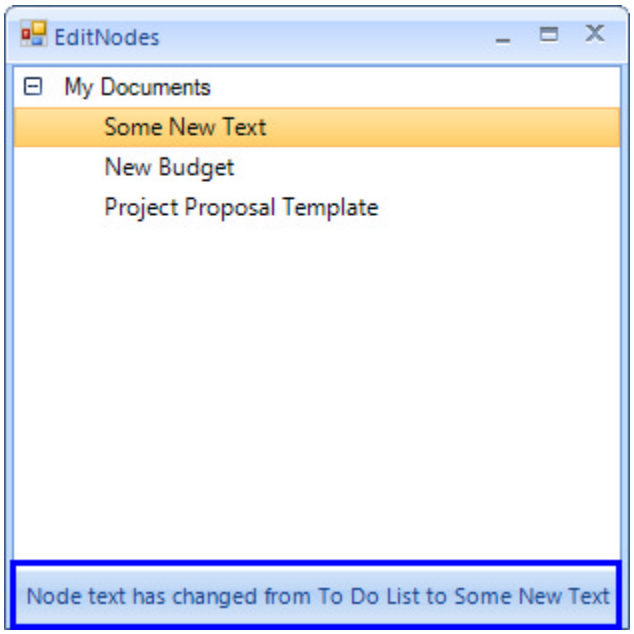
```
private void EditNodes_Load(object sender, EventArgs e)
{
radTreeView1.AllowEdit = true;
}
private string _oldNodeText;
private void radTreeView1_Editing(object sender, TreeNodeEditingEventArgs e)
{
_oldNodeText = e.Node.Text;
// disallow editing root nodes.
if (e.Node.Level == 0)
{
e.Cancel = true;
lblStatus.Text = "You cannot edit a root node";
}
}
private void radTreeView1_ValueValidating(object sender, TreeNodeValidatingEventArgs e)
{
string newNodeText = radTreeView1.ActiveEditor.Value as string;
// disallow blank entries
if (newNodeText.Equals(String.Empty))
{
e.Cancel = true;
lblStatus.Text = "Cannot be blank - Enter a new value";
}
}
private void radTreeView1_Edited(object sender, TreeNodeEditedEventArgs e)
{
```

```
// display text before and after editing
string newNodeText = e.Node.Text;
if (!_oldNodeText.Equals(newNodeText))
{
lblStatus.Text = String.Format("Node text has changed from {0} to {1}",
_oldNodeText, newNodeText);
}
}
private void radTreeView1_MouseDoubleClick(object sender, MouseEventArgs e)
{
// edit double-clicked node
RadTreeNode node = radTreeView1.GetNodeAt(e.X, e.Y);
if (node != null)
{
node.BeginEdit();
}
}
```

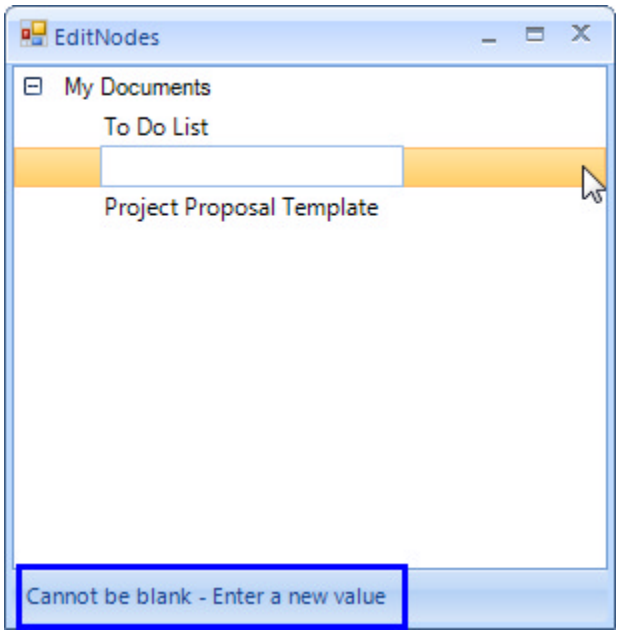
When you run this application and attempt to press F2 with the root node selected, the Editing event handler prevents the edit and displays an error message.



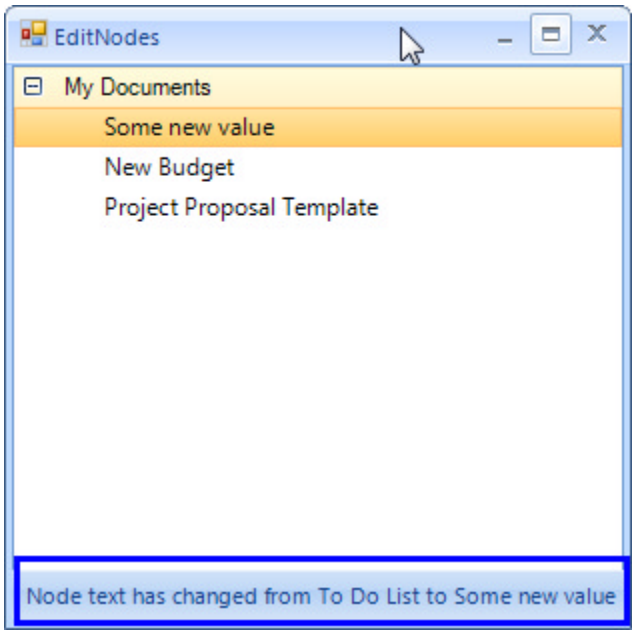
The child nodes below the root can be edited...



...but blank values are disallowed in the **ValueValidating** event. The Cancel argument is set to true, the old value is restored and an error message is displayed in the status strip.



On completion, the ValueValidating event fires and displays both new and old text values.



Drag and Drop

RadTreeView has rich drag-and-drop functionality where you can drag nodes:

- Within the treeview
- From the treeview to another treeview
- From the treeview to another control
- From another control to the treeview

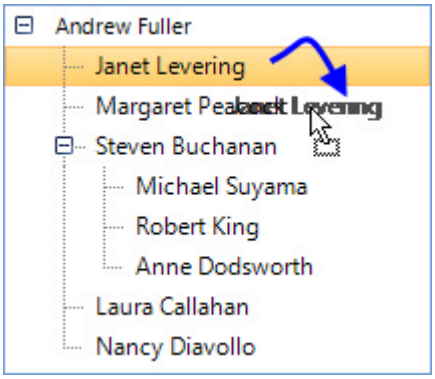
Dragging within the treeview or between treeviews is handled by the RadTreeView control. Dragging to or from other non-telerik controls is handled by standard Windows form control drag methods. Dragging between RadControls for WinForms can be handled by the RadDragDropService.

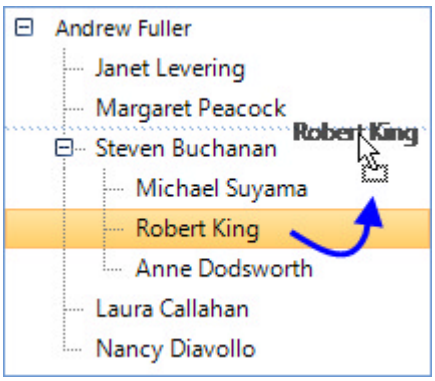
Three key RadTreeView properties enable Drag and Drop functionality:

- **AllowDragDrop** enables the drag and drop functionality for a tree view control. Drag and drop functionality can also be used to reorder nodes within the tree.
- **AllowDrop** when true accepts data that the user drags onto it. This event is used to interact with drag and drop for standard Windows controls.

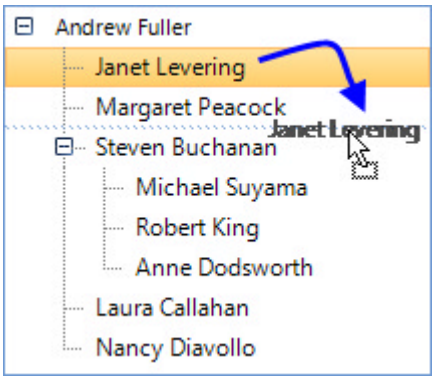
The AllowDragDrop is completely automatic. Turn it on and you get enhanced dragging capabilities within the treeview and between treeviews. The selected node and child nodes are all moved to the target location with user interaction only. No code is required.

The "Janet Leverling" node is dropped onto the "Margaret Peacock" node, so the "Janet Leverling" node becomes a child node of the "Margaret Peacock". This behavior is indicated by the fact that the dotted line which usually demonstrates where a node will be placed is gone.



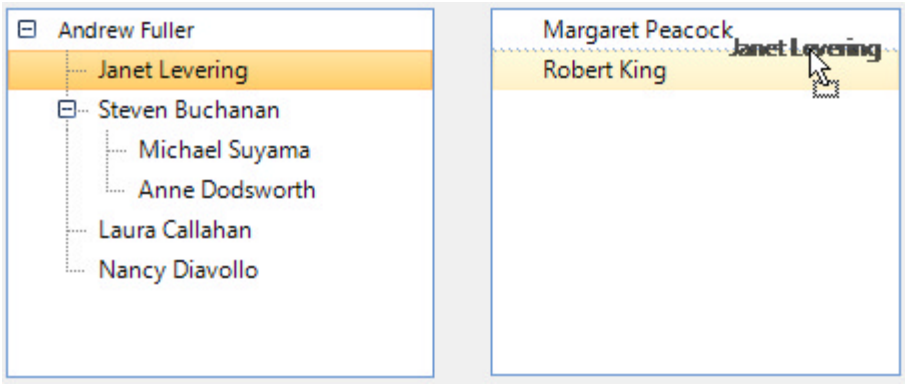


The dragged node "Robert King" will be located as a sibling just above the "Steve Buchanan" node. The new position is indicated by a dotted line.



The dragged node "Janet Leverling" is dropped as a sibling just below the "Margaret Peacock" node. Again this behavior is indicated by a dotted line.

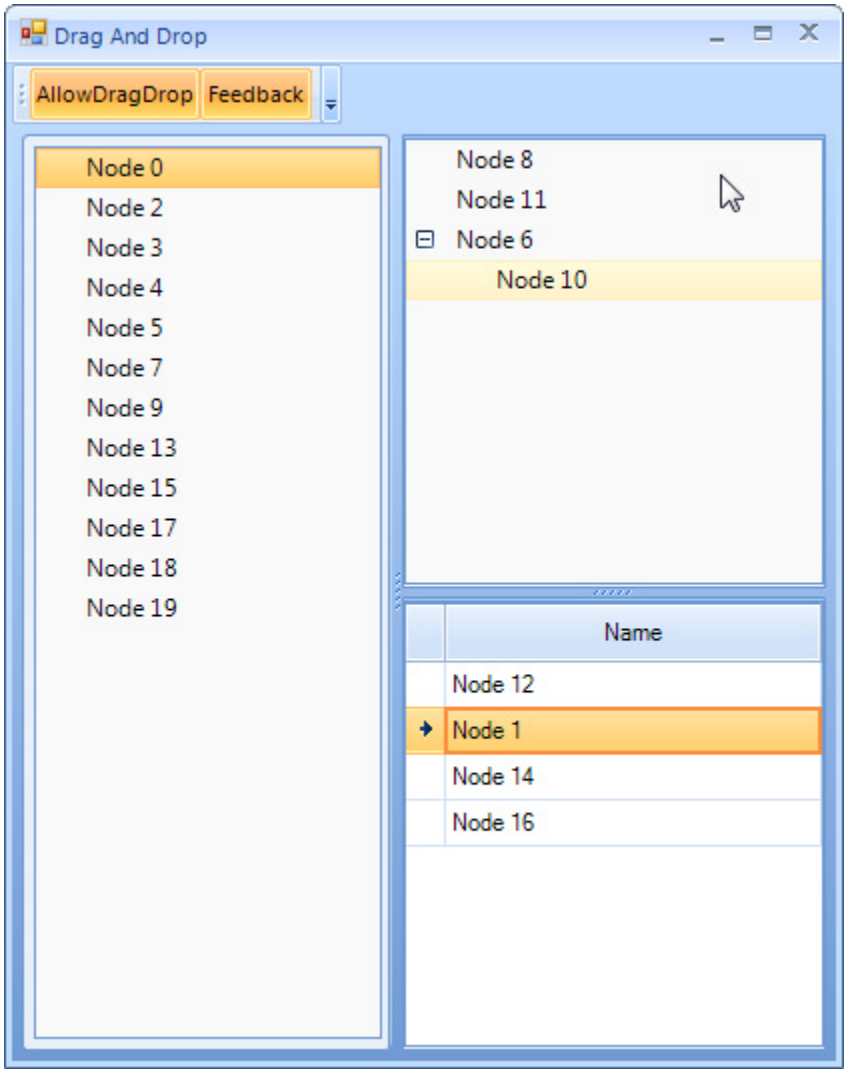
The same visual cues apply when dragging between tree views.



When dragging from the tree view to other controls you can use the standard Windows drag and drop events, or you can use the RadDragDropService when dragging between RadControls for WinForms.

The integrated Drag and drop functionality between controls or in the same RadTreeView control is only possible in **unbound** mode.

You can drag tree nodes to a RadGridView for example. In this example project, There is a RadTreeView on the left of the form, RadTreeView in the upper right of the form and an RadGridView in the lower right of the form. A RadCommandBar at the top of the form lets you toggle drag-and-drop related properties to see their effect on behavior. Nodes can be dragged between the two tree views and from the left tree view to the grid.



You can find the complete source for this project at:

\TreeView\<VB|CS>\DragAndDrop

Form Appearance and Layout For the Drag and Drop Example

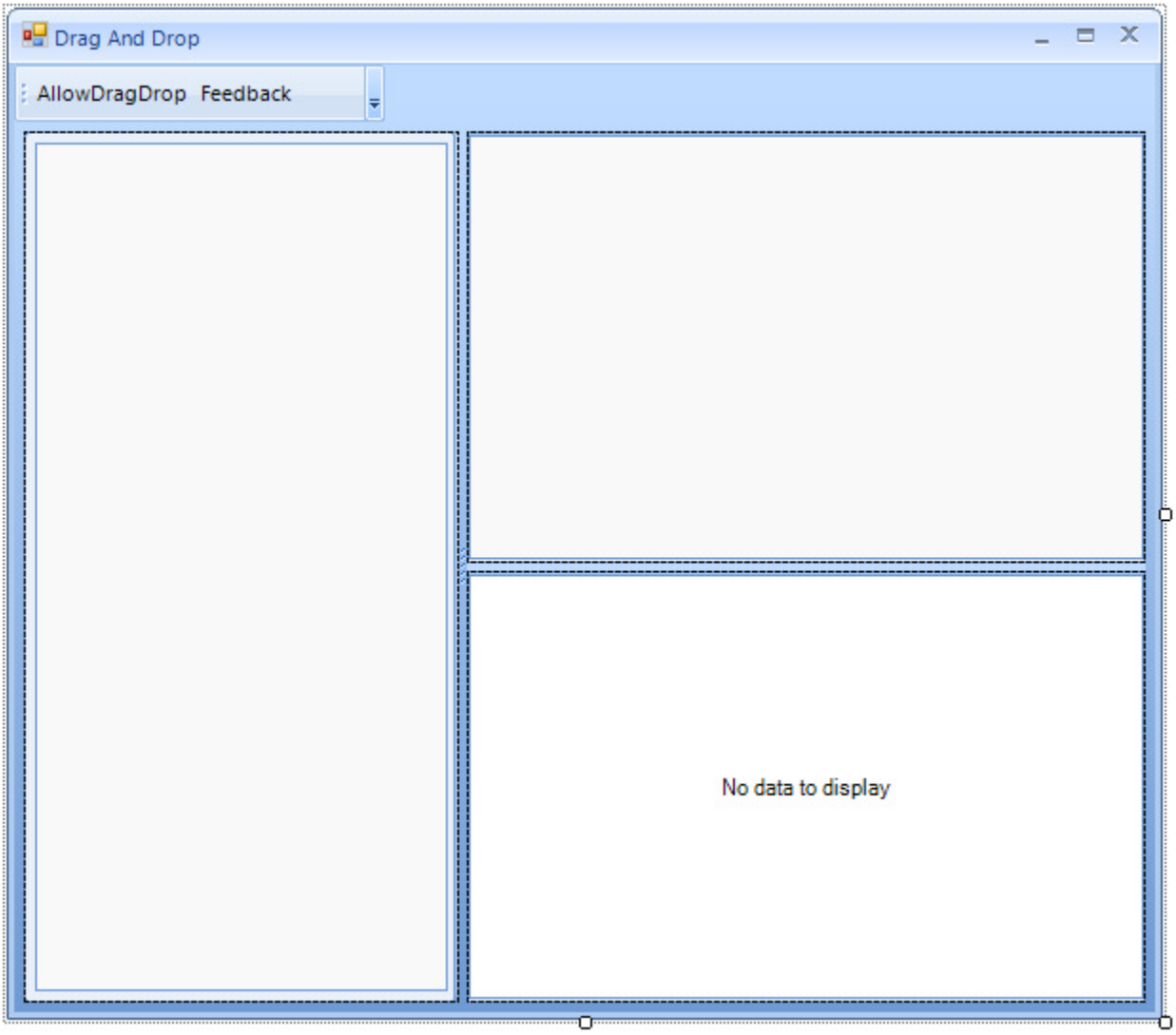
First a few notes on how the form is setup.

