RadControls
FOR ASP.NET

a step by step learning guide

telerik
deliver more than expected

Telerik RadControls

Printed: July 2007 in San Jose, CA USA

Authors
Noel Rice
John Waters
Xavier Pacheco
Ramesh Theivendran
Rick Miller

Team Coordinator
Noel Rice

Marketing & Design:
Sofia Batalova
Slavka Petrova
Assen Tzekin

Achnowledgment:
Ivo Nedkov
Stefan Rahnev
Dimitre Taslakov
Vassil Terziev
Lino Tadros

Production
Falafel Software Inc.
www.falafel.com

Authored by:

Table of Contents

Foreword

It has been a true pleasure for our team to work with the Telerik suite of RadControls for the last couple of years. We have built enterprise level solutions for major corporations using the RadControls technologies. Its ease of use, flexibility and great performance enabled us to reach our goals as well as those of our customers. We were honored to be asked by the Telerik team to build this courseware for the Telerik community. This hands-on training material will help individuals and teams get up to speed fast with RadControls and experience their true value. I hope you enjoy this courseware and that it helps you master RadControls, bringing you success in your valuable projects.

Lino Tadros
President & CEO
Falafel Software Inc.

Day

1 Day 1

1.1 Introduction

1.1.1 Introduction to RadControls for ASP.NET

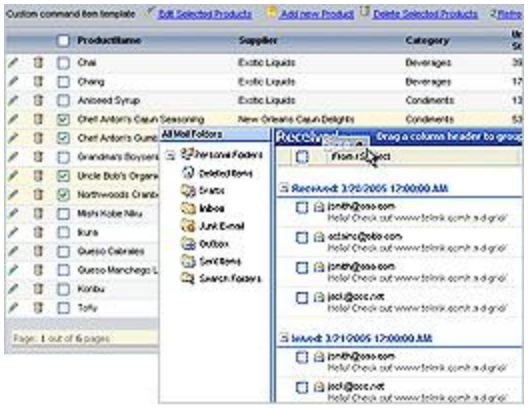
About RadControls

RadControls for ASP.NET are a powerful set of components that let you create web applications with the richness and responsiveness of desktop applications. 18 market leading UI and data controls with proven reliability provide extensive AJAX capabilities for a superior user experience.

RadControls for ASP.NET are not just about AJAX alone, although the controls do a first-class job of implementing AJAX to make it easy for developers to use. RadControls for ASP.NET are cross-browser compatible, XHTML/accessibility compliant, have extensive skinning and style support for look-and-feel customization, and have excellent stability, performance and ease-of-use.

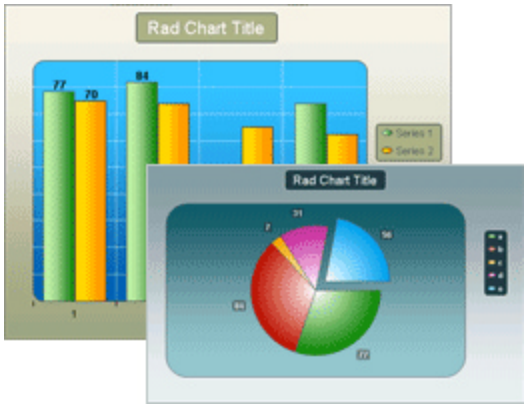
What does this mean to you as a web developer? You can leave AJAX plumbing and browser peculiarities to RadControls and spend your time building web applications that are so agile your users may not realize they are using web applications. Here are just a few highlights to get you thinking:

Menu and navigation: Menus, collapsing panel bars, tab strips and toolbars with a variety of skins provide interactive navigation and user control in the UI. You can develop without code at design time or achieve fine grain control in server or client side code.



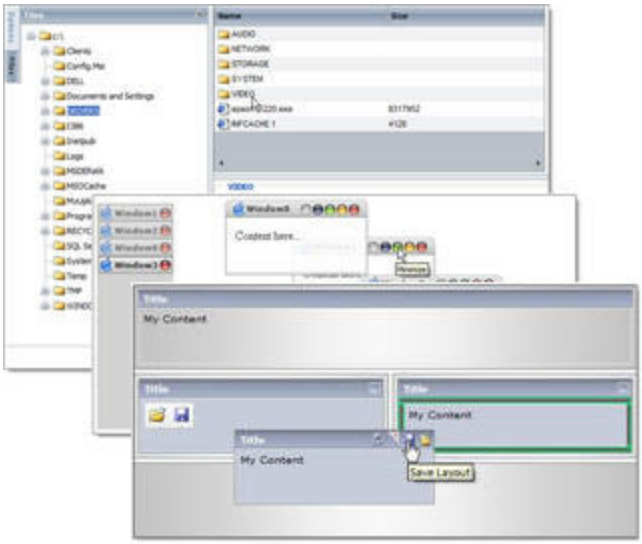
Datagrid: This feature intense control provides a fast, desktop-like user experience and capabilities:

- AJAX mode for real-time performance
- Hierarchical Structures with Many Tables
- Automatic data editing operations
- Filtering
- Outlook-style Grouping
- Multi-Column Sorting
- Column and Row Resizing
- Column Reordering with Drag-and-Drop
- Keyboard navigation
- Automatic AJAX enablement of embedded controls



Charting: Easy-to-use, business oriented data presentation:

- Comprehensive Chart Wizard
- Different chart types on a single chart
- Drill-Down
- Numerical X Axis
- Image Maps
- Automatic scaling of axes
- Gradient fills, hatch fills, image fills
- Negative values

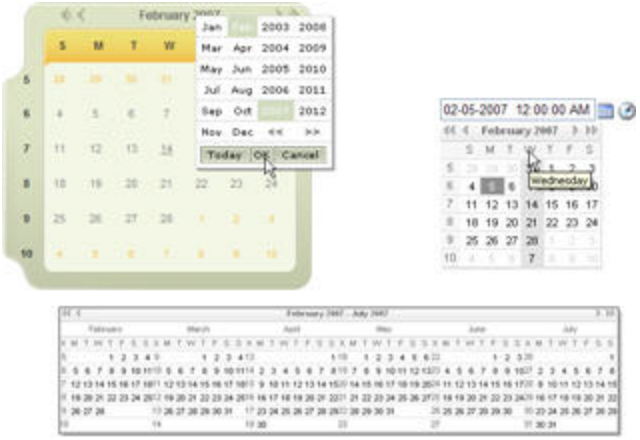


Screen Real Estate: Splitters, windows and docking give your web application flexible, desktop-like control over the user interface layout. The appearance of these components is completely customizable. Here are some other key features:

Splitters have resizable panels, sliding panels, allow unlimited nesting and can be oriented either vertically or horizontally.

Windows in RadControls can be dragged, minimized or maximized like true desktop windows. Implement modal dialogs, splash screens or use one of the predefined Alert, Confirm or Prompt dialogs.

The docking controls produce a true drag-and-drop UI with full control of where objects are dropped.



Calendaring: Calendars and validated date/time inputs provide quick intuitive entry. A few key features:

- Integrated DatePicker Control
- Customizable Day Matrix
- Multiview Presentation
- Multiple Selection Options
- Template Support
- Horizontal and Vertical Orientation
- Scrolling and Navigation

About Telerik

Telerik Corporation was founded in 2002 and is a leading vendor user interface components for ASP.NET

and Windows forms.
Teleriks customers include Fortune 500 companies, educational, government and non-profit organizations all over the world: Microsoft, United Nations, US Army, Harvard University, Intel, Siemens, Pfizer, Citigroup and NASA, just to name a few. Telerik staff has an impressive 11 MCSDs, 1 MCSE, 1 MCDBA and 1 MCAD on its team of over 70 highly qualified professionals.

1.1.2 Who Should Read this Courseware

You should read this courseware if:

- You have never used AJAX or an AJAX control and want to learn what it's all about.
- You have used AJAX or some kind of AJAX based controls and want to learn the Telerik approach using RadControls.
- You have used RadControls and want to make your knowledge more comprehensive.

1.1.3 What Do You Need to Have Before You Read this Courseware?

Computer Setup

- Windows XP Professional
- Microsoft .NET Framework 2.0
- Internet Information Services 5.x
- Internet Explorer 6.x. Note: RadAjax requires 6.x support. For complete info on browser compatibility and compliance see http://www.telerik.com/support/requirements-and-compliance/browser-support.aspx.
- Visual Studio 2005
- Visual Studio 2005 Web Application Projects extension: http://msdn2.microsoft.com/en-us/asp.net/aa336618.aspx
- Some examples use the Microsoft MS SQL desktop engine MSDE. See the "Appendix 1 - MSDE" for installation instructions. If you're using Vista you can try Sql Server 2005 Express Edition as a replacement but the projects in this courseware have not been tested for that environment. For Sql Server 2005 Expression Edition information see: http://msdn.microsoft.com/vstudio/express/sql/download/

Setup: RadControls for ASP.net

You can purchase radcontrols from: http://www.telerik.com/purchase/purchase-online.aspx

...or download the trial version at: http://www.telerik.com/products/aspnet/download.aspx

1.1.4 What Do You Need to Know Before Reading this Courseware?

The courseware assumes that you are familiar with ASP.NET using either VB.NET or C# code. You will also need a basic understanding of the differences between server and client code.

The courseware uses Visual Studio 2005 and assumes you know your way around this environment. You should be able to navigate the basic functional areas of the IDE (e.g. Solution Explorer, Properties, design/source for web pages, etc.) and be able to run and debug web applications. Many of the lab projects in this course use the Web Application Projects extension for Visual Studio 2005 so you will need to know how to load both web sites directly and web application projects.

1.1.5 How the Courseware is Organized

Day 1

AJAX Fundamentals

Learn the underlying mechanisms that make AJAX work. This is AJAX "the hard way", without using Telerik components. You may want to skip ahead to the RadAjax chapter if you're not interested in the nuts and bolts. This chapter focuses on how JavaScript, XML, CSS and the DOM are combined to make AJAX web clients.

RadAjax

This chapter introduces the Telerik approach to AJAX and explores how to use the components in the RadAjax assembly to perform various AJAX tasks. The chapter covers how to use RadAjaxManager, RadAjaxServiceManager, RadAjaxPanel, AjaxLoadingPanel and RadAjaxTimer. You'll learn how to create a simple AJAX web application, display a loading "spinny" graphic, AJAX-retrofit an existing application and learn when to use RadAjaxPanel vs RadAjaxManager. The chapter demonstrates how RadAjaxManager works within user controls, master pages and inside a grid. You will also learn how to call web services using AJAX, how to use the RadAjaxTimer and how to measure performance.

RadComboBox

This chapter demonstrates how to populate and manipulate RadComboBox, both at design time and programmatically. The chapter includes how to bind declaratively to a datasource, create a multiple column drop down using templates, simple load on demand and loading using external "streamer" pages. You will also see how to implement cascading combo boxes.

RadInput

This section covers using RadDateInput and RadMaskedTextBox to retrieve valid, properly formed input from the user. You'll see how to build input masks and date formats for RadMaskedTextBox and RadDateInput as well as how to manipulate output properties. You will also see how to handle client side events for instantaneous feedback during erroneous input.

RadCalendar

This chapter helps organize the many properties and methods of the RadCalendar control set RadCalendar, RadDatePicker, RadTimePicker and RadDateTimePicker. You will learn to setup RadCalendar at design time, including how to define templates and special days. The chapter includes a demonstration of custom rendering for calendar headers and day cells. Finally, you'll learn how to use the RadCalendar control set in JavaScript by using the controls client script methods and events.

Day 2

RadGrid

This chapter aims to make you productive right away with this workhorse control by leading you through common useful tasks such as connecting the grid to a data source, AJAX enabling the grid, paging,

scrolling, grouping, sorting and filtering, displaying hierarchical data, editing in-place, using templates in the grid, creating custom edit forms and changing grid appearance using skins. You'll also learn the basics of exporting data to Word or Excel.

RadMenu

Learn how to build just about any type of drop-down or context menu for your ASP.NET applications: multi-row, vertical, hierarchical or scrollable menus. You learn how to bind RadMenu control to XML, hierarchical data and programmatically. You will also learn how to modify menu appearance in the designer using skins and how to trigger menu items using keyboard shortcuts.

RadToolBar

This section on RadToolBar covers the basics of configuring the tool bar at design time, explains building and responding to RadToolBar at runtime, and explores the capabilities of the client side API and event model. The chapter demonstrates populating the toolbar programmatically and with data binding. You will see how custom toolbar buttons are created. You will learn to create a database explorer demo complete with filtering and later will extend the example to respond to client side events.

RadTabStrip

This chapter covers how to use RadTabStrip controls in the designer to control visual appearance and behavior, design-time binding to data sources for xml and hierarchical data, and using tabstrip as a vehicle for navigating websites. You will learn how RadMultiPage works with RadTabStrip to manage user interfaces with multiple sets of components that should display based on tabstrip button selection.

RadPanelBar

This section covers the basics of configuring at design time, binding to flat and hierarchical data, demonstrates building and responding to RadPanelBar at runtime, and explores the capabilities of the client side API and event model.

RadSplitter

This chapter demonstrates techniques for handling screen real estate using the rad splitter, panes and split bars. The splitter is used to create an Explorer style user interface and a more involved File Explorer example demonstrates integration between the splitter and multiple rad components.

RadTreeView

This chapter covers the practical aspects of using RadTreeView. It demonstrates linking your RadTreeView control to a number of different data sources, programmatic loading of the tree, and using templated controls to display text and images. You will discover how to create context menus for RadTreeView elements, how to use node attributes to store custom data and how to deal with issues surrounding drag-and-drop.

RadUpload

This section explains how to configure your application to use RadUpload, how to validate the upload, perform custom validation, deal with large file uploads, and how to display progress during the upload. This section also describes common troubleshooting scenarios and solutions.

Day 3

RadChart

This chapter explores the RadChart component with special focus on the chart wizard. This section is designed to help you become productive by organizing and presenting RadCharts many properties in logical groupings. The chapter demonstrates combining different chart types, responding to click events with and without postback and looks at RadChart interacting with RadAjax.

RadEditor

This chapter shows methods for getting data in and out of the editor, how to persist content, how to customize the editor at multiple levels, how to handle the controls appearance using skins, how to localize the editor/dialogs/tools, and how to control the editing permissions. You will learn how to use the File Managers to handle external content.

RadSpell

This section covers common scenarios for RadSpell including basic setup and usage, localizing dictionaries and the spell check dialog, custom user dictionaries, creating your own dictionaries for specific languages or special purposes, integrating with standard ASP.NET validation, and skinning. It also looks at RadSpell usage on the client to trigger spell checking, respond to events and the many uses of assigning custom text sources.

RadWindow

This section explores RadWindowManager and RadWindow, from basic opening, positioning and behavior to transferring data to and from parent and popup windows. The chapter looks at various flavors of built-in windows like the splash screen as well as alert, confirm and prompt dialogs. The chapter also shows how to manipulate templates that control window content. The client scripting portion of the chapter provides an overview of the methods and properties available to RadWindowManager and RadWindow.

RadDock

This chapter looks at how dock manager, docking zones and the dockable objects work together to create drag and drop web applications. The chapter demonstrates how to control the appearance and behavior of dockable objects.

RadRotator

This chapter discusses the utility of RadTicker and RadRotator and how they can be used in concert or separately. The RadTicker is populated in code and several methods of binding data are demonstrated. The chapter works with progressively more complex templates for RadRotator.

Appendix 1 - MSDE

Several labs use the Microsoft SQL Server Desktop Engine (MSDE) to demonstrate binding to a full featured database (vs binding to XML or Access). The chapter contains installation instructions, how to connect to MSDE on the command line, connecting to the MSDE from Visual Studio and installing the Northwind database.

Appendix 2 - Firebug

This chapter tours the functionality of the Firebug JavaScript debugger, paying particular attention to its CSS capabilities for use in modifying RadControl skins, for help identifying both performance and functionality problems in script debugging, and to show how the Net tab exposes AJAX requests and responses.

1.2 Ajax Fundamentals

1.2.1 Introduction

This chapter is designed to give you an understanding of underlying AJAX mechanisms and the effort involved to produce even a minimal AJAX application "from scratch". If you want to get straight into using the controls skip ahead to the RadAjax chapter to see the RadControls approach to AJAX.

1.2.2 AJAX In A Nutshell

A decade ago web applications started as simple, static HTML web pages. Then application frameworks like ASP and Java Servlets emerged to help deliver dynamic content. As web applications became more complex and difficult to manage, technologies like ASP.NET, JavaServer Pages (JSP), JavaServer Faces (JSF) and Apache Struts surfaced to provide a rich component-based, event-driven framework for developing Web UI.

These server side technologies vary slightly but the fundamental nature of web applications remains the same. The client browser...

- Makes a request for a URL.
- Web server processes the request, interacting with business objects.
- Web server returns an HTML response.
- Finally the browser renders the page.

When the user makes changes the page is posted back, processed and rendered again. It's the post back and rendering of the entire page that brings up two important issues. First, the amount of network traffic between browser and server is much larger than needed. Second, the UI flickers as the browser renders the page. The result is a slow, verbose web application that doesn't provide a rich user experience.

This was adequate until sites like Google Suggest, Google maps and Flickr became main stream and developers wondered how they could develop similar rich UI. What was once a web application limitation vanished due to technologies such as DHTML, CSS, DOM, JavaScript and XmlHttpRequest (XHR). In February 2005, Jesse James Garrett of Adaptive Path coined the name "AJAX" (Asynchronous JavaScript and XML) for this group of technologies which today allows us to develop rich presentation on the web. What is AJAX ultimately? A collection of technologies that provide a rich, responsive web client. The technologies that make up AJAX are:

- XmlHttpRequest: Data transfer
- XML or JSON: Data objects
- DOM: Client elements
- CSS: Visual representation

AJAX is about exchanging data asynchronously between browser and web server, and updating only portions of the UI as responses arrive. XmlHttpRequest is the foundation object that makes asynchronous data exchange and AJAX possible.

1.2.3 XmlHttpRequest(XHR)

The heart of AJAX is the XmlHttpRequest ActiveX object which was originally developed by Microsoft for Outlook Web Access 2000 and released with Internet Explorer 5. It enabled IE 5 client-side script to

access server-side components. XmlHttpRequest didn't get early widespread adoption but is now supported by all leading browsers.

XHR is not limited to transferring XML data. It can also supports other data formats such as JSON (JavaScript Object Notation), JavaScript Arrays and Custom Serialization (like in Google Web Toolkit, Atlas e.t.c). XHR requests from the client can be through HTTP GET or POST and can send SOAP envelopes or plain XML data.

XHR exposes methods and properties callable from client script that transfer data to and from web servers via HTTP. XHR uses a separate communication channel between the client browser and the web server.

XHR methods and properties

Method	**Description**
abort()	Cancel the current request
getResponseHeader(headerName)	Returns the value of the specified HTTP header
getAllResponseHeaders()	Returns the complete set of HTTP headers as a string
open(method, URL) open(method, URL, async) open(method, URL, async, userName) open(method, URL, async, userName, password)	Specifies the method, URL and other optional parameters for the request The method parameter can be a any of the HTTP methods "GET", "POST','HEAD","PUT" e.t.c The URL specifies the location of the resource The async parameter specifies whether the request should be handled synchronously or asynchronously. If "true", the request is handled asynchronously and script execution continues after send(). If "false", the request is handled synchronously and script execution waits for a response.
send(content)	Send the request
setRequestHeader(name, value)	Adds a name/value pair to the HTTP request header
onreadystatechange	Specifies a reference to the event handler that will be called on every state change

readyState	Returns the state of the XHR 0 = uninitialized 1 = open 2 = sent 3 = receiving 4 = complete
responseXML	Returns the response as XML
responseText	Returns the response as string
status	Returns the HTTP status code
statusText	Returns the HTTP status as string

1.2.4 Asynchronous Request and Response with XHR

To send an asynchronous request from a browser you first instantiate an XMLHttpRequest object. How the XHR object is instantiated varies by browser vendor (as shown in the following Sample 2). Once you have a valid XHR object, you call open() to specify a new HTTP GET or POST to a particular URL. The first open() parameter defines the type of HTTP request, the second parameter is the requested URL and the last parameter specifies the request as synchronous or asynchronous. A callback function is assigned to the onreadystatechange property and the callback function is called on every state change. Finally, calling send() will make the HTTP request to the HTTP server. Following is the code needed to send a HTTP request using XHR

```
XmlHttp = new XMLHttpRequest();
XmlHttp.open("GET", "MyData.txt", true);
XmlHttp.onreadystatechange = Update;
XmlHttp.send(null);
```

As the HTTP request state changes the callback function is called. The readyState property can be used to determine if the request has been completed. The status property can be used to determine if the HTTP request completed without any errors. Following is the code needed to process a HTTP response using XHR

```
function Update()
{
      if (XmlHttp.readyState == 4)
     {
         if (XmlHttp.status == 200 )
        {
             alert(XmlHttp.responseText);
        }
     }
}
```

1.2.5 XHR and ASP.NET

Now that you know how to send HTTP request and process HTTP response using XHR, see how a few simple ASP.NET applications that demonstrate retrieving and processing data from text, XML and JSON:

1. Get a text file and display it on the browser,
2. Get XML data and process the XML document
3. Get JSON data and update the DOM and apply some CSS.

Sample 1: Getting data using XmlHttpRequest

While the page is loading, GetXmlHttpRequest() is called automatically to instantiate the XmlHttp ActiveX object. The GET request to "MyData.txt" follows, then the callback is hooked up and finally send() makes the actual server request. When the Callback() function fires the responseText member of XmlHttp contains the text data.

```
//------------
//GetData.aspx
//------------

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

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

<html  xmlns="http://www.w3.org/1999/xhtml"  >

<script  type="text/javascript"  language="javascript">
var XmlHttp = false;

GetXmlHttpRequest();
if (XmlHttp)
{
   XmlHttp.open("GET", "MyData.txt", true);
   XmlHttp.onreadystatechange = Callback;
   XmlHttp.send(null);
}

function  Callback()
{
  if (XmlHttp.readyState == 4)
  {
     if (XmlHttp.status == 200 )
     {
 //responseText returns the response as a String
 alert(XmlHttp.responseText);
     }
     else
     {
         alert("Requested resource not found");
     }
  }
}

//This will only work in IE as other browsers use a different way to
//instantiate an XMLHttpRequest object.
```

```
// (see sample 2 for a more
//   generic method that works in different browsers)
function   GetXmlHttpRequest()
{
  try
  {
      XmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (e)
  {
       alert("Failed to get XmlHttpRequest");
      XmlHttp = false;

  }
}
</script>

<head  runat="server">
     <title>GetData using XmlHttpRequest</title>
</head>
<body>
     <form id="form1" runat="server">
    <div>
    </div>
    </form>
</body>
</html>
```

Sample 2: Getting XML data using XmlHttpRequest

The OnClientClick button event is the trigger in this example. The differences from the previous example are:

- The GetXmlHttpRequest() method is more generic and allows for other browsers than IE to instantiate the XmlHttp object.
- The callback can return either a the response text as before or a "responseXML" document element.

```
//  ---------------
//  GetXmlData.aspx
//  ---------------

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

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

<html  xmlns="http://www.w3.org/1999/xhtml"  >

<script  type="text/javascript"  language="javascript">
var XmlHttp = false;

GetXmlHttpRequest();

function  OnClick()
{
    if (XmlHttp)
    {
        XmlHttp.open("GET", "MyData.xml", true);
        XmlHttp.onreadystatechange = Callback;
```

```
       XmlHttp.send(null);
    }
}

function  Callback()
{
    if (XmlHttp.readyState == 4)
   {
       if (XmlHttp.status == 200 )
      {
          //responseXML returns the response as XML
           var xmldoc = XmlHttp.responseXML.documentElement;
          var text = XmlHttp.responseText;
          //Use DOM to get the div and display the XML data in the div
           var div1 =  document.getElementById("Div1");
         div1.innerText = text;
          alert("Done.");
     }
     else
     {
        alert("Requested resource not found");
     }
   }
}

//A generic way to instantiate an XMLHttpRequest object that will work
//with most browsers
function   GetXmlHttpRequest()
{
  try
  {
      XmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
  }
  catch (e)
  {
      GetIEXmlHttpRequest();
  }
}

function   GetIEXmlHttpRequest()
{
  try
  {
      //Internet Explorer
      XmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
  }
  catch (e)
  {
       GetMozillaXmlHttpRequest();
  }
}

function   GetMozillaXmlHttpRequest()
{
   try
     {
         //Mozilla browsers
         XmlHttp  = new XmlHttpRequest();
     }
     catch(ex)
     {
```

```
         alert("Failed to get XmlHttpRequest");
        XmlHttp = false;
     }
}
</script>

<head  runat="server">
     <title>Get XML data using XmlHttpRequest</title>
</head>
<body>
     <form id="form1" runat="server">
      <asp:Button ID="Button1" runat="server" Text="Get Data"
       OnClientClick="OnClick()" />
    <div id="Div1">
    </div>
    </form>
</body>
</html>
```

Sample 3: JavaScript, XHR, CSS, DOM, JSON at work (AJAX!)

This sample shows AJAX in action. Here we are using XHR to make asynchronous HTTP request and the response returned as JSON. The JavaScript eval() method parses JSON and returns JavaScript objects. We use the DOM to access the div element and use CSS to apply some style.

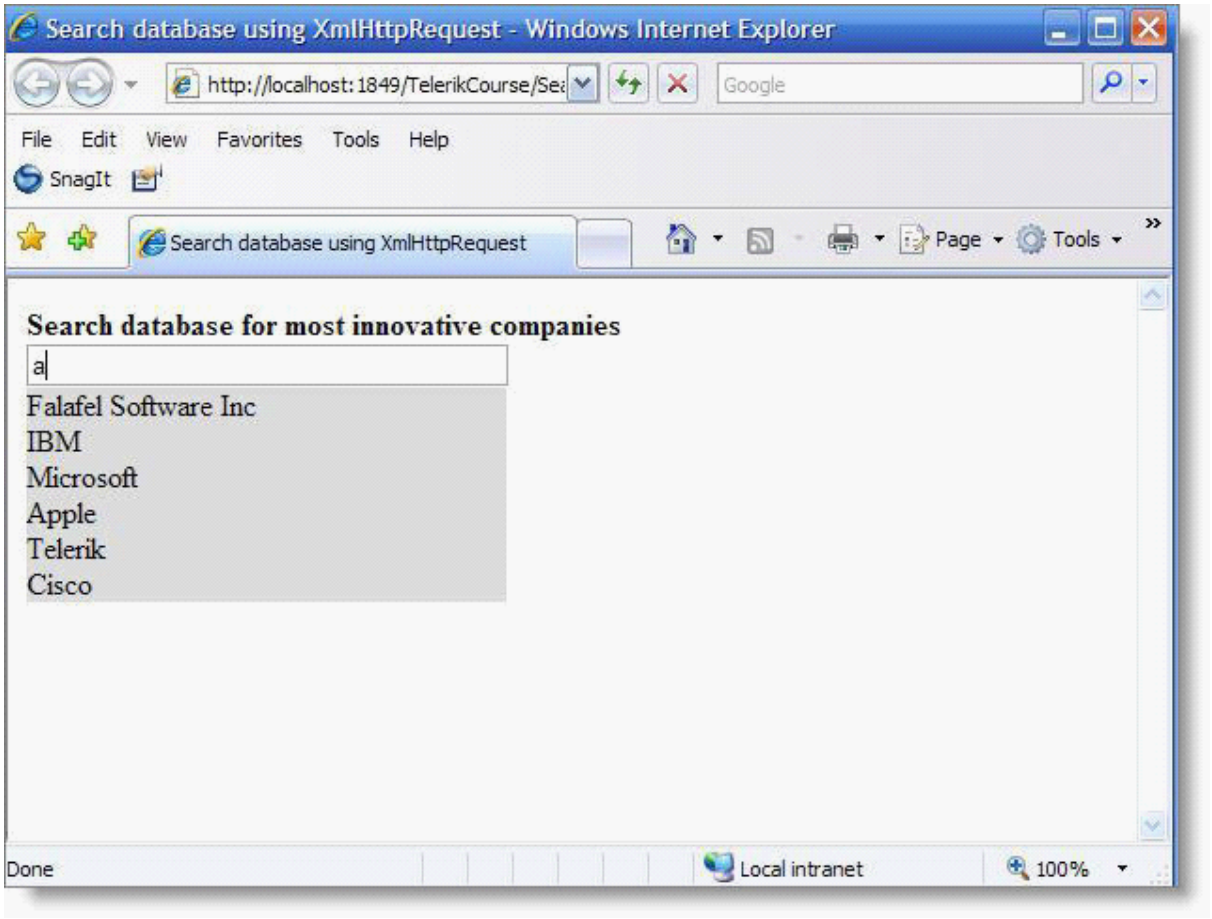
```
//  --------------
//  SearchDB.aspx
//  --------------

. . .

<html   xmlns="http://www.w3.org/1999/xhtml">

<script  type="text/javascript"  language="javascript">
var XmlHttp = false;
var  searchURL  =  "DBAccess.aspx?searchKey=";

GetXmlHttpRequest();

function  GetData(searchKey)
{
     if (searchKey.length > 0 )
    {
         var requestURL = searchURL + searchKey;
        if (XmlHttp)
        {
            XmlHttp.open("GET", requestURL, true);
            XmlHttp.onreadystatechange = ProcessResponse;
           XmlHttp.send(null);
        }
    }
    else
    {
         var resultsDiv = document.getElementById("DivSearchResults");
        resultsDiv.className ='hide';
    }
}

function  ProcessResponse()
  {
     if (XmlHttp.readyState == 4)
```

```
    {
        if (XmlHttp.status == 200 )
       {
           //Convert JSON to JavaScript objects
           eval("var res = " + XmlHttp.responseText);
          var resulttext = "";

           for(var i=0; i < res.SearchResults.Rows.length; i++)
         {
              resulttext += res.SearchResults.Rows[i].Value + "<br>";
         }
          if ( resulttext.length > 0 )
         {
               var resultsDiv = document.getElementById("DivSearchResults");
             resultsDiv.className ='show';
             resultsDiv.innerHTML = resulttext;
         }
       }

    }
  }
. . .

</script>

<head  runat="server">
     <title>Search database using XmlHttpRequest</title>
    <style type="text/css">
    .hide
    {
       display:none;
    }
    .show
    {
      display:block;
      width:255px
    }
    #DivSearchResults
    {
       background-color:silver;
    }
    </style>
</head>
<body>
     <form id="form1" runat="server">
        <div>
             <asp:Label ID="Label1" runat="server" Font-Bold="True"
            Text="Search database for most innovative companies">
           </asp:Label>
          <br />
           <input type="text" id="keyword"
               autocomplete="off"
               onkeyup="GetData(this.value);"
               style="width: 250px" />
        </div>
         <div id="DivSearchResults">
        </div>
    </form>
</body>
</html>
```

```
//  -------------
//  DBAccess.aspx
//  -------------
<%@  Page  Language="C#"  AutoEventWireup="true"  CodeFile="DBAccess.aspx.cs"
     Inherits="DBAccess" %>
```

```
//  ----------------
//  DBAccess.aspx.cs
//  ----------------

using  System;
using  System.Data;
using   System.Configuration;
using   System.Collections;
using  System.Web;
using   System.Web.Security;
using  System.Web.UI;
using   System.Web.UI.WebControls;
using    System.Web.UI.WebControls.WebParts;
using   System.Web.UI.HtmlControls;

using  System.Text;

public  partial  class  DBAccess  :  System.Web.UI.Page
{
     protected  void  Page_Load(object  sender,  EventArgs  e)
    {
          String searchKey = Request.QueryString["searchKey"];
         if (searchKey.Length > 0)
        {
             //Do DB search here

             //Return the data fetched from DB as JSON
              //For simplicity the results are hardcoded here.
              StringBuilder strB = new StringBuilder();
               strB.Append("{\"SearchResults\": {\"Rows\": [");
               strB.Append("{\"Value\":\"" + "Falafel Software Inc" + "\"},");
              strB.Append("{\"Value\":\"" + "IBM" + "\"},");
               strB.Append("{\"Value\":\"" + "Microsoft" + "\"},");
               strB.Append("{\"Value\":\"" + "Apple" + "\"},");
               strB.Append("{\"Value\":\"" + "Telerik" + "\"},");
               strB.Append("{\"Value\":\"" + "Cisco" + "\"}");
              strB.Append("]}}");

              Response.Write(strB.ToString());

        }
    }
}
```

The user experience is that as characters are typed in, a list of matching companies dynamically appears. A very slick effect...