Add **RadCommandBar** to the form with two **CommandBarToggleButtons**. Set each button **DrawText** property to true, remove the **Image** setting and set the **Text** properties to

- AllowDragDrop

- Feedback

A **RadDock control** is placed underneath the command bar and its **Dock** property is set to "Fill". The Dock control has a document window on the left and two tool windows in the upper and lower right side. Both tool windows in RadDock has its **CaptionVisible** and **TabStripVisible** set to False and the **TabStripVisible** property of the document tab strip is set to false. The two RadTreeViews and one RadGridView are placed inside the windows. Your form should look like something similar to the following screen shot:



The Form Load Event Handler For the Drag and Drop Example

The form Load event handler fills the tree view on the left named "tvLeft" with some data. The treeview and grid view drag drop service PreviewDragDrop and PreviewDragOver event handlers are hooked up to implement drag-and-drop from the treeview to the grid and vice versa. Remember that the drag-and-drop in the treeview and between treeviews is already enabled by the **AllowDragDrop** property. The RadGridView works in unbound mode simply by adding new GridViewTextBoxColumn to the Columns collection.

[VB] Handling the Form Load Event

```
For i As Integer = 0 To 19
tvLeft.Nodes.Add("Node " & i)
Next
' configure treeview and hook up events
```

```
tvLeft.ExpandAll()
AddHandler tvLeft.TreeViewElement.DragDropService.PreviewDragDrop, AddressOf
DragDropService_PreviewDragDrop
AddHandler tvLeft.TreeViewElement.DragDropService.PreviewDragOver, AddressOf
DragDropService_PreviewDragOver
' create columns and configure grid
Dim nameColumn As New GridViewTextBoxColumn("Name")
grdLowerRight.Columns.Add(nameColumn)
grdLowerRight.AllowRowReorder = True
grdLowerRight.MasterTemplate.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill
Dim gridDragDropService As RadDragDropService = grdLowerRight.GridViewElement.GetService(Of
RadDragDropService)()
AddHandler gridDragDropService.PreviewDragDrop, AddressOf
gridDragDropService_PreviewDragDrop
AddHandler gridDragDropService.PreviewDragOver, AddressOf
gridDragDropService_PreviewDragOver
```

[C#] Handling the Form Load Event

```
private void DragAndDropForm_Load(object sender, EventArgs e)
{
for (int i = 0; i < 20; i++)
{
tvLeft.Nodes.Add("Node " + i);
}
// configure treeview and hook up events
tvLeft.ExpandAll();
tvLeft.TreeViewElement.DragDropService.PreviewDragDrop += new EventHandler<RadDropEventArgs>
(DragDropService_PreviewDragDrop);
tvLeft.TreeViewElement.DragDropService.PreviewDragOver += new
EventHandler<RadDragOverEventArgs>(DragDropService_PreviewDragOver);
// create columns and configure grid
GridViewTextBoxColumn nameColumn = new GridViewTextBoxColumn("Name");
grdLowerRight.Columns.Add(nameColumn);
grdLowerRight.AllowRowReorder = true;
grdLowerRight.MasterTemplate.AutoSizeColumnsMode = GridViewAutoSizeColumnsMode.Fill;
RadDragDropService gridDragDropService =
grdLowerRight.GridViewElement.GetService<RadDragDropService>();
gridDragDropService.PreviewDragDrop += new EventHandler<RadDropEventArgs>
(gridDragDropService_PreviewDragDrop);
gridDragDropService.PreviewDragOver += new EventHandler<RadDragOverEventArgs>
(gridDragDropService_PreviewDragOver);
}
```

Handling the events For the Drag and Drop Example

In the **PreviewDragOver** event handler for both RadTreeView and RadGridView get the **HitTarget** from the event arguments and check if it is the correct destination (GridTableElement or GridDataRowElement for RadGridView and TreeNodeElement for RadTreeView). Once that correct destination is reached, allow it to accept a drop operation through the **CanDrop** property.

The **PreviewDradDrop** event handler is responsible for the transition operation. Here get the dragged row/node element and the target element, insert the dragged object as new row/node at the same index position of the hovered node/row and remove the dragged object from the drag instance. Set the **Handled** property to true in order to notify the service that the dragging process is finished.

[VB] Handling Dragging Events

```
Private Sub gridDragDropService_PreviewDragOver(ByVal sender As Object, ByVal e As
RadDragOverEventArgs)
'if the drop target is TreeNodeElement enable dropping
Dim nodeElement As TreeNodeElement = TryCast(e.HitTarget, TreeNodeElement)
If nodeElement IsNot Nothing Then
e.CanDrop = True
End If
End Sub
Private Sub gridDragDropService_PreviewDragDrop(ByVal sender As Object, ByVal e As
RadDropEventArgs)
Dim rowElement As GridDataRowElement = TryCast(e.DragInstance, GridDataRowElement)
Dim nodeElement As TreeNodeElement = TryCast(e.HitTarget, TreeNodeElement)
If nodeElement IsNot Nothing Then
'insert the node at the place under the currently hovered node
tvLeft.Nodes.Insert(nodeElement.Data.Index, New RadTreeNode(grdLowerRight.CurrentCell.Text))
'remove the dragged row from RadGridView
grdLowerRight.Rows.Remove(rowElement.Data)
e.Handled = True
End If
End Sub
Private Sub DragDropService_PreviewDragOver(ByVal sender As Object, ByVal e As
RadDragOverEventArgs)
Dim tableElement As GridTableElement = TryCast(e.HitTarget, GridTableElement)
Dim dataRow As GridDataRowElement = TryCast(e.HitTarget, GridDataRowElement)
Trace.WriteLine(e.HitTarget)
'if dragged in RadGridView or over a data row in RadGridView, enable dropping
If tableElement IsNot Nothing OrElse dataRow IsNot Nothing Then
e.CanDrop = True
End If
End Sub
Private Sub DragDropService_PreviewDragDrop(ByVal sender As Object, ByVal e As
RadDropEventArgs)
Dim tableElement As GridTableElement = TryCast(e.HitTarget, GridTableElement)
Dim dataRow As GridDataRowElement = TryCast(e.HitTarget, GridDataRowElement)
Dim nodeElement As TreeNodeElement = TryCast(e.DragInstance, TreeNodeElement)
'if dragged over RadGridView add new row to the Rows collection with the node data
If tableElement IsNot Nothing Then
tableElement.ViewTemplate.Rows.Add(nodeElement.Data.Text)
tvLeft.Nodes.Remove(nodeElement.Data)
e.Handled = True
'if dragged over a row, insert new row with the node data at the hovered row place
ElseIf dataRow IsNot Nothing Then
Dim rowInfo As GridViewRowInfo = dataRow.ViewTemplate.Rows.NewRow()
rowInfo.Cells(0).Value = nodeElement.Data.Text
dataRow.ViewTemplate.Rows.Insert(dataRow.RowInfo.Index, rowInfo)
tvLeft.Nodes.Remove(nodeElement.Data)
e.Handled = True
End If
End Sub
```

[C#] Handling Dragging Events

```
void gridDragDropService_PreviewDragOver(object sender, RadDragOverEventArgs e)
{
//if the drop target is TreeNodeElement enable dropping
```

```
TreeNodeElement nodeElement = e.HitTarget as TreeNodeElement;
if (nodeElement != null)
{
e.CanDrop = true;
}
}
void gridDragDropService_PreviewDragDrop(object sender, RadDropEventArgs e)
{
GridDataRowElement rowElement = e.DragInstance as GridDataRowElement;
TreeNodeElement nodeElement = e.HitTarget as TreeNodeElement;
if (nodeElement != null)
{
//insert the node at the place under the currently hovered node
tvLeft.Nodes.Insert(nodeElement.Data.Index, new RadTreeNode
(grdLowerRight.CurrentCell.Text));
//remove the dragged row from RadGridView
grdLowerRight.Rows.Remove(rowElement.Data);
e.Handled = true;
}
}
private void DragDropService_PreviewDragOver(object sender, RadDragOverEventArgs e)
{
GridTableElement tableElement = e.HitTarget as GridTableElement;
GridDataRowElement dataRow = e.HitTarget as GridDataRowElement;
Trace.WriteLine(e.HitTarget);
//if dragged in RadGridView or over a data row in RadGridView, enable dropping
if (tableElement != null || dataRow != null)
{
e.CanDrop = true;
}
}
private void DragDropService_PreviewDragDrop(object sender, RadDropEventArgs e)
{
GridTableElement tableElement = e.HitTarget as GridTableElement;
GridDataRowElement dataRow = e.HitTarget as GridDataRowElement;
TreeNodeElement nodeElement = e.DragInstance as TreeNodeElement;
///if dragged over RadGridView add new row to the Rows collection with the node data
if (tableElement != null)
{
tableElement.ViewTemplate.Rows.Add(nodeElement.Data.Text);
tvLeft.Nodes.Remove(nodeElement.Data);
e.Handled = true;
}
//if dragged over a row, insert new row with the node data at the hovered row place
else if (dataRow != null)
{
GridViewRowInfo rowInfo = dataRow.ViewTemplate.Rows.NewRow();
rowInfo.Cells[0].Value = nodeElement.Data.Text;
dataRow.ViewTemplate.Rows.Insert(dataRow.RowInfo.Index, rowInfo);
tvLeft.Nodes.Remove(nodeElement.Data);
e.Handled = true;
}
}
```

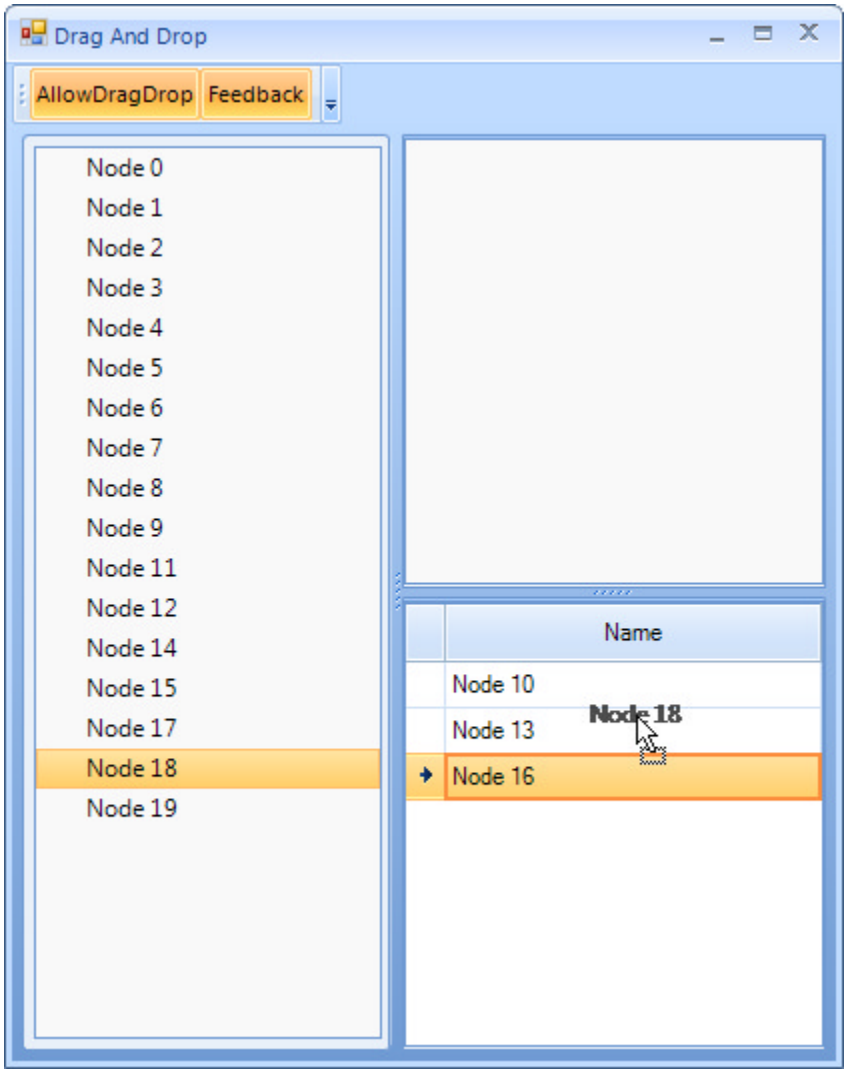
CommandBarToggleButtons Event Handlers For the Drag and Drop Example

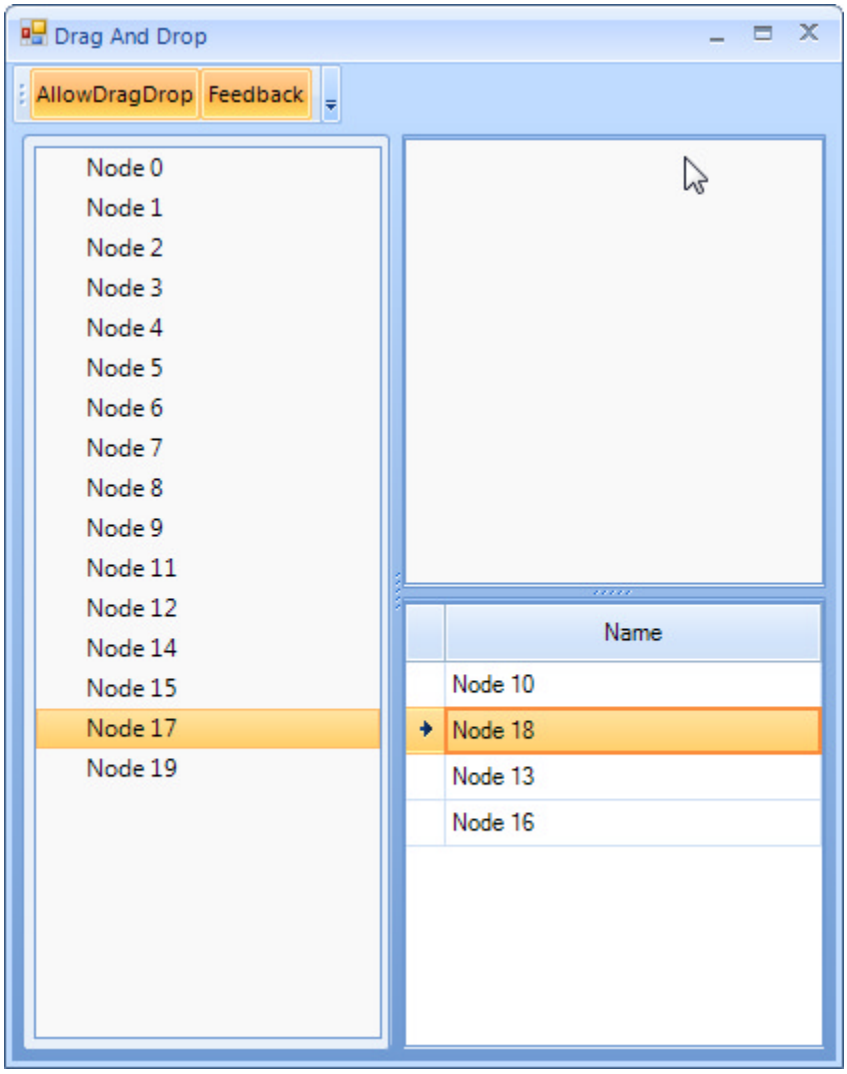
[VB] Handling ToggleButtons ToggleStateChanged Events

```
Private Sub btnAllowDragDrop_ToggleStateChanged(ByVal sender As System.Object, ByVal args As
Telerik.WinControls.UI.StateChangedEventArgs) Handles btnAllowDragDrop.ToggleStateChanged
tvLeft.AllowDragDrop = Convert.ToBoolean(args.ToggleState)
tvTopRight.AllowDragDrop = Convert.ToBoolean(args.ToggleState)
End Sub
Private Sub btnFeedback_ToggleStateChanged(ByVal sender As System.Object, ByVal args As
Telerik.WinControls.UI.StateChangedEventArgs) Handles btnFeedback.ToggleStateChanged
tvLeft.ShowDragHint = Convert.ToBoolean(args.ToggleState)
tvLeft.ShowDropHint = Convert.ToBoolean(args.ToggleState)
End Sub
```

[C#] Handling ToggleButtons ToggleStateChanged Events

```
private void commandBarToggleButton1_ToggleStateChanged(object sender, StateChangedEventArgs
args)
{
tvLeft.AllowDragDrop = Convert.ToBoolean(args.ToggleState);
tvTopRight.AllowDragDrop = Convert.ToBoolean(args.ToggleState);
}
private void commandBarToggleButton3_ToggleStateChanged(object sender, StateChangedEventArgs
args)
{
tvLeft.ShowDragHint = Convert.ToBoolean(args.ToggleState);
tvLeft.ShowDropHint = Convert.ToBoolean(args.ToggleState);
}
```


Drag And Drop
AllowDragDrop
Feedback
Node 0
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Node 8
Node 9
Node 11
Node 12
Node 14
Node 15
Node 17
Node 18
Node 19
Name
Node 10
Node 13
Node 16
Node 18



Dragging From Another Control to the RadTreeView

For this purpose we are going to use the standard Windows Forms drag and drop functionality. **AllowDrop** enables Control descendants to accept data dropped onto it to set the tvTopRight **AllowDrop** to true. This event can be used to allow interaction between standard Windows controls and RadTreeView. For example, to drag from a TextBox to the tree view.

For the purpose of this example we are going to extend a bit the previous code. Add CommandBarTextBox to RadCommandBar and set its **Text** to "" and its MinSize property to 120,0.

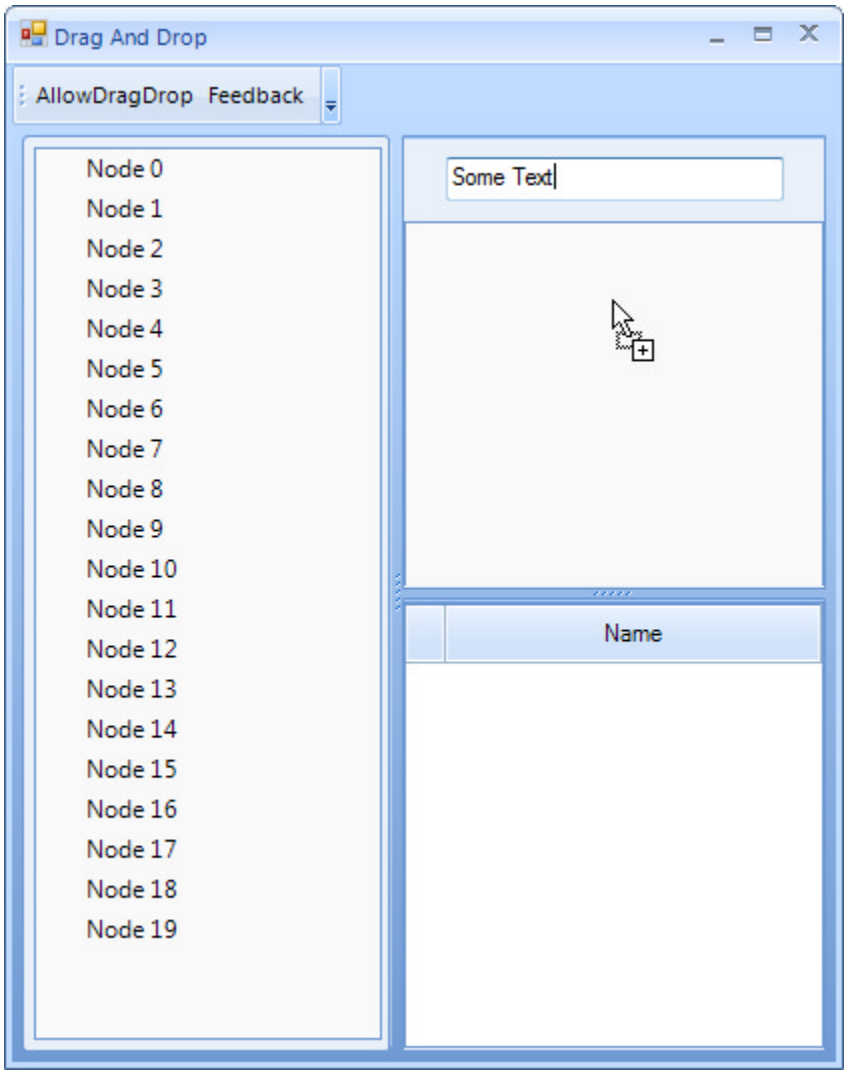
[VB] Dragging to RadTreeView

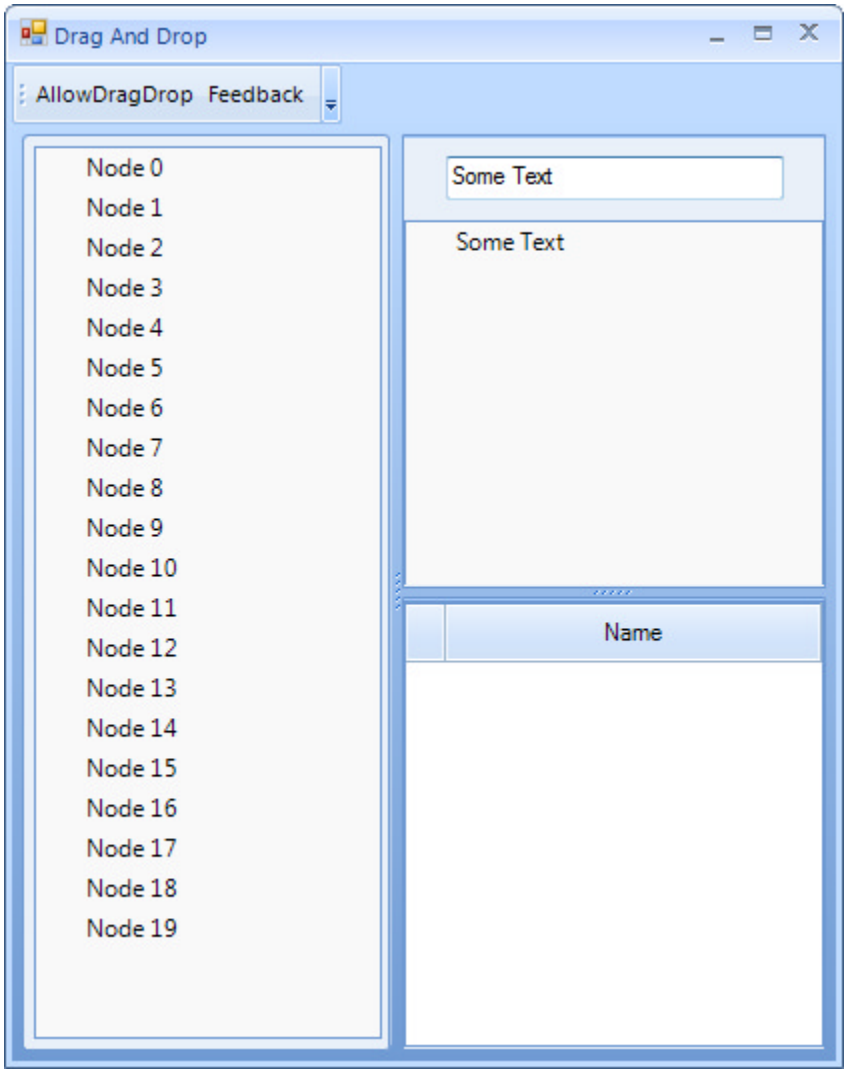
```
Private Sub TextBox1_MouseDown(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.MouseEventArgs) Handles TextBox1.MouseDown
TextBox1.DoDragDrop(TextBox1.Text, DragDropEffects.Copy Or DragDropEffects.Move)
End Sub
Private Sub tvTopRight_DragEnter(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles tvTopRight.DragEnter
If e.Data.GetDataPresent(DataFormats.Text) Then
e.Effect = DragDropEffects.Copy
Else
e.Effect = DragDropEffects.None
End If
```

```
End Sub
Private Sub tvTopRight_DragDrop(ByVal sender As System.Object, ByVal e As
System.Windows.Forms.DragEventArgs) Handles tvTopRight.DragDrop
Dim p As Point = tvTopRight.PointToClient(New Point(e.X, e.Y))
Dim hoverNode As RadTreeNode = tvTopRight.GetNodeAt(p.X, p.Y)
If hoverNode Is Nothing Then
tvTopRight.Nodes.Add(e.Data.GetData(DataFormats.Text).ToString())
Return
End If
hoverNode.Nodes.Add(e.Data.GetData(DataFormats.Text).ToString())
End Sub
```

[C#] Dragging to RadTreeView

```
private void textBox1_MouseDown(object sender, MouseEventArgs e)
{
textBox1.DoDragDrop(textBox1.Text, DragDropEffects.Copy | DragDropEffects.Move);
}
private void tvTopRight_DragEnter(object sender, DragEventArgs e)
{
if (e.Data.GetDataPresent(DataFormats.Text))
{
e.Effect = DragDropEffects.Copy;
}
else
{
e.Effect = DragDropEffects.None;
}
}
private void tvTopRight_DragDrop(object sender, DragEventArgs e)
{
Point p = tvTopRight.PointToClient(new Point(e.X, e.Y));
RadTreeNode hoverNode = tvTopRight.GetNodeAt(p.X, p.Y);
if (hoverNode == null)
{
tvTopRight.Nodes.Add(e.Data.GetData(DataFormats.Text).ToString());
return;
}
hoverNode.Nodes.Add(e.Data.GetData(DataFormats.Text).ToString());
}
```

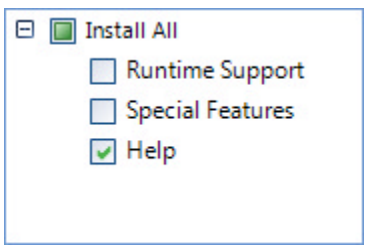

Drag And Drop
AllowDragDrop
Feedback
Node 0
Node 1
Node 2
Node 3
Node 4
Node 5
Node 6
Node 7
Node 8
Node 9
Node 10
Node 11
Node 12
Node 13
Node 14
Node 15
Node 16
Node 17
Node 18
Node 19
Some Text
Name



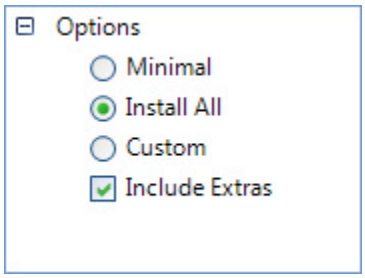
Checkboxes

Set **CheckBoxes** to True to display checkboxes or radio buttons next to each node in the tree. To check a node programmatically, use the **CheckState** property of RadTreeNode. The **NodeCheckChanged** event will fire whenever the user clicks a checkbox.

Like RadCheckBox, "tri-state" checkboxes have checked/un-checked/indeterminate states. To get this behavior, set the tree view **TriStateMode** property to True. The "Install All" checkbox in the screenshot is displayed in an indeterminate state because the child "Help" node is unchecked.



Check boxes and radio buttons can be mixed in a single RadTreeNodeCollection. The RadTreeNode property that control the type of option element to be shown (check box or radio button) is **CheckType**.



[VB] Setting the CheckType

```
' display checkboxes next to nodes
radTreeView1.CheckBoxes = True
' create a root node and set its CheckType to None
Dim root As New RadTreeNode("Options")
root.Expanded = True
root.CheckType = CheckType.None
' create three nodes with radio buttons and select the second node
Dim node1 As New RadTreeNode("Minimal")
node1.CheckType = CheckType.RadioButton
Dim node2 As New RadTreeNode("Install All")
node2.CheckType = CheckType.RadioButton
node2.CheckState = Telerik.WinControls.Enumerations.ToggleState.[On]
Dim node3 As New RadTreeNode("Custom")
node3.CheckType = CheckType.RadioButton
' add a checkbox node and select it
Dim node4 As New RadTreeNode("Include Extras")
node4.CheckType = CheckType.CheckBox
node4.CheckState = Telerik.WinControls.Enumerations.ToggleState.[On]
' add nodes to the root node
root.Nodes.Add(node1)
root.Nodes.Add(node2)
root.Nodes.Add(node3)
root.Nodes.Add(node4)
' add root to the treeview nodes
radTreeView1.Nodes.Add(root)
```

[C#] Setting the ChecktType

```
// display checkboxes next to nodes
radTreeView1.CheckBoxes = true;
// create a root node and set its CheckType to None
RadTreeNode root = new RadTreeNode("Options");
root.Expanded = true;
root.CheckType = CheckType.None;
// create three nodes with radio buttons and select the second node
RadTreeNode node1 = new RadTreeNode("Minimal");
node1.CheckType = CheckType.RadioButton;
RadTreeNode node2 = new RadTreeNode("Install All");
node2.CheckType = CheckType.RadioButton;
node2.CheckState = Telerik.WinControls.Enumerations.ToggleState.On;
RadTreeNode node3 = new RadTreeNode("Custom");
node3.CheckType = CheckType.RadioButton;
// add a checkbox node and select it
RadTreeNode node4 = new RadTreeNode("Include Extras");
node4.CheckType = CheckType.CheckBox;
node4.CheckState = Telerik.WinControls.Enumerations.ToggleState.On;
```

```
// add nodes to the root node
root.Nodes.Add(node1);
root.Nodes.Add(node2);
root.Nodes.Add(node3);
root.Nodes.Add(node4);
// add root to the treeview nodes
radTreeView1.Nodes.Add(root);
```

23.6 Summary

In this chapter you learned how to display, navigate and manage hierarchical data using the RadTreeView and RadBreadCrumb controls. You learned how to add, remove and move nodes programmatically, how to iterate nodes and how to locate nodes. You learned how to handle node editing. You saw how to implement drag and drop within the RadTreeView, how to drag and drop to other tree views and how to drag from the treeview to other controls. You learned how to attach context menus to particular nodes. You also learned about RadTreeView specific databinding issues including binding to self referencing data, binding to related data, writing to and from XML and load-on-demand.

24 Wizard

24.1 Objectives

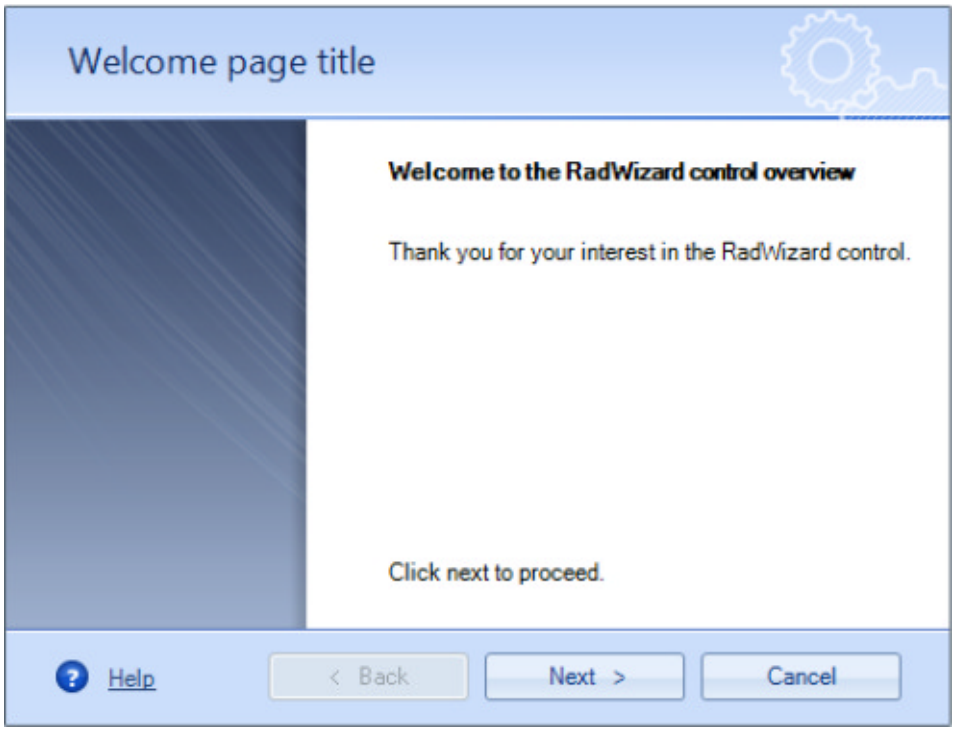
- Learn how to create applications with RazWizard.
- Learn the basic functionalityes of the control.
- Explore the ways of customization.
- Create s sample installation form.

24.2 Introduction

The **RadWizard** control will help you differentiate a complex process into separate steps and provide your users with the ability to govern the process upon their decisions. It is useful for creating installation, registration and other types of wizards.

Each **WizardPage** contains panel where you can host the desired controls needed to create the required functionality.

RadWizard supports both *Wizard 97* and *Wizard Aero* specifications. The control provides the developers with predesigned Welcome, Completion and Internal pages. Follows is an example of a RadWizard Welcome page:



RadWizardElement encapsulates the UI representation and functionality of **RadWizard**. This element initializes the view of the control:

- **Wizard97View** for the **Wizard 97 mode**.
- **WizardAeroView** for **Wizard Aero mode**.

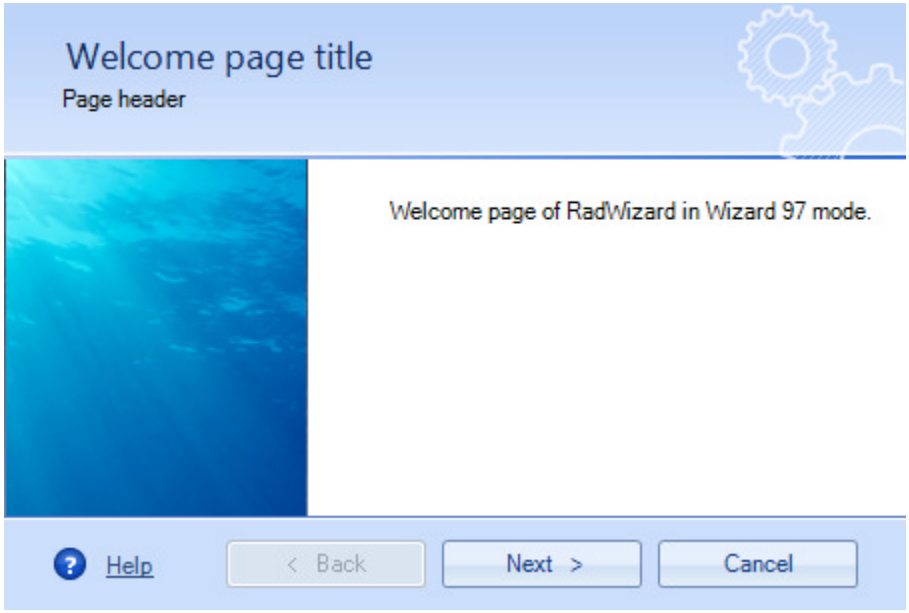
The view contains the following elements:

- **Pages collection** – collection which contains Internal, Welcome and Completion RadWizard pages.