Search results AJAX'd back to the client dynamically

1.2.6 AJAX Frameworks

Without AJAX Frameworks, developing real-world AJAX applications would be tedious and time consuming as you need good exposure to JavaScript, DOM, CSS, XML, JSON or other data remoting formats and a good understanding of different browser capabilities. AJAX frameworks ease the work for web developers by offering client-side JavaScript functions to send requests to the server and server-side processing that handles sending responses back to the browser. There are several AJAX Frameworks available today and one such framework for .NET is the Telerik RadAjax framework.

The Telerik RadAjax framework:

1. Allows you to AJAX enable any existing application or new application without any major changes to the application.
2. Allows application developers to focus on the server side application logic and eliminate the need to write and debug tons of JavaScript.
3. Increases developer productivity by using simple drag and drop controls.
4. **Preserves the ASP.NET page life cycle**: the AJAX-enabled controls don't really know that they are working in AJAX mode rather than Postback mode.
5. **Handles ASP.NET ViewState** and **EventValidation**: unlike other AJAX Frameworks

RadAjax updates the ViewState of the page after the AJAX request in order to keep it in a consistent state regardless of the request (AJAX callback or Postback)

6. **Persists form values**: all form values are sent to the server for processing during the AJAX request and are persisted between AJAX requests. All form controls can be referenced in the code-behind as if a postback occurred. This feature guarantees that your AJAX-enabled applications will not be limited to mere rendering of controls with AJAX, but can cover even the most complex scenario.
7. **Supports ASP.NET Client Validation**: RadAjax controls support server- and client-side validation implemented with the standard ASP.NET validation controls.
8. **Search-engine optimization**: RadAjax controls automatically detect and handle the web crawlers (search engines) and render alternative content, suitable for indexing.
9. **Microsoft ASP.NET AJAX Native Integration:** ASP.NET AJAX extends .NET 2.0 to provide AJAX capabilities. The Telerik RadAjax framework provides native integration to ASP.NET AJAX, combining ASP.NET AJAX abilities with the functional richness of the Telerik suite.

For a comparison of various AJAX frameworks you can refer to this article by Daniel Zeiss: http://www.daniel-zeiss.de/AJAXComparison/Results.htm.

1.2.7 Summary

You have a seen AJAX under the chrome: the component technologies that make it work, how those technologies work together and a little history about how AJAX addresses web application performance and user experience issues. By describing the basics of AJAX the development limitations come into focus and the need for AJAX frameworks becomes clear. Finally, we looked at some of the Telerik AJAX framework advantages. The following chapter shows the Telerik approach to AJAX web application development in action.

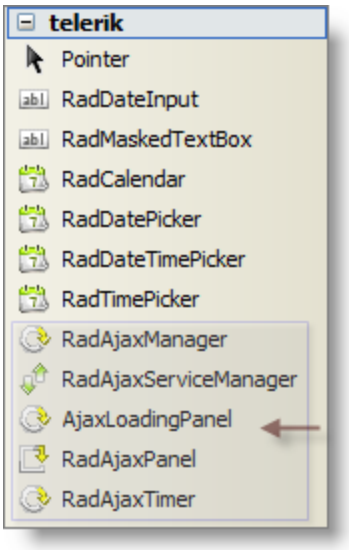
1.3 RadAjax

1.3.1 Getting started

Telerik's RadAjax framework allows you to take a regular ASP.Net application and AJAX-enable it with no coding or changes to your page life cycle. No client side javascript, no server side coding - just drop a component on your form and configure it with the smart tag editor and you are good to go!

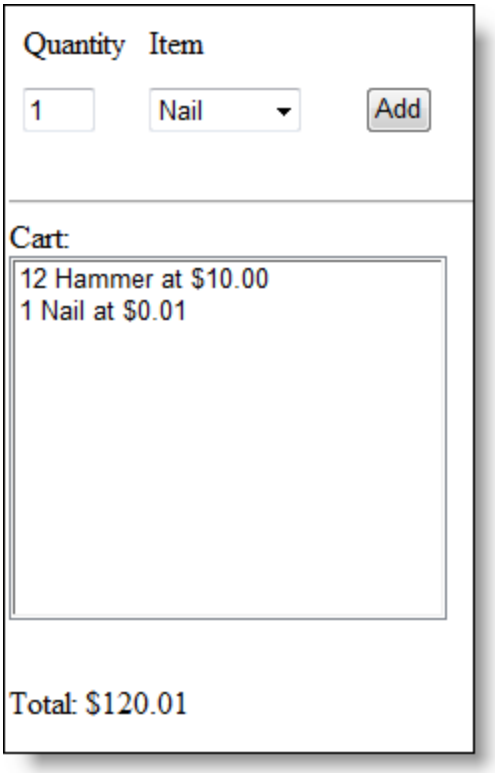
The framework automatically replaces Postbacks with AJAX callbacks. It preservers changes to viewstate, preserves the normal page life cycle, persists form values, reruns client side javascript as needed, and supports standard client side validation. In other words, it works well out of the box with most ASP.Net 1.1 or 2.0 applications, and many third party components.

In this chapter we will explore how to use the components in the RadAjax assembly to perform various AJAX tasks:



RadAjax components in the Toolbox

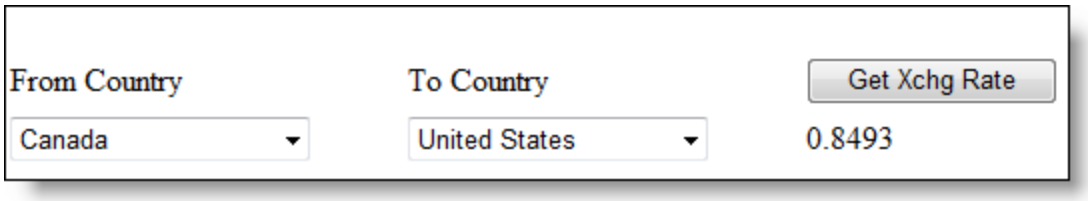
We will see how to use the RadAjaxPanel to create an efficient data entry form that has no Postbacks, refreshes or flicker.



AJAX enabled Data Entry Form

We will also see how to utilize the fine grained control provided by the RadAjaxManager, to update just a portion of the form when some client event happens.

We will explore how to download data from a web service using AJAX:



Calling a Web Service through AJAX

Finally, you will see how to build a timer driven client that periodically refreshes data using AJAX.

1.3.2 Using the RadAjaxPanel

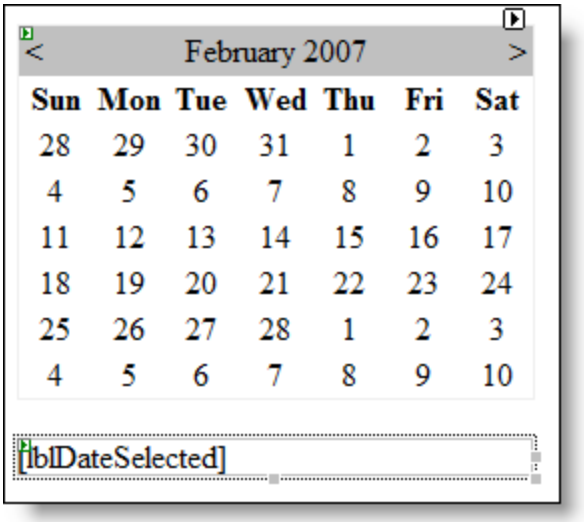
The RadAjaxPanel allows you to AJAX enable a form just by placing the controls that you want to use callbacks instead of Postbacks inside the panel. You can surround everything in the form with an AJAX panel, or just a portion of it.

Lab: A simple RadAjaxPanel scenario

Let's start off with a really simple demonstration of the capabilities of the RadAjaxPanel.

1. Create a new ASP.Net Web Application project named AjaxCalendar.

2. Drop a standard ASP.Net calendar control on the main form

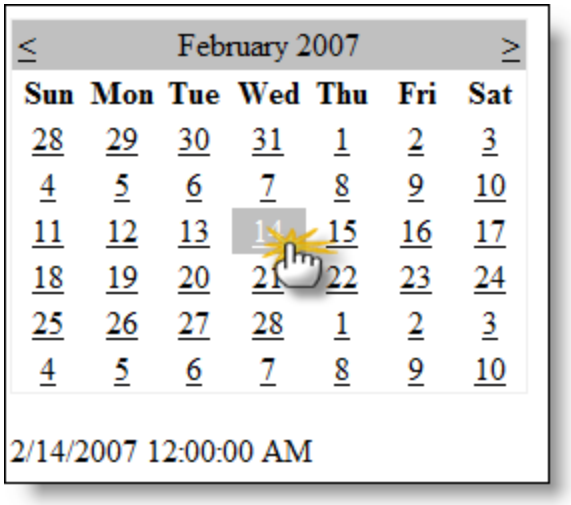


The form

3. Hook up an event handler to the Calendars SelectionChanged event, setting the text of the label to be the currently selected date:

```
C# Example:
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
    lblDateSelected.Text = Calendar1.SelectedDate.ToString();
}

VB Example:
Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, ByVal e As EventArgs)
    lblDateSelected.Text = Calendar1.SelectedDate.ToString()
End Sub
```

Not try running the application. You will see that when you click on a date in the calendar, the date is displayed below. You will also note that each click on a date, or on the arrows that take you to the next or previous month, results in a Postback. The page flickers as it is refreshed:

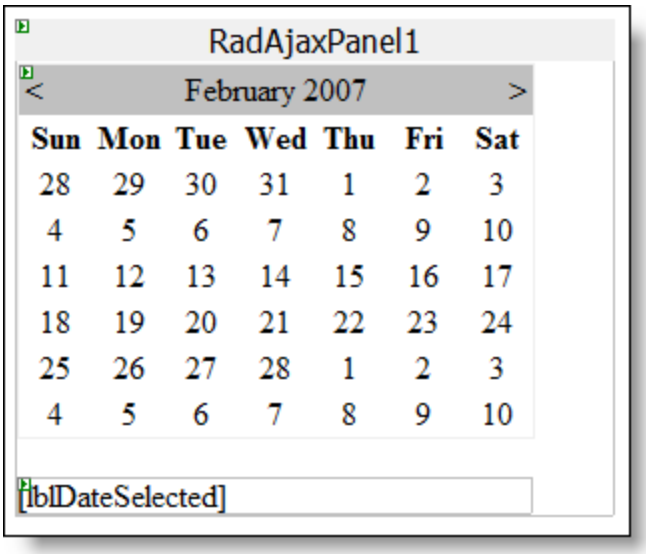


Application output

Now, lets add in the AJAX magic!

5. Drag a RadAjaxPanel from the Toolbox onto the form

6. Drag the Calendar and the Label onto the RadAjaxPanel:



The RadAjaxPanel wraps the Calendar and Label

The markup should now look something like this if you view the source:

```
<form id="form1" runat="server">
  <div>
     <radA:RadAjaxPanel ID="RadAjaxPanel1" runat="server"
       Height="200px" Width="300px">
     <asp:Calendar id="Calendar1" runat="server"
       __designer:wfdid="w1"
        OnSelectionChanged="Calendar1_SelectionChanged">
    </asp:Calendar>
    <BR />
     <asp:Label id="lblDateSelected" runat="server"
       __designer:wfdid="w2"
       Width="258px"></asp:Label>
     </radA:RadAjaxPanel>
  </div>
</form>
```

Note that the only real change is that the tag <radA:RadAjaxPanel> surrounds the other two controls. Now try running the application again...

The magic works! Now the form no longer behaves as a traditional web application, with Postbacks and screen refreshes. In fact, it is now more like a WinForms application! And with no code changes.

Now that your appetite has been whetted, lets look at some more interesting examples...

Properties of the RadAjaxPanel

If you examine the properties of the RadAjaxPanel you will see some that merit further explanation:

Property	Description
EnableAJAX	Normally, this is set to True, and the RadAjaxPanel will do AJAX callbacks. You can also set it to False and the panel will revert to regular Postbacks. This can be useful when debugging, if you suspect that some problem is caused by the AJAX mechanism itself, then you can temporarily turn AJAX off to test the hypothesis
LoadingPanel	The RadAjaxPanel can display an image while the AJAX callback is in progress. This allows the end user to see that something is going on while the callback in progress. See section below on how to hook up the LoadingPanel.
EnableOutsideScripts	Normally when an AJAX callback is performed, javascripts outside of the RadAjaxPanel are not run. Setting this property true allows you to run javascripts that are outside of the updated area

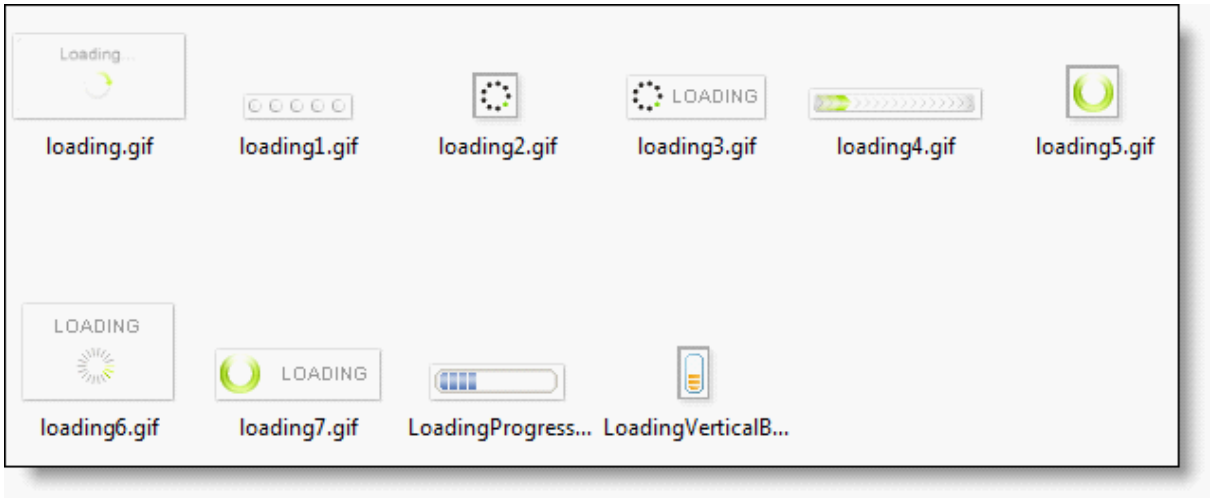
Lab: Hooking up an AjaxLoadingPanel

The AjaxLoadingPanel displays an image while the AJAX call is in progress, providing feedback to the user. It can be used with both the RadAjaxPanel and the RadAjaxManager. You can have a number of different AjaxLoadingPanels on your page with different images and other properties.

Two properties of AjaxLoadingPanel that merit further explanation:

Property	Description
MinDisplayTime	This setting is the minimum time (in milliseconds) that the loading panel is displayed. If the AJAX postback returns before this time has elapsed, the loading panel keeps displaying until MinDisplayTime has passed (the underlying controls are actually updated by the result of the AJAX request while the loading panel is still displaying). Why would you want to slow down your responses this way? It's a matter of user experience, a flickering loading panel that shows up and then disappears immediately may feel more annoying than one that stays around for say half a second and looks smoother.
InitialDelayTime	Sets a value specifying the delay in milliseconds, after which the AjaxLoadingPanel will be shown. If the request returns before this time, the AjaxLoadingPanel will not be shown. This is like the opposite of MinDisplayTime. You can use them in conjunction. For instance, set InitialDisplayTime to 200 ms for the loading panel not to show if the request is serviced faster than 200 ms, and MinDisplayTime to 500 ms to have it show for at least 500 ms if the request does take at least 200ms.

To use the stock loading images, you need to copy the RadControls/Ajax/Skins/Default directory from the Telerik RadControls installation directory (typically something like C:\Program Files\Telerik\r.a.d. controlsQ4 2006\NET2\RadControls\Ajax) to a sub folder of your web application. Here you can see the contents of that folder:



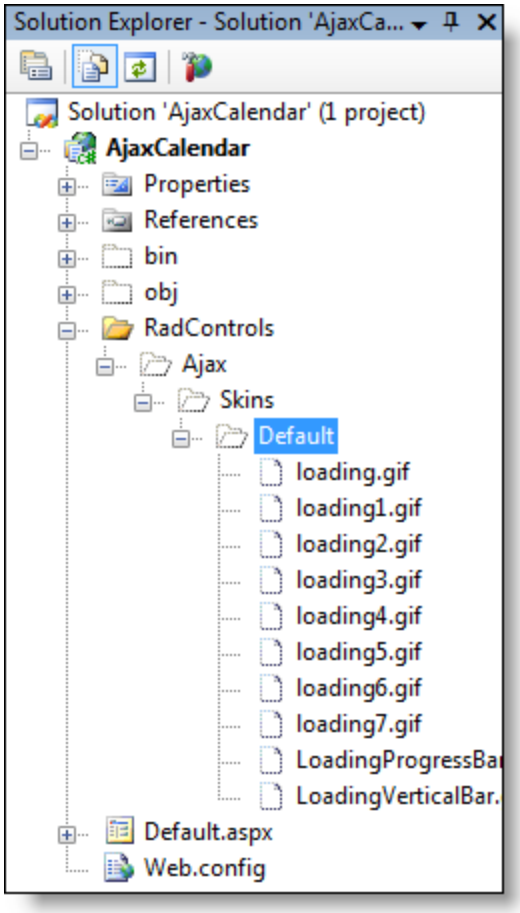
Stock loading images

These are all animated GIF files.

Lets add a loading image to our AjaxCalendar application

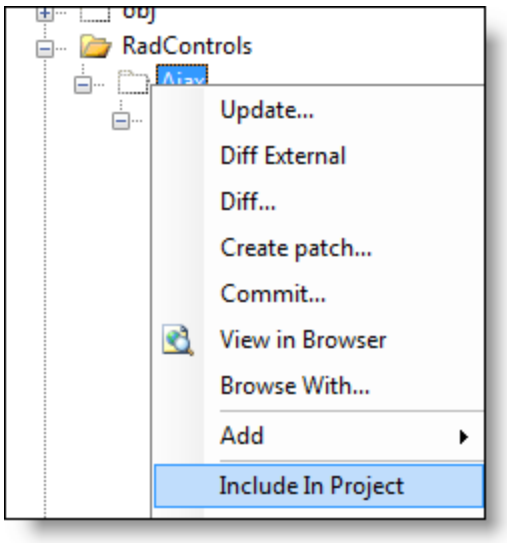
1. Open the RadCalendar project that you created above, and use the Solution Explorer to create a new Project Folder called RadControls.

2. Copy the Ajax/Skins/Default directory from the RadControls directory of your Telerik installation to this location. Click the Show All Files icon at the top of the Solution Explorer. Now you see the files that you added:



The images copied to the correct location

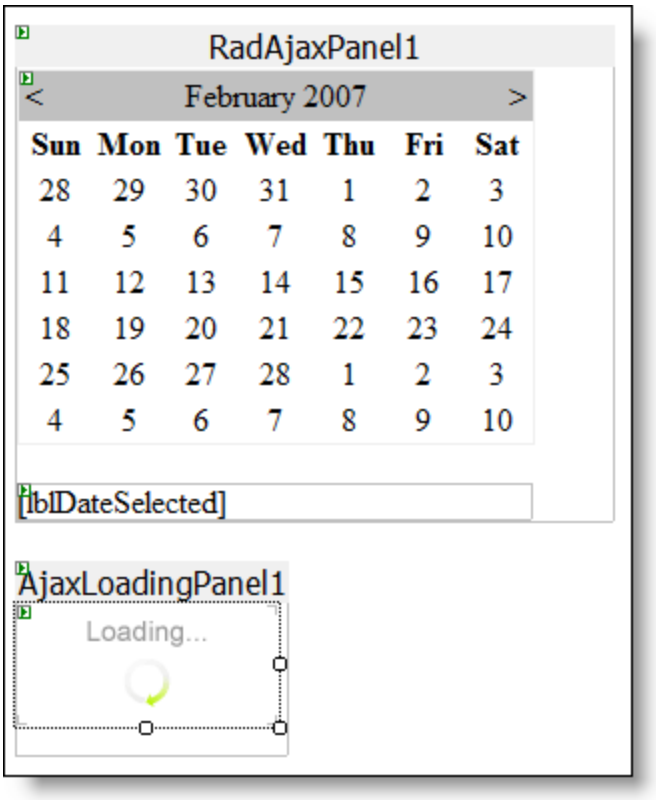
3. Right click the node Ajax and select Include In Project:



Including the files

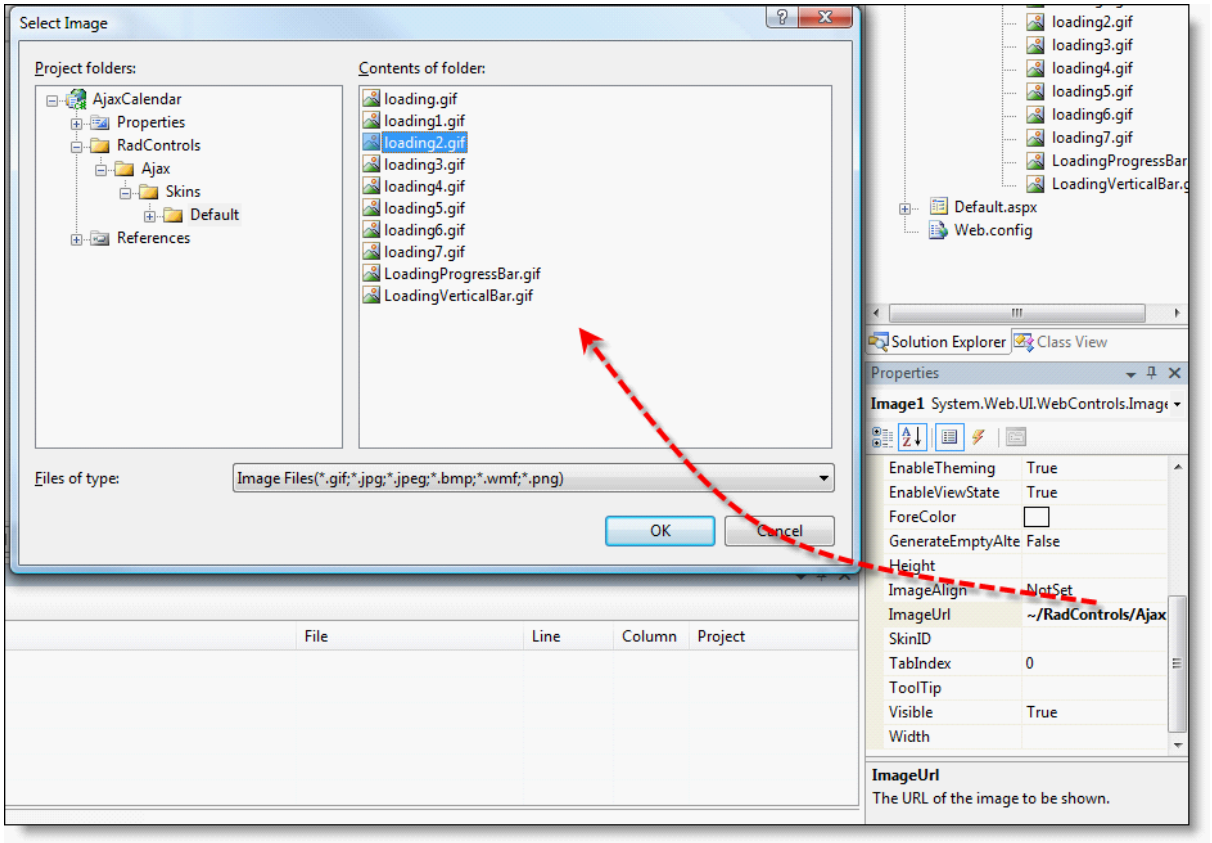
4. Drag an AjaxLoadingPanel onto the main form. Click on the inner Image component contained within the loading panel (this is a regular ASP.Net Image control). It should by default display the animated GIF

below:



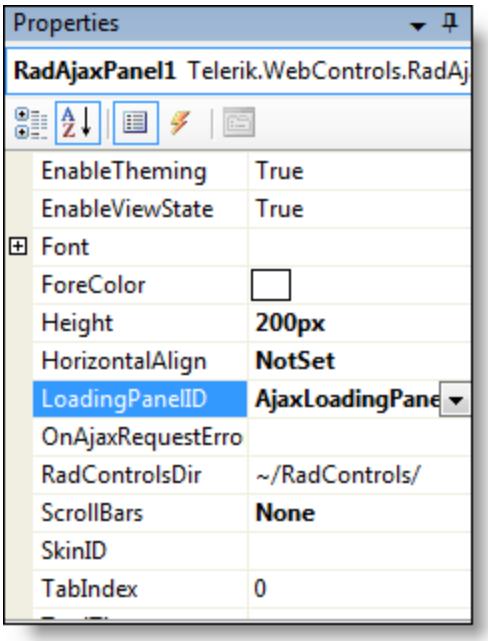
The Loading Panel on the form

5. In the Property Inspector, select whichever animation you want to use by editing the ImageUrl property. It has a picker that allows you to browse the project's folders for the image to use:



Choosing an image

6. Click on the RadAjaxPanel and set its LoadingPanelID to the ID of your AjaxLoadingPanel (AjaxLoadingPanel1):



Setting the Loading Panel

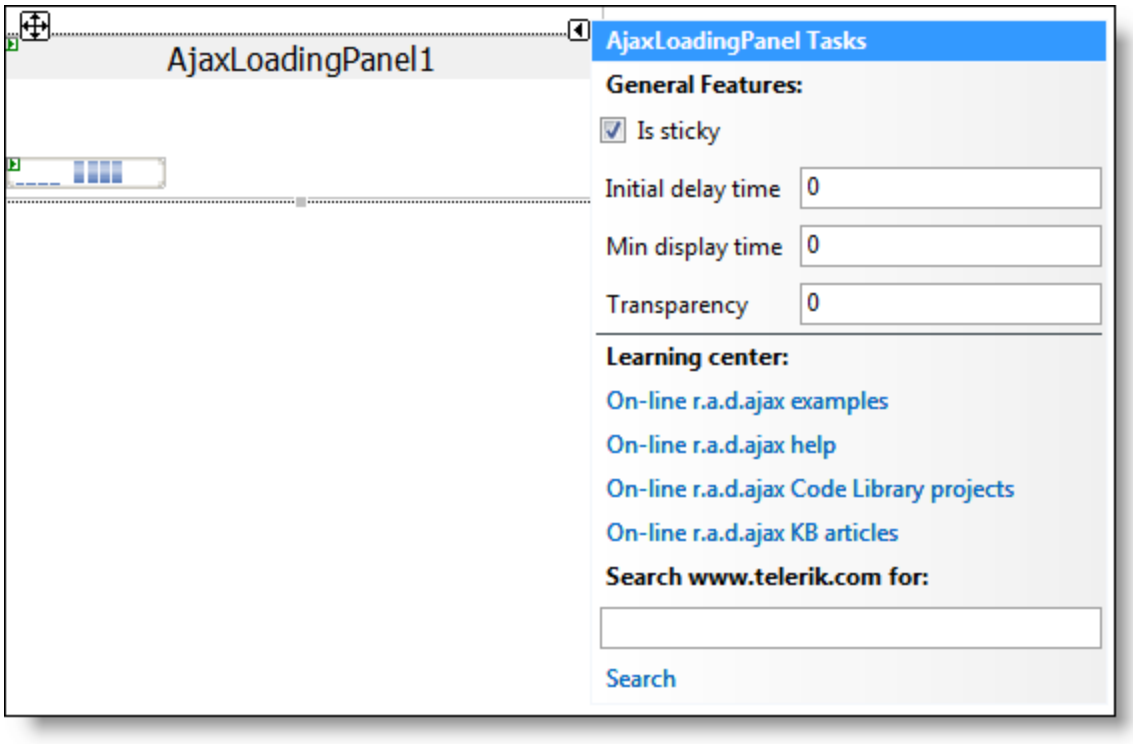
7. The application is actually too fast right now to even see the AjaxLoadingPanel, so to simulate a longer response time, lets add an intentional delay to the page load event handler:

```
C# Example:
protected void Page_Load(Object sender, EventArgs e)
{
    System.Threading.Thread.Sleep(200);
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    System.Threading.Thread.Sleep(200)
End Sub
```

By default, the AjaxLoadingPanel renders on top of the control that initiates the callback (the Calendar in this case). You can set the IsSticky property to make it render where you placed it on the form instead.