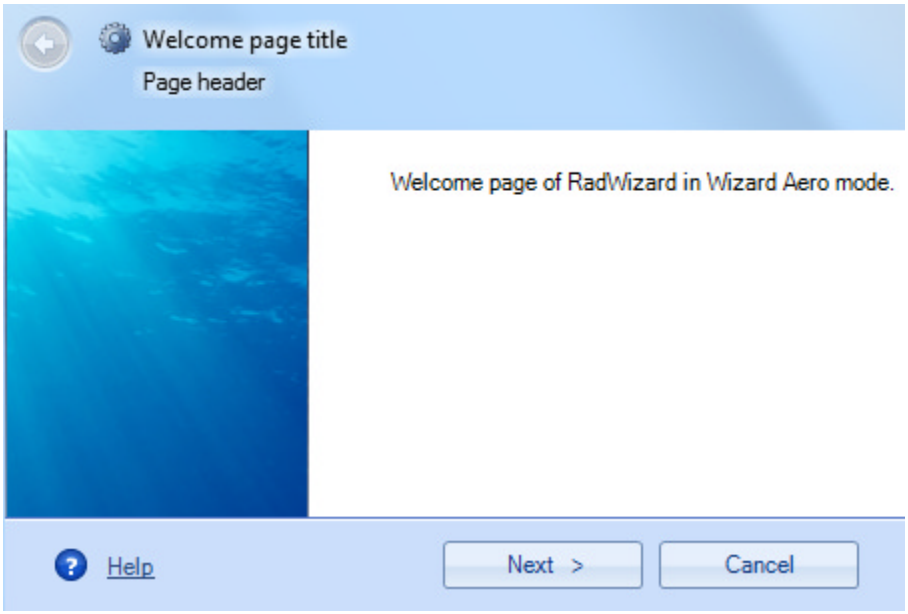
- **Page header** – element which is located above each page and contains elements for title text, header text and page icon.
- **Command area** – element which is located below each page and contains command button – Back (Wizard 97), Next, Cancel, Finish, and Help.
- **Welcome image** – element which contains the image of the Welcome page.
- **Completion image** - element which contains the image of the Completion page.
- **Top element** – element which contains the Back button of Wizard Aero view.

On the screenshots below, you can see the Welcome pages for both wizard mode with the described elements in them:

Wizard97



Wizzard Aero



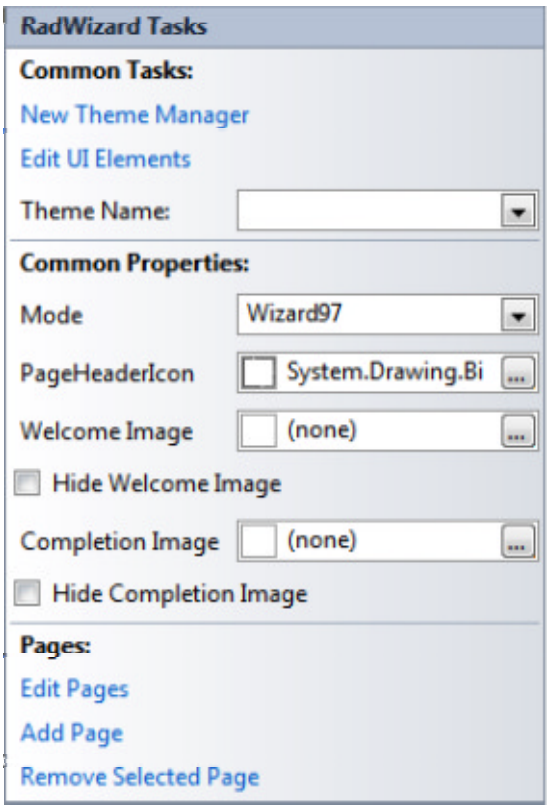
24.3 Using the Design Time Interface

RadWizard provides handful Smart Tag menu and Pages Collection Editor in design time.

The Smart Tag contains options for defining the following RadWizard properties:

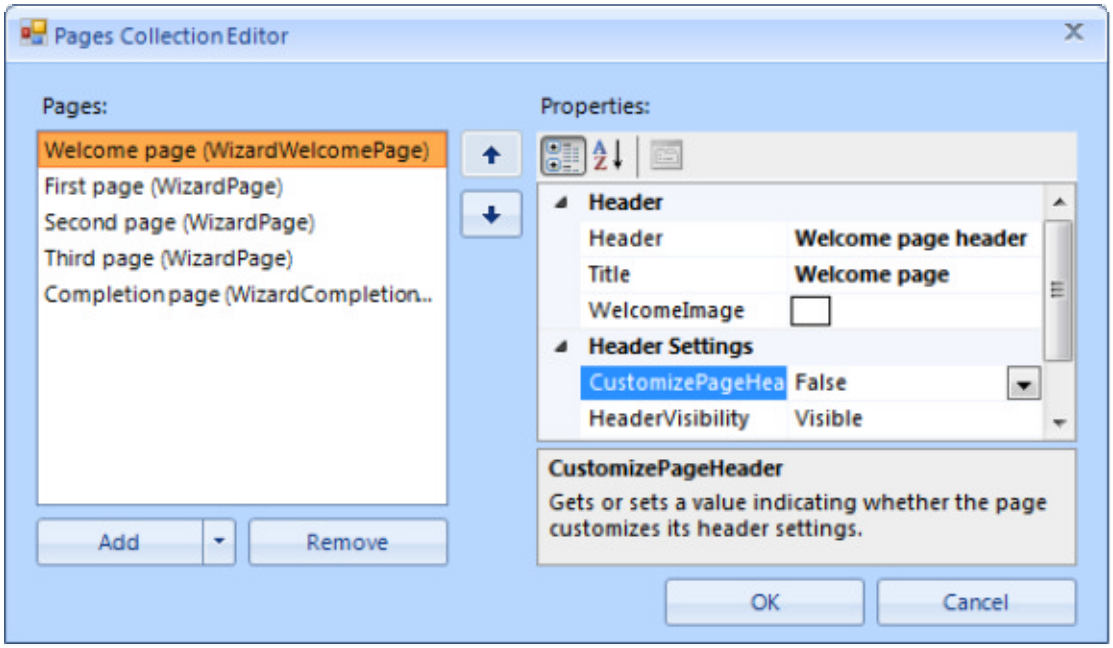
- **Mode**
- **Page header icon**
- **Welcome image**
- **Completion image**

The Smart tag also allows you to add and remove Internal pages.



Pages Collection Editor allows you to:

- Add **Internal**, **Welcome** and **Completion** pages
- Remove pages
- Rearrange pages
- Customize page properties



24.4 Getting Started

In this article, we will create s simple application with RadWizard in order to demonstrate the customization and functionality abilities of the control.

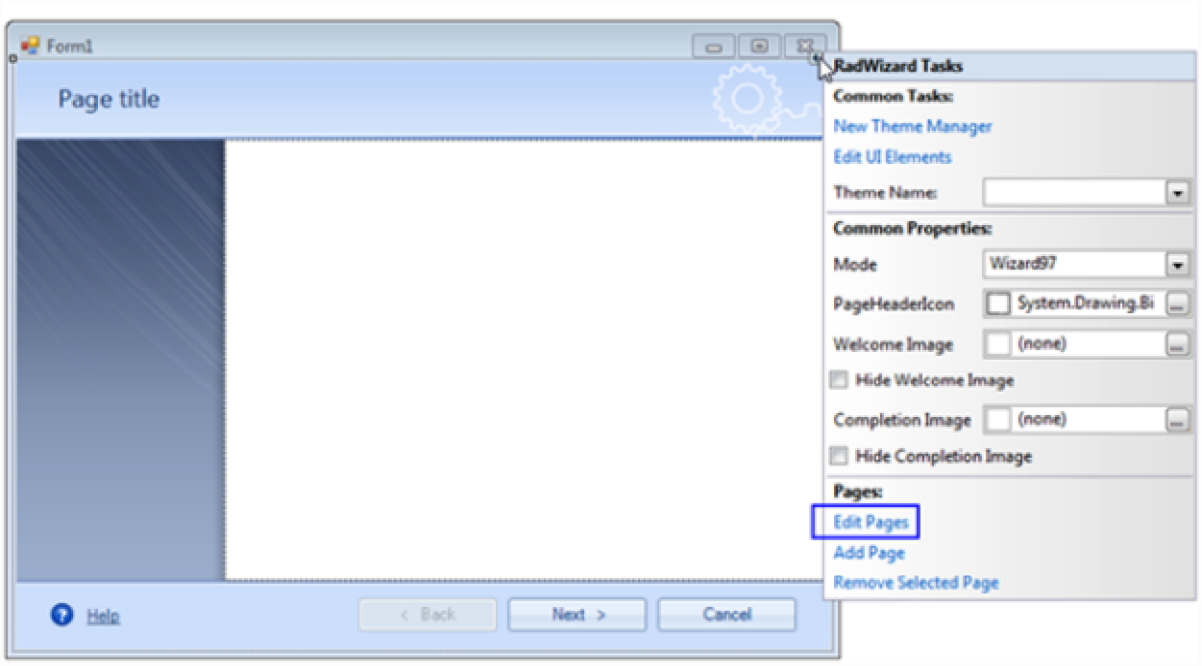


You can find the complete source for this project at:

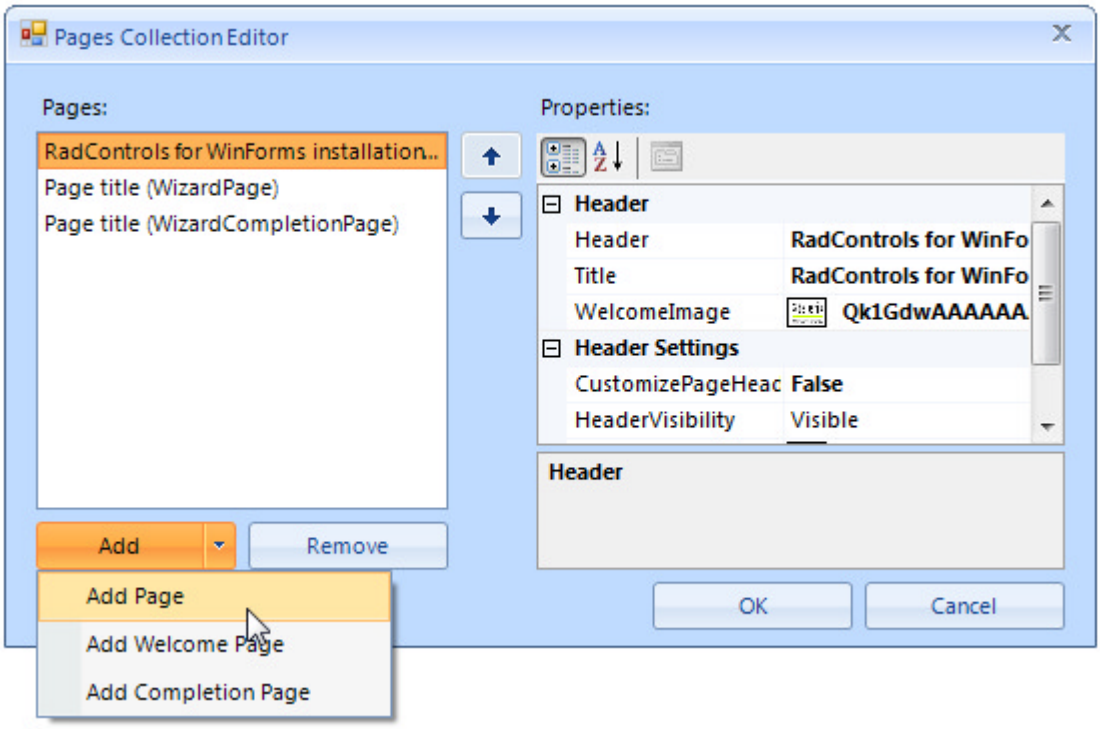
\Wizard\<VB|CS>\GettingStarted

Create a new Windows Forms application.

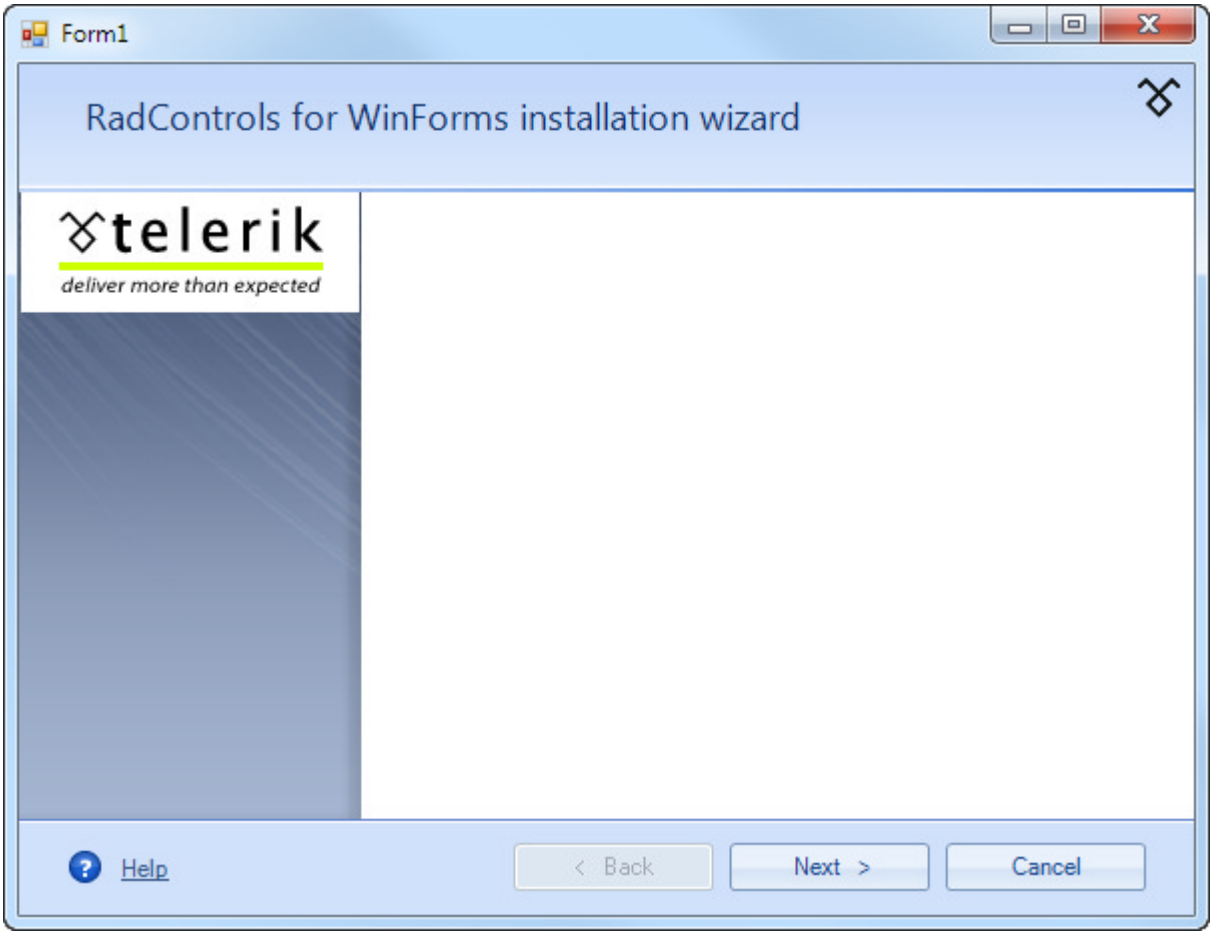
2. Drag RadWizard control from the toolbox to the form.
3. Click the Smart Tag of RadWizard and select Edit Pages option to open the Pages Collection Editor.



4. Set the following properties to the Welcome page:
   - **Title** = "RadControls for WinForms installation wizard"
   - **WelcomeImage** = some appropriate image
   - **Icon** = some appropriate image
   - **CustomizePageHeader** = true
   - **Header** = ""
5. One Wizard page is added by default. Click the Add drop down button and use the "Add Page" option to add two more pages.



6. On the first WizardPage set the title to "Choose installation type", on the second one to "Choose components to install" and the third to "Installation progress" . Remove the header settings for these pages.
7. Set the Title property for the WizardCompletionPage to "Installation Successful" and click OK to close the collection editor.
8. At this point your application should like like this.



9. Double click the form to create Load event handler. Center the image of the wellcome page by setting the WelcomeImageLayout of RadWizard control.

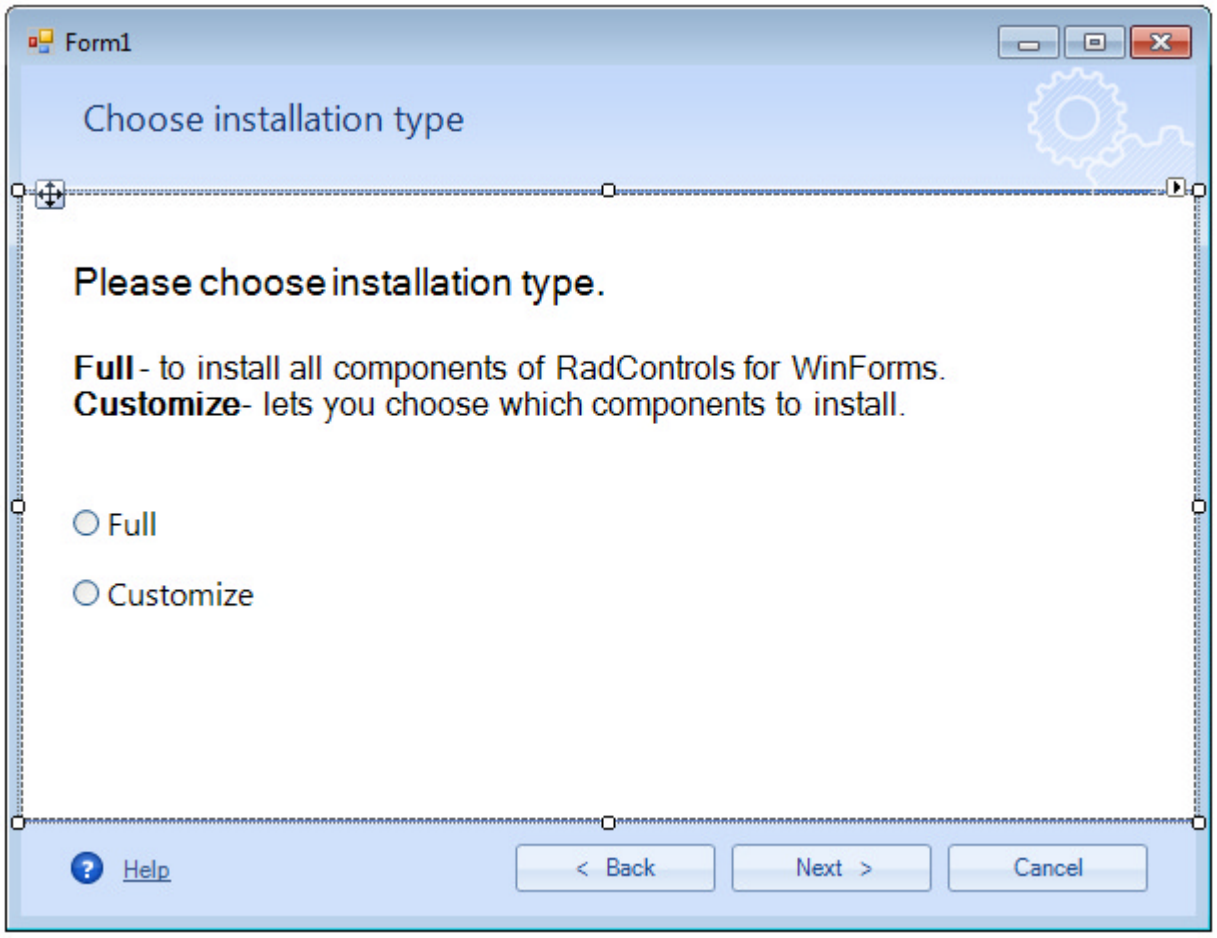
   [VB] Set the image alignment

   ```
   RadWizard1.WelcomeImageLayout = ImageLayout.Center
   ```

   [C#] Set the image alignment

   ```
   radWizard1.WelcomeImageLayout = ImageLayout.Center;
   ```

10. Go back to design time and add RadLabel with Text "<html> Wellcome, Thank you for choosing Telerik RadControls for WinForms. </html>" to the welcome page. Set the MinSize to 0,50 and the font size to 12 pt.
11. Click next button to open the first WizardPage. Add two RadRadioButtons to the panel and RadLabel above them.
12. Set the following properties of RadLabel:
    - **Text** = "<html> Please choose installation type. Full - to install all components of RadControls for WinForms. Customize - lets you choose which components to install. </html>"
    - **MinSize** = 0,100
13. Set the buttons **Text** to "Full" and "Customize" and the font size to 12pt.



14. Click next to navigate to the second WizardPage. Add RadLabel and eight RadCheckBoxes. Set the label font to 14 pt and the Text for the controls as follows:

    - **radLabel1** = "Select features to install"
    - **radCheckBox1** = "RadControls for WinForms"
    - **radCheckBox2** = "Components"
    - **radCheckBox3** = "Demos"
    - **radCheckBox4** = "Visual Studio Extensions"
    - **radCheckBox5** = "Visual Studio 2005"
    - **radCheckBox6** = "Visual Studio 2008"
    - **radCheckBox7** = "Visual Studio 2010"
    - **radCheckBox8** = "Documentation"

15. Click next to navigate to the next page. Add RadLabel with Text "Installation progress" and font size 14 pt, and RadWaitingBar.
16. On the last page add label with text "<html> Congratulations, RadControls for WinForms is successfully installed on your computer. </html>".
17. The wizard layout is ready now all that is left is to add the desired functionality to it. Lets handle the Help event of RadWizzard where we are going to open a webpage and the SelectedPageChanged event of the control where we will start the waiting bar, when needed. Additionally, we can handle the Cancel event of RadWizard to stop the application when the user wants and the Finish event to stop the application when

the process if finished.

[VB] Add functionality

```
Private Sub radWizard1_Help(sender As Object, e As EventArgs)
        Process.Start("http://www.telerik.com")
End Sub

Private Sub radWizard1_SelectedPageChanged(sender As Object, e As
Telerik.WinControls.UI.SelectedPageChangedEventArgs)
        If e.SelectedPage = wizardPage3 Then
                radWaitingBar1.StartWaiting()
        Else
                radWaitingBar1.StopWaiting()
        End If
End Sub



Private Sub radWizard1_Cancel(ByVal sender As Object, ByVal e As EventArgs)If
RadMessageBox.Show("Are you sure that you want to stop the installation?", "",
MessageBoxButtons.YesNo) = DialogResult.Yes ThenApplication.[Exit]()End IfEnd Sub Private
Sub radWizard1_Finish(ByVal sender As Object, ByVal e As EventArgs)Application.[Exit]()E
Sub
```

[C#] Add functionality

```
void radWizard1_Help(object sender, EventArgs e)
{
Process.Start(http://www.telerik.com);
}

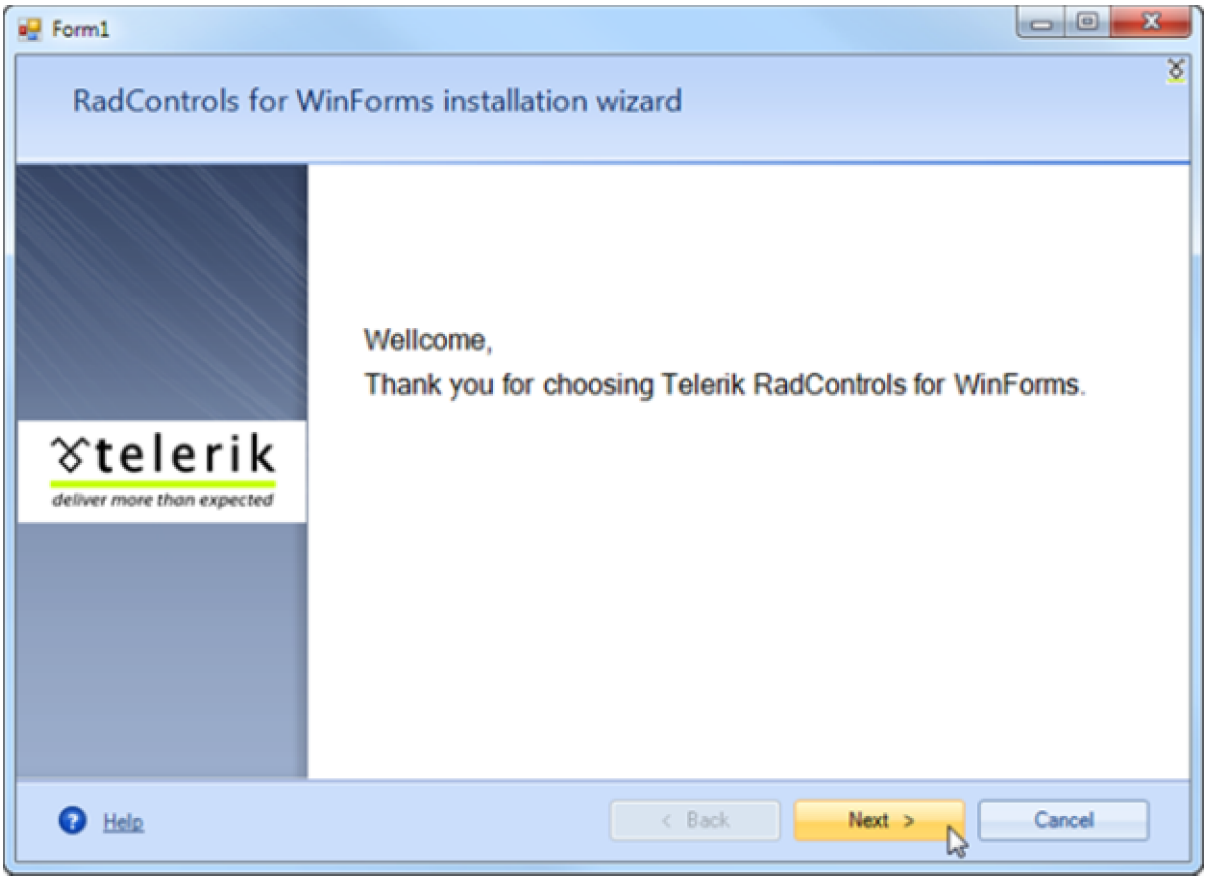
void radWizard1_SelectedPageChanged(object sender,
Telerik.WinControls.UI.SelectedPageChangedEventArgs e)
{
if (e.SelectedPage == wizardPage3)
{
radWaitingBar1.StartWaiting();
}
else
{
radWaitingBar1.StopWaiting();
}
}

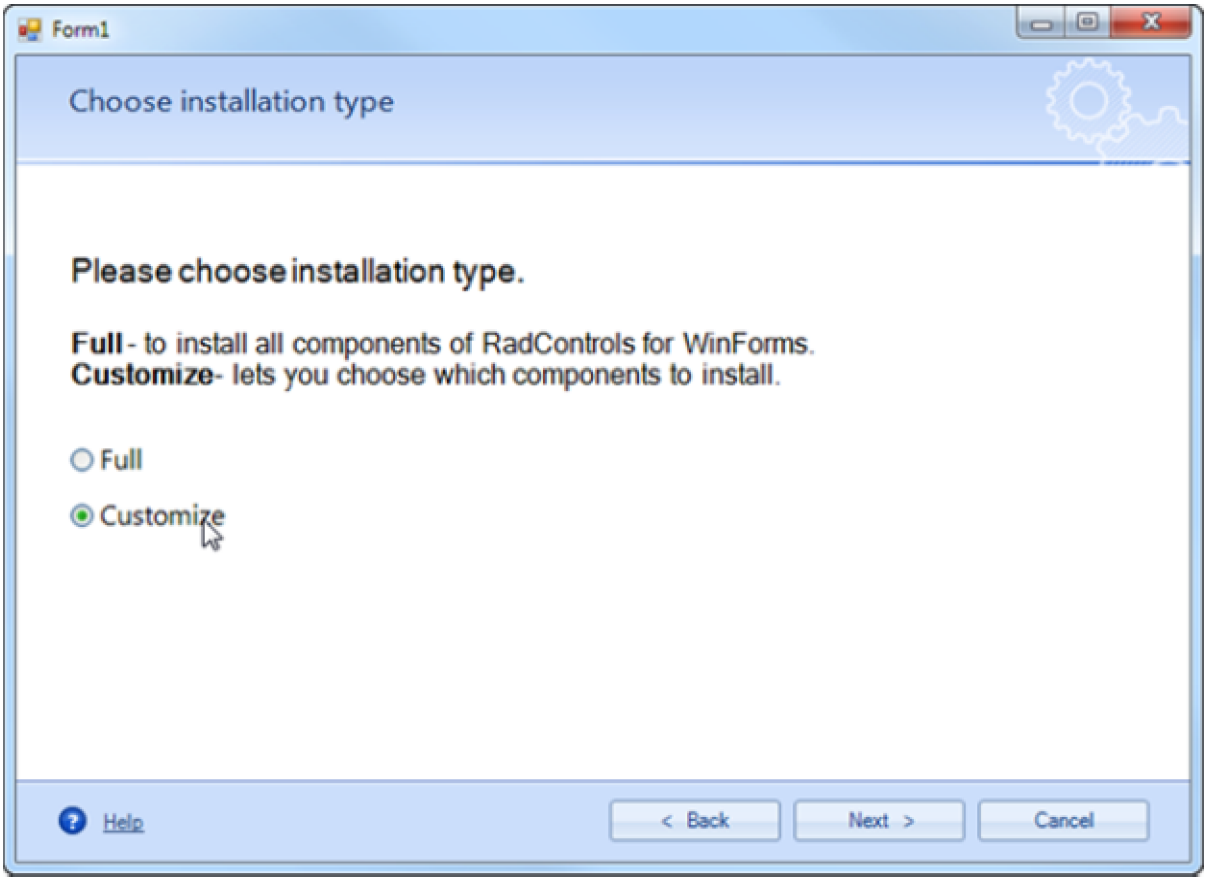
void radWizard1_Cancel(object sender, EventArgs e)
{
if (RadMessageBox.Show("Are you sure that you want to stop the installation?", "",
MessageBoxButtons.YesNo)
== DialogResult.Yes)
{
Application.Exit();
}
}

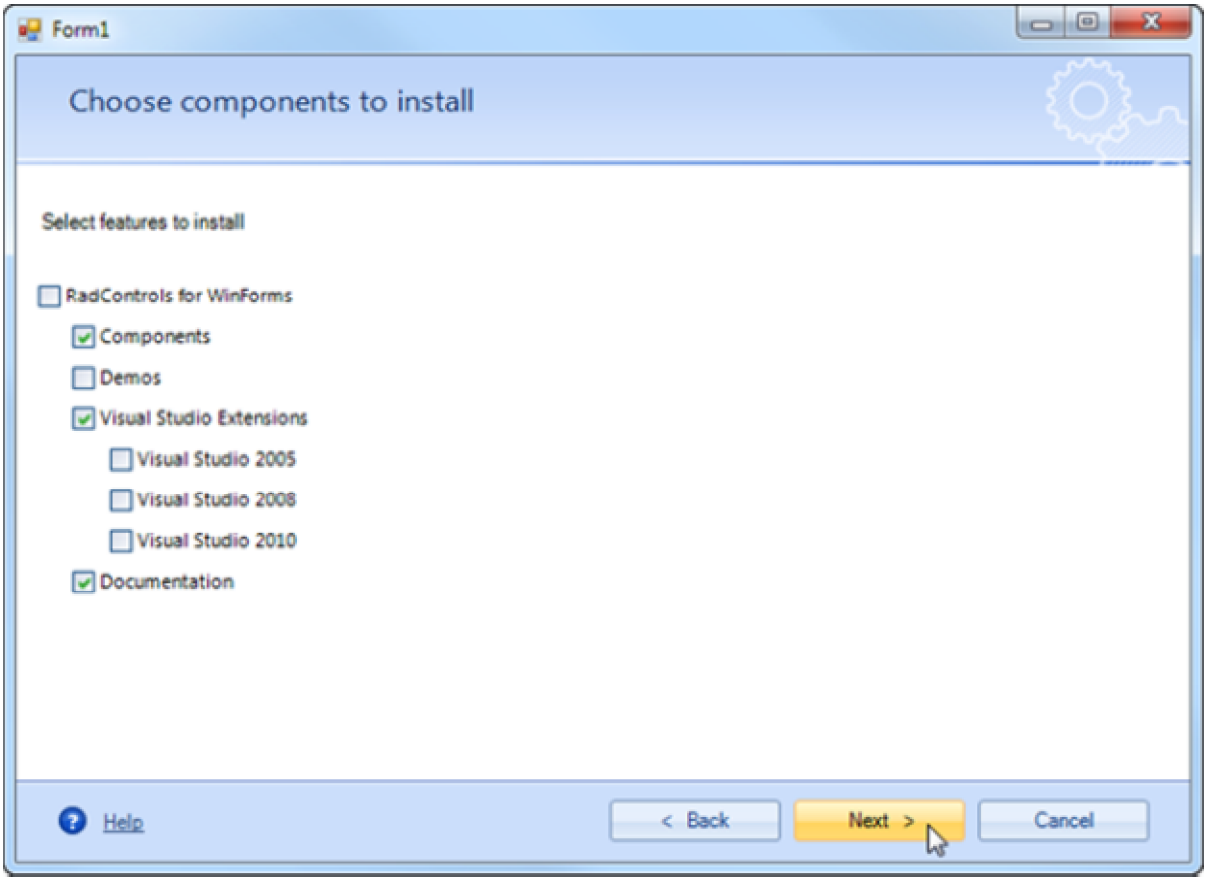
void radWizard1_Finish(object sender, EventArgs e)
```