The AjaxLoadingPanel renders on top of whatever is rendered beneath it, but you can set the Transparency property to let the underlying control shine through. Initially it is set to 0, meaning no transparency, You can set it to say 50 for a semi-transparent look. Also, if you don't want the spinny to flash by too quickly, you can set it's MinDisplayTime to a number of milliseconds, and it will stay on the screen for at least that amount of time. It's funny - now we have made our application much more responsive and are working on ways for it to seem not too quick! You can also set InitialDelayTime to say 200ms, which will not display the AjaxLoadingPanel at all if the AJAX callback returns faster than this.

These three properties are so commonly used that you will find them on the Smart Tag:

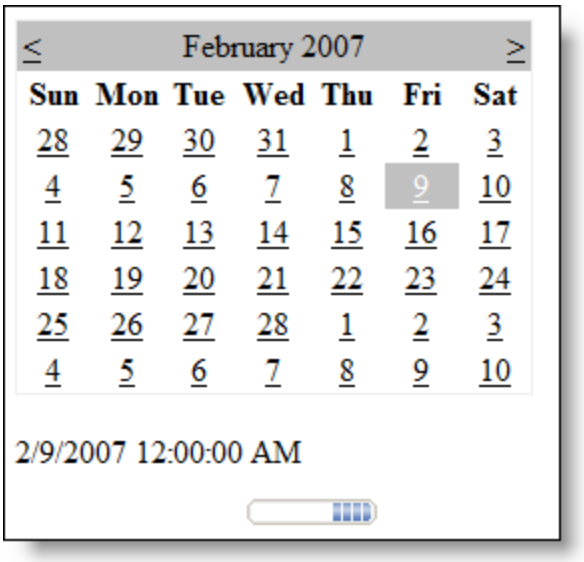


Smart Tag

One final touch: to make the spinny render under the Label, add a couple of BR tags above the Image control in the markup of the Loading Panel:

```
<radA:AjaxLoadingPanel   ID="AjaxLoadingPanel1"   runat="server"
   Height="34px"  HorizontalAlign="Center"
   IsSticky="True"  MinDisplayTime="0"  Width="297px">
  <br />
  <br />
   <asp:Image  ID="Image1"  runat="server"  AlternateText="Loading..."
       ImageUrl="~/RadControls/Ajax/Skins/Default/LoadingProgressBar.gif"  />
</radA:AjaxLoadingPanel>
```

Now run the application again and you will see the AJAX spinny doing it's thing!



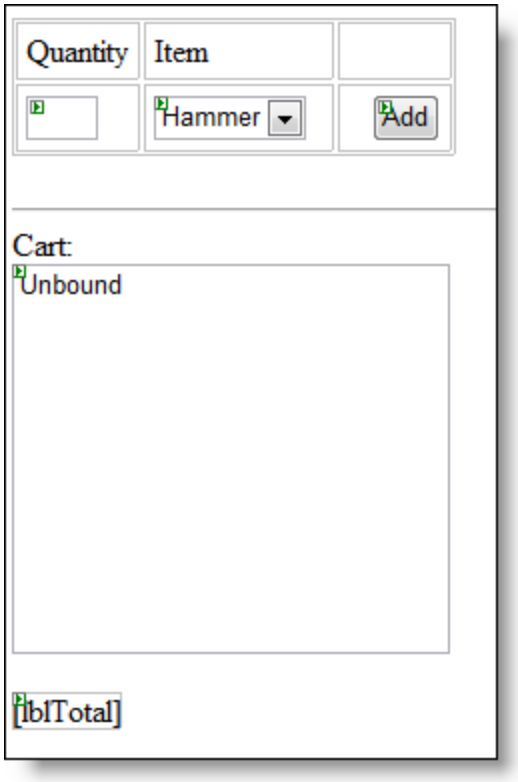
Application running with Loading Panel

Lab: An AJAX enabled Data Entry form

Now that we have seen the basic RadAjaxPanel functionality, lets look at a more complex example. We will create an order entry screen, where the user can select a quantity of an item from a list of available items, and add them to their cart.

1. Create a new Web Application Project called AjaxCart

2. Layout the controls below in a table (for layout) - a TextBox named tbQuantity, a DropDownList ddlItems, a Button pbAdd, a ListBox lbItemsInCart, and a Label lblTotal:



The form layout

3, Add some items to the DropDownList, with prices stored in the Value of each item:

```
<asp:DropDownList  ID=ddlItems  runat=server>
   <asp:ListItem  Text="Hammer"  Value="10"/>
   <asp:ListItem  Text="Nail"  Value="0.01"/>
   <asp:ListItem  Text="Snail"  Value="1"/>
</asp:DropDownList>
```

4.Switch to the code behind file and define a protected property that will be used to store a running total for cart in ViewState:

```
C#  Example:
protected  Decimal  Total
{
    get
    {
         object o = ViewState["Total"];
        if (o == null)
            return 0;
        else
             return (Decimal)o;
    }
    set
    {
        ViewState["Total"] = value;
    }
}

VB  Example:
```

```
Protected Property Total() As Decimal
   Get
        Dim o As Object =   ViewState("Total")
        If o Is Nothing Then
            Return 0
        Else
            Return CType(o, Decimal)
        End If
   End Get
   Set (ByVal Value As Decimal)
         ViewState("Total") = value
   End Set
End Property
```

The total could also have been stored in Session, but storing it in ViewState will serve to illustrate that ViewState is indeed updated in AJAX callbacks.

5. Databind the lblTotal to this property:

```
<asp:Label ID=lblTotal runat=server
   Text='<%# String.Format( "Total: {0:c}", Total ) %>'></asp:Label>
```

6. Make sure the Page calls DataBind() when rendering: add this event handler (which will be automatically hooked up with the Page set to AutoEventWireup="true"

```
C# Example:
protected void Page_PreRender(object sender, EventArgs e)
{
     Page.DataBind();
}

VB Example:
Protected Sub Page_PreRender(ByVal sender As Object, ByVal e As EventArgs)
     Page.DataBind()
End Sub
```

7. Double click the Add button and add an event handler that adds an item to the cart, and updates the running total:

```
C# Example:
protected void pbAdd_Click(object sender, EventArgs e)
{
     int qty = Int32.Parse( tbQuantity.Text );
      Decimal price = Decimal.Parse( ddlItems.SelectedItem.Value );
      lbItemsInCart.Items.Add( new ListItem(
       String.Format( "{0} {1} at {2:c}",
         qty, ddlItems.SelectedItem.Text, price )));
     Total += qty * price;
}

VB Example:
Protected Sub pbAdd_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim qty As Integer =   Int32.Parse(tbQuantity.Text)
    Dim price As Decimal =   Decimal.Parse(ddlItems.SelectedItem.Value)
```

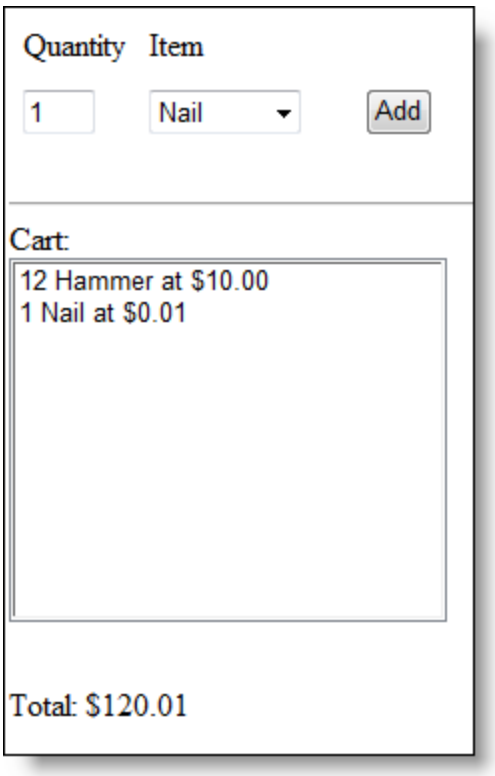
```
     lbItemsInCart.Items.Add(New ListItem( _
       String.Format("{0} {1} at {2:c}", _
        qty, ddlItems.SelectedItem.Text, price)))
    Total += qty * price
End Sub
```

Of course, in a real application we would be fetching the items from a database and storing the whole cart in session, but this is just to illustrate the concepts.

8. To illustrate that client side validators also work in AJAX, add a RequiredFieldValidator and a CompareValidator to validate the tbQuantity TextBox:

```
<asp:RequiredFieldValidator ID="rfvQty" runat="server" ControlToValidate="tbQuantity"
    Display=dynamic ErrorMessage="Must specify quantity"></asp:RequiredFieldValidator>
<asp:CompareValidator ID="cvQty" runat="server" ControlToValidate="tbQuantity"
   Display=dynamic ErrorMessage="Must be postitive whole number" Type=Integer
    ValueToCompare="0" Operator=GreaterThan></asp:CompareValidator>
```

Before adding in AJAX, let's run the application to check that it is working. Try adding some items and verify that the total is being updated:



The data entry form running with no AJAX

The application is now fully functional, but it flickers and refreshes as you add items to the cart. Now lets bring it into the new world of AJAX!

9. Add a RadAjaxPanel that wraps the contents of the form. I find the easiest way to add in a RadAjaxPanel to an existing form is to drag and drop it onto the from the toolbox (which also inserts the necessary TagPrefix in the form and adds a project reference to the Rad Ajax assembly), and then switch to Source view and manually move the start and end tags to where they should be:

```
<form id="form1" runat="server">
```

```
      <radA:RadAjaxPanel ID="RadAjaxPanel1" runat="server" Height="200px" Width="300px">
        <div>
             <table cellpadding="5" border="0">
             ...
        </div>
     </radA:RadAjaxPanel>
</form>
```

10. Now run the Ajax enabled version of the cart. Note that Viewstate and validation still work as they should.

When to use the RadAjaxPanel

Using the RadAjaxPanel is so easy that it is tempting to just wrap the contents of each form in one, and bingo, your application is completely AJAX enabled! Unfortunately, that is not always the best solution.

The problem is that *everything* that is inside the panel is re-rendered for every AJAX callback, whether it changed or not. Our examples so far have been very small, and so the time it takes to re-render the page on the client side is negligible. But for more complex forms, with one or more grids, the client side redrawing of the page becomes noticeable. If all you need to do is to update the contents of one or two controls, then the RadAjaxPanel actually slows your application down instead of speeding it up.

You can actually have several RadAjaxPanels on your page for different areas, and that can work if each area contains the control that does the Postback that you want to convert to an AJAX callback, plus the controls that are updated by the callback, and each area is a self contained group of controls that affect each other. But more often, the initiating control is in one region of the page (maybe a button or an AutoPostback DropDownList), and the data to be updated is somewhere else (say a grid).

In all these more complex case, the solution is to use the RadAjaxManager instead of a RadAjaxPanel. This is the topic of the next section.

1.3.3 Using the RadAjaxManager

The RadAjaxManager can be used instead of the RadAjaxPanel to more finely control which controls update which other controls on a page. It maintains a list of AjaxSettings, where each AjaxSetting designates one control as an *Initiator*, and one or more controls as *Targets*:

- The Initiator is the control that causes a Postback that you wish to convert to an AJAX callback, for instance a Button, or some control that has it's AutoPostback property set.
- The Targets are the controls that are to be re-rendered after the callback (presumably because the callback changes them). The target can be a list of individual controls, or containers, like ASP Panels, Grids, or even divs.

A requirement for this to work is that the controls that are specified in the AjaxSettings are rendered with an ID attribute in the client side HTML. Thus you can not have a User Control as the target, as it does not actually render itself with an ID attribute. However, there are other ways to accomplish this. Once we have looked at the basic usage scenarios we will look into more tricky cases like User Controls, Master Pages and dynamically created controls.

You may only have one RadAjaxManager per page. RadAjaxManagers and RadAjaxPanels are supposed to be used exclusively; don't combine them on one page. The RadAjaxManager can do everything the RadAjaxPanel can do, so there is no need to combine them anyway.

Using the RadAjaxManager in the Designer

For simple scenarios, the RadAjaxManger can configure the AJAX interactions on your page just by dropping it on your form and using the Smart Tag to set up your AjaxSetting: no coding required.

Lab: A Date Range Picker

Let's create a simple AJAX enabled date range picker. The picker will have a DropDownList of named date ranges like Last Week, Last Month, Last Year and Today, and two RadDatePickers that can be used to specify the start and end date of the date range. When you select a named date range, the Start and End date will be populated accordingly.

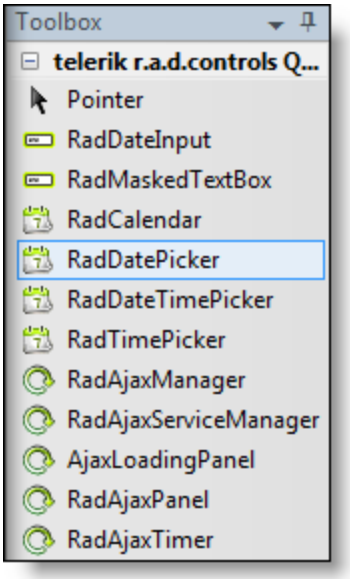
1. Create a new Web Application Project named AjaxDateRange

2. Switch to Source view of the main form and add a simple table to structure the layout:

```
<table  border="0"  cellpadding="3">
    <tr>
         <td  width="150px"></td>
         <td  width="110px"></td>
         <td  width="110px"></td>
    </tr>
</table>
```

3. Switch to design mode and drag a standard ASP DropDownList from the Toolbox into the first cell. Name it ddlDateRange.

4. Locate the Telerik RadDatePicker component in the Toolbox:



The RadDatePicker in the Toolbox

5. Drop two RadDatePickers into the second and third cells, and name them dpStart and dpEnd:

The controls laid out

6. Set the AutoPostback property of the DropDownList, and add these items:

```
<asp:DropDownList ID="ddlDateRange" runat="server" Width="150px" AutoPostBack="True">
    <asp:ListItem Value="0">Today</asp:ListItem>
    <asp:ListItem Value="1">Last Week</asp:ListItem>
    <asp:ListItem Value="2">Last Month</asp:ListItem>
    <asp:ListItem Value="3">Last Year</asp:ListItem>
</asp:DropDownList>
```

7. Double click the DropDownList to add an Event Handler for the SelectedIndexChanged event:

```
C# Example:
protected void ddlDateRange_SelectedIndexChanged(object sender, EventArgs e)
{
     SetDates();
}

private void SetDates()
{
    switch (ddlDateRange.SelectedValue)
    {
        case "0":
              dpStart.SelectedDate = DateTime.Today;
              dpEnd.SelectedDate = DateTime.Today;
            break;
        case "1":
              dpStart.SelectedDate = DateTime.Today.AddDays(-7);
              dpEnd.SelectedDate = DateTime.Today.AddDays(-1);
            break;
        case "2":
              dpStart.SelectedDate = DateTime.Today.AddMonths(-1);
              dpEnd.SelectedDate = DateTime.Today.AddDays(-1);
            break;
        case "3":
              dpStart.SelectedDate = DateTime.Today.AddYears(-1);
              dpEnd.SelectedDate = DateTime.Today.AddDays(-1);
            break;
    }
}

VB Example:
Protected Sub ddlDateRange_SelectedIndexChanged(_
  ByVal sender As Object, ByVal e As EventArgs)
    SetDates()
End Sub

Private Sub SetDates()
    Select Case ddlDateRange.SelectedValue
        Case "0"
              dpStart.SelectedDate = DateTime.Today
              dpEnd.SelectedDate = DateTime.Today
            Exit Sub
        Case "1"
              dpStart.SelectedDate = DateTime.Today.AddDays(-7)
              dpEnd.SelectedDate = DateTime.Today.AddDays(-1)
            Exit Sub
        Case "2"
              dpStart.SelectedDate = DateTime.Today.AddMonths(-1)
              dpEnd.SelectedDate = DateTime.Today.AddDays(-1)
            Exit Sub
```

```
        Case "3"
              dpStart.SelectedDate = DateTime.Today.AddYears(-1)
              dpEnd.SelectedDate = DateTime.Today.AddDays(-1)
            Exit Sub
    End Select
End Sub
```

8. Also make sure that the SetDates method is called on Page Load the first time the page is loaded:

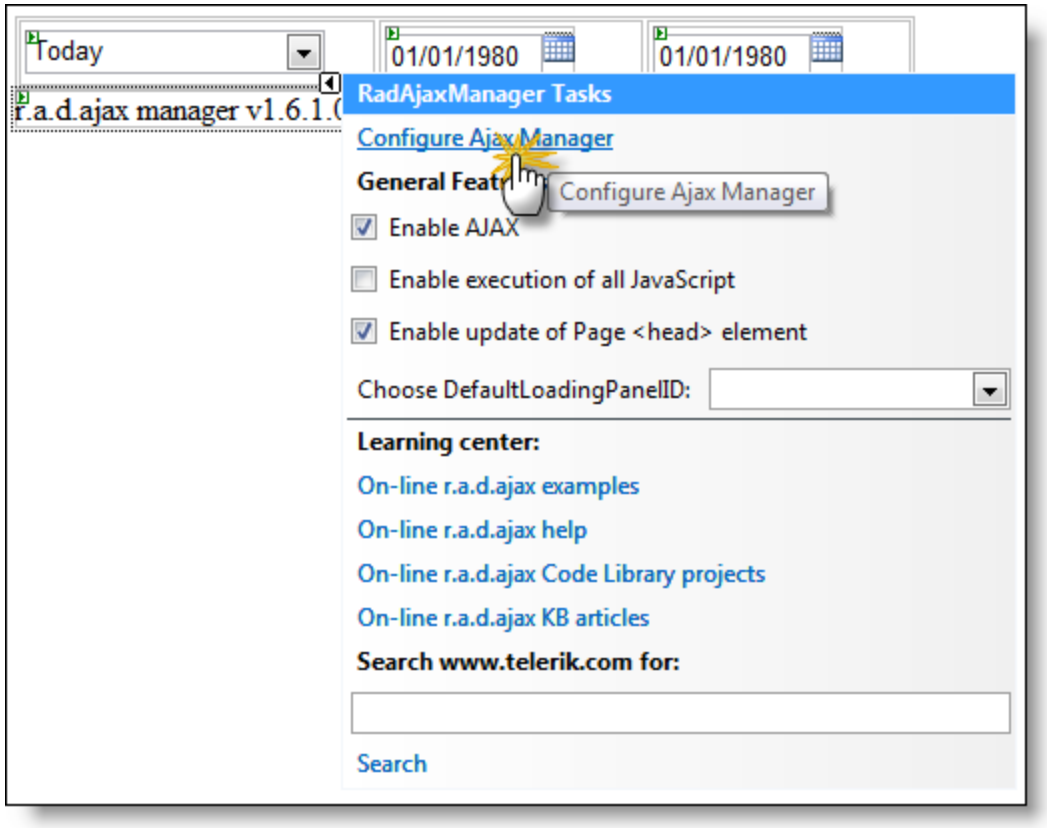
```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
     if (!IsPostBack)
         SetDates();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    If Not IsPostBack Then
         SetDates()
    End If
End Sub
```

9. Now try running the application. Notice how each time you select a different named date range from the list, there is a Postback and the dates on the RadDatePickers are change:

The Date Range Picker running

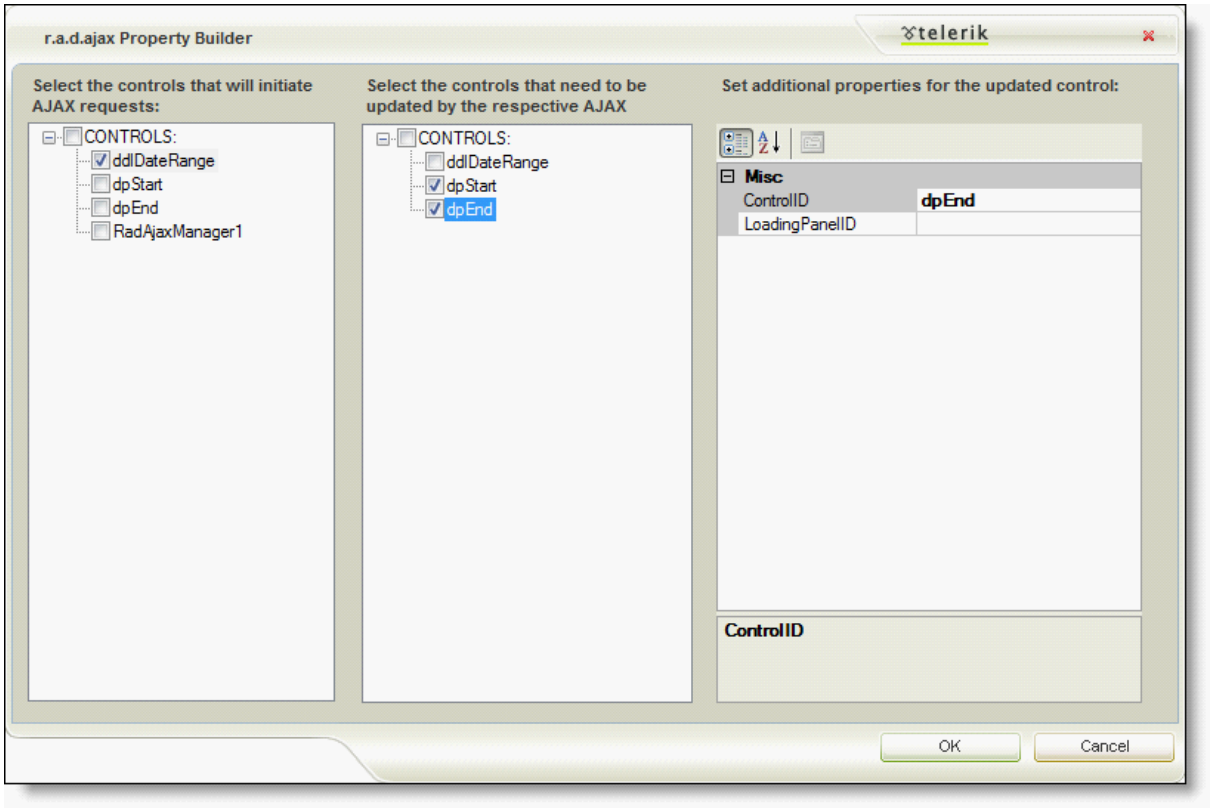
Now we will AJAX enable this application.

10. Drop a RadAjaxManager on to the form, and click the Configure Ajax Manager link on the Smart Tag:



The RadAjaxManager Smart Tag

11. The RadAjax Property Builder is shown. In the first pane, you will see all of the controls that the AJAX manager 'sees' as potential Initiators of AJAX callbacks. Check ddlDateRange. In the second pane, when you click on an item in the first pane, the dialog show you which Targets will be updated when that Initiator fires a callback. Check the dpStart and dpEnd. We will revisit the third pane shortly. Close the dialog.



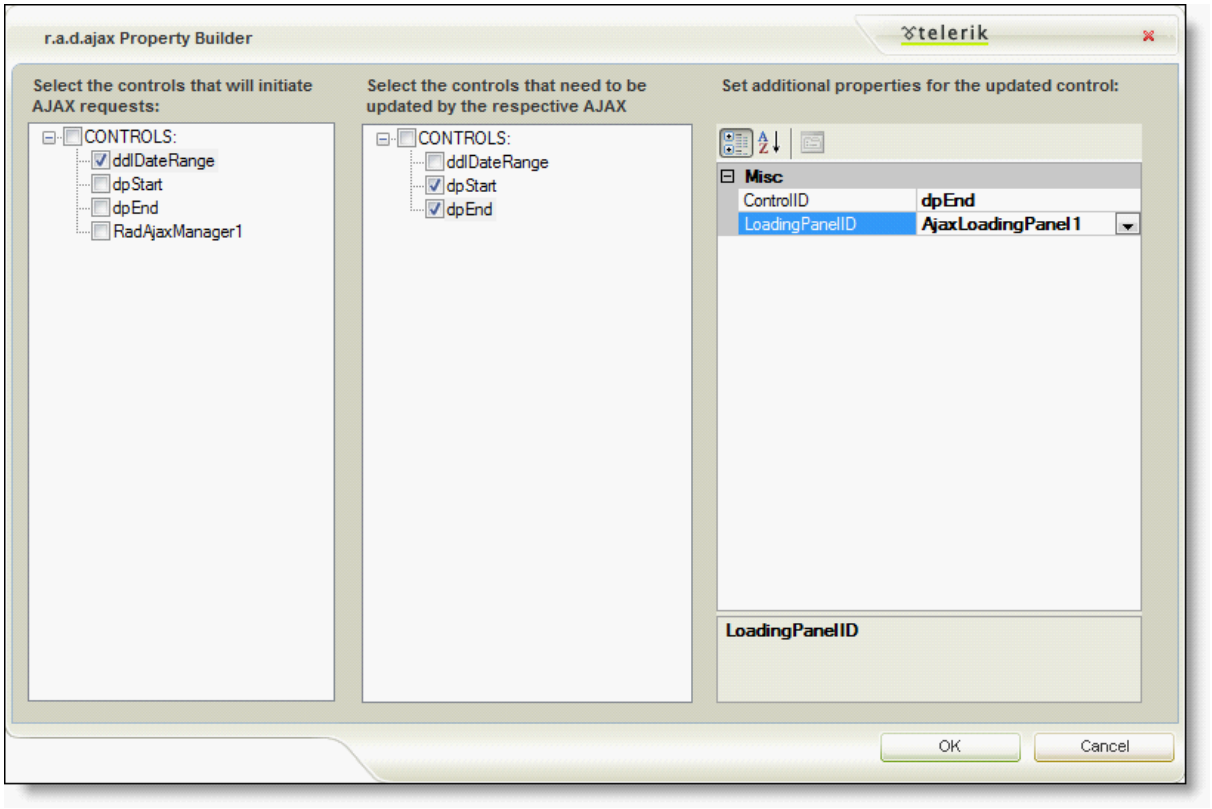
Configuring the Ajax Manager

12. Now run the application again and notice that the Postback has been replaced by a Callback!

The only problem now is that in real world apps where the screen is a little more complex and the callback takes longer, the user has no feedback telling them that something is going on. This can be remedied by adding in one ore more AjaxLoadingPanels.

13. Follow steps 1-4 in the lab "Hooking up an AJAX panel" to copy the standard loading images to your project and add an AjaxLoadingPanel to your form. For the image, select LoadingImage1.gif.

14. Click the Smart Tag on the RadAjaxManager, and click the link to open the Property Builder. Click on dpStart in the second pane, and then in the third pane choose LoadingPanel1 in the drop down list next to the property LoadingPanelID. Repeat for dpEnd. Note that the two targets can use the same instance of AjaxLoadingPanel, you don't need to add two AjaxLoadingPanels to the form.



Configuring the Loading Panels

15. Now run the application again and see the feedback:

Loading panels in place

This kind of feedback becomes very important when you have more complex pages and slower response times.

Lab: Invisible or dynamic controls

Let's keep working on our DateRangePicker and add the capability to Hide the end date when you choose the named date Today:

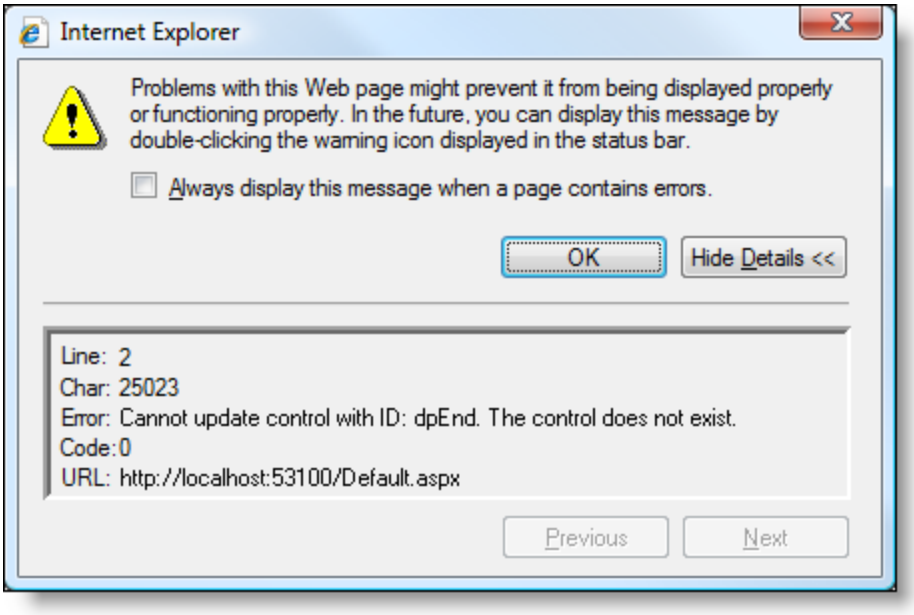
1. Change the SetDates method by adding this line at top of the method:

```
C# Example:
dpEnd.Visible = ddlDateRange.SelectedValue != "0";

VB Example:
dpEnd.Visible = ddlDateRange.SelectedValue <> "0"
```

2. Run the application.

Initially, it looks like this works, the end date is hidden. Now select another named date range, for instance Last Week. Something unexpected happens: the loading panel keeps showing, and the dates

are not updated. You will see an error icon saying there was an error on the page (a yellow warning triangle in the status bar in Internet Explorer). Click to see the error:



Error on page

The Ajax manager is telling us that the control dpEnd does not exist! And in fact, if you view the HTML source rendered to the browser, you will see this is correct. You can see the dpStart, but not the dpEnd. This is because we set the Visible property to False on the dpEnd, so it doesn't render itself. Now, when the AJAX Callback returns, the RadAjaxManager client side code looks for a control with the ID dpEnd in currently rendered HTML of the page, in order to replace it's contents with what was returned by the Callback, but it can't be found!