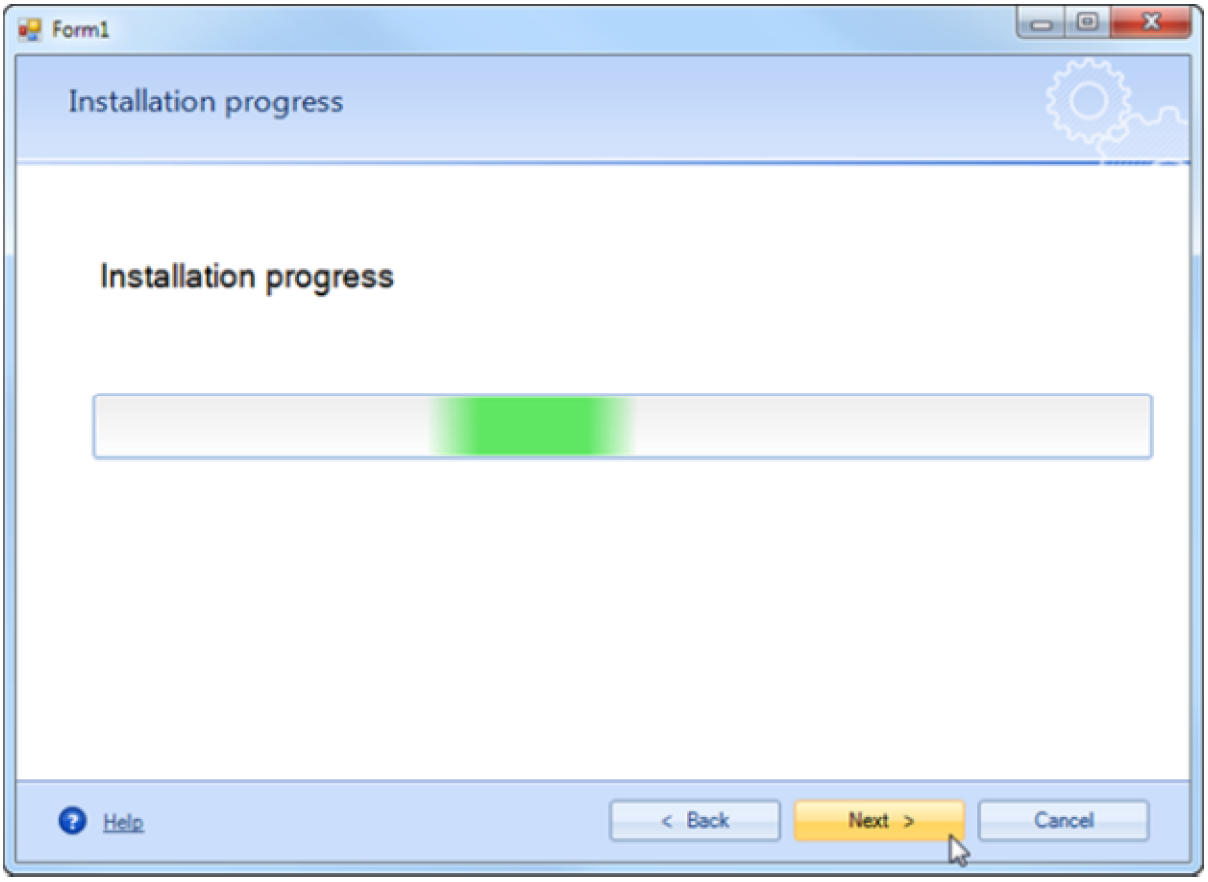
```
{
Application.Exit();
}
```

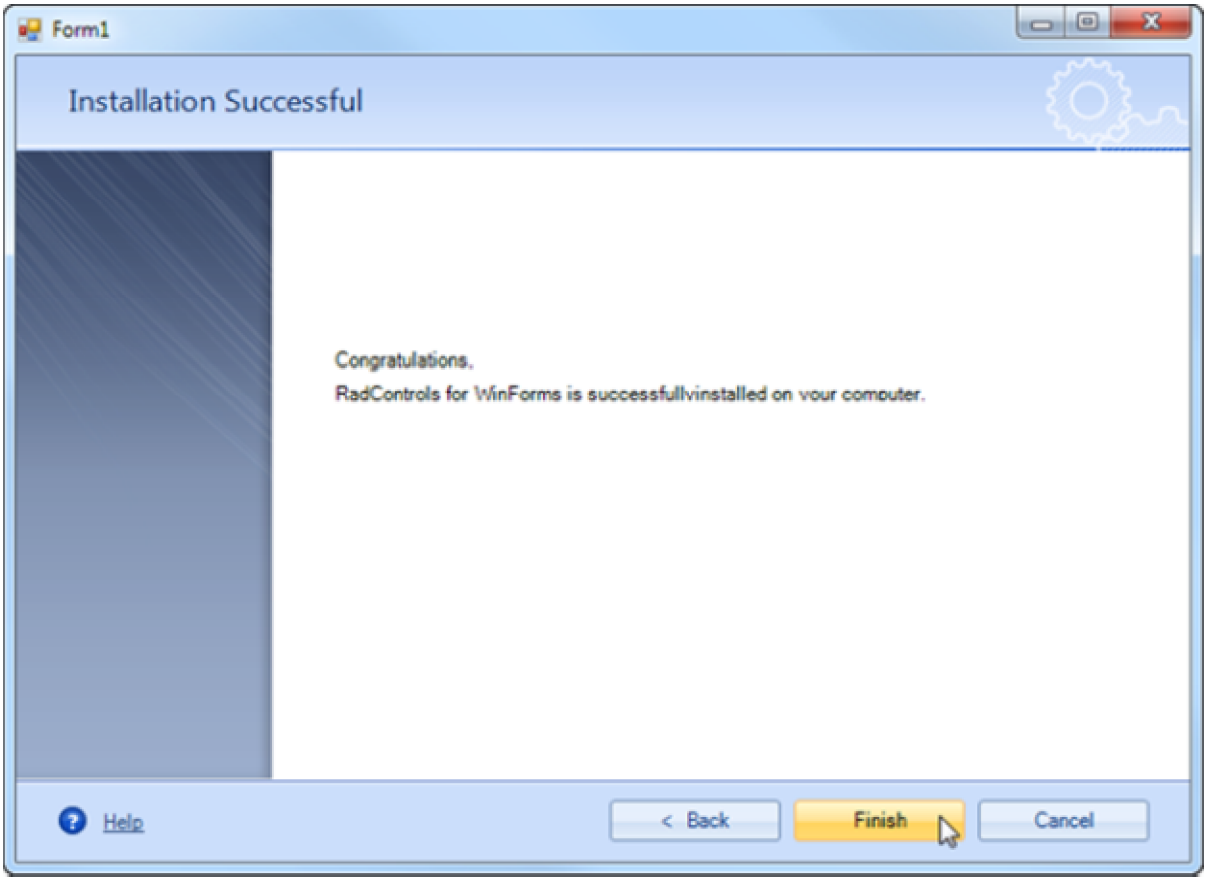
Here is how the application looks like.




Form1
Choose installation type
Please choose installation type.
Full - to install all components of RadControls for WinForms.
Customize - lets you choose which components to install.
Full
Customize
Help
< Back
Next >
Cancel


Form1
Choose components to install
Select features to install
RadControls for WinForms
Components
Demos
Visual Studio Extensions
Visual Studio 2005
Visual Studio 2008
Visual Studio 2010
Documentation
Help
< Back
Next >
Cancel


Form1
Installation progress
Installation progress
Help
< Back
Next >
Cancel



24.5 Summary

In this chapter you learnt how to create applications with RadWizard control and how to place controls on wizard pages and embed functionality. Also, few ways of customization were explored.

25 Appearance and Styling

25.1 Objectives

- Understand the basic tools and concepts used to style RadControls.
- Learn how to use themes and apply application level themes.
- Learn how to use the color blending feature.

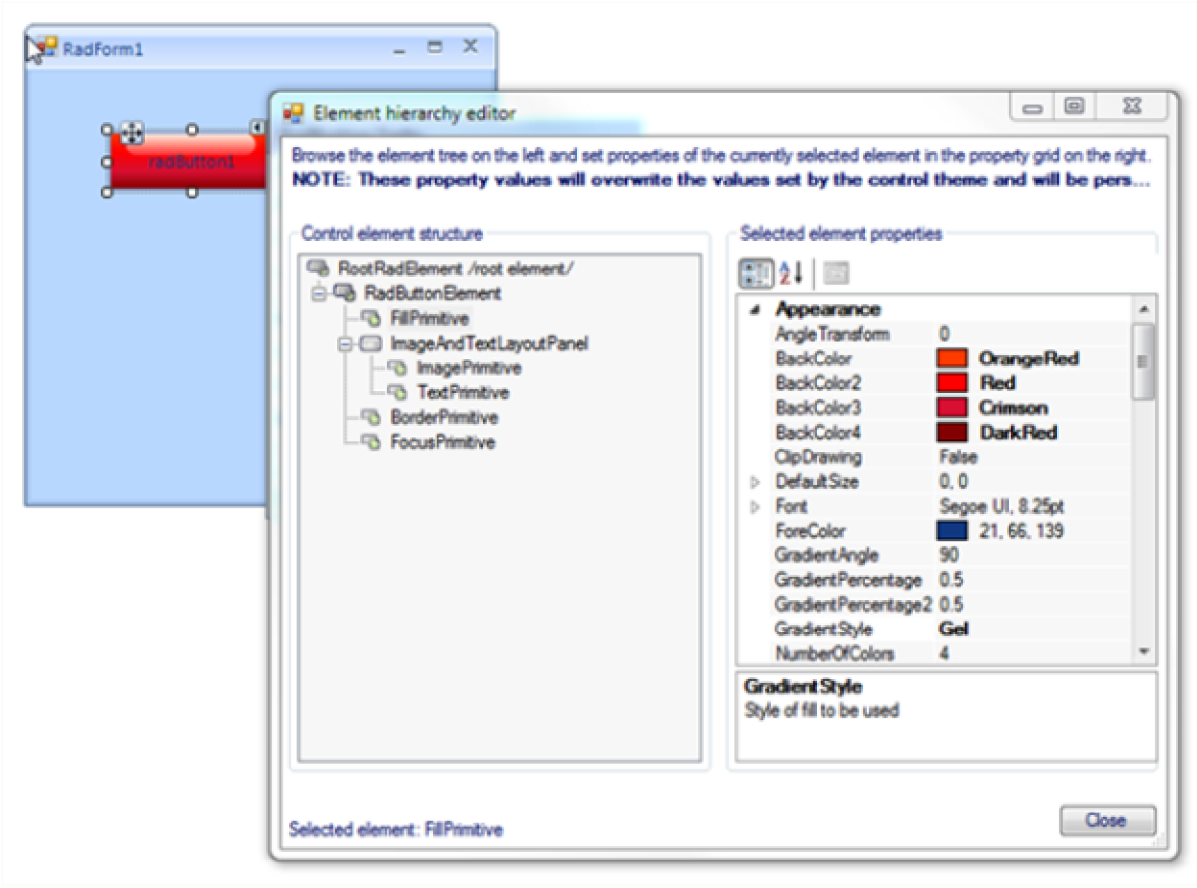
25.2 Introduction

You have multiple routes to completely style all visual aspects of your application:

- **RadElement properties** are loaded with eye-candy properties for tweaking color gradient, multiple aspects of background and border colors, gradient styles, angle and scale transforms. These can be reached through individual elements of RadControls in code or using the Element Hierarchy Editor (see the Telerik Presentation Foundation chapter in "Accessing Elements" for more information).
- **Predefined themes** available in the ToolBox style your entire application with a unified look-and-feel in one shot. In addition, the ThemeResolutionService object queries for existing themes and lets you apply themes application wide.
- Use the **Visual Style Builder** to build your own theme codelessly.
- The **Color Blending** feature takes themes a step further by allowing you to apply new colors to existing themes.

25.3 Using RadElements for Styling

As we've seen in previous chapters, you can change every visual element at a low level through the RadControls elements. At design-time, use the Element Hierarchy Editor (through the Smart Tag "Edit UI Elements" link) to access the layout panels and primitives that make up each control.



At run-time, drill down through the tree of RadControl child elements when you need to access specific style related properties.

[VB] Accessing RadElement Properties

```
(TryCast(btnIncrease.ButtonElement.Children(0), FillPrimitive)).BackColor = Color.Crimson
```

[C#] Accessing RadElement Properties

```
(btnIncrease.ButtonElement.Children[0] as FillPrimitive).BackColor =
 Color.Crimson;
```

Class selectors are used to apply the same customization to all elements that belong to the same class. This behavior is very similar to CSS class selectors. The "Class" referred to here is the **ClassName** property used to identify a RadElement. In the example below we're looking for elements with the ClassName "CalendarVisualCellElement". This chunk of code colors the text of every cell element in a RadCalendar.

[VB] Using the ClassSelector

```
Dim selector As New ClassSelector("CalendarVisualCellElement")
For Each cellElement As CalendarCellElement In selector.GetSelectedElements
(radCalendar1.CalendarElement)
 cellElement.ForeColor = Color.Purple
Next
```

[C#] Using the ClassSelector

```
ClassSelector selector = new ClassSelector("CalendarVisualCellElement");
foreach (CalendarCellElement cellElement in selector.GetSelectedElements
(radCalendar1.CalendarElement))
{
```

```
 cellElement.ForeColor = Color.Purple;
}
```

One major advantage to the ClassSelector is that you can get at elements that are buried many levels deep. The element in the calendar example above is at least 5 levels below the base RadCalendarElement.

You can also reset properties to their original values (much like selecting "Reset" from the context menu for a property) using the RadObject **ResetValue()** method and passing it the RadProperty that should be reset.

[VB] Resetting Property Values

```
Dim fillSelector As New TypeSelector(GetType(RadPageViewStripItem))
For Each page As RadPageViewPage In RadPageView1.Pages
For Each stripItem As RadPageViewStripItem In fillSelector.GetSelectedElements(page.Item)
stripItem.ResetValue(LightVisualElement.BackColorProperty)
stripItem.ResetValue(LightVisualElement.BackColor2Property)
stripItem.ResetValue(LightVisualElement.BackColor3Property)
stripItem.ResetValue(LightVisualElement.BackColor4Property)
Next
Next
```

[C#] Resetting Property Values

```
TypeSelector fillSelector = new TypeSelector(typeof(RadPageViewStripItem));
foreach (RadPageViewPage page in radPageView1.Pages)
{
foreach (RadPageViewStripItem stripItem in fillSelector.GetSelectedElements(page.Item))
{
stripItem.ResetValue(LightVisualElement.BackColorProperty);
stripItem.ResetValue(LightVisualElement.BackColor2Property);
stripItem.ResetValue(LightVisualElement.BackColor3Property);
stripItem.ResetValue(LightVisualElement.BackColor4Property);
}
}
```



To enable themes again after setting property locally in code you should call element.ResetValue (RadProperty) and then element.ForceReApplyStyle() method.



Practical Notes on Color Schemes

To get a consistent look across your entire application UI, make the color scheme pervasive. But how do you get colors to match, particularly if you are "color blind"? One way is to access the BackColor, ForeColor and border color properties and let the built-in tools do most of the work for you. Lets say you have the elements shown in the screenshot below, a shaped form with a RadTitleBar, a RadPanel, a standard Picture box with an image and a RadLabel.



The unique set of color on this form comes from the image, so the best approach here is to extract bits of color from this image. Starting with the RadTitleBar, you can locate the underlying Fill primitive, click the BackColor..BackColor4 properties and use the eye dropper tool from the Color dialog.



Continue to "steal" colors for all the elements of the form until all the form colors work together. This method can be adapted to work with RadElements or used within the Visual Style Builder. This approach us relatively quick and lets you manually blend your control colors nicely with colors already in the application without needing to consult a creative designer.





You can find the complete source for this project at:

\Appearance and Styling\<VB|CS>\ElementsAndHTMLStyling



Visual Property Value Precedence

Properties that are set using the Element UI Editor at design time or modified at runtime take precedence and will not be affected by settings in the theme.

25.4 HTML-Like Text Formatting

The rich text formatting mechanism uses plain HTML tags to display formatted text such as font style, font color, font size options and simple layouts. To turn on Html-like formatting the text must start with an <html> tag. Use <size=[+|-]value> to set font size, and <br> to create a new line feed. To bold, underline and italic text, use the corresponding opening and closing tags. Font family is set through <font=Family>.



Gotcha! The <html> tag is not optional. It must exist for the text to be interpreted as HTML. Also, the table below indicates which tags don't have an end tag. This is also not optional. If you try to add these end tags they will be shown as literal text.

Tag	End Tag	Description
<font>	n/a	Font family
<color>	n/a	Text color
<size>	n/a	Font size
		Bold text
		Italic text
<u>	</u>	Underlined text
 	n/a	Line break

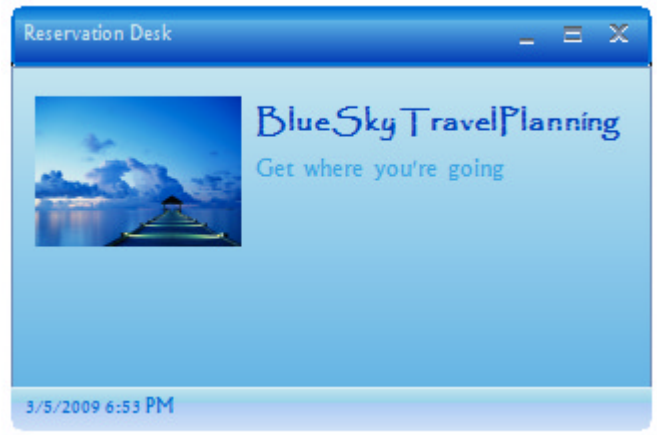
If we add HTML tagged text to the title RadLabel in the earlier "Blue Sky Travel Planning" example you'll see a large "Papyrus" font using one of the image colors, a line break, then a slightly smaller font in another image color.