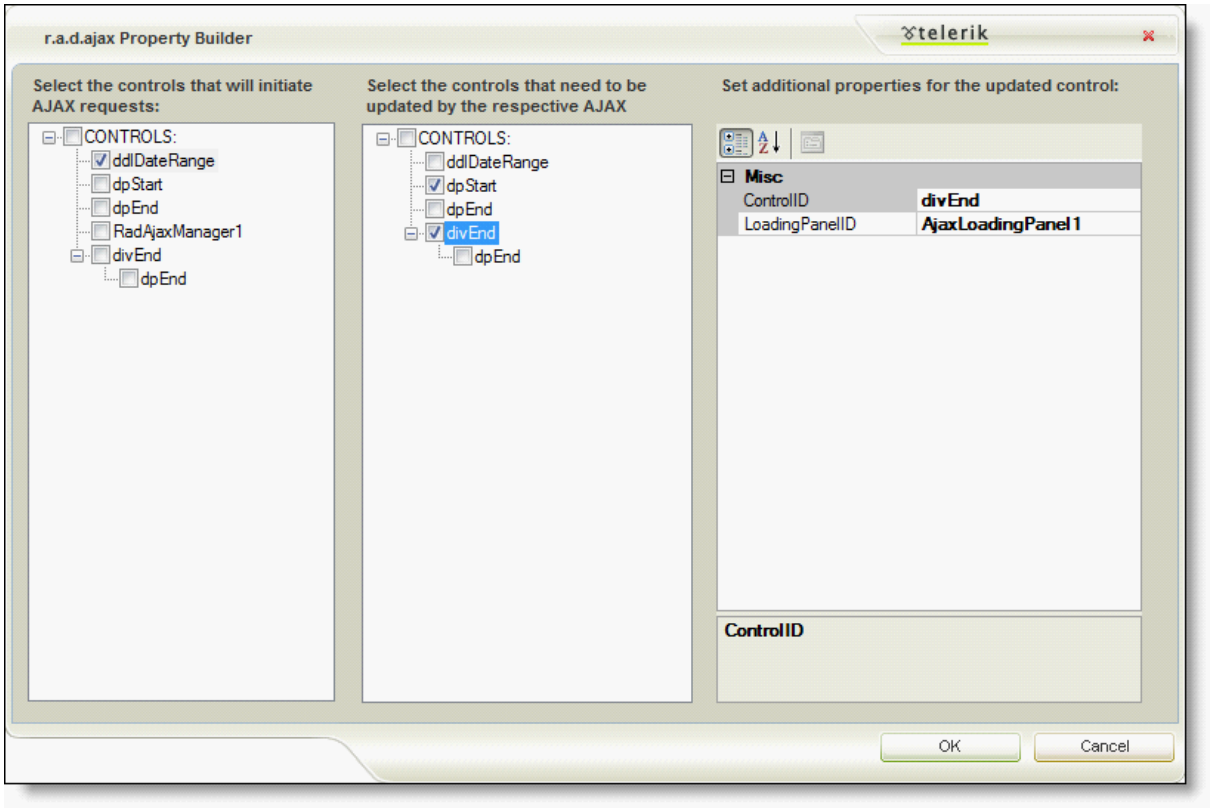
This is a problem that will occur every time the Target of an AjaxSetting is something that did not exist prior to the callback, but that does exist later. This includes controls whose visibility is changed from True to False during the callback, or controls that are dynamically created in the callback.

To get around this, we need to wrap the problematic Target control in something that is always rendered. The control may already be in a container like a Panel that is always there and renders a HTML tag with an ID attribute (you can not use a PlaceHolder control, because it does not render itself with an ID tag in HTML). In that case you may be able to use that container as the Target. Or, if there is no such container, or the container contains too many other things that you don't need to re-render, just add a div or a panel around the control. This is what we will do here:

3. Switch to Source mode for the form and wrap dpEnd in a DIV element as follows (add the runat=Server tag or it wont be visible to the RadAjaxManager designer).

```
<div  id="divEnd"  runat=server>
      <radCln:RadDatePicker  ID="dpEnd"  runat="server"  Width="100px">
          <DateInput  Title=""  PromptChar="  "  TitleIconImageUrl=""
              DisplayPromptChar="_"  CatalogIconImageUrl=""
              TitleUrl=""  Description=""></DateInput>
     </radCln:RadDatePicker>
</div>
```

4. Bring up the property builder for the RadAjaxManager and change the configuration to the following (don't forget to add the AjaxLoadingPanel to the divEnd):



New configuration

Now run the application and it will work again.

Take a moment to inspect the markup generated by the Property Builder:

```
<radA:RadAjaxManager   ID="RadAjaxManager1"   runat="server">
     <AjaxSettings>
          <radA:AjaxSetting  AjaxControlID="ddlDateRange">
             <UpdatedControls>
                  <radA:AjaxUpdatedControl  ControlID="dpStart"
                     LoadingPanelID="AjaxLoadingPanel1"    />
                  <radA:AjaxUpdatedControl  ControlID="divEnd"
                     LoadingPanelID="AjaxLoadingPanel1"    />
             </UpdatedControls>
         </radA:AjaxSetting>
     </AjaxSettings>
</radA:RadAjaxManager>
```

You can see that there is an outer level <AjaxSettings> element, that contains the Initiator in an <AjaxSetting> element, with the ID of the Initiator in its AjaxControlID attribute, and then the targets along with their Loading Panels in the <UpdatedControls> section, each Target represented by an <AjaxUpdatedControl> element. You can edit the markup yourself if you don't want to use the Property Builder. You can of course have any number of Initiators. This will create several <AjaxSetting> sections, one for each Initiator.

Properties of the RadAjaxManager

If you examine the properties of the RadAjaxManager you will recognize them from the RadAjaxPanel:

Property	Description
EnableAJAX	Normally, this is set to True, and the RadAjaxManager will do AJAX callbacks. You can also set it to False and the manager will revert to regular Postbacks. This can be useful when debugging, if you suspect that some problem is caused by the AJAX mechanism itself, then you can temporarily turn AJAX off to test the hypothesis
EnableOutsideScripts	Normally when an AJAX callback is performed, javascripts outside of the Targets updated in the AjaxSettings of the Initiator are not run. Setting this property true allows you to run javascripts that are outside of the updated area (s).

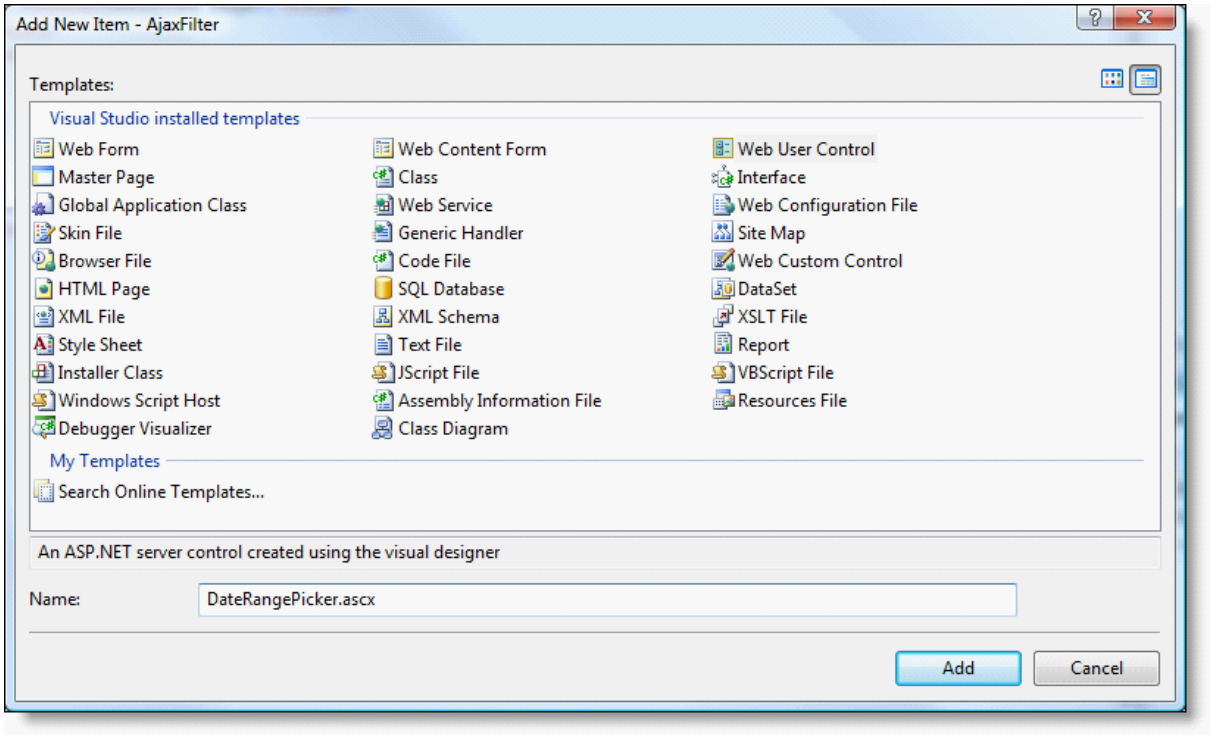
Programmatic usage

Now that we have seen the basic design time usage of the RadAjaxManager, let's turn to some more complex scenarios where you will need to tap in to the run time capabilities of the RadAjaxManager component. One such case is when the Ajax Initiator is inside of a User Control (or some other container that hides it from view: the RadAjaxManager can not "see into" a user control in design time).

Lab: Using the RadAjaxManager with User Controls

To this end, lets redesign the Date Range Picker as a reusable user control:

1. Create a new ASP.Net Web Application Project called AjaxFilter
2. Copy over the Loading images as per steps 1-3 in the section "Lab: Hooking up an Ajax Loading Panel"
3. Right click on the project in the Solution Explorer and choose Add | New Item. Select Web User Control and name it DateRangePicker:

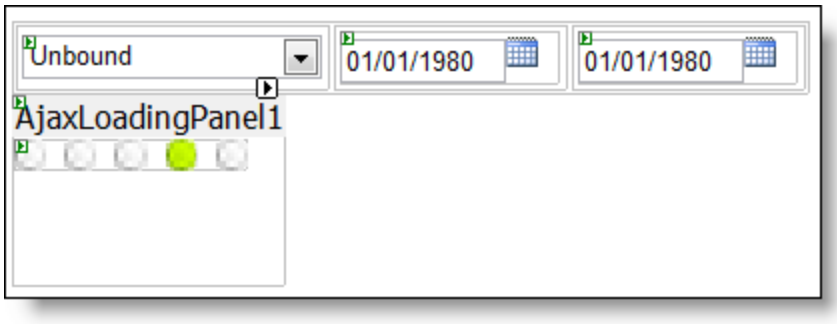


Adding the User Control

4. Switch to Source view for the new DateRangePicker.aspx file, and paste in the following simple table to structure the layout:

```
<table  border="0"  cellpadding="3"  style="width:  450px">
    <tr>
         <td  width="150px"></td>
         <td  width="110px"></td>
         <td  width="110px"></td>
    </tr>
</table>
```

5. Follow steps 3-8 in "Lab: A Date Range Picker" to set up the controls in the picker. Then add an AjaxLoadingPanel to the control and set the Image to Loading1.gif. Here is the end result:



Layout in DateRangePicker control

What about the RadAjaxManager, you may ask? Well, we can't add it into the Web User Control, because the RadAjaxManager needs to go on the page that hosts this control. However, to make the control more self contained, we will add in a public method that can be called by the containing form to

initialize the Ajax interactions.

6. Add this using/Imports statement to the top of the code-behind file for the DateRangePicker control, and this method:

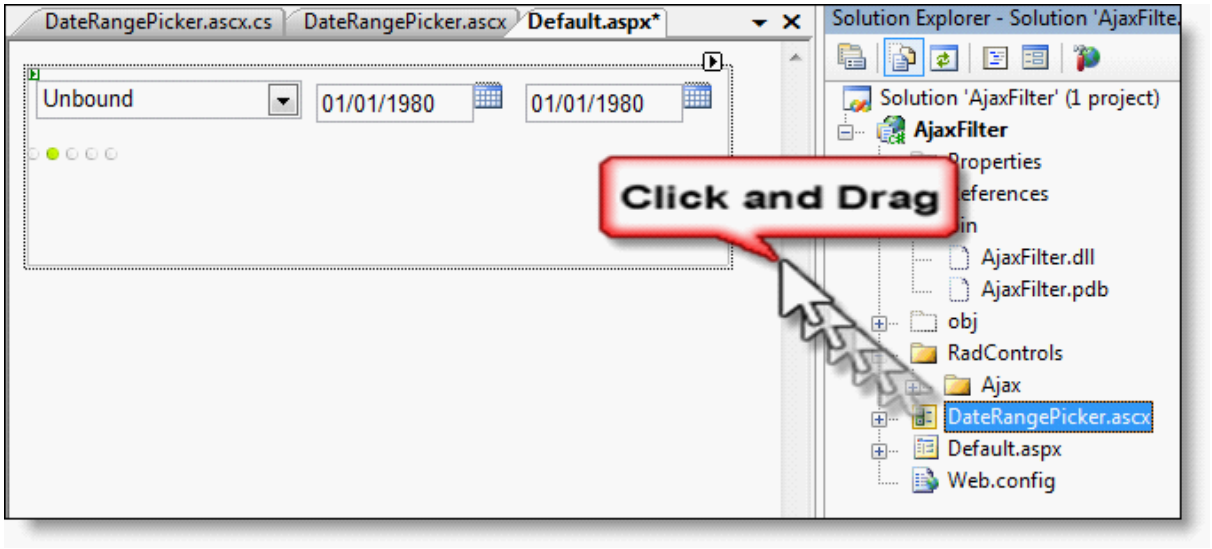
```
C# Example:
using Telerik.WebControls;

public void ConfigureAjax( RadAjaxManager ajm )
{
      ajm.AjaxSettings.AddAjaxSetting( ddlDateRange, dpStart, AjaxLoadingPanel1 );
      ajm.AjaxSettings.AddAjaxSetting( ddlDateRange, divEnd, AjaxLoadingPanel1 );
}

VB Example:
Imports Telerik.WebControls

Public Sub ConfigureAjax(ByVal ajm As RadAjaxManager)
       ajm.AjaxSettings.AddAjaxSetting(ddlDateRange, dpStart, AjaxLoadingPanel1)
       ajm.AjaxSettings.AddAjaxSetting(ddlDateRange, divEnd, AjaxLoadingPanel1)
End Sub
```

A few things to note about this code:

- We pass in the RadAjaxManager instance to use as a parameter
- Each pair of Initiator and Target is added with one separate call to ajm.AjaxManager.AddAjaxSetting
- AddAjaxSetting comes in two flavors. Both take two Control instances, the Initiator and the Target. One overload also takes a AjaxLoadingPanel instance as a third parameter
- In our code, we have instance variables for all of the parameters needed. In some cases, that it not so, and we need to use FindControl on the appropriate naming container to get a handle to the instance (for instance, when setting up an Ajax pair within cells of an ItemTemplate in a Grid).

Now that our control is good to go, lets set about using it in a web page.

7. Switch to the main form of the application and view it in Design mode. Drag the Web User Control DateRangePicker.ascx from the Solution Explorer onto the design surface. Note that a member variable is created for you called DateRangePicker1:



Dragging the control onto the design surface

8. Now drop a RadAjaxManager on to the form

9. Switch to the code behind file for the main form, and change the Page_Load event to pass the RadAjaxManager instance contained in the form down to the DateRangePicker:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    DateRangePicker1.ConfigureAjax(RadAjaxManager1);
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    DateRangePicker1.ConfigureAjax(RadAjaxManager1)
End Sub
```

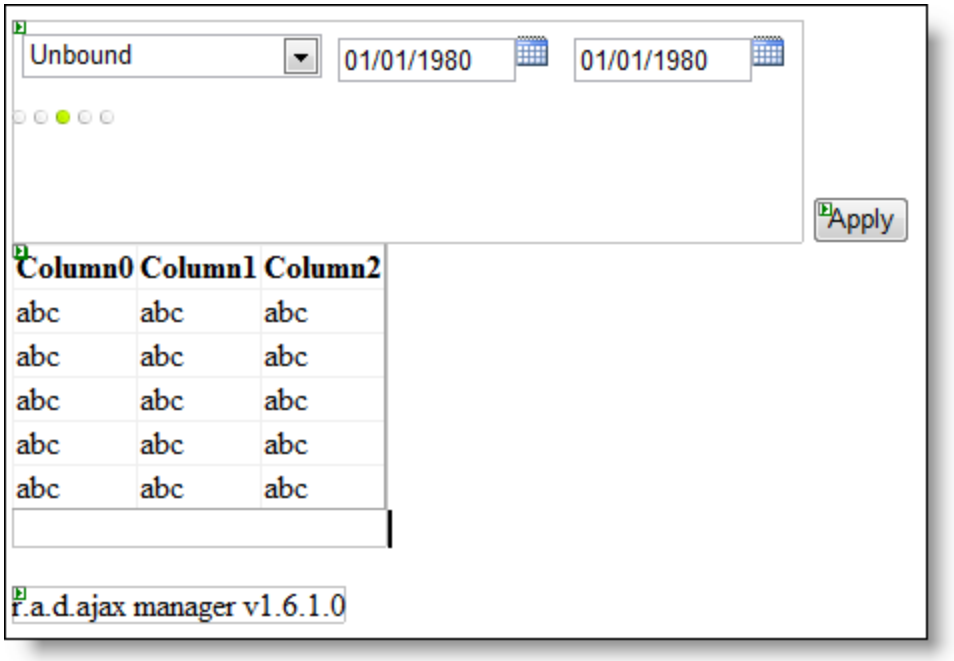
Try running the application. It should behave as before. However, the DateRangePicker is now a reusable control, that can be hooked up to the RadAjaxManager of the containing form.

Now, to make this a more interesting example, lets try adding a Grid to the form with some data in it that includes a date column, and then use the DateRangePicker to filter that data. We will have an apply button that will apply whatever date range you pick to the data in the grid. The grid will be refreshed using AJAX. All of these AJAX interactions will be handled by the same RadAjaxManager instance...

10. Add a Button called pbAdd to the page, and set the Text to Apply

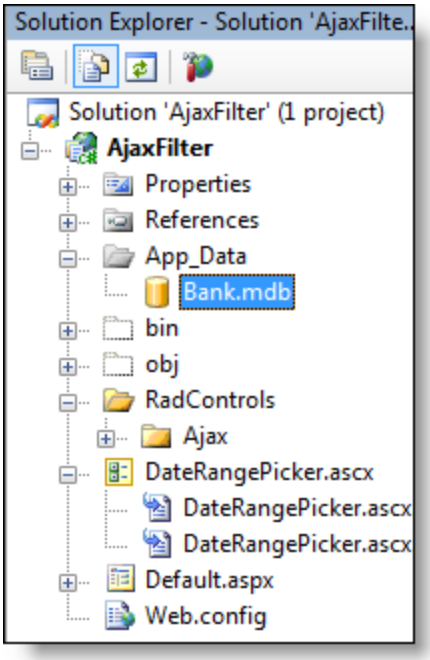
11. Add a Panel under the DateRangePicker and Button, call it pnlGrid. We will need to place our GridView inside this panel, because a GridView does not render itself when it has 0 records bound to it, and thus we will get the 'invisible control' page error if we choose a filter that excludes all records and then one that doesn't.

12. Place a GridView inside of the Panel. The form should now look something like this:



Form layout

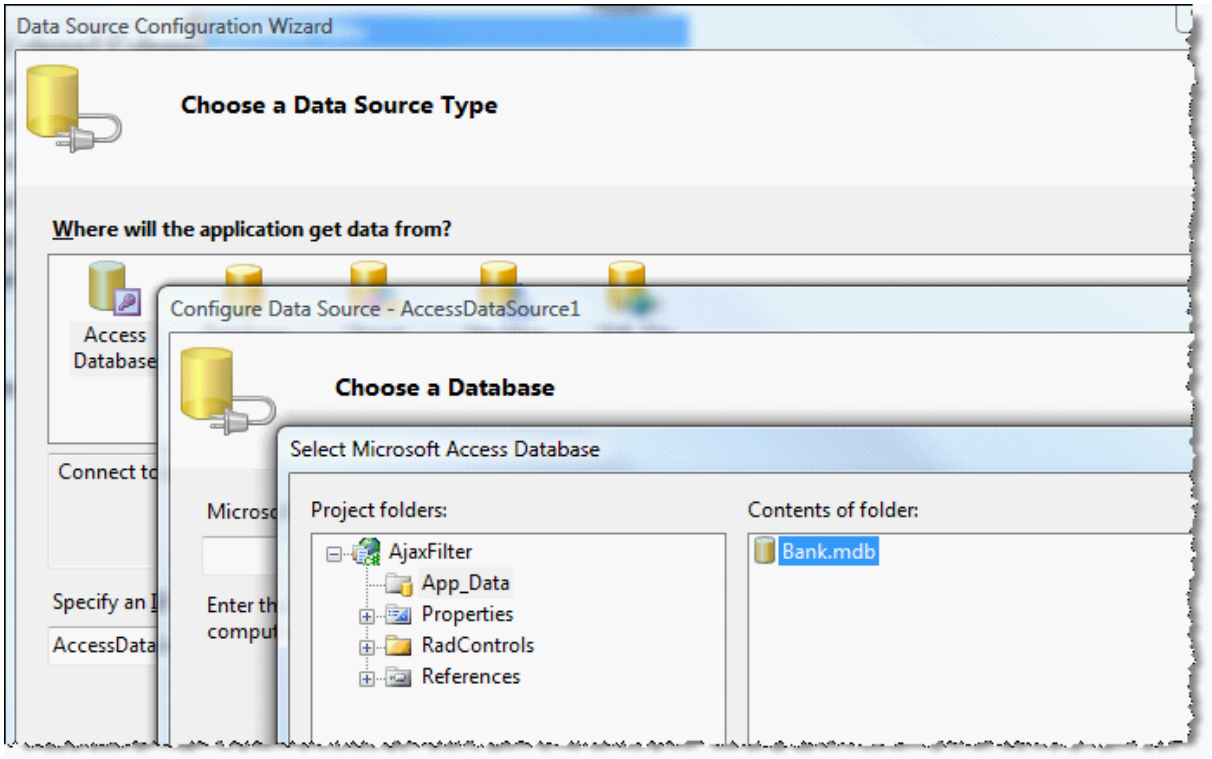
13. Right click the Solution Explorer and choose Add, then Add ASP.NET Folder, then App_Data
14. Right click the App_Data folder in the Solution Explorer and choose Add, then Existing Item.
15. Navigate to the Bank.mdb file under Rad Ajax/Projects/Data, and click add to copy this Access Database to your App_Data folder:



Adding the Access database

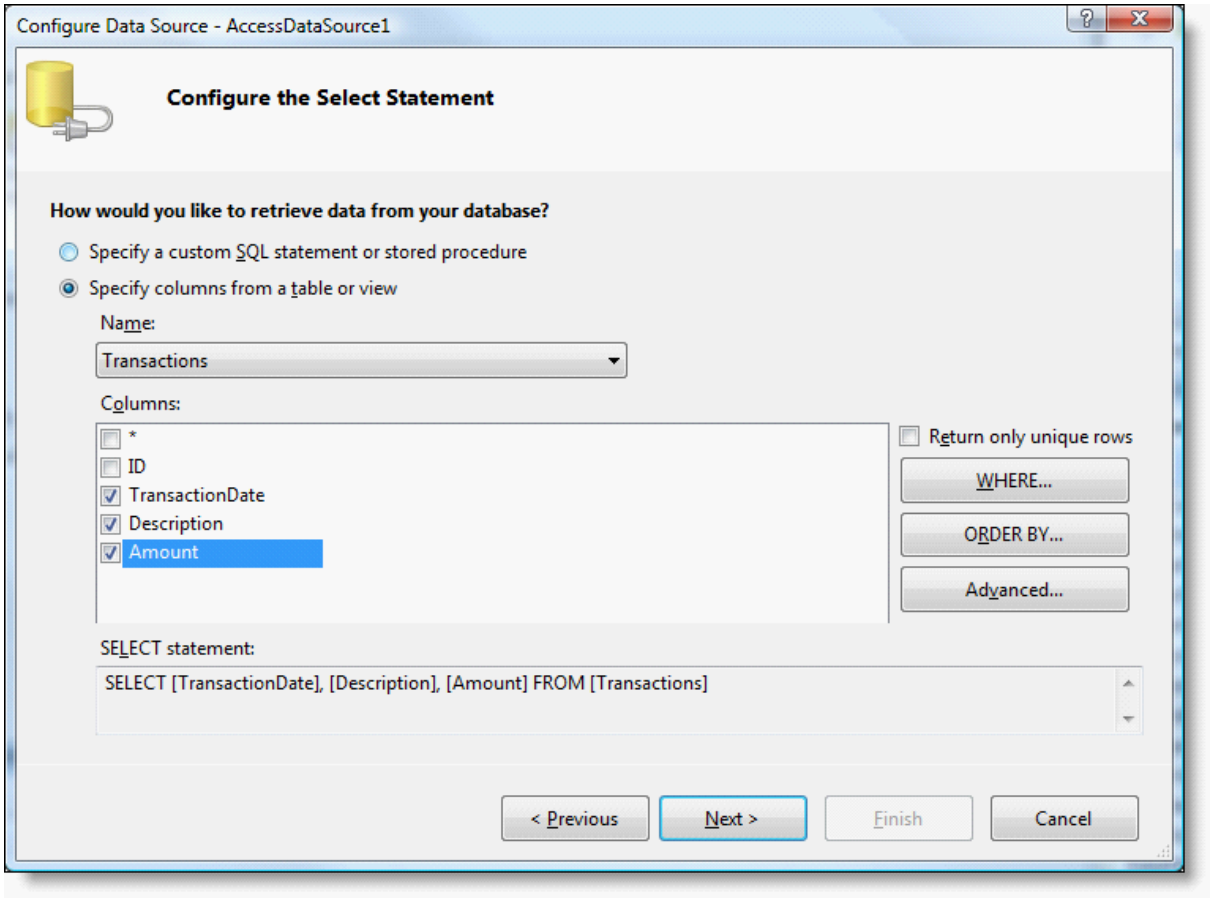
16. On the Smart Tag of the GridView, where it says Choose Data Source, select <New Datasource...>. When the Choose a Data Source Type dialog is shown, click on Access Database, then OK. In the

Choose a Database dialog, click Browse. In the Select Microsoft Access Database dialog, drill down to your database:



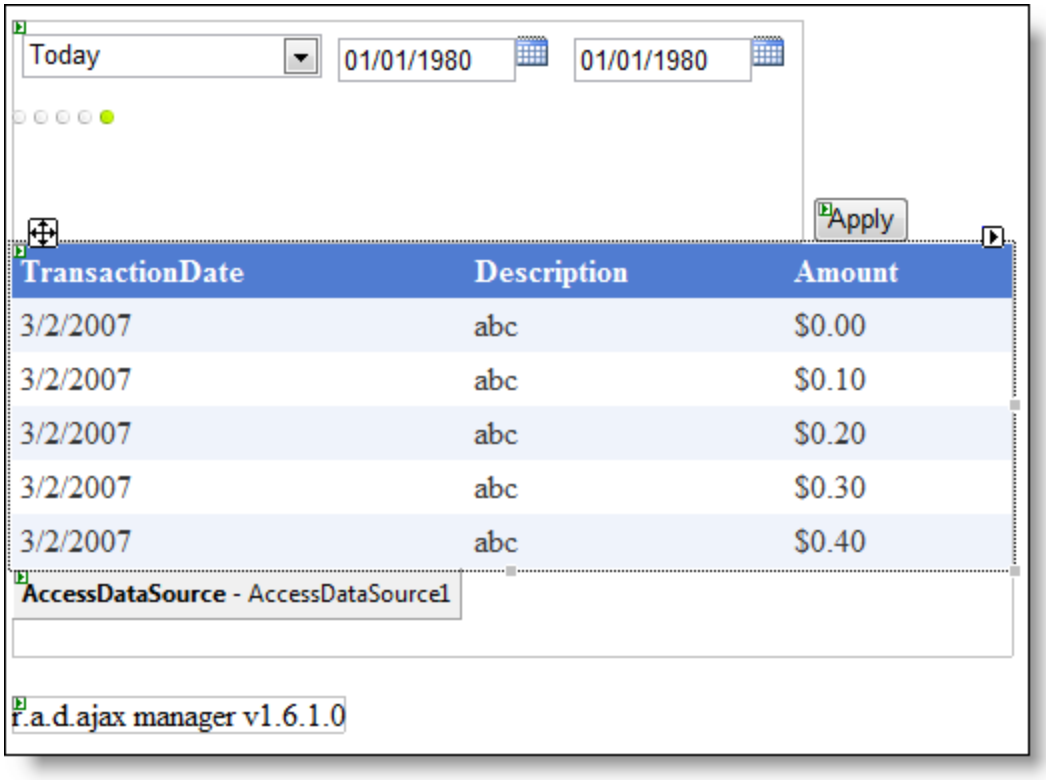
Selecting the Access database

17. Click OK. Click OK, then Next. Check the columns TransactionDate, Description and Amount. Click Next, then Finish.



Configuring the data source

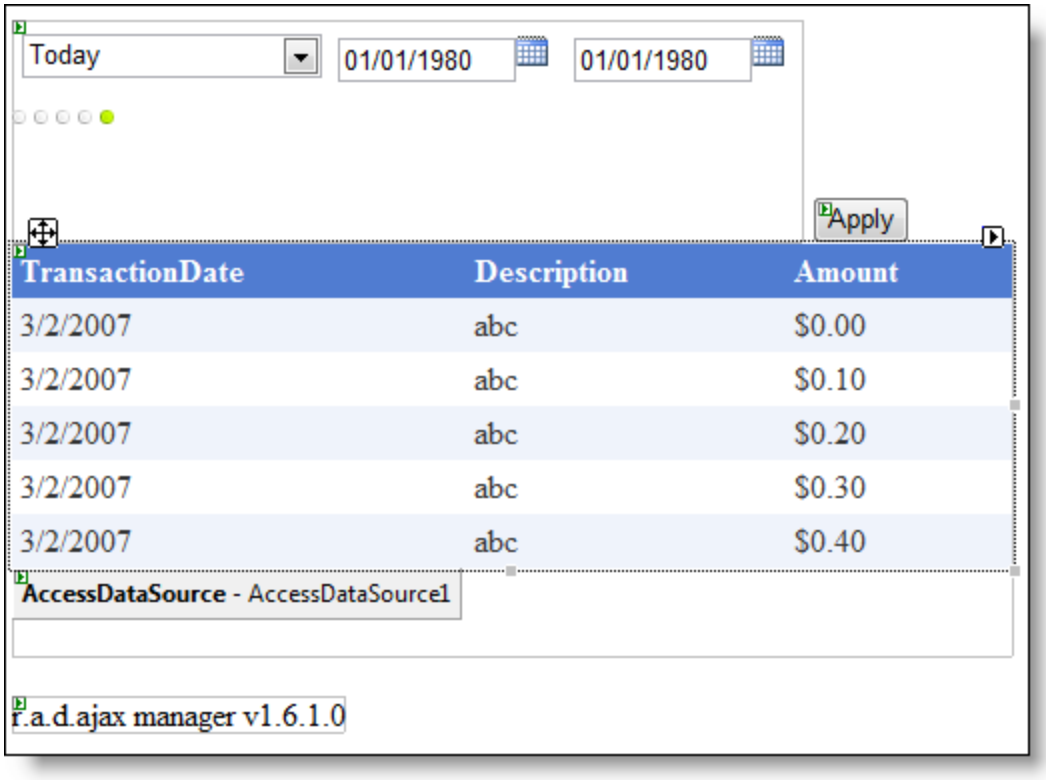
18. Back in the Form Designer, click the Smart Tag on the GridView and then click the link Edit Columns. Click the TransactionDate column, scroll down to DataFormatString and set it to {0:d}. Set the HtmlEncode property to False (if you don't, the DataFormatString is not applied correctly!). For the Amount column, set the DataFormatString to {0:c} and HtmlEncode to False. Click OK.



Setting format strings

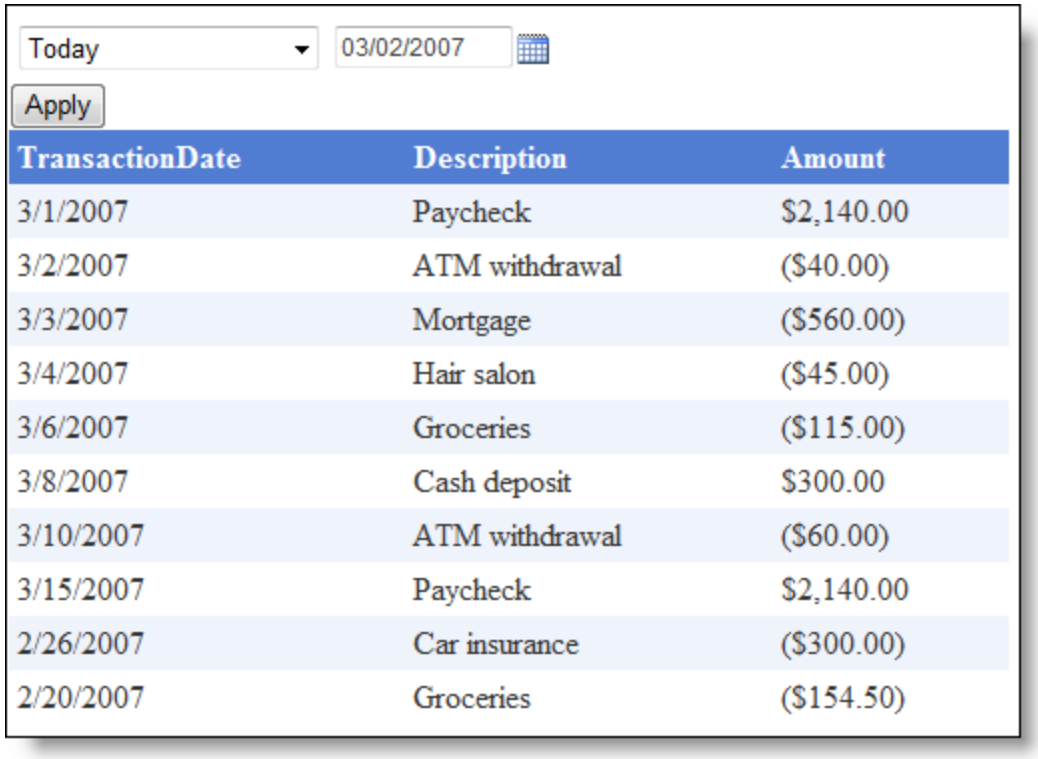
19. Click the Smart Tag of the GridView again and choose Auto Format.... Select the Classic format. Set the Width of the GridView to 500px. Set the HorizontalAlign property of the GridView's HeaderStyle property to Left.

Your form should now look something like this:



The finished layout

Now lets try running the application so far. You will see that the GridView binds to the Access database and shows all of the transactions:



The application running

OK, so far, so good. Next, we want to hook up the filter logic. First, we need to expose the currently selected Start and End Date in the DateRangePicker.

20. Add these two properties to the code behind file of the DateRangePicker control:

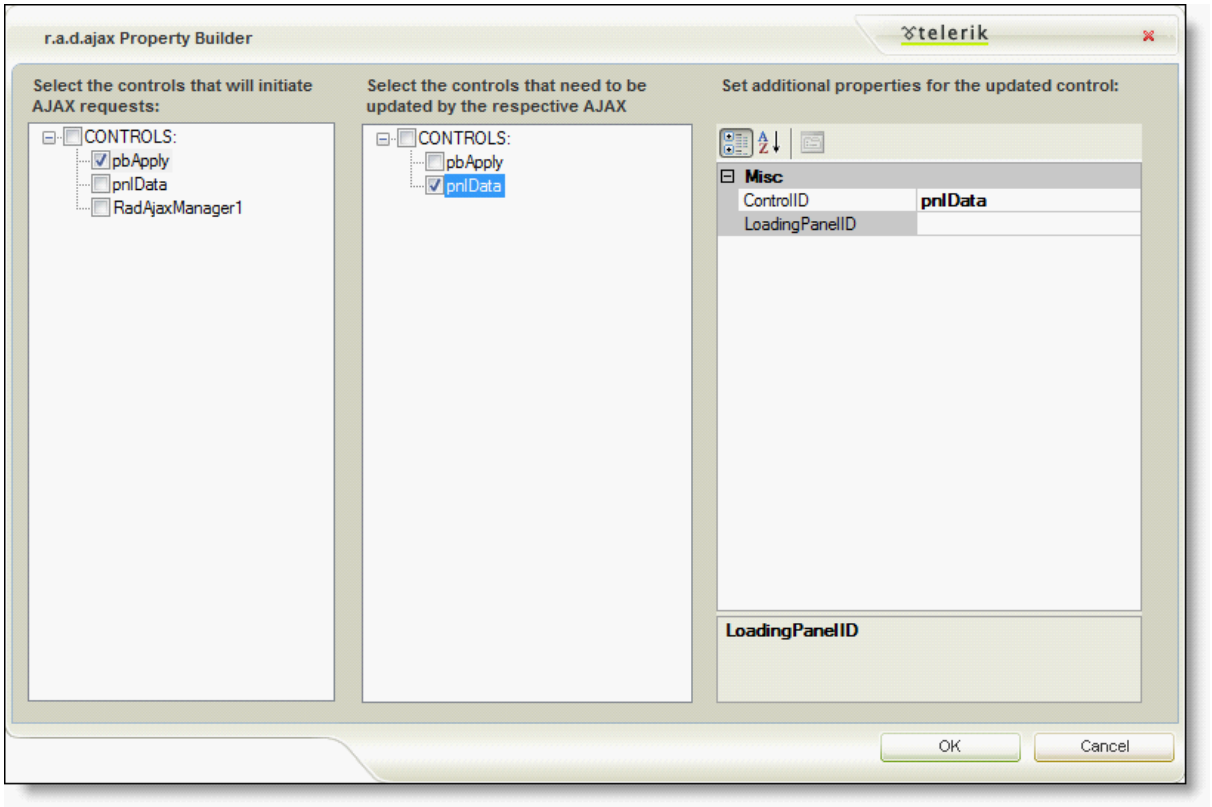
```
C# Example:
public DateTime StartDate
{
    get
    {
        return dpStart.SelectedDate;
    }
}

public DateTime EndDate
{
    get
    {
        return dpEnd.SelectedDate;
    }
}

VB Example:
Public ReadOnly Property StartDate() As DateTime
   Get
        Return dpStart.SelectedDate
   End Get
End Property

Public ReadOnly Property EndDate() As DateTime
   Get
        Return dpEnd.SelectedDate
```

```
   End Get
End Property
```

21. Now double click the Apply button on the main form and write an Event Handler that sets the FilterExpression property of the AccessDataSource to one that matches the currently selected dates:

```
C# Example:
protected void pbApply_Click(object sender, EventArgs e)
{
      AccessDataSource1.FilterExpression = String.Format(
        "TransactionDate>='{0}' AND TransactionDate<='{1}'",
          DateRangePicker1.StartDate, DateRangePicker1.EndDate);
}

VB Example:
Protected Sub pbApply_Click(ByVal sender As Object, ByVal e As EventArgs)
      AccessDataSource1.FilterExpression = String.Format(
          "TransactionDate>='{0}' AND TransactionDate<='{1}'",
          DateRangePicker1.StartDate, DateRangePicker1.EndDate)
End Sub
```

22. Finally, we want the Apply button to work in Ajax mode. Bring up the Property Builder on the RadAjaxManager and tell it that the pbApply will update the pnlData:



Configuring the AjaxManager

Note that this form now uses a hybrid approach for configuring AJAX: the Designer is used to set up the relationships between the controls on the page, and the relationships within the DateRangePicker are set up programmatically in run time. Both use the same RadAjaxManager instance.

This is a pretty typical example of a more complex AJAX setup. To make it even more realistic, you might have a Filter User Control that in turn contains a DateRangePicker and some other filters, and the Apply button would be in that filter Control. To set up the AJAX, you would then need to call FindControl to get a reference to the Apply button and set up the AJAX relationship in runtime between that control and the Panel that surrounds the GridView.

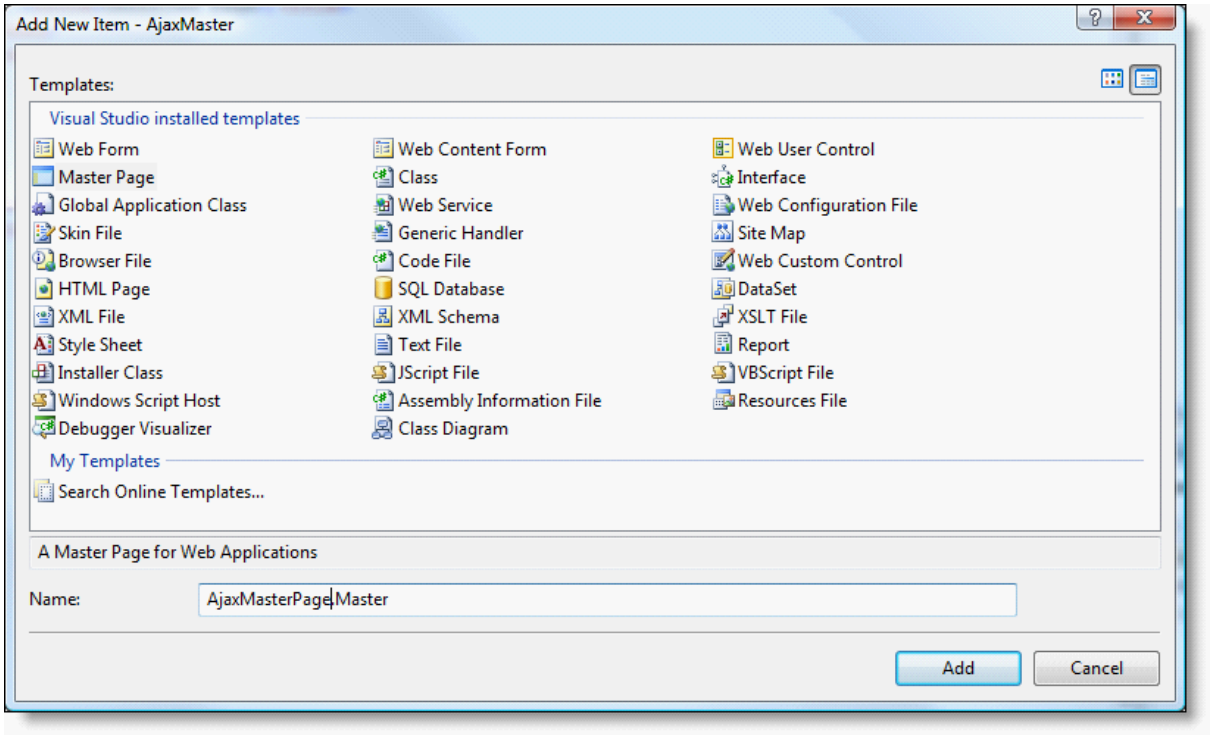
Lab: Master Pages

If you want to use the RadAjaxManager in a Master Page context, it is most likely that you will have the RadAjaxManager in the Master Page, and the Ajax Initiators and Targets could be in any combination of the Master Page or the Content Forms. Either way, you will need to set up your AjaxSettings programmatically, as the RadAjaxManager Property Builder will not be able to see into the Content Pages.

Lets create a simple demo to illustrate how this would work.

1. Create a new ASP.Net Web Application Project called AjaxMaster.

2. Right click the project in the Solution Explorer and choose Add, then New Item. Select Master Page and name it AjaxMasterPage.Master:



Adding the Master Page

3. While in Source mode, add in a HTML table to structure the layout into a left pane and a right pane that has the ContentPlaceHolder in it. Rename the ContentPlaceHolder cphMain. Note the use of background color so we can see which part of the output is which:

```
<body>
     <form id="form1" runat="server">
        <div>
              <table width=100% border=0 cellpadding=2>
```

```
                <tr>
                       <td width=20% valign=top style="background-color:whitesmoke">
                    </td>
                       <td valign=top style="background-color:lightyellow">
                    </td>
                </tr>
            </table>
        </div>
    </form>
</body>
```

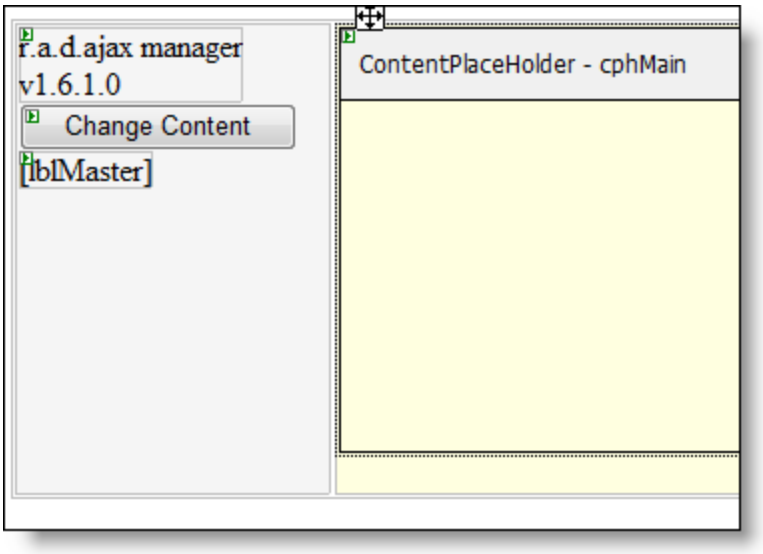
4. Switch to Design mode on the Master Page and drag the following into the left cell of the table:

- a RadAjaxManager (name it ajmMaster)

- a Button (name it pbMaster, set the Text to "Change Content")

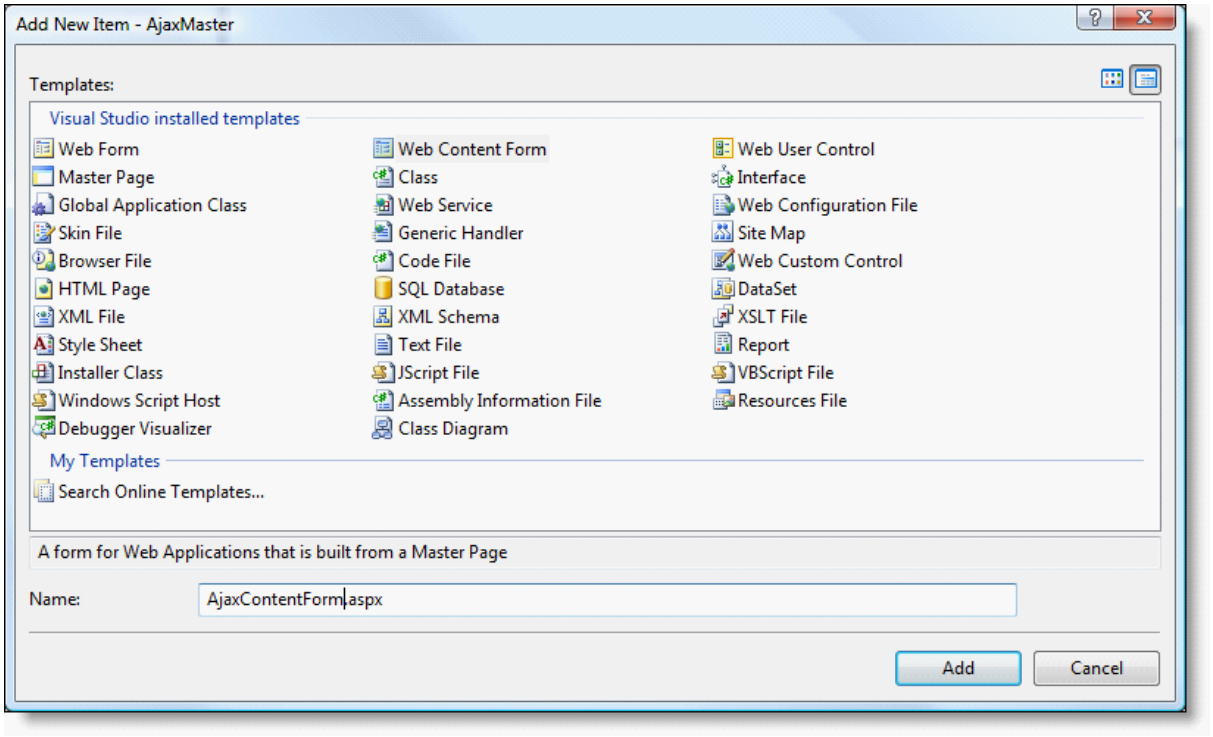
- a Label (name it lblMaster, set the Text to "")

Your master page should now look like this:



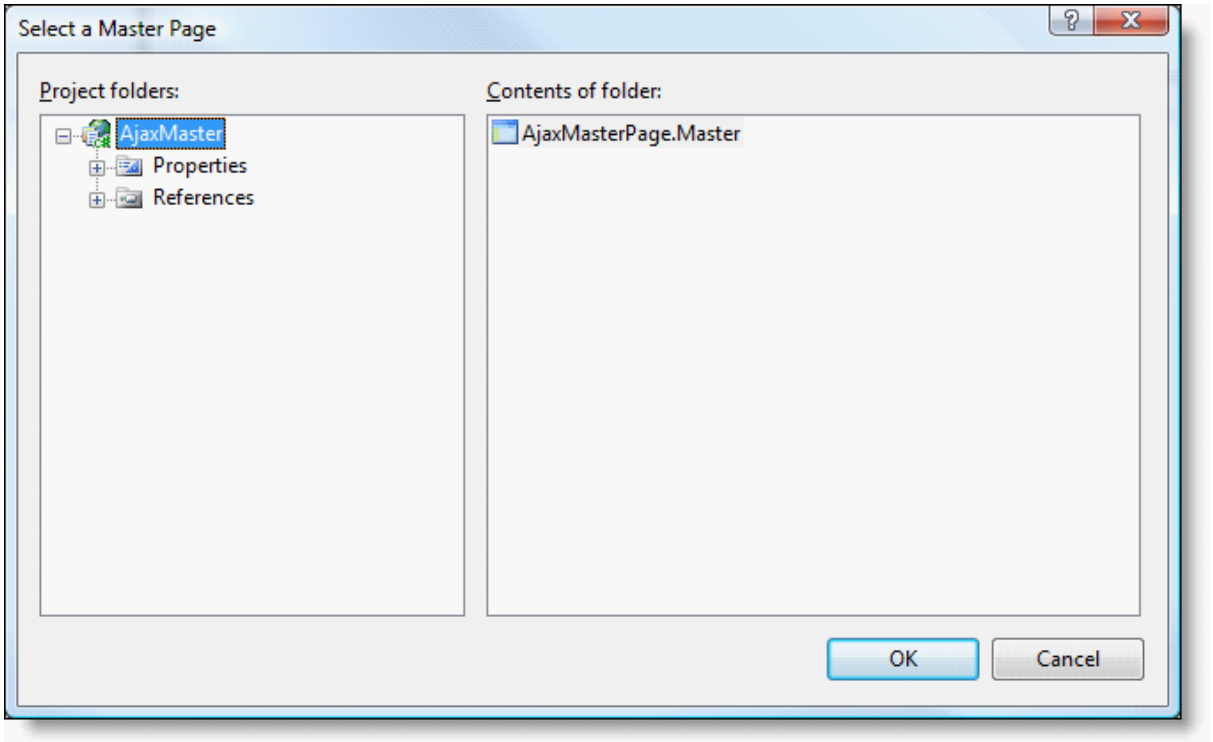
Master Page layout

5. Right click the project in the Solution Explorer and choose Add, then New Item. Select Web Content Form and name it AjaxContentForm.aspx:



Adding the Content Form

6. Click Add. In the dialog "Select a Master Page", select AjaxMasterPage.Master and click OK.



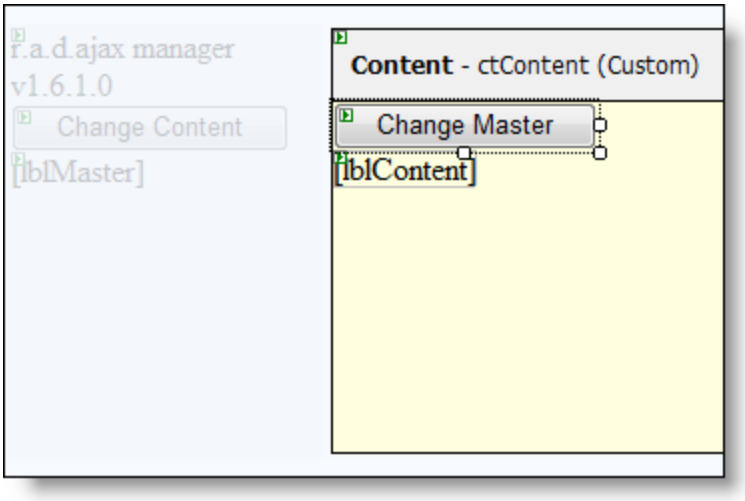
Setting the Master Page of the Content Form

7. Switch to Design mode for the content page. Change the name of the Content to ctContent. Add the following controls:

- a Button (call it pbContent, change the Text to "Change Master")

- a Label (call it lblContent, change the text to "")

The content form should now look like this:



The Content Form

8. Go back to Design mode on the Master Page. Double click pbMaster and add the following to the OnClick Event Handler:

```
C# Example:
protected void pbMaster_Click(object sender, EventArgs e)
{
     Label lbl = (Label) this.cphMain.FindControl("lblContent");
    lbl.Text = "Changed from Master " + DateTime.Now.ToString();
}

VB Example:
Protected Sub pbMaster_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim lbl As Label = CType(Me.cphMain.FindControl("lblContent"), Label)
     lbl.Text = "Changed from Master " + DateTime.Now.ToString()
End Sub
```

Note the use of FindControl to get a reference to the Label that will be in the content Form

9. Switch to the Content Form in Design mode, double click the pbContent and add the following code to the event handler:

```
C# Example:
protected void pbContent_Click(object sender, EventArgs e)
{
     Label lbl = (Label)this.Master.FindControl("lblMaster");
    lbl.Text = "Changed from Content " + DateTime.Now.ToString();
}

VB Example:
Protected Sub pbContent_Click(ByVal sender As Object, ByVal e As EventArgs)
    Dim lbl As Label = CType(Me.Master.FindControl("lblMaster"), Label)
     lbl.Text = "Changed from Content " + DateTime.Now.ToString()
End Sub
```

This code is just the mirror of the Master Page code.

Now lets hook up the Ajax, so that when you click the Button pbMaster it changes the text of the Label lblContent in the Content Form, and when you click the Button pbContent in the Content Form it changes the text of the Label lblMaster in the Master Page:

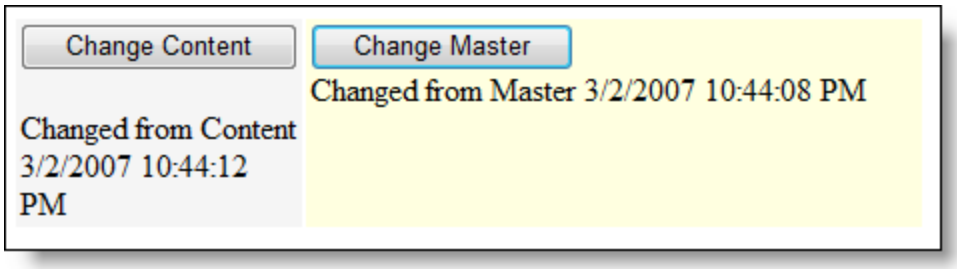
10. In the code-behind of the Content Form, add this using/Imports statement and this code to Page_Load:

```
C# Example:
using Telerik.WebControls;

protected void Page_Load(object sender, EventArgs e)
{
   RadAjaxManager ajm = (RadAjaxManager)this.Master.FindControl("ajmMaster");
   Button btn = (Button)this.Master.FindControl("pbMaster");
    ajm.AjaxSettings.AddAjaxSetting(btn, this.lblContent);

   Label lbl = (Label)this.Master.FindControl("lblMaster");
    ajm.AjaxSettings.AddAjaxSetting(pbContent, lbl);
}

VB Example:
Imports Telerik.WebControls

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  Dim ajm As RadAjaxManager = CType(Me.Master.FindControl("ajmMaster"), RadAjaxManager)
  Dim btn As Button = CType(Me.Master.FindControl("pbMaster"), Button)
    ajm.AjaxSettings.AddAjaxSetting(btn, Me.lblContent)

  Dim lbl As Label = CType(Me.Master.FindControl("lblMaster"), Label)
    ajm.AjaxSettings.AddAjaxSetting(pbContent, lbl)
End Sub
```

Note again the use of FindControl to get hold of the RadAjaxManager, the Button and the Label in the Master Page from the code in the Content Form.

11. Delete Default.aspx from the project and set AjaxContentForm.aspx to be the startup Page.

12. Run the application and click the buttons. The output is as expected, and of course the updates are now done through AJAX callbacks instead of Postbacks!



The output of the running application

We could actually have hooked up the Ajax settings in the Master Page instead, or done part of it in the Master and part of it in the Content. It is probably more normal to hook it up in the Content Form, because you will generally have many different Content Forms for any given Master Page, so the Master code won't know what Content Form it is hosting, whereas the Content will know which Master it is hosted in.

13. Just to demonstrate the reversibility of the solution, comment out the code in Page_Load of the Content Form and write this code in the Page_Load of the Master Page instead:

```
C# Example:
using Telerik.WebControls;

protected void Page_Load(object sender, EventArgs e)
{
   Button btn = (Button)cphMain.FindControl("pbContent");
    ajmMaster.AjaxSettings.AddAjaxSetting(btn, this.lblMaster);

   Label lbl = (Label)cphMain.FindControl("lblContent");
    ajmMaster.AjaxSettings.AddAjaxSetting(pbMaster, lbl);
}

VB Example:
Imports Telerik.WebControls

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  Dim btn As Button = CType(cphMain.FindControl("pbContent"), Button)
    ajmMaster.AjaxSettings.AddAjaxSetting(btn, Me.lblMaster)