[VB] Setting HTML Text

```
lblTitle.Text = "<html>" + "<size=11><b><color=1,78,194><font=Papyrus>Blue Sky Travel
Planning</b>" + "<br>" + "<size=10><color=72,173,229><font=Narkisim>Get where you're going"
```

[C#] Setting HTML Text

```
lblTitle.Text =
 "<html>" +
 "<size=11><b><color=1,78,194><font=Papyrus>Blue Sky Travel Planning</b>" +
 "<br>" +
 "<size=10><color=72,173,229><font=Narkisim>Get where you're going";
```



You can find the complete source for this project at:

\Appearance and Styling\<VB|CS>\ElementsAndHTMLStyling

25.5 Themes

Themes supplied with RadControls for WinForms are available as standalone components, apart from the controls' assemblies. This simplifies the way themes are added to the application and improve application loading time.

- AquaTheme
- BreezeTheme
- DesertTheme
- HighContrastBlackTheme
- Office2007BlackTheme
- Office2007SilverTheme
- Office2010BlackTheme
- Office2010BlueTheme
- Office2010SilverTheme
- TelerikTheme
- Windows7Theme

To use Theme components, drag and drop from the VS Toolbox to the form. This adds a reference to the corresponding theme assembly in the project and displays the theme in the ThemeName drop-down of all Telerik controls on the form. As we've seen in earlier chapters, you can avoid the work of setting themes individually for each control by using the **ThemeResolutionService** to set the **ApplicationThemeName**:

[VB] Setting ApplicationThemeName

```
Private Sub ShapedForm1_Load(sender As Object, e As EventArgs)
 ThemeResolutionService.ApplicationThemeName = "Office2010Blue"
End Sub
```

[C#] Setting ApplicationThemeName

```
private void ShapedForm1_Load(object sender, EventArgs e)
{
 ThemeResolutionService.ApplicationThemeName = "Office2010Blue";
```

```
}
```

The ThemeResolutionService also has useful methods for getting all the available themes or themes just for a specific kind of control. With this method you can build a list of themes that can be changed on the fly.

[VB] Getting the Available Themes

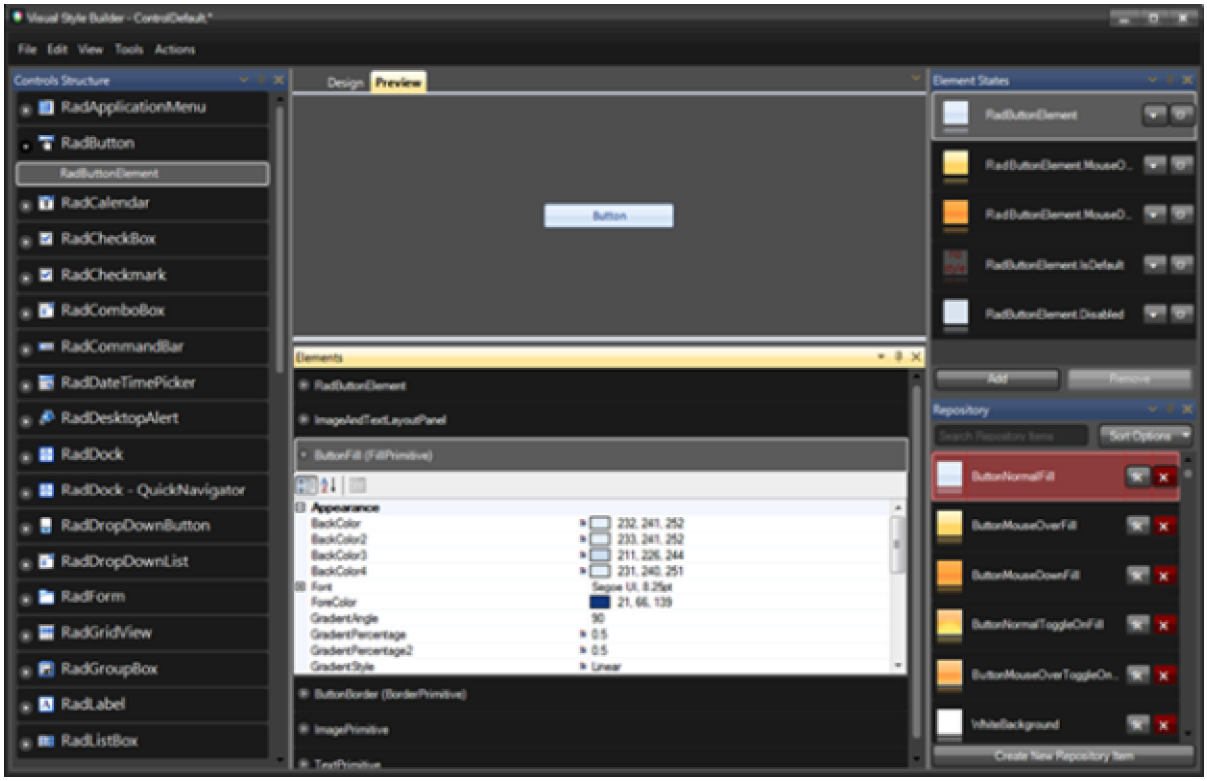
```
Dim list As ThemeList = ThemeResolutionService.GetAvailableThemes(radTitleBar1)
```

[C#] Getting the Available Themes

```
ThemeList list = ThemeResolutionService.GetAvailableThemes(radTitleBar1);
```

25.6 Visual Style Builder

VisualStyleBuilder provides key flexibility behind RadControls for WinForms visual presentation. Visual Style Builder runs as a stand alone application.



With Visual Style Builder you can:

- Set properties

  Using Visual Style Builder you can alter any property of any control. Because all Telerik controls are composed of primitives, this customization can be applied at a very fine level of detail. For example, if you are working with a RadMenu control you can easily change the background color used for submenu items without changing the color used for main menu items.

- Inherit properties

  When working with complex controls in Visual Style Builder, you can inherit property values (such as colors, or fonts) from parent elements to child elements, or override them at the child element level.

- Animate changes

  Visual Style Builder lets you animate changes in response to events, so that a property moves smoothly

through a range of values in response to a mouse movement, state of the control or other event. This makes it easy to create complex, dynamic user interfaces where elements are emphasized in response to user actions such as selections and movements. You can fine-tune these animations by specifying their speed and granularity. See the Photo Album Viewer, Business Card, Magnifier or BubbleBar applications in the RadControls for Winforms example program to see some of the possibilities

- Save themes

  Themes provide a way to reuse your work. Visual Style Builder can save themes to be applied consistently across controls, forms, or applications. This is similar to the way that cascading style sheets (CSS) works in Web applications. Suppose you have developed a custom theme for RadCommandBar that has particular colors, fonts, mouseover behavior, and so on. By saving these themes into an XML theme file (or a compressed .tssp file), you can apply it to every command bar control in your application. Using themes frees you from the repetitive work of setting properties on multiple controls.

25.7 Color Blending

The new Telerik Presentation Foundation color blending feature works similar to many professional graphics design tools that take a basic set of themes colors and recalculates colors based on a substitution color. The result is an elegant new approach to customizing applications appearance automatically on-the-fly. The "Desert" theme for example has a smooth orange color all over. But if we take away all the orange and replace it with other colors we can get new themes.

The code used to achieve this calls the Themes AddColorBlend() method, passes a "ThemeParameterName" value of "BaseColor" and a color that will be the base color to be replaced. Then by accessing the same "BaseColor" property in the ThemeProperties collection, set the new color.

To apply the color blend, TPF will search through the theme's color RGB settings, convert them to HSL color space and calculate the new RGB color values, based on the newly selected color Hue.

[VB] Adding a Color Blend

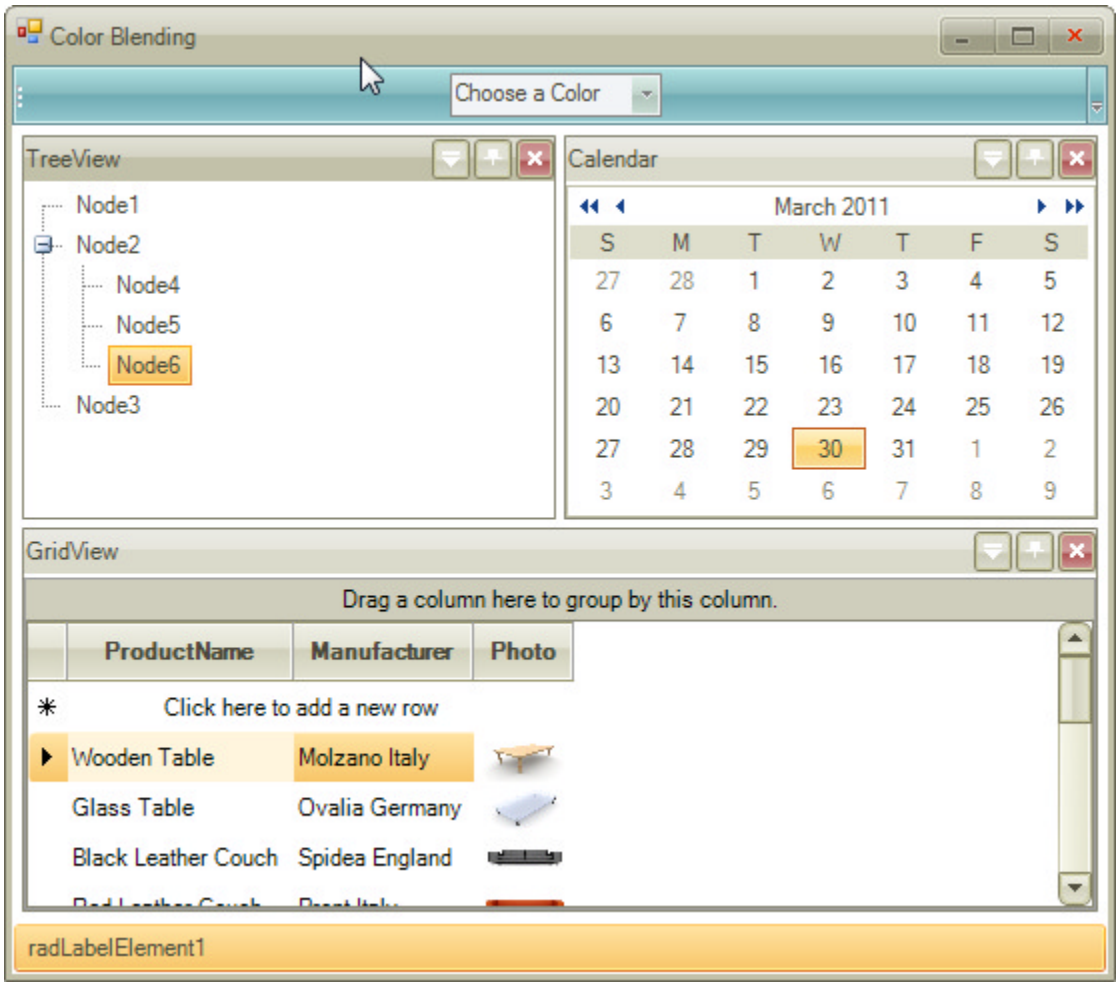
```
Dim currentTheme As Theme = Telerik.WinControls.ThemeResolutionService.GetTheme("Desert")
currentTheme.AddColorBlend("BaseColor", HslColor.FromColor(Color.Orange))
currentTheme.ThemeProperties("BaseColor") = HslColor.FromColor(Color.PaleGreen)
```

[C#] Adding a Color Blend

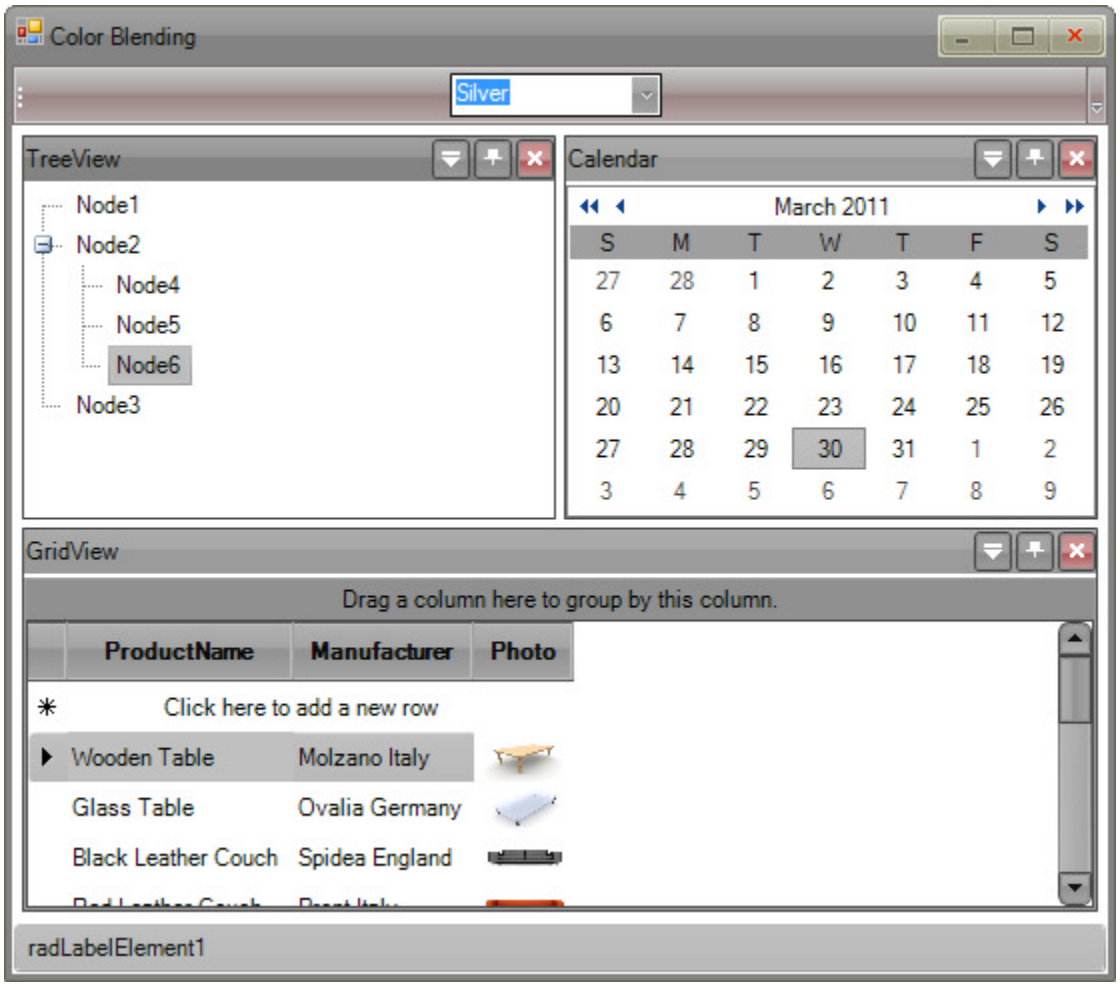
```
Theme currentTheme = Telerik.WinControls.ThemeResolutionService.GetTheme("Desert");
currentTheme.AddColorBlend("BaseColor", HslColor.FromColor(Color.Orange));
currentTheme.ThemeProperties["BaseColor"] = HslColor.FromColor(Color.PaleGreen);
```

You can apply multiple blends using different base colors. The ColorBlending demo example loads all the KnownColor enumerations into RadListDataItems added to the Items collection of CommandBarDropDownList. When the user selects a color, two color blends are added: one for Orange and the other for the teal color used in RadCommandBar. Both blend to the selected KnownColor.

Here is the example form with the "Desert" theme using a number of RadControls...



After blending the orange and teal colors the entire form still has the "Desert" theme but the base color is Silver.



[VB] Applying Mulitple Color Blends

```
Private Sub CommandBarDropDownList1_SelectedIndexChanged(ByVal sender As Object, ByVal e As
Telerik.WinControls.UI.Data.PositionChangedEventArgs)
Dim item As RadListDataItem = TryCast((TryCast(sender,
RadDropDownListElement)).SelectedItem, RadListDataItem)
Dim color As Color = CType(item.Value, Color)
Dim currentTheme As Theme = ThemeResolutionService.GetTheme
(ThemeResolutionService.ApplicationThemeName)
currentTheme.AddColorBlend("BaseColor", Telerik.WinControls.HslColor.FromColor
(color.Orange))
currentTheme.ThemeProperties("BaseColor") = Telerik.WinControls.HslColor.FromColor(color)
currentTheme.AddColorBlend("CommandBarColor", Telerik.WinControls.HslColor.FromColor
(color.FromArgb(170, 215, 218)))
currentTheme.ThemeProperties("CommandBarColor") = Telerik.WinControls.HslColor.FromColor
(color)
End Sub
```

[C#] Applying Mulitple Color Blends

```
void commandBarDropDownList1_SelectedIndexChanged(object sender,
Telerik.WinControls.UI.Data.PositionChangedEventArgs e)
{
RadListDataItem item = (sender as RadDropDownListElement).SelectedItem as RadListDataItem;
Color color = (Color)item.Value;
```

```
Theme currentTheme = ThemeResolutionService.GetTheme
(ThemeResolutionService.ApplicationThemeName);
currentTheme.AddColorBlend("BaseColor", HslColor.FromColor(Color.Orange));
currentTheme.ThemeProperties["BaseColor"] = HslColor.FromColor(color);
currentTheme.AddColorBlend("CommandBarColor", HslColor.FromColor(Color.FromArgb(149, 201,
204)));
currentTheme.ThemeProperties["CommandBarColor"] = HslColor.FromColor(color);
}
```



You can find the complete source for this project at:

\Appearance and Styling\<VB|CS>\ElementsAndHTMLStyling

25.8 Summary

This chapter briefly described the concepts used to style RadControls, provided a short description of the Visual Style Builder tool, discussed themes and the use of the ThemeResolutionService. You also learned how to use the Color Blending feature.