  Dim lbl As Label = CType(cphMain.FindControl("lblContent"), Label)
    ajmMaster.AjaxSettings.AddAjaxSetting(pbMaster, lbl)
End Sub
```

Run this version of the application and you will get the same results.

Lab: AJAX inside Grid

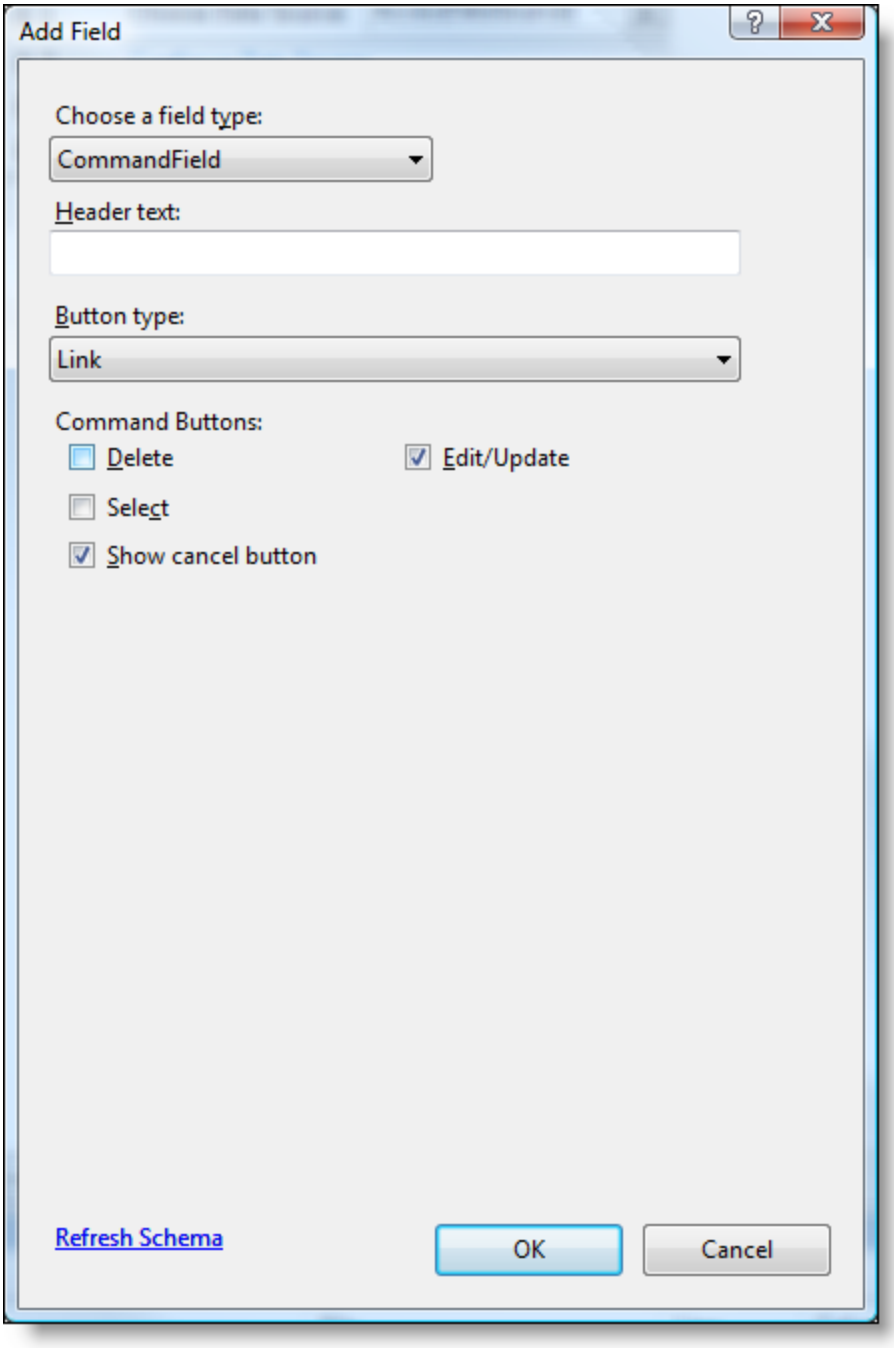
As our last example of complex AJAX scenarios, lets look at how we would AJAX enable one control in an ItemTemplate of a GridView updating another control on that same row.

1. Create a new ASP.NET Web Application Project called AjaxGrid

2. Switch to Design mode on the main form and drop a GridView into the form.

3. Follow steps 13-17 in the section "Lab: Using the RadAjaxManager with User Controls" to add the Bank Access Database to the project and connect it to the GridView through an AccessDataSource.

4. From the Smart Tag editor of the GridView, click Add New Column, and add a CommandField as follows:



Adding a command field

5. Click OK, then choose Add New Field again, and this time add a TemplateField

6. Click Edit Columns on the Smart Tag of the GridView. Select the column Amount and then click the link "Convert this field into a Template Field". Click OK.

7. Switch to Source view, and move the new template column to the top of the list of GridView Columns. Add in an EditItemTemplate as follows:

```
<asp:TemplateField>
     <EditItemTemplate>
          <asp:Button ID=btnAdd runat=server Text="Add" CommandName="AddOne"/>
```

```
    </EditItemTemplate>
</asp:TemplateField>
```

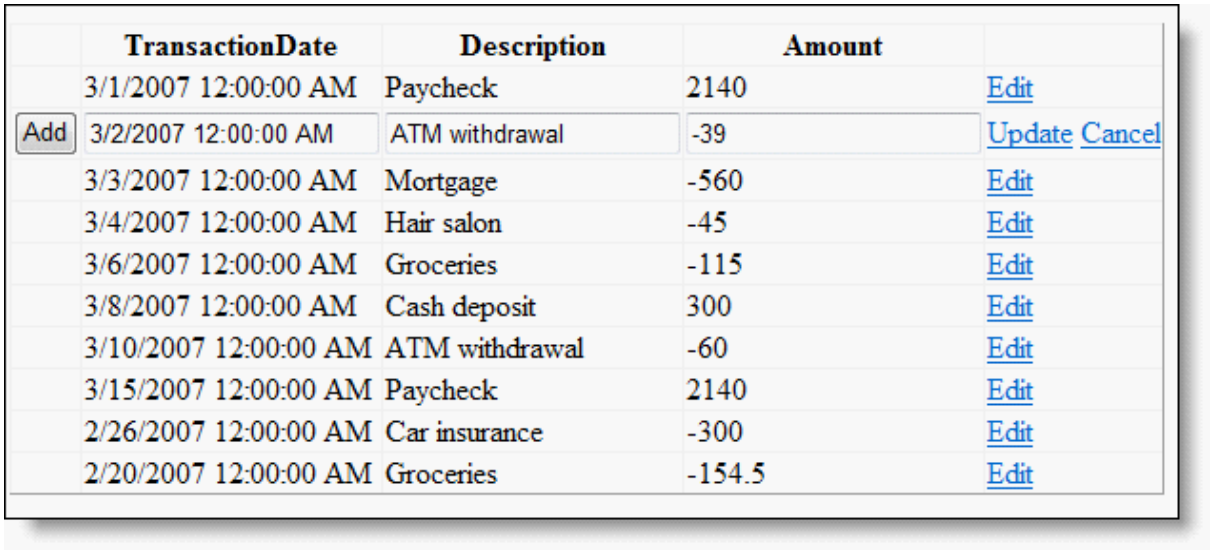
8. Rename the TextBox1 in the EditItemTemplate of the AmountTemplateColumn to tbAmount:

```
<asp:TemplateField   HeaderText="Amount"   SortExpression="Amount">
    <EditItemTemplate>
        <asp:TextBox ID="tbAmount" runat="server"
        Text='<%# Bind("Amount") %>'></asp:TextBox>
    </EditItemTemplate>
    <ItemTemplate>
        <asp:Label ID="Label1" runat="server"
        Text='<%# Bind("Amount") %>'></asp:Label>
    </ItemTemplate>
</asp:TemplateField>
```

9. Switch back to Design mode. Select the GridView and double click the RowCommand event. Add this code to the event handler, to increment the number in the tbAmount TextBox by one:

```
C# Example:
protected void GridView1_RowCommand(object sender, GridViewCommandEventArgs e)
{
    if (e.CommandName == "AddOne")
    {
        GridViewRow row = (GridViewRow)((Button)e.CommandSource).Parent.Parent;
        TextBox tbAmount = (TextBox)row.FindControl("tbAmount");
        Decimal amount = Decimal.Parse(tbAmount.Text);
        tbAmount.Text = Convert.ToString(amount + 1);
    }
}

VB Example:
Protected Sub GridView1_RowCommand(ByVal sender As Object, _
  ByVal e As GridViewCommandEventArgs)
    If e.CommandName = "AddOne" Then
        Dim row As GridViewRow = _
            CType((CType(e.CommandSource, Button)).Parent.Parent, GridViewRow)
        Dim tbAmount As TextBox = _
            CType(row.FindControl("tbAmount"), TextBox)
        Dim amount As Decimal =  Decimal.Parse(tbAmount.Text)
        tbAmount.Text = Convert.ToString(amount + 1)
    End If
End Sub
```

10. Try running the application now and click the Edit Link. When the row goes into Edit Mode, click the Add button a few times and see the Amount field increase by one each time. Note that this is still in Postback mode and the screen flickers, both when you click Edit and when you click the Add button:



	TransactionDate	Description	Amount	
	3/1/2007 12:00:00 AM	Paycheck	2140	Edit
Add	3/2/2007 12:00:00 AM	ATM withdrawal	-39	Update Cancel
	3/3/2007 12:00:00 AM	Mortgage	-560	Edit
	3/4/2007 12:00:00 AM	Hair salon	-45	Edit
	3/6/2007 12:00:00 AM	Groceries	-115	Edit
	3/8/2007 12:00:00 AM	Cash deposit	300	Edit
	3/10/2007 12:00:00 AM	ATM withdrawal	-60	Edit
	3/15/2007 12:00:00 AM	Paycheck	2140	Edit
	2/26/2007 12:00:00 AM	Car insurance	-300	Edit
	2/20/2007 12:00:00 AM	Groceries	-154.5	Edit

The application running

11.Drop a RadAjaxManager on the form.

12. Follow steps 1-4 in the lab "Hooking up an AJAX panel" to copy the standard loading images to your project and add an AjaxLoadingPanel to your form. For the image, select LoadingImage1.gif. Set the MinDisplayTime to 300.

13. Now to start hooking up some AJAX! Click on the GridView and double click the RowDataBound event in the Property Inspector. Add this code to the event handler:

```
C# Example:
protected void ConfigureAjax(GridViewRow row)
{
   Button btn = (Button)row.FindControl("btnAdd");
   TextBox tbAmount = (TextBox)row.FindControl("tbAmount");
  if ((btn != null) && (tbAmount != null))
      RadAjaxManager1.AjaxSettings.AddAjaxSetting(
       btn, tbAmount, AjaxLoadingPanel1);
}

protected void GridView1_RowDataBound(object sender, GridViewRowEventArgs e)
{
    if (e.Row.RowType == DataControlRowType.DataRow)
        ConfigureAjax(e.Row);
}

VB Example:
Protected Sub ConfigureAjax(ByVal row As GridViewRow)
  Dim btn As Button = CType(row.FindControl("btnAdd"), Button)
  Dim tbAmount As TextBox = CType(row.FindControl("tbAmount"), TextBox)
  If (btn <> Nothing) &&(tbAmount <> Nothing) Then
      RadAjaxManager1.AjaxSettings.AddAjaxSetting( _
       btn, tbAmount, AjaxLoadingPanel1)
  End If
End Sub

Protected Sub GridView1_RowDataBound(ByVal sender As Object, _
  ByVal e As GridViewRowEventArgs)
```

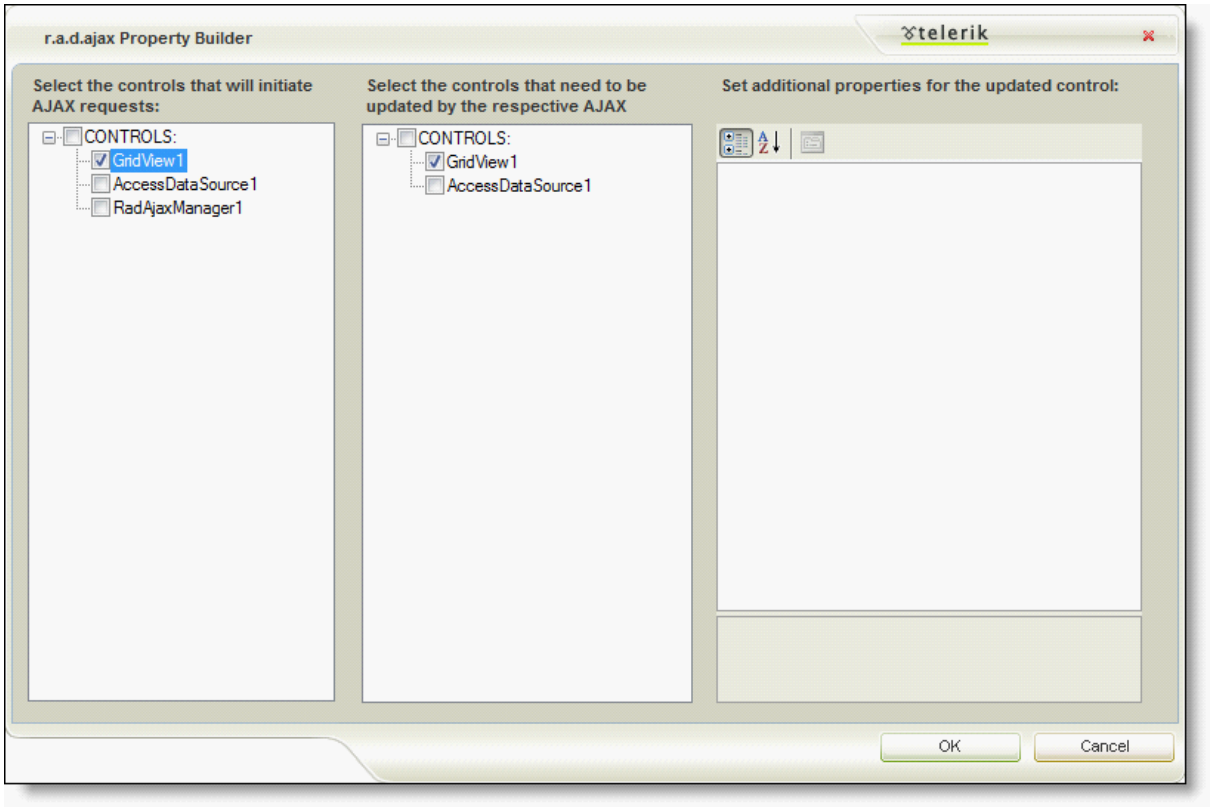
```
    If  e.Row.RowType  =  DataControlRowType.DataRow Then
         ConfigureAjax(e.Row)
    End If
End Sub
```

Some things to point out in this code:

- We need to use FindControl to locate the Button and the Textbox inside the current row.
- The check for null is necessary after the two FindContols in ConfigureAjax, because this method is called for each row in the grid, but only the row that you are editing will actually contain these two controls.
- We hook up the AjaxLoadingPanel

If you run the application now there are two problems. First, the form still does a Postback when you enter and exit Edit mode for a row.

14. To eliminate this, use the RadAjaxManager's Smart Tag editor to display the Property Builder and tell the GridView to update itself:



The GridView updates itself

The next problem is that the RowDataBound event only fires when you first enter Edit Mode. On subsequent callbacks, it doesn't fire, so we don't get to hook up our AJAX pair.

15. To fix this problem, add the following row at the end of the RowCommand handler:

```
C#  Example:
protected  void  GridView1_RowCommand(object  sender,  GridViewCommandEventArgs  e)
{
   if  (e.CommandName  ==  "AddOne")
```

```
  {
      GridViewRow row = (GridViewRow)((Button)e.CommandSource).Parent.Parent;
     TextBox tbAmount = (TextBox)row.FindControl("tbAmount");
     Decimal amount = Decimal.Parse(tbAmount.Text);
    tbAmount.Text = Convert.ToString(amount + 1);
     ConfigureAjax(row);
  }
}

VB Example:
Protected Sub GridView1_RowCommand(ByVal sender As Object, _
  ByVal e As GridViewCommandEventArgs)
  If e.CommandName = "AddOne" Then
    Dim row As GridViewRow = _
        CType((CType(e.CommandSource, Button)).Parent.Parent, GridViewRow)
    Dim tbAmount As TextBox = CType(row.FindControl("tbAmount"), TextBox)
    Dim amount As Decimal = Decimal.Parse(tbAmount.Text)
     tbAmount.Text = Convert.ToString(amount + 1)
     ConfigureAjax(row)
  End If
End Sub
```

Now that the problems are resolved, run the application again and see how all events are now AJAX enabled!

	TransactionDate	Description	Amount	
Add	3/1/2007 12:00:00 AM	Paycheck	2142	Update Cancel
	3/2/2007 12:00:00 AM	ATM withdrawal	-40	Edit
	3/3/2007 12:00:00 AM	Mortgage	-560	Edit
	3/4/2007 12:00:00 AM	Hair salon	-45	Edit
	3/6/2007 12:00:00 AM	Groceries	-115	Edit
	3/8/2007 12:00:00 AM	Cash deposit	300	Edit
	3/10/2007 12:00:00 AM	ATM withdrawal	-60	Edit
	3/15/2007 12:00:00 AM	Paycheck	2140	Edit
	2/26/2007 12:00:00 AM	Car insurance	-300	Edit
	2/20/2007 12:00:00 AM	Groceries	-154.5	Edit

The final result

One final note: I went out of my way here to demonstrate how to hook up AJAX within a grid row. A much simpler solution in this case would be to just put the Grid in a RadAjaxPanel, or just to do step 15. Still, the technique is useful.

1.3.4 Setting focus

Sometimes you want to move the focus to a different control in server side logic when processing an AJAX callback. Using Page.SetFocus() or Control.Focus() does not work will, because that just causes a javascript to be appended to the end of the form body, but when the AJAX callback returns, that script is not executed.

To get around this, use RadAjaxManager.FocusControl or RadAjaxPanel.FocusControl instead. For

instance, in the AJAX shopping cart lab, you could add this code to the end of the pbAdd_Click event handler to clear the quantity field and return the focus to that field:

```
C# Example:
tbQuantity.Text = "";
RadAjaxPanel1.FocusControl(tbQuantity);

VB Example:
tbQuantity.Text = ""
RadAjaxPanel1.FocusControl(tbQuantity)
```

Try this out and see the result. This kind of technique is important when AJAX enabling data entry forms, as it allows you to build fast, responsive, keyboard driven web applications that almost feel like regular desktop applications.

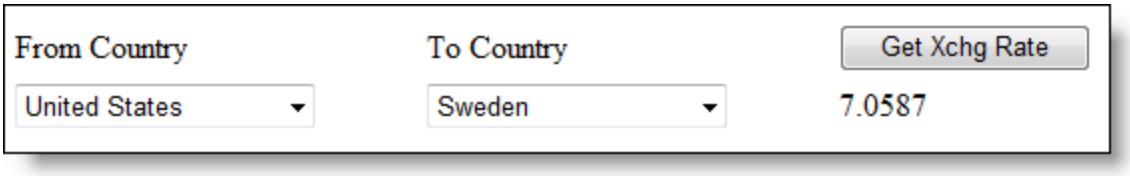
1.3.5 Web Services

An exciting new feature that Telerik added in a recent release was the ability to call Web Services using AJAX. This requires a little more Javascript then we are accustomed to with Telerik's framework, but it is a very powerful capability. You could for instance look up the price of an item when you type the SKU. The Web Service can be written to be lean and mean, and not be burdened with the whole page life cycle of a web from, and the data passed and returned can use bandwidth much more efficiently than large chunks of viewstate and HTML typically returned by an AJAX callback. There are some gotchas, but they come with workarounds.

Lets jump straight in with an example!

Lab: An AJAX Web Service

In this Lab, we will create a form that allows you to choose two countries from DropDownLists, and then calculate the Exchange Rate between the two currencies using an AJAX Web Service Call:

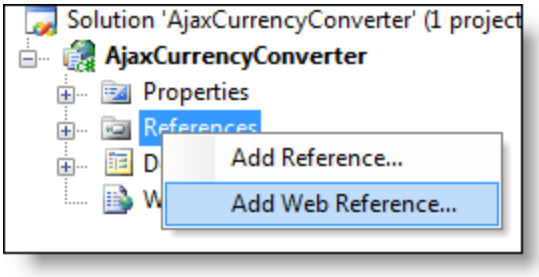


The end result

To power this service, we are going to use a free online web service found at XMethods: http://soapclient.com/XmethodsServices.html. The web service is called Currency Exchange Rate, you can try it out interactively here: http://www.soapclient.com/soapclient?fn=soapform&template=/clientform.html&soaptemplate=/soapresult.html&soapwsdl=http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl. The WSDL for the service can be found here: http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl. One of the caveats of the Web Service AJAX support is that you can only call Web Services that are hosted in the same domain as the web page you are calling from. To get around this, we will write our own Web Service, that forwards the call to the service above.

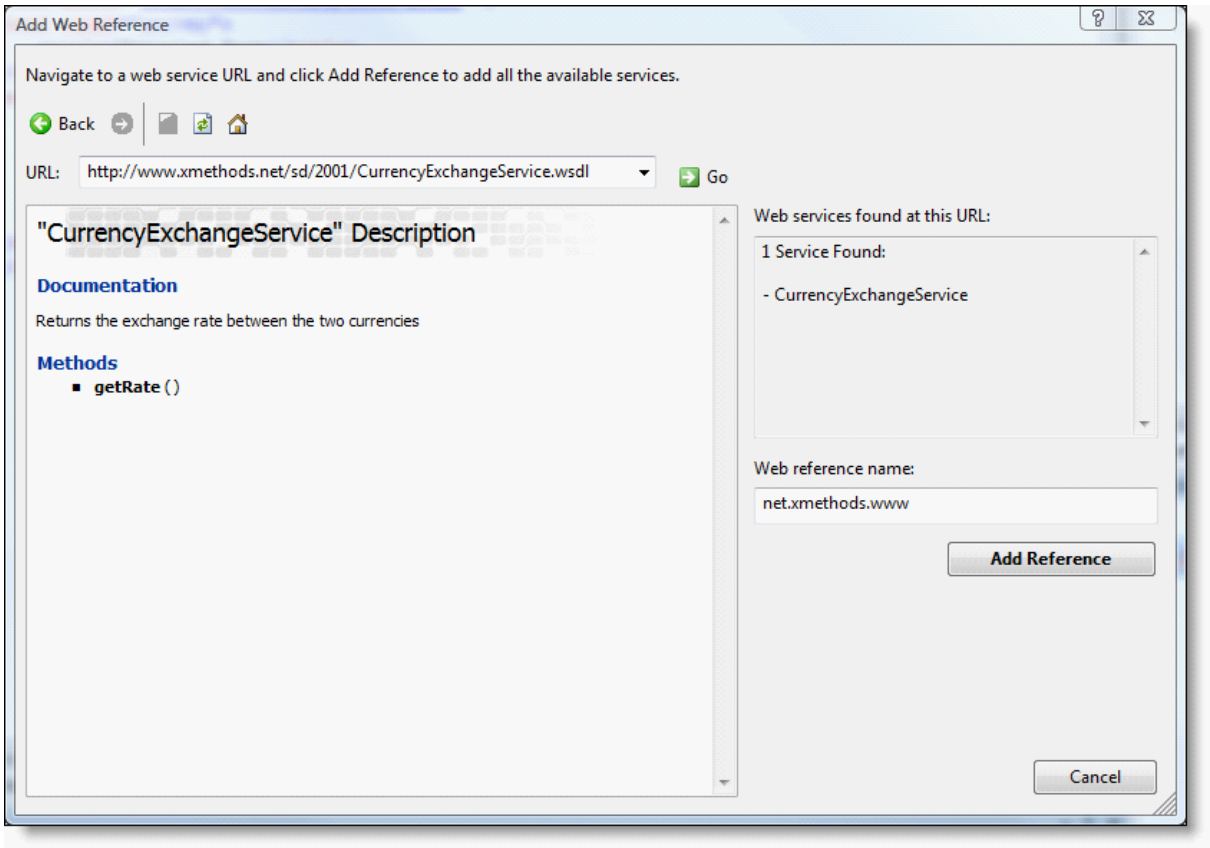
1. Create a new ASP.Net Web Application Project called AjaxCurrencyConverter.

2. Right click the References node in the Solution Explorer and choose Add Web Reference:



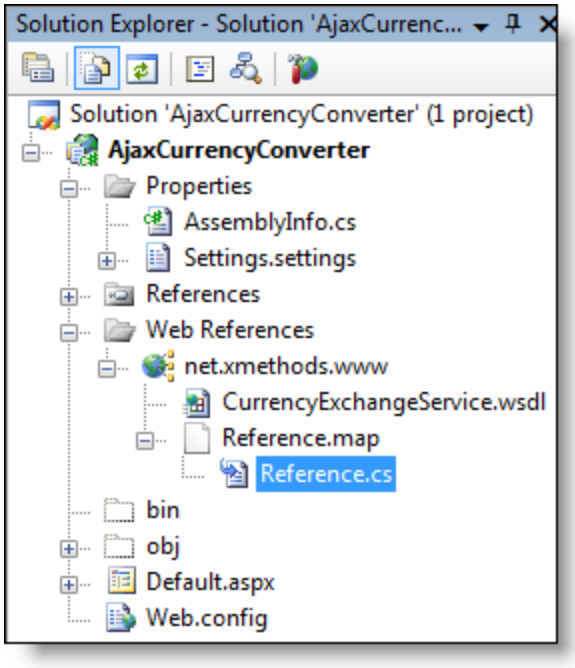
Adding a Web Reference

3. In the Add Web Reference dialog, paste in the URL to the WSDL of the CurrencyExchangeService (http://www.xmethods.net/sd/2001/CurrencyExchangeService.wsdl). Click Go. You will see the service description below. Note that it only has one method, getRate(). This is the method that we wish to call. Click the button Add Reference.



Selecting the Web Service to call

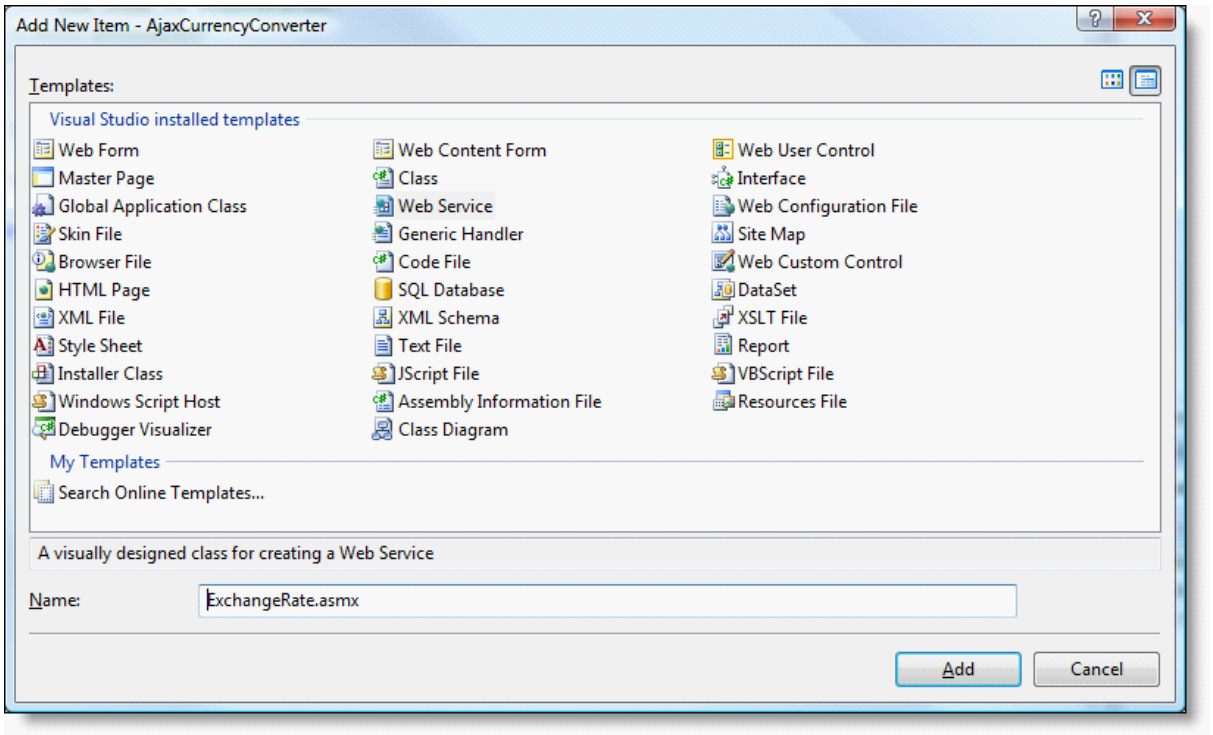
4.The dialog closes and Visual Studio generates a proxy class that can be used to call the Web Service. To see the code that was generated, choose Show All Files at the top of the Solution Explorer, then drill down into Web References, net.xmethods.www, Reference.map, Reference.cs (or vb) :



The generated proxy

5. Double click this file and study the contents. You will see that the namespace is AjaxCurrencyConverter.net.xmethods.www and the proxy class is called CurrencyExchangeService.

6. Now let's wrap this foreign Web Service in one of our own, that is callable from our form. Right click the project in the Solution Explorer and choose Add, New Item. In the Add New Item dialog, click Web Service and call it ExchangeRate.asmx. Click Add.



Adding the Web Service

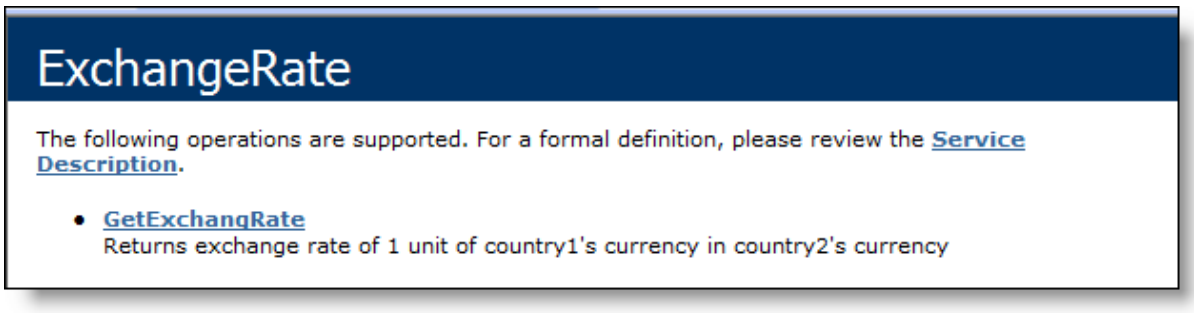
7. In the ExchangeRate.asmx code behind file, remove the auto generated HelloWorld WebMethod and add this using/Imports statement and WebMethod declaration:

```
C# Example:
using AjaxCurrencyConverter.net.xmethods.www;

[WebMethod(Description =
  "Returns exchange rate of 1 unit of country1's currency in country2's currency")]
public float GetExchangRate(string country1, string country2)
{
   CurrencyExchangeService ws =
    new CurrencyExchangeService();
  return ws.getRate(country1, country2);
}

VB Example:
Imports AjaxCurrencyConverter.net.xmethods.www

<WebMethod(Description = _
   "Returns exchange rate of 1 unit of country1's currency in country2's currency")> _
Public Function GetExchangRate(ByVal counTry1 As String, ByVal counTry2 As String) As Single
   CurrencyExchangeService ws =
    New CurrencyExchangeService()
  Return ws.getRate(counTry1,counTry2)
End Function
```

Some points to note about this code:

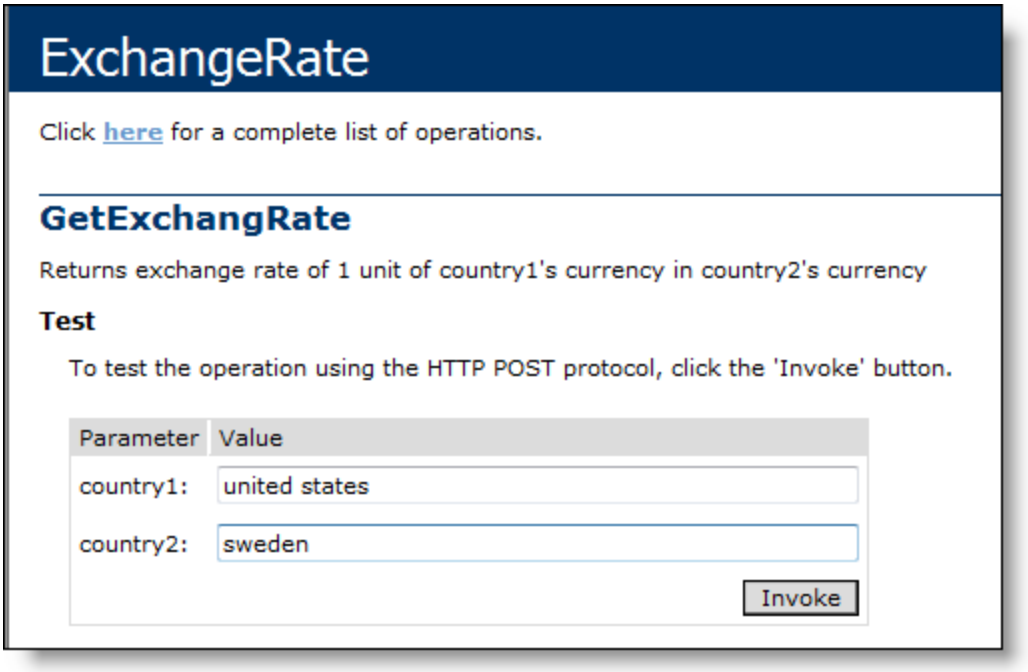
- The using/Imports statement pulls in the namespace that we discovered in step 5 above
- Note the description to the WebMethod attribute, this make it easier for users of our service to understand what it does
- To call the CurrencyExchangeService, I just create a CurrencyExchangeService instance, and call it's getRate() method, passing through the parameters received by the WebMethod call. The proxy class then takes care of connecting to the foreign web service, passing the parameters, waiting for the reply and returning the results, which we just return right back to our caller.

8. Lets test that our forwarding trick works. Set ExchangeRate.asmx to be the Start Page of the application and hit F5 to run. You should see this Service Description page:



The auto generated description page

9. Click the GetExchangeRate link. Fill out country1 and country2 with "united states" and "sweden". Click Invoke:



Invoking the service

10. You should get a response something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<float xmlns="http://tempuri.org/">7.0587</float>
```

Response from our chained web service

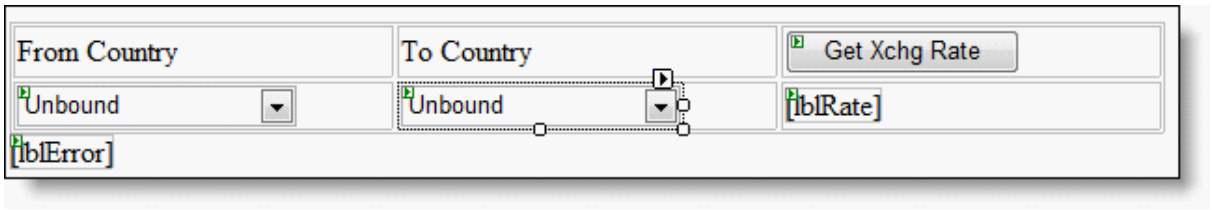
So, to recap, we now have a Web Service in our project that wraps an external Web Service, thus making it callable from our form. Next, lets layout the form.

11. Make Default.aspx the Start Page for the application, and switch to Source view. Add the following markup:

```
<table  border=0  cellpadding=2>
    <tr>
         <td width=200px>From Country</td>
         <td width=200px>To Country</td>
        <td width=200px>
             <asp:Button ID=pbCalc runat=server Text="Get Xchg Rate" />
        </td>
    </tr>
    <tr>
        <td>
             <asp:DropDownList ID=ddlFrom runat=server width=150px>
            </asp:DropDownList>
        </td>
        <td>
             <asp:DropDownList ID=ddlTo runat=server width=150px>
            </asp:DropDownList>
        </td>
        <td>
             <asp:Label ID=lblRate runat=server Text=""></asp:Label>
        </td>
```

```
    </tr>
</table>
<asp:Label   ID=lblError   runat=Server></asp:Label>
```

This gives us a table that structures the layout, with two DropDownLists used to select the countries, a Button to execute the call, one Label to show the result and one Label to show any errors:

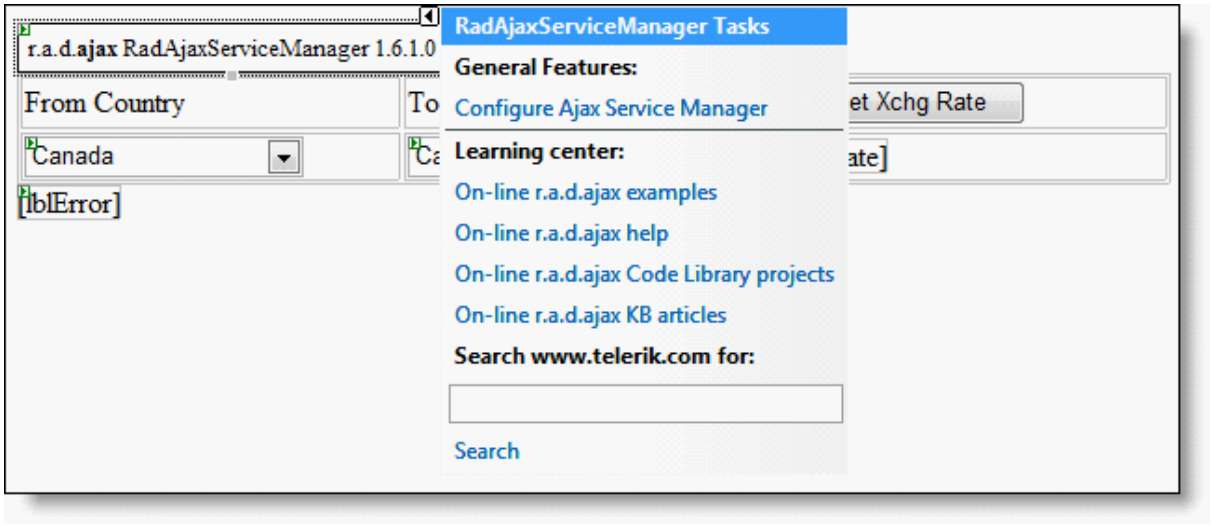


The layout

12. Now we need to populate the country lists with some valid lists. The countries that can be used are described on the service page of the CurrencyExchangeRate service. Lets just add in some of them hard coded (add the same list to ddlTo as well):

```
<asp:DropDownList  ID=ddlFrom  runat=server  width=150px>
      <asp:ListItem  Text="Canada"></asp:ListItem>
      <asp:ListItem  Text="Finland"></asp:ListItem>
      <asp:ListItem  Text="Germany"></asp:ListItem>
      <asp:ListItem  Text="Sweden"></asp:ListItem>
       <asp:ListItem  Text="Switzerland"></asp:ListItem>
      <asp:ListItem  Text="United  States"></asp:ListItem>
</asp:DropDownList>
```

13. The next step is to start hooking up the AJAX. First, drag a RadAjaxServiceManager onto the form:



The RadAjaxServiceManager

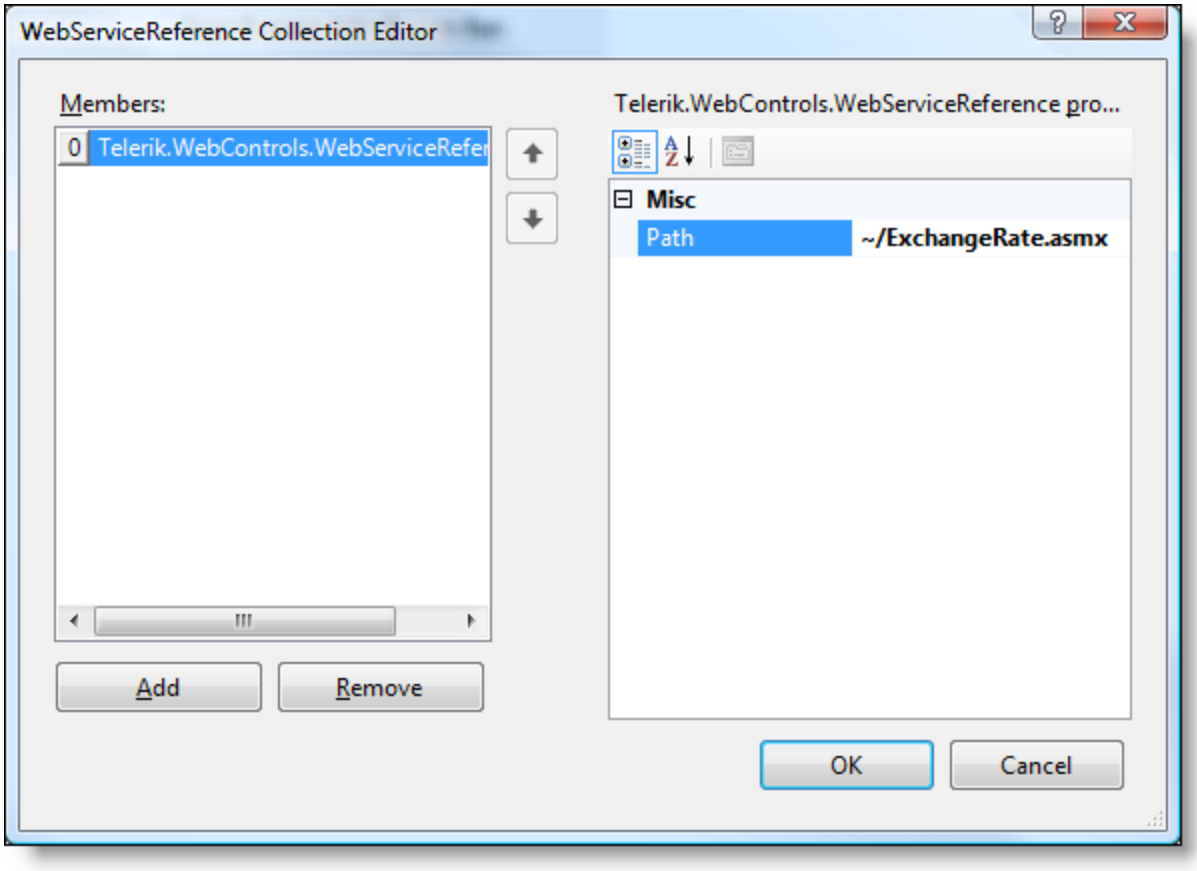
14. Actually, in my Visual Studio 2005 SP1 I can't drag it onto the form... I had to cheat a little. I dragged a RadAjaxManager onto the from instead, which adds the RadAjax.Net2 assembly to the Project References, and inserts the correct Register directive at the top of the form:

```
<%@ Register Assembly="RadAjax.Net2" Namespace="Telerik.WebControls"
TagPrefix="radA" %>
```

15. I then deleted the RadAjaxManager and manually added this markup within the form (it can go anywhere) :

```
<radA:RadAjaxServiceManager ID="RadAjaxServiceManager" runat=server>
</radA:RadAjaxServiceManager>
```

16. From the Smart Tag of the RadAjaxServiceManager (what a mouthful that is!), select the Configure Ajax Service Manager Link. In the WebServiceReference Collection Editor dialog, click the Add button. Type in the Path below and click OK:



Adding the Web Service to the Service Manager

17. Now the RadAjaxServiceManager knows about our Web Service, and when the page is rendered, it automatically injects a Javascript proxy that can be used to invoke the service through AJAX. The proxy method has the same class name as the WebService, and the same method name as the WebMethod. It takes the same parameters as the WebMethod, plus two additional ones, ServiceCompleteCallback and ServiceErrorCallback. The former is called when the AJAXified Web Service call returns successfully, and the latter if an error occurs. So, the call should look something like this:

```
ExchangeRate.GetExchangRate(
   country1, country2,
    ServiceCompleteCallback,  ServiceErrorCallback);
```

18. We want to call the method when the user clicks the Button on the from , so lets add some client side script to the button:

```
<asp:Button ID=pbCalc runat=server Text="Get Xchg Rate"
   OnClientClick="CallService(); return false;" />
```

19. Now, lets implement the CallService javascript function:

```
<script type="text/javascript">
function CallService()
{
    var from=document.getElementById("ddlFrom");
    var to=document.getElementById("ddlTo");

   if(from.value != "" && to.value !="" )
   {
       ExchangeRate.GetExchangRate(from.value, to.value,
         ServiceCompleteCallback, ServiceErrorCallback);
   }
   else
   {
      alert("Please select two countries!");
   }
}
</script>
```

Some things worth mentioning here are:

- First, we need to find the client side references to the two DropDownLists (which are actually rendered as SELECT elements).
- If both are found, we check to make sure they have values and then pass them to the proxy, along with the names off the ServiceCompleteCallback and ServiceErrorCallback.
- Otherwise, we show an error message. Since we are using DropDownLists, this error can never occur, the code is just in their to illustrate how to structure the call setup.
- Note that the call to the Web Service is asynchronous, and the return value is not available until the ServiceCompleteCallback is reached.

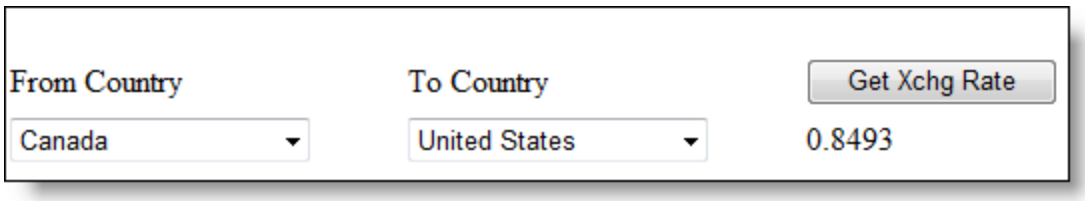
20. Now, add the definitions of the success and failure callbacks (place these under the CallService function, within the same script block):

```
function ServiceCompleteCallback(ResponseAsJSON, ResponseAsXml, ResponseAsText)
{
 var rate= ResponseAsText;
 var element = document.getElementById("lblRate");
 if(element != null)
 {
    element.innerHTML = rate;
 }
}

function ServiceErrorCallback(args)
{
   var label = document.getElementById("lblError");
   label.innerHTML = "Error occurred: " + args.ErrorText + "<br />" + "Details: " + args.Text;
}
```

Some comments:

- The return value of the AJAXed Web Service call is served up in three formats: as a JSON object, as XML, and as Text. Read more about JSON in the chapter "Introduction to AJAX". In our case, we just need a simple float return value, so the ResponseAsText is what we need.
- To display the result, we need to find the client side representation of the Label control lblRate. It is rendered as a SPAN element. We then set it's innerHTML to the returned string. In more complex scenarios, you could return something like an HTML table of result to assign to the innerHTML of a DIV, or you could return an array as a JSON object that you convert to a javascript array object and iterate over
- In the error callback, we use the same technique to find the client side rendering of the lblError Label, and then extract some error information from the args.ErrorText and args.Text

OK, that wraps up the coding, start the application, choose two countries and hit the Get XChange Rate button!



The final output

Note the instantaneous response, with no Postback or screen refresh. Consider what is happening here when you click the button:

- Some javascript gathers parameter values from the form and calls a generated web method proxy class
- The proxy sends an AJAX request to your Web Service
- The Web Service instantiates a proxy to the foreign Web Service and calls it, passing in the parameters.
- The foreign Web Service returns a result,
- Your web service returns the result back to the AJAX caller
- The client side AJAX framework code now invokes your ServiceComplete callback
- The callback get the returned result and renders it in the output area.

Pretty complex stuff going on here!

1.3.6 The RadAjaxTimer

The final control to cover in this chapter is the RadAjaxTimer. This is a component that can be set to periodically raise a Postback event, which if channeled through an AJAX callback can be used to update a Grid or some other monitor/dashboard of information.

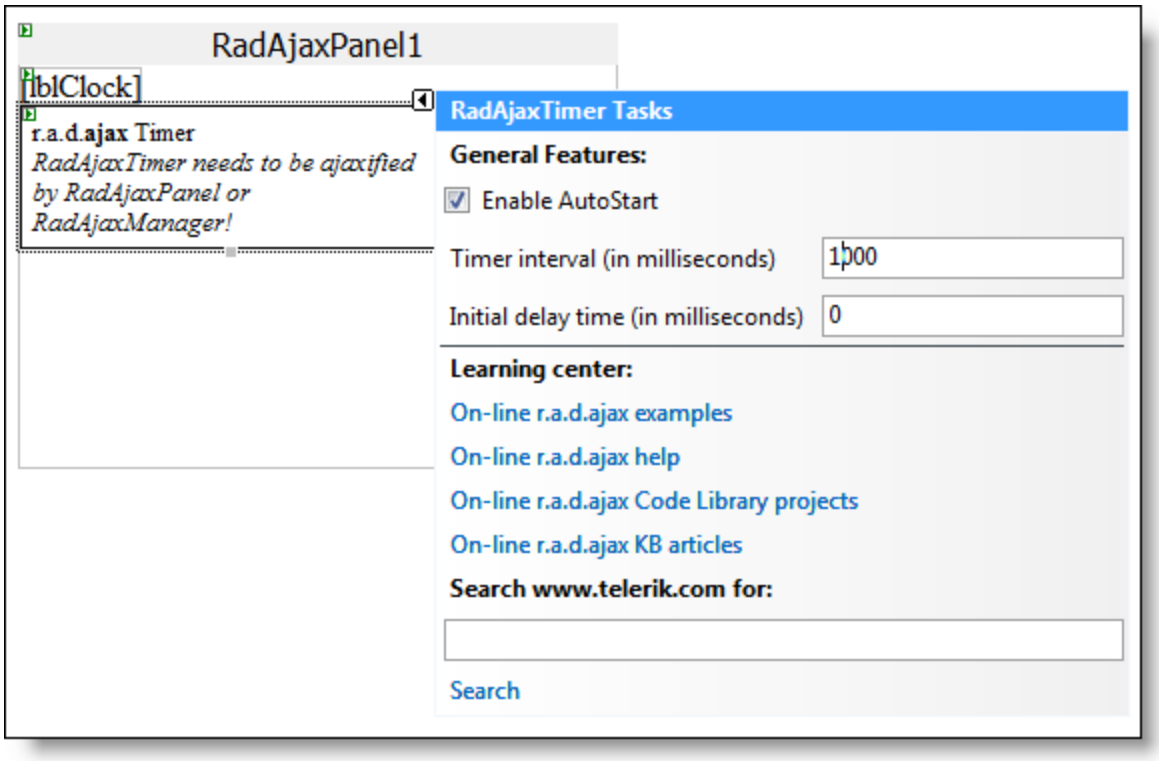
Lab: Using the RadAjaxTimer

Lets try a really trivial example to show how the bits and pieces fit together: the timer will fire once a second, and in response the current time will be rendered...in other words, a server based clock!

1. Create a new ASP.NET Web Application Project called AjaxTimer

2. Drag a RadAjaxPanel onto the form

3. Drag a Label (lblClock) and a RadAjaxTimer into the RadAjaxPanel. Clear the Text of lblClock. Use the Smart Tag of the RadAjaxTimer to set the Timer interval to 1000 milliseconds:

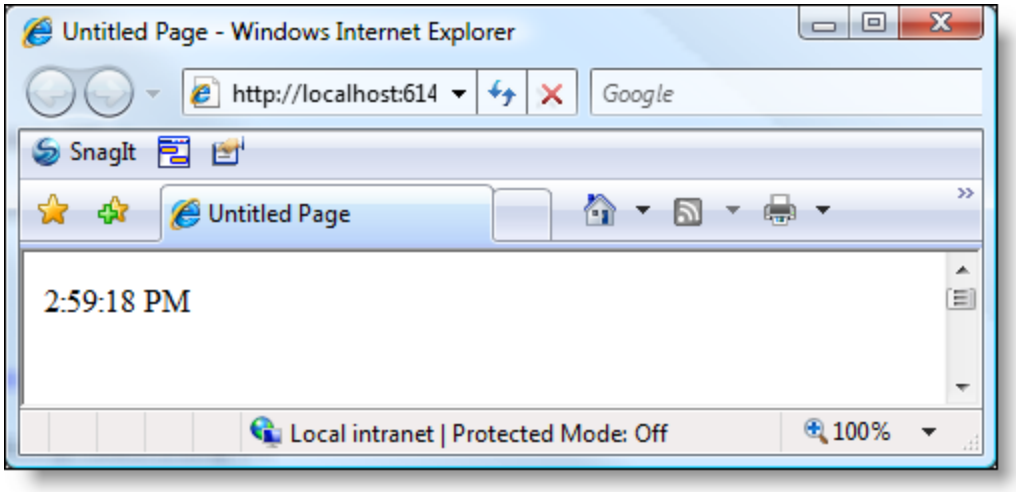


The form layout

4. Double click the RadAjaxTimer and add this code to the event handler for the Tick event:

```
C# Example:
protected void RadAjaxTimer1_Tick(object sender, Telerik.WebControls.TickEventArgs e)
{
    lblClock.Text = DateTime.Now.ToLongTimeString();
}

VB Example:
Protected Sub RadAjaxTimer1_Tick(ByVal sender As Object, _
  ByVal e As Telerik.WebControls.TickEventArgs)
    lblClock.Text = DateTime.Now.ToLongTimeString()
End Sub
```

5. That's all there is to it! Run the application and admire the fantastic AJAX clock that you have created!



The AJAX clock

1.3.7 Measuring Ajax Performance

One reason to use AJAX is to minimize network traffic for better performance. But in some complex pages with several controls updating each other performance may be lacking. AJAX performance depends on two factors:

1. Amount of time taken for XHR communication between browser and the web server.

2. Amount of time taken for the DOM to update and render.

If you are using the Telerik RadAjaxManager or RadAjaxPanel you can hook up client events OnRequestStart, OnResponseReceived and OnResponseEnd to measure the time for the AJAX request to complete. The time difference between RequestStart and ResponseReceived will give you the time taken for XHR Communication and the time difference between ResponseReceived and ResponseEnd is the time taken for DOM update on the client. Add the following code to your page .aspx.cs:

```
protected void Page_Init(object sender, EventArgs e)
 {
      RadAjaxManager1.AjaxSettings.AddAjaxSetting(myInitiatorControl,
       myUpdatedControl, AjaxLoadingPanel1);
     RadAjaxManager1.ClientEvents.OnRequestStart = "RequestStart";
     RadAjaxManager1.ClientEvents.OnResponseReceived = "ResponseReceived";
     RadAjaxManager1.ClientEvents.OnResponseEnd = "ResponseEnd";
}
```

...and the following javascript to your .aspx file

```
<script type="text/javascript">
var start = null;
var received = null;

function RequestStart()
{
document.body.style.cursor = "wait";
start = new Date();
}

function ResponseReceived()
{
received = new Date();
```

```
}

function  ResponseEnd()
{
document.body.style.cursor  =  "default";
   var end = new Date();
   document.title = "Server time: " + (received - start) +
     ", Client time: " + (end - received) + ", Total time: " + (end - start);
}
</script>
```

1.3.8 Summary

In this chapter, you have learned how to:

- Wrap a set of controls in a RadAjaxPanel in order to convert Postbacks to AJAX callbacks, with no coding required
- Use the AjaxLoadingPanel to supply feedback to the user while the AJAX request is running
- Use the RadAjaxManager for more fine grained control of AJAX pairs (Initiators and Targets), including usage with User Controls, Master Pages and within Grids.
- Set the focus using the FocusControl method of the RadAjaxManager or RadAjaxPanel
- Use the RadAjaxServiceManager to call Web Services through AJAX
- Use a RadAjaxTimer to periodically refresh data using AJAX.

1.3.9 Other sources of information

Be sure to check the Telerik online help for RadAjax at http://www.telerik.com/help/aspnet/ajax/. Here you will find material not covered in this chapter, including how to:

- hook into four client side javascript events as AJAX requests are executed,
- disable controls during the processing of an AJAX request
- cancel an AJAX request
- disable AJAX for browsers that don't support XmlHttpRequest
- make a control within a RadAjaxPanel perform a Postback instead of a callback
- perform Response.Redirects
- work with uploading and downloading files in an AJAX context
- and much more.

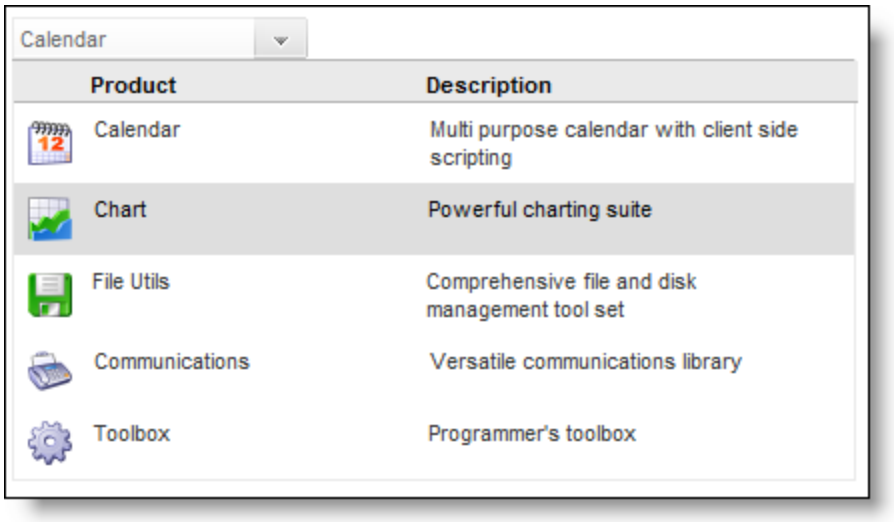
Also, check out the online demos at Telerik's site.

1.4 RadComboBox

1.4.1 Getting Started

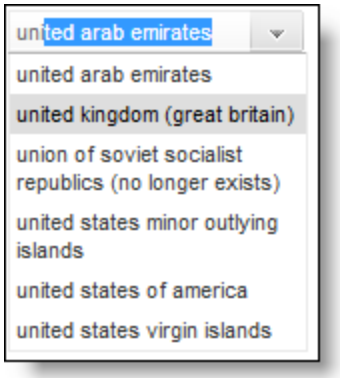
The Telerik RadComboBox is a versatile control that allows you to create powerful user interfaces. It has intrinsic support for Ajax in a few different modes, including fetching items on demand as you type, and can be used to create compelling multi column drop down lists with it's support for templates. The control can be used as replacement for the standard ASP.Net DropDownList, although it does not descend from it and converting code to use it instead of the DropDownList has a few gotcha's. The control can be bound declaratively to .NET 2.0 datasources.

Here is an example of a multi column RadComboBox using templates:



A multi column RadComboBox displaying Images

And here is a load on demand RadComboBox:

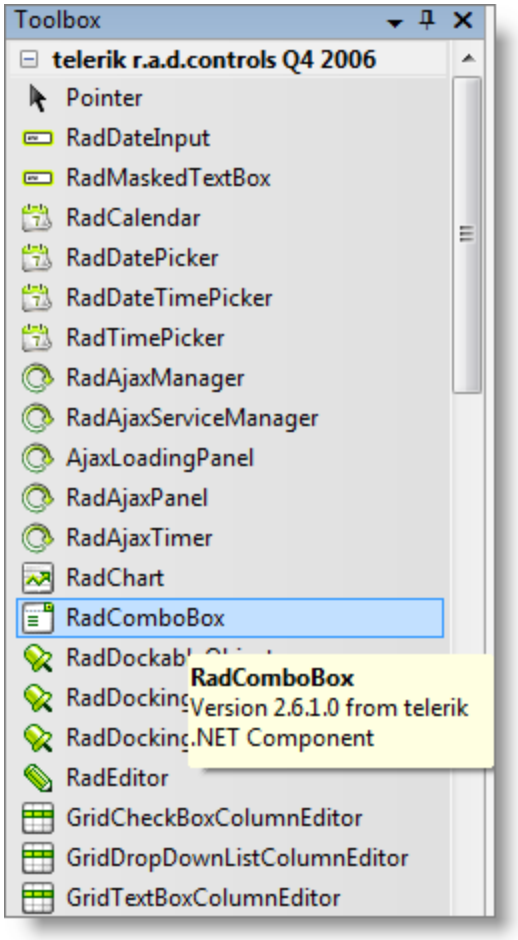


Load on Demand RadCombBox

In this chapter we will build both of these types of combos and more.