Index

AccessibleInterval, 480-481
Activated, 386-390
ActiveViewType, 480-481
Add, 314-315
Add New Chunk..., 129-139
Add Service Reference..., 320-322
Add Tabbed Document, 418-422
AddingNewEventArgs, 315-320
AddNew, 314-315, 315-320
Advanced Formatting and Binding dialog, 312-314
Alignment, 335-340
AllowAppointmentMove, 480-481
AllowAppointmentResize, 480-481
AllowColumnHeaderSelectors, 446-447, 447-451
AllowFastNavigation, 447-451
AllowFishEye, 446-447
AllowMultipleSelect, 446-447, 447-451
AllowMultipleView, 447-451
AllowNavigation, 447-451
AllowRowHeaderSelectors, 446-447, 447-451
AllowViewSelector, 446-447, 447-451
Anchor, 13-22, 39-45
AppointmentTitleFormat, 480-481
AquaTheme, 13-22, 335-340, 446-447, 558-560, 560-563
ArrowButtonMinWidth, 659-662
ArrowPrimitive, 149-154
AutoFilter, 659-662
AutoGenerateHierarchy, 592-625
AutoSizeColumnsMode, 558-560, 592-625
AxisMode, 550-556
BackColor, 13-22, 39-45
BackgroundImage, 340-344
BasePrimitive, 148-149, 149-154
Basic, 38-39
Before You Begin..., 11-12
BestFitColumns(), 560-563
Binding Image Data, 325-328

Binding RadGridView, 328-333
Binding to Business Objects, 315-320
Binding to LINQ Data Sources, 322-325
Binding to Services, 320-322
Binding to Simple Controls Using DataBindings, 312-314
BindingNavigator, 314-315
BindingSource, 304-312, 314-315, 315-320, 322-325
BorderPrimitive, 144-145, 145-148, 149-154
BorderStyle, 13-22
BorderVisible, 144-145
BreezeTheme, 211-215, 655-659
Button Groups, 129-139
ButtonItems, 129-139
Buttons Programming, 24-37
CancelEdit, 314-315
CarouselBezierPath, 418-422
CarouselEllipsePath, 418-422
CarouselPath, 418-422
CausesValidation, 39-45
Cell and Row Formatting, 569-573
CellValueNeeded, 625-630
CheckOnClick, 105-110
CheckPrimitive, 149-154
ChildColumnNames, 592-625
ChildGridViewTemplate, 592-625
ChildTemplate, 592-625
Chunks, 129-139
Class Hierarchy, 148-149
ClearButton, 447-451
Collapsed, 139-143
Color Blending, 752-755
Column Chooser, 563-564
Column Manipulation, 560-563
CommandTab, 129-139
CommandTabs, 129-139
Conditional Formatting, 564-569
ContextMenu, 665-669
Contextual Tabs, 129-139
ContextualTabGroups, 129-139

CreateChildElements(), 154
Creating Shortcuts, 464-466
CurrencyManager, 314-315
CurrentItemChanged, 314-315, 315-320
CurrentItemChanging, 315-320
CurrentViewColumn, 447-451
CurrentViewRow, 447-451
Custom Cells and Elements, 573-581
Dash, 340-344
Data Source Configuration Wizard, 304-312
DataBindings, 294-304, 312-314
DataBoundItem, 659-662
DataMember, 294-304
DataSet, 294-304, 304-312, 314-315
DataSet Designer, 592-625
DataSource, 294-304, 304-312, 312-314, 314-315, 315-320, 322-325, 474-480, 474-480, 655-659
DataTable, 314-315
DateTime, 446-447
DefaultTreeView, 665-669
DesertTheme, 13-22
DisplayMember, 294-304, 304-312, 659-662
DisplayStyle, 129-139, 139-143, 414-418
Dock, 13-22
Dock New Window, 418-422
DockingManager, 414-418
DockState, 414-418
DriveInfo, 211-215
DropDown, 665-669
DropDownSizingMode, 655-659
Dynamic LINQ Query Library, 630-635
Dynamic.cs, 630-635
Edit in DataSet Designer, 592-625
EditorControl, 655, 659-662
Element Hierarchy Editor, 145-148
Elements, 129-139
Empty Values, 547-550
EnableAutoLoop, 414-418
EndEdit, 314-315, 315-320
EndWaiting(), 340-344

EventProvider, 474-480
Excel, 641-653
ExcelML, 641-653
ExitButton, 139-143
Exporting RadGridView Data, 641-653
Expression Editor, 635-641
F.A.Q, 81-83
FileSystemInfo, 211-215
Fill, 558-560
FillPrimitive, 144-145, 145-148, 149-154
Filter, 13-22
FilterExpression, 659-662
FocusedDate, 447-451, 480-481
FocusPrimitive, 145-148, 149-154
FormatString, 558-560
Formatting API, 93
Galleries, 129-139
GetDrives(), 211-215
GetFileSystemInfos, 211-215
Getting Started, 13-22, 39-45, 83-85, 105-110, 145-148, 172-180, 211-215, 280-284, 304-312, 335-340, 369-375, 386-390, 414-418, 446-447, 474-480, 502-520, 558-560, 655-659, 665-669, 735-745
GettingStarted, 248-256
Global Shortcuts, 466-469
GradientStyle, 145-148
Grid Settings, 592-625
GridElement, 625-630
GridFilterCellElement, 659-662
GridViewCellValueEventArgs, 625-630
GridViewDataRowInfo, 659-662
GridViewDecimalColumn, 560-563
GridViewTemplate, 560-563, 592-625
GripPrimitive, 149-154
GroupBox, 13-22, 39-45
Grouping, Sorting and Filtering, 581-592
Hatch, 340-344
HideSelection, 105-110
Hierarchy Support, 592-625
History, 92-93
How This Courseware Is Organized, 2-5

How To, 550-556
HTML Support, 97-98
HTML-Like Text Formatting, 749-750
IBindingList, 314-315, 315-320, 663-665
IComponent, 148-149
IList, 294-304, 663-665
IListSource, 294-304, 663-665
ImageAlignment, 340-344, 414-418
ImageKey, 105-110, 139-143
ImageList, 105-110
ImagePrimitive, 149-154
Import/Export, 93-97
Interval, 625-630
Introducing RadControls, 5-11
Introduction, 13, 38-39, 104-105, 144-145, 165-172, 210-211, 247-248, 279-280, 294-304, 334-335, 367-369, 385-386, 411-414, 445-446, 463-464, 470-474, 502, 557-558, 655, 663-665, 732-734, 746
IsLogarithmic, 547-550, 547-550
IsVisible, 655-659
Items, 13-22
Language-Integrated Query, 322-325
LargeChange, 340-344
Layout Modes, 86-87
LayoutPanel, 144-145, 148-149
LinePrimitive, 149-154
LINQ, 322-325, 630-635
LogarithmBase, 547-550
Logarithmic Y-Axis, 547-550, 547-550
Mask, 39-45
MaskType, 39-45
MasterGridViewTemplate, 592-625, 630-635
Maximum, 335-340, 340-344
MDI, 129-139
Minimum, 340-344
MinimumSize, 39-45
MiscellaneousTheme, 13-22, 592-625
MS SQL, 335-340
MultiColumnComboBoxElement, 655, 659-662
Multi-Line Labels, 547-550
Multiple Document Interface, 129-139

MultiViewColumns, 447-451
MultiViewRows, 447-451
Navigation Management, 447-451
New Theme Manager, 45-46
NullText, 38-39, 39-45
Objectives,
13, 38, 104, 144, 165, 210, 247, 279, 294, 334, 367, 385, 411, 445, 463, 470, 502, 557, 655, 663, 732, 7
Office2007BlackTheme, 474-480
Office2007Silver, 335-340, 335-340, 386-390, 414-418
Office2007SilverTheme, 13-22, 39-45, 105-110, 335-340, 665-669
Open Theme Builder, 45-46
OpenFileDialog, 13-22, 105-110
OptionsButton, 139-143
Orientation, 129-139, 340-344
OverflowPrimitive, 149-154
Overview, 80-81
Padding, 335-340
PaintPrimitive(), 148-149, 149-154
ParentColumnNames, 592-625
ParentTemplate, 592-625
PIA, 641-653
PictureBox, 13-22
PictureBoxSizeMode, 13-22
PlotArea, 547-550
Positioning, 91-92
Primary Interop Assemblies, 641-653
Primitives, 149-154
Professional, 38-39
Programming, 286-293, 379-384
Programming ListControl and DropDownList, 216-246
Programming ListView , 264-277
Programming Rad Form and Message Box , 390-410
Programming RadPageView, 194-208
Programming the "Real Estate" Controls, 422-444
Programming the Calendar, 451-462
Programming the Chart, 527-547
Programming the Multi Column Combo Box, 659-662
Programming the Scheduler, 481-501
Programming the Telerik Presentation Foundation, 154-164

Programming the Tree View, 672-731
Programming the User Feedback Controls, 344-365
ProgressBarPrimitive, 149-154
ProgressOrientation, 340-344
Properties.Resources, 335-340
PropertyMetadata, 148-149
Quick Access Tool Bar, 129-139
QuickAccessToolbarBelowRibbon, 129-139
QuickAccessToolBarHeight, 129-139
QuickAccessToolBarItems, 129-139, 139-143
RadAboutBox, 386-390, 390
RadApplicationMenu, 129-139
RadBreadCrumb, 665-669
RadButton, 13-22, 22-24, 39-45, 211-215, 315-320, 334-335
RadButtonElement, 145-148, 386-390, 414-418, 474-480
RadCalendar, 446-447
RadCarousel, 294-304, 414-418
RadChart, 294-304
RadCheckBox, 13-22
RadColorDialog, 38-39
RadComboBox, 210-211, 211-215, 215-216, 294-304, 314-315, 655-659, 659-662
RadComboBoxItem, 211-215, 215-216
RadComponentElement, 148-149
RadContextMenu, 665-669
RadControl, 148-149, 149-154
RadDateTimePicker, 38-39
RadDropDownButton, 13-22, 22-24, 129-139
RadElement, 144-145, 148-149
RadForm, 13-22, 105-110, 334-335, 390
RadGalleryElement, 129-139
RadGridView, 294-304, 314-315, 655, 659-662
RadioPrimitive, 149-154
RadItem, 13-22, 22-24, 148-149
RadItem Collection Editor, 110-112
RadItem Editor, 215-216
RadLabel, 13-22, 144-145, 211-215, 312-314, 334-335, 340-344, 365-366
RadLabelElement, 144-145, 386-390
RadListBox, 210-211, 211-215, 215-216, 294-304, 304-312, 320-322
RadListBoxItems, 215-216

RadMaskedEdit, 38-39
RadMenu, 13-22, 105-110, 110-112, 139-143
RadMenuButtonItem, 22-24
RadMenuComboItem, 22-24
RadMenuHeaderItem, 22-24
RadMenuItem, 13-22, 22-24, 110-112
RadMenuSeparatorItem, 22-24, 105-110
RadMessageBox, 39-45, 39-45, 446-447
RadMultiColumnComboBox, 655, 655-659, 659-662
RadObject, 148-149
RadPanel, 13-22, 665-669
RadProgress, 340-344
RadProgressBar, 334-335, 340-344, 365-366
RadProperty, 148-149
RadPropertyMetaData, 148-149
RadRadioButtonElement, 154
RadRadioButtons, 13-22
RadRepeatButton, 13-22
RadRibbonBar, 129-139, 139-143, 144-145
RadRibbonBar and RadApplicationMenu, 129-139
RadRibbonBar Walk Through, 139-143
RadRibbonBarChunk, 129-139
RadRibbonBarForm, 386-390
RadRibbonForm, 390
RadRotator, 414-418
RadScheduler, 474-480
RadSpinEditor, 38-39
RadSplitButton, 13-22, 22-24
RadStatusStrip, 334-335, 335-340, 340-344, 365-366, 386-390
RadTabStrip, 144-145
RadTabStripElement, 144-145
RadTextBox, 38-39, 294-304, 312-314, 312-314
RadToggleButton, 13-22
RadToolStrip, 386-390
RadTrackBar, 334-335, 335-340, 340-344, 365-366
RadTrackBarElement, 154
RadTreeView, 294-304, 663-665, 665-669
RadWaitingBar, 334-335, 340-344, 365-366
ReadOnly, 93

RectanglePrimitive, 149-154
Relations, 592-625
Relations Collection Editor, 592-625
RemoveCurrent, 314-315
ResetBindings, 315-320
RibbonBar UI, 101-103
RichTextBox, 105-110
RightColumnItems, 129-139
RightColumnWidth, 129-139
RootRadElement, 144-145, 145-148
Running, 414-418
SaveFileDialog, 105-110
Scale Breaks, 547-550
ScaleBreaks, 547-550
ScaleTransform, 414-418
SchedulerBindingDataSource, 474-480
Search, 88-91
SelectedDates, 446-447
SelectedIndex, 211-215
SelectedIndexChanged, 211-215
SelectedItem, 659-662
SelectedValue, 659-662
Selection, 87-88
SelectionChanged, 105-110
Separator1, 340-344
Separator2, 340-344
SeparatorWidth, 340-344, 340-344
SetThemeName(), 39-45
ShapedForm, 386-390, 390
Show Advanced Layout Designer, 418-422
Show Advanced Layout Designer..., 414-418
ShowAllDayAppointmentStatus, 480-481
ShowAppointmentStatus, 480-481
ShowColumnHeaders, 447-451
ShowFastnavigationButtons, 447-451, 447-451
ShowFooter, 447-451
ShowNavigationElements, 480-481
ShowOtherMonthDays, 447-451
ShowPopup(), 659-662

ShowProgressIndicators, 340-344
ShowRowHeader, 446-447
ShowRowheaders, 447-451
ShowSelectionMargin, 105-110
ShowSlideArea, 340-344
ShowTicks, 335-340, 340-344
ShowViewSelector, 446-447, 447-451
SlideAreaColor1, 340-344
SlideAreaColor2, 340-344
SlideAreaGradientAngle, 340-344
SlideAreaWidth, 340-344
SliderAreaColor1, 335-340
SliderAreaColor2, 335-340
SortType, 630-635
Spellcheck, 98-101
Spring, 335-340
SqlDataRead, 314-315
StackLayoutPanel, 144-145
StartWaiting(), 340-344
StepWidth, 340-344
StretchVertically, 335-340
Strict Mode, 547-550
StripLayoutPanel, 144-145
Summary, 37, 79, 143, 164, 208-209, 246, 277, 293, 333, 365-366, 384, 410, 444, 462, 501, 556, 653-654, 662, 731, 745, 755
SweepAngle, 340-344
System, 38-39
System.IO, 211-215
System.Linq.Dynamic, 630-635
TabIndex, 39-45
TableAdapter, 304-312
Tag, 39-45
Telerik Presentation Framework, 144-145, 210-211
Telerik Reports, 641-653
Telerik.WinControls.UI, 211-215, 335-340
TelerikTheme, 335-340
TextAlignment, 144-145
TextBlock, 547-550
TextBox, 38-39

TextImageRelation, 105-110, 145-148, 340-344, 386-390

TextImageRelationship, 335-340

TextPrimitive, 144-145, 149-154

Theme Manager, 22-24, 45-46

ThemeName, 13-22, 39-45, 105-110, 211-215, 335-340, 340-344, 340-344, 446-447, 474-480

Themes, 750-751

ThumbWidth, 340-344

Tick, 625-630

TickFrequency, 340-344

TicksColor, 335-340, 340-344

Timer, 625-630

TitleAlign, 447-451

TitleFormat, 447-451

TodayButton, 447-451

ToggleStateChanged, 39-45, 105-110, 139-143

TPF, 144-145

TrackBarPrimitive, 149-154

Unique RadChart Features, 547-550

UniqueName, 560-563

UpDownAndRightBottom, 655-659

Using BindingSource, 314-315

Using RadElements for Styling, 746-749

Using RadGridView with Dynamic LINQ Query Library, 630-635

Using the Design Time Interface, 22-24, 45-46, 110-112, 180-182, 215-216, 256-264, 284-286, 340-344, 375-379, 390, 418-422, 447-451, 480-481, 520-527, 669-672, 734-735

Value, 340-344

Value1, 340-344

Value2, 340-344

ValueChanged, 340-344

ValueMember, 294-304, 304-312, 659-662

Virtual Mode, 625-630

Visibility, 139-143

Visual Style Builder, 751-752

VisualElement, 148-149, 149-154

WaitingIndicatorWidth, 340-344

WaitingSpeed, 340-344

WaitingStep, 340-344

WCF, 320-322

WCF Service Library, 320-322

Web, 38-39
What Do You Need To Have Before You Read This Courseware?, 2
What Do You Need To Know Before Reading This Courseware?, 2
Who Should Read This Courseware, 2
Windows Communication Foundation, 320-322
Windows Presentation Frameowrk, 144-145
Working with RadPageView, 182-190
Working with RadPageViewPages, 190-194
Working with RichTextBox , 85-86
Working With the Editor Controls, 46-79
Working with the Menu Controls, 112-129
WPF, 144-145
YAxis, 547-550
YAxis2, 547-550