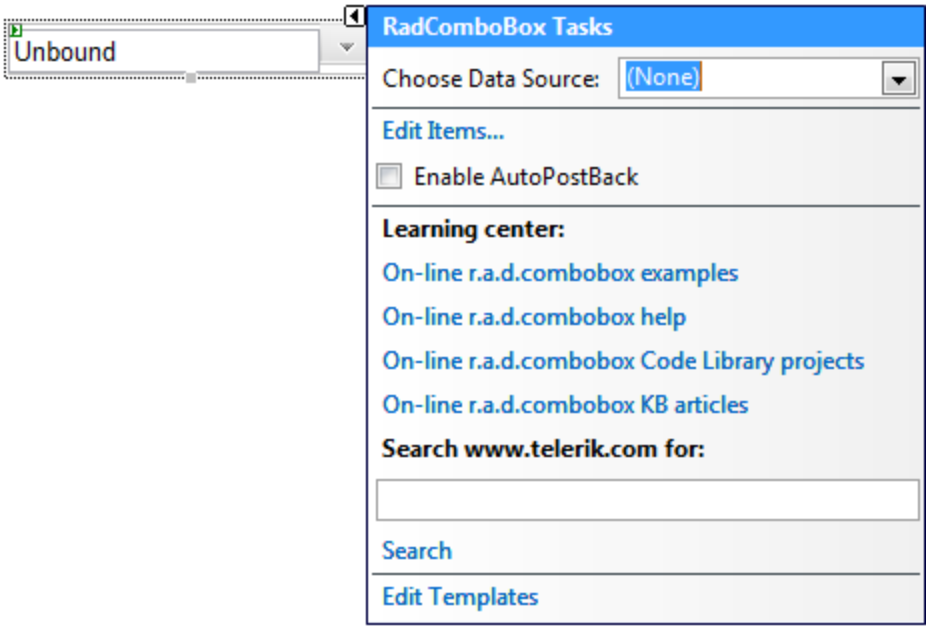
1.4.2 Using RadComboBox in the Designer

Start a new Web Application Project. Find the RadComboBox under the Telerik RadControls section of your Toolbox:



The RadComboBox in the Toolbox

Drag a RadComboBox from the toolbox onto your form and click the Smart Tag. Notice the links for Telerik site search, examples, help, sample projects and knowledge base articles. Also notice the Edit Templates link.



The RadComboBox Smart Tag editor

Lab: Bind to an AccessDataSource declaratively

In this lab you will learn how to bind a RadComboBox to a table in an Access database declaratively, using the Visual Studio designer.

1) Create a new ASP.NET project called "Templates".

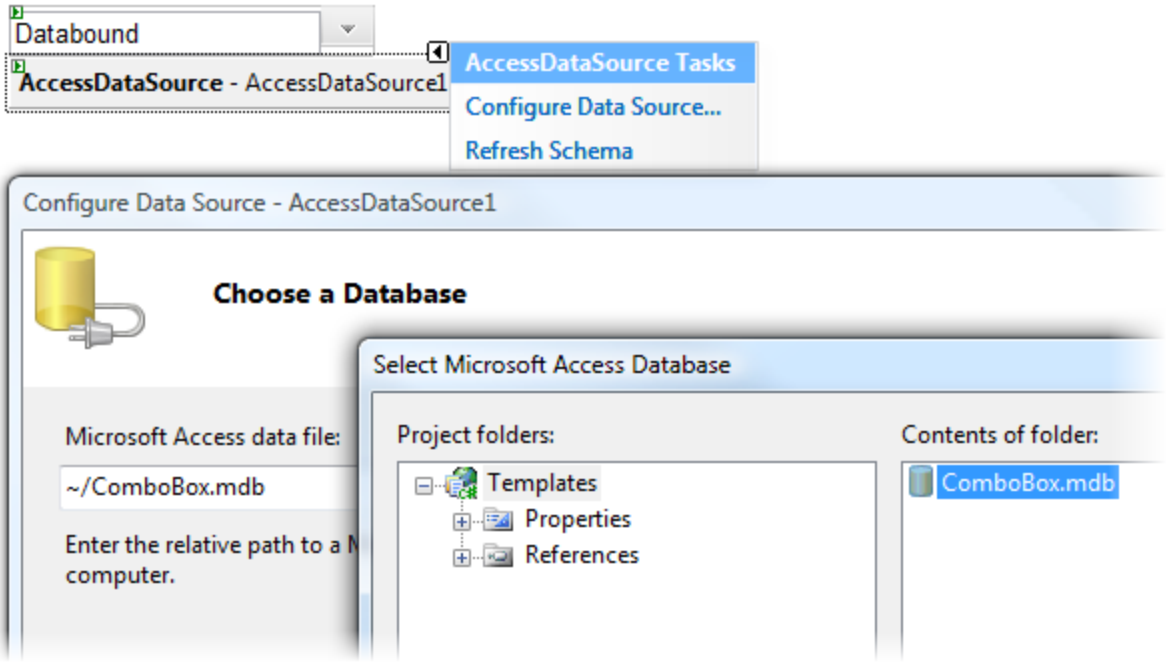
2) Add a RadComboBox control to the main form.

3) Add the file "ComboBox.mdb" to the project. ComboBox.mdb can be found in the \Rad ComboBox \Projects\Data directory. If the "Data Source Configuration Wizard" appears, cancel the dialog:

The CombBox.mdb added to the project

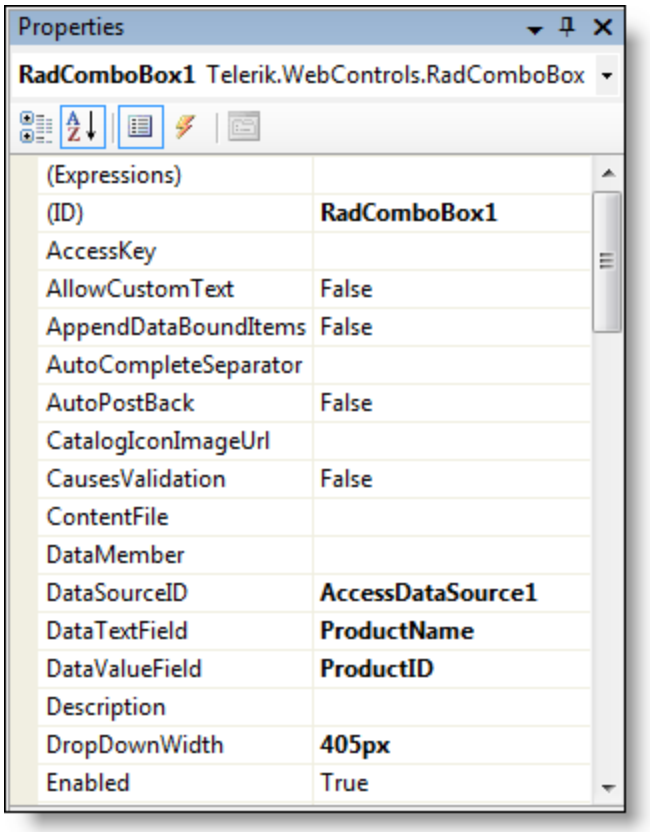
4) From the ToolBox drag an AccessDataSource to the page.

5) Choose the "Configure Data Source" Smart Tag. In the Configure Data Source dialog, browse to ComboBox.mdb and select it. The Microsoft Access data file entry should now read "~/ComboBox. mdb". Click the Next button. Select all fields of the table "Product". Click the Next button. Click the Finish button.



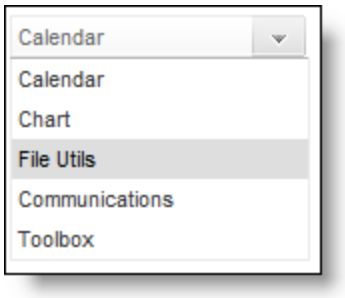
Configuring the AccessDataSource

6) In the properties window for the RadComboBox set the following properties:



Basic RadComboBox properties

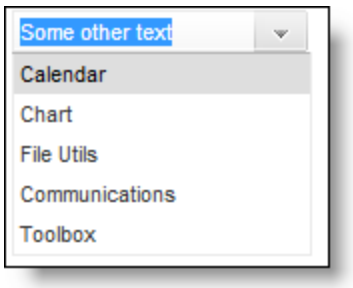
7) Run the application:



Output of finished lab

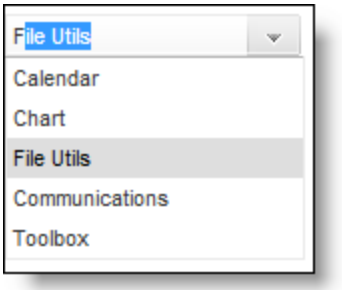
Here are some things worth noting about this simple example:

- The DataSourceID, DataTextField and DataValueField work just like the regular DropDownList of ASP.Net 2.0. There is no DataTextFormatString in the RadComboBox, but the template support is much more powerful (this is the topic of the next lab)
- The AllowCustomText=False prevents you from typing into the combo. If you set it to True, you can type things in the edit box portion of the combo that are not in the list itself:



Typing in the Textbox

- However, when it is set to False, you can not type in the Textbox.
- Setting the property MarkFirstMatch=True when AllowCustomText=False does allow you to type in the Textbox, but only strings that match one of the items in the list, and the matching entry is displayed as you type (this is also referred to as auto complete):



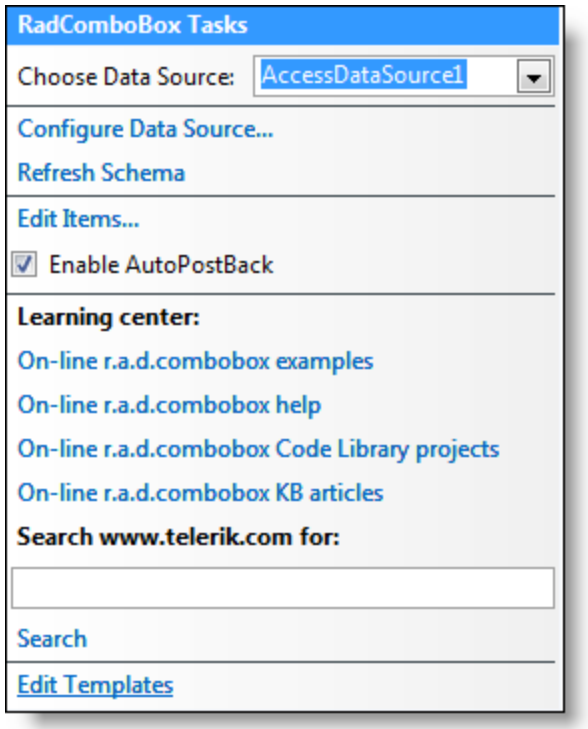
Match displays as you type

- In this mode you can erase what you have typed to clear the current match.
- Also, note that the built in keyboard support. Alt+Arrow Down will drop down the list. You can then use the Arrow Up and Arrow Down keys to scroll through the list. Alt+Arrow Up will close the list again. Enter or Tab will select the currently highlighted item in the list.

1.4.3 Using Templates to create Multi Column DropDowns

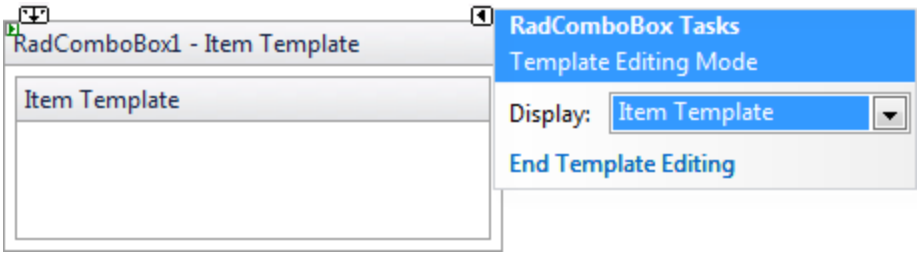
In the previous Lab you saw how to use the RadComboBox to create a typical DropDownList. But what if you wanted the drop down portion to contain multiple columns for each row, maybe showing a product description, even an image? This can be achieved using the built in template support in the RadComboBox.

There are two templates that you can edit, the Header Template and the Item Template. The Header Template is rendered at the top of the drop down portion, and the item template is rendered for each of the items in the list. The templates can contain pretty much anything, and support data binding. You can access the templates from the Smart Tag on the RadComboBox in the designer:



The Edit Templates link is displayed by the Smart Tag editor

However, when you click it, your only choice is Item Template - there is no way to choose the Header Template from here!



Editing the template

It is easier to just switch to source view and edit the templates directly from there. Use this general structure:

```
<radC:RadComboBox   ID="RadComboBox1"   runat="server" DropDownWidth="408px">
     <HeaderTemplate>
```

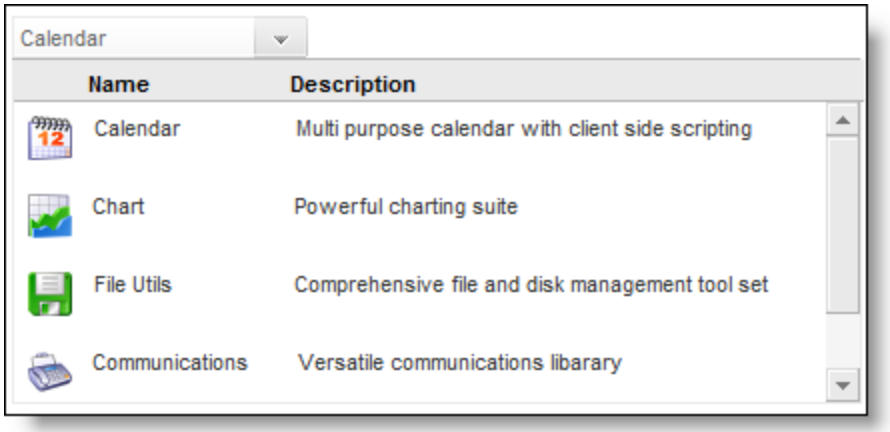
```
    </HeaderTemplate>
    <ItemTemplate>
    </ItemTemplate>
</radC:RadComboBox>
```

You will usually want the drop down portion to be wider than the actual Textbox part of the combo. To this end you can set the DropDownWidth property.

To structure the layout in the header and item templates, and make them line up, it is helpful to use HTML tables. For instance:

```
<HeaderTemplate>
    <table width=400px  border=0>
        <tr>
            <td width=30px></td>
            <td width=100px>Name</td>
            <td width=270px>Description</td>
        </tr>
    </table>
</HeaderTemplate>
<ItemTemplate>
    <table width=400px  border=0>
        <tr>
            <td width=30px valign=top align=left></td>
            <td width=100px valign=top align=left></td>
            <td width=270px valign=top align=left></td>
        </tr>
    </table>
</ItemTemplate>
```

It is generally a good idea to make the table say 8 pixels narrower than the DropDownWidth of the combo, so that there is room to render a vertical scroll bar if more items are rendered than fit into the allotted height (which is determined by the Height property of the RadComboBox. Note: If you make the table the exact same width as DropDownWidth you get horizontal scrollbars where you don't need them.
When there are more items than fit heightwise, vertical scroll bars appear, and those eat into the horizontal space, causing horizontal bars to appear as a side effect. If the width accommodates the vertical bars to start with, the horizontal ones don't show.



Leave room for the vertical scroll bar

In the next lab you will learn how to build the multi column drop down shown above!

Lab: Create a Multi Column Drop Down List

In this lab you will extend the previous lab by providing a custom multi column dropdown that show an image and two text fields for each item.

We will continue to use the same Access database. We are binding to the Product table, which looks like this:

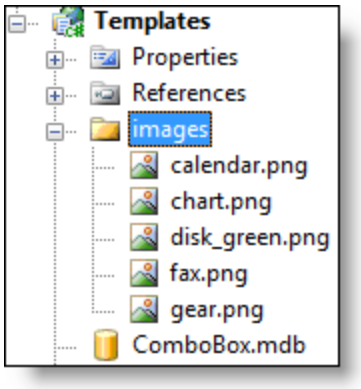
Product

ProductID	ProductNam	ProductDescription	ImageFile
1	Calendar	Multi purpose calendar with client side scripting	calendar.png
2	Chart	Powerful charting suite	chart.png
3	File Utils	Comprehensive file and disk management tool set	disk_green.png
4	Communicatio	Versatile communications libarary	fax.png
5	Toolbox	Programmer's toolbox	gear.png

The Product table

Each row has a unique ProductID, a ProductName, a ProductDescription and an ImageFile (which is just the filename of the image for that product, not the actual image). First, we need to make the images available to the web site:

1) Create an image folder in your project, and copy the images in the \Rad ComboBox\Projects \Templates\Images directory to that folder (or use your own images, but you will then have to change the names of the images to match the names in the product table, or change the filenames in the ImageFile column):



The images in the project

2) Now, open the default.aspx page and view the source. Paste in the following code to define the header and item templates:

```
<radC:RadComboBox  ID="RadComboBox1"  runat="server"  DataSourceID="AccessDataSource1"
    DataTextField="ProductName"  DataValueField="ProductID"
    SkinsPath="~/RadControls/ComboBox/Skins"
   Width="150px"  MarkFirstMatch="True"  DropDownWidth="408px"
   HighlightTemplatedItems=true  Height=150px>
     <HeaderTemplate>
     </HeaderTemplate>
     <ItemTemplate>
     </ItemTemplate>
</radC:RadComboBox>
```

3) Flesh out the header template. This sets up a table with one row and three columns. It is important to specify the column widths, so that we can make the columns line up with the item template columns.

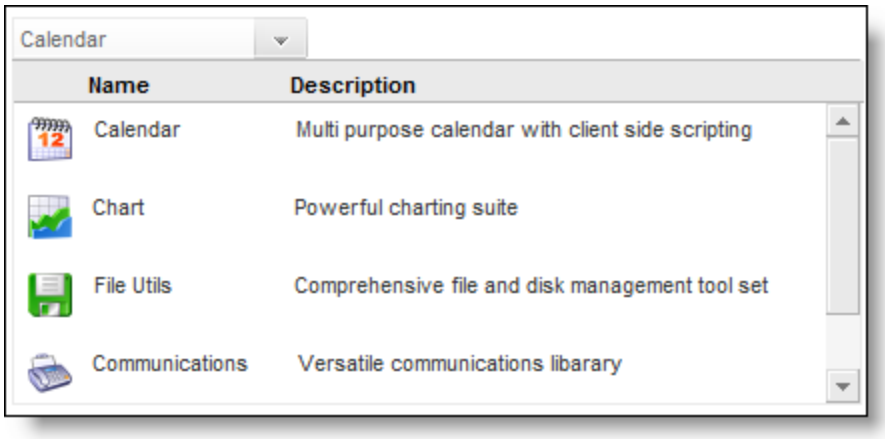
Also note that the table is 8 pixels narrower than the DropDownWidth of the combo.

```
<HeaderTemplate>
   <table width=400px border=0>
     <tr>
        <td width=30px></td>
        <td width=100px>Name</td>
        <td width=270px>Description</td>
     </tr>
   </table>
</HeaderTemplate>
```

4) Add in code for the item template. Use the same table widths. Set the vertical and horizontal alignment of the cells. Note that to display the image, we use a regular ASP.Net Image server control, whereas the product name and text are rendered just as inline text in the cells.

```
<ItemTemplate>
   <table width=400px border=0>
     <tr>
        <td width=30px valign=top align=left>
          <asp:Image runat=server ID=imgProduct
              ImageUrl='<%#  "~/Images/"+DataBinder.Eval(Container.DataItem,"ImageFile")%>'/>
       </td>
        <td width=100px valign=top align=left>
          <%# DataBinder.Eval( Container.DataItem, "ProductName" ) %>
       </td>
        <td width=270px valign=top align=left>
          <%# DataBinder.Eval( Container.DataItem, "ProductDescription" )   %>
       </td>
     </tr>
   </table>
</ItemTemplate>
```

5) Run the project. It should look something like this:



Lab output

Note the data binding expressions used to bind the item templates contents to the fields in the data source. Also, note that these fields do not have to be the same fields as the DataTextField and DataValueField of the RadComboBox. When you select a given row, the RadComboBox will set it's Text and Value to the correct values from that row, even if those values are not present in the bindings of the item template.

With this technique, you actually end up having one table for the header, and one table for each item. If there had also been a footer element, we could have started the TABLE tag in the header, created the table header row and TH tags for the column headings, and then had TRs and TDs in each item, and

finally closed the TABLE tag in the footer.

1.4.4 Load on Demand

One of the most powerful built in Ajax features of the RadComboBox is the Load on Demand capability. This is how it works:

1. The user types some text into the Textbox of the combo.
2. After a customizable delay (so that several characters can be typed), say 300 milliseconds, the combo box send an Ajax callback request to a server side method (which can be on the same page as the combo box, or another page, more on that later). The callback supplies the text that was typed (plus an optional string send from client side java script).
3. The server side method can then fetch data that partially matches the typed characters, for instance by querying a back end database with a LIKE expression. The resulting items are added to the combo box server side.
4. The items are returned to the browser and appear in the drop down list of the combo.

This is how it might look:

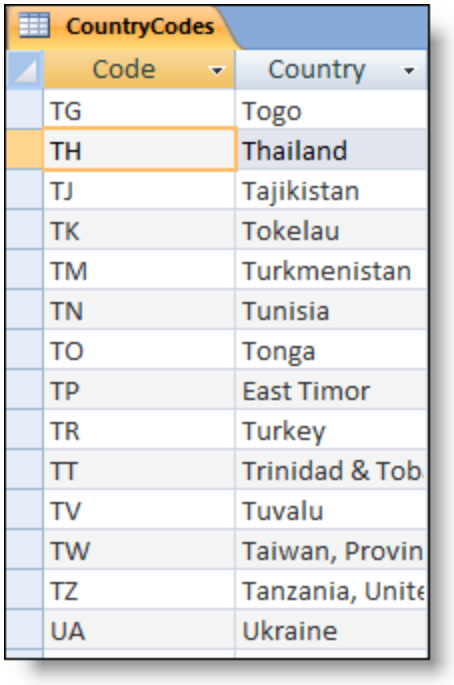
Load on demand in action

The whole cycle above is extremely fast using Ajax, and to the end user, it looks as if the drop down list is changing to match their typing with each key press. What makes this so useful is that the number of items could potentially be way to many to just show in a regular drop down list, maybe thousands, but with each key press the number is reduced drastically to a manageable amount of data that can be streamed over even a slow network. As with all Ajax, there is no postback, no screen refresh, no flicker. To the user it seems like all this data is ready available client side, yet it is actually being served up on demand by the server.

The labs in this section will show you how to create a basic load on demand combo, and then show some of the more advanced ways using this technique with External Streamer Pages - techniques that are a must to build reusable components that contain load on demand combos.

Lab: A simple load on demand combo

In this lab we will build a load on demand combo that will allow you to select a country. There are 245 such countries, each having a two letter country code, for instance TH for Thailand. The data for this lab is in an access database table, that looks like this:



CountryCodes table

1) Create a new ASP.NET project called "CountryCodes".

2) Add the file "ComboBox.mdb" to the project. ComboBox.mdb can be found in the \Rad ComboBox \Projects\Data directory. If the "Data Source Configuration Wizard" appears, cancel the dialog:

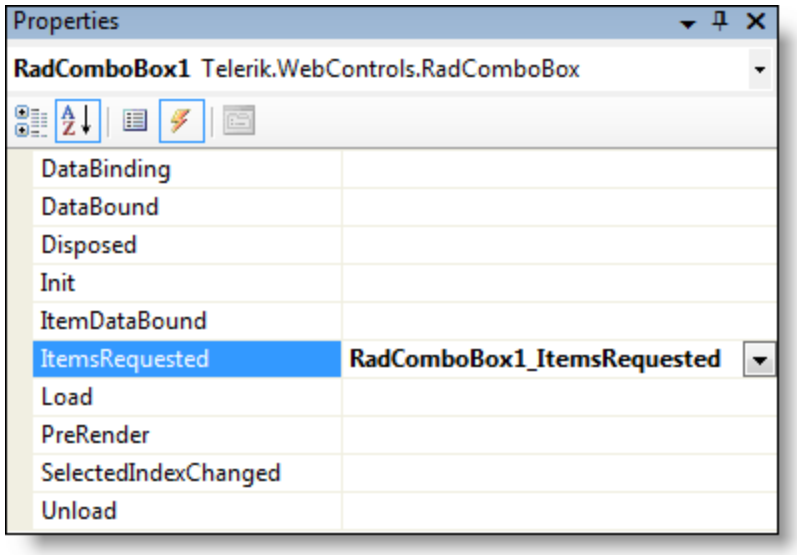
The CombBox.mdb added to the project

3) Add a RadComboBox control to the main form.

4) In the properties window for the RadComboBox set the following properties:

Property	Value	Comments
AllowCustomText	True	Must be set to allow typing in the Textbox
EnableLoadOnDemand	True	Setting this enables the Ajax callback when you type
MarkFirstMatch	True	This will cause the first partial match to be displayed in the textbox as you type
HighlightTemplatedItems	True	Highlights the current match in the drop down list as you type
ItemRequestTimeout	200	The delay in milliseconds before the Ajax callback is made after each keypress.

5) Double click the ItemsRequested event in the properties inspector of the RadComboBox



Creating the ItemsRequested event handler

6) Add this to the top of the code behind file:

```
C# Example:
using Telerik.WebControls;
using System.Data.OleDb;
using System.Data;

VB Example:
Imports Telerik.WebControls
Imports System.Data.OleDb
Imports System.Data
```

7) Create this method to fetch all the matching country code rows from the Access database, given a partial country name, and return them in a DataTable:

```
C# Example:
private DataTable FetchMatchingCountries(string partialName)
{
  string mdbPath = Server.MapPath("~/Combobox.mdb");
   OleDbConnection dbCon = new OleDbConnection(
     "Provider=Microsoft.Jet.OLEDB.4.0;Data    Source=" + mdbPath);
   dbCon.Open();
  string sql = "SELECT * from CountryCodes WHERE Country LIKE '" +
     partialName.Replace("'", "''") + "%'";
   OleDbDataAdapter adapter = new OleDbDataAdapter(sql, dbCon);
   DataTable dt = new DataTable();
    adapter.Fill(dt);
   dbCon.Close();
  return dt;
}

VB Example:
Private Function FetchMatchingCountries(ByVal partialName As String) As DataTable
  Dim mdbPath As String = Server.MapPath("~/Combobox.mdb")
  Dim dbCon As OleDbConnection = New OleDbConnection( _
```

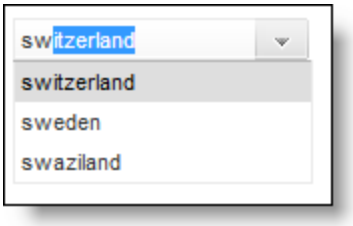
```
      "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + mdbPath)
   dbCon.Open
  Dim sql As String = "SELECT * from CountryCodes WHERE Country LIKE '" + _
     partialName.Replace("'", "''") + "%'"
  Dim adapter As OleDbDataAdapter = New OleDbDataAdapter(sql, dbCon)
  Dim dt As DataTable = New DataTable
   adapter.Fill(dt)
   dbCon.Close
  Return dt
End Function
```

8) In the event handler for ItemsRequested, call this method, passing in e.Text (which contains the text typed so far in the Textbox of the combo). Then, clear the combobox and loop through all the rows in the table, adding them to the comboboxes list of items:

```
C# Example:
protected void RadComboBox1_ItemsRequested(object o,
    Telerik.WebControls.RadComboBoxItemsRequestedEventArgs e)
{
   DataTable dt = FetchMatchingCountries( e.Text );
   RadComboBox combo = (RadComboBox)o;
   combo.Items.Clear();
  foreach (DataRow row in dt.Rows)
  {
     RadComboBoxItem item = new RadComboBoxItem(
      row["Country"].ToString().ToLower());
     item.Value = row["Code"].ToString();
     combo.Items.Add(item);
  }
}

VB Example:
Protected Sub RadComboBox1_ItemsRequested(ByVal o As Object, _
  ByVal e As    Telerik.WebControls.RadComboBoxItemsRequestedEventArgs)
Dim dt As DataTable = FetchMatchingCountries(e.Text)
Dim combo As RadComboBox = CType(o, RadComboBox)
combo.Items.Clear
For Each row As DataRow In dt.Rows
  Dim item As RadComboBoxItem = New    RadComboBoxItem(row("Country").ToString.ToLower)
   item.Value = row("Code").ToString
   combo.Items.Add(item)
Next
End Sub
```

Note that the RadComboBox contains RadComboBoxItems, not ListItems! If you are converting a ASP.Net DropDownList to a RadComboBox, you will most likely need to change some markup and/or code to accommodate this difference. Another gotcha is that the RadComboBox by default does not select its first item, whereas the DropDownList does. So any code assuming that an item is selected will break.

9) Run the application. Start typing... see what happens when you erase using backspace, or type something that doesn't match anything in the list...



Output of lab

There is one flaw with the current implementation - it returns all records in the database if you just click the arrow down icon on the combo or click in the combo textbox without typing anything. If the database has say 1000 or 10000 records in it, you don't want to populate the list until at least one or two characters have been typed. This can easily be done server side.

10) Add the following code to the top of the ItemsRequested event handler, and try running again:

```
C# Example:
if (e.Text.Length < 1)
  return;

VB Example:
If e.Text.Length < 1 Then
   Return
End If
```

The server event will still fire, but will mot fetch any data unless at least the first character has been supplied. You can also abort the fetch on the client side using the client side object model in javascript:

11) Add the following attribute to the markup of the RadComboBox:

```
OnClientItemsRequesting="RequestingHandler"
```

12) Now the client side javascript function RequestingHandler will be called just before the Ajax callback is made. To abort it, return false. Here is the function:

```
<script language="javascript">
function RequestingHandler(comboBox)
{
   if (comboBox.GetText().length<1)
  {
     comboBox.ClearItems();
    return false;
  }
}
</script>
```

Note the call to ClearItems to zero out the drop down list. Without this call, if you type a letter and then erase it by backspacing, the list of matches for the letter stays in the drop down list.

Now that we have the basic load on demand functionality in place, we can start working on more advanced usage scenarios.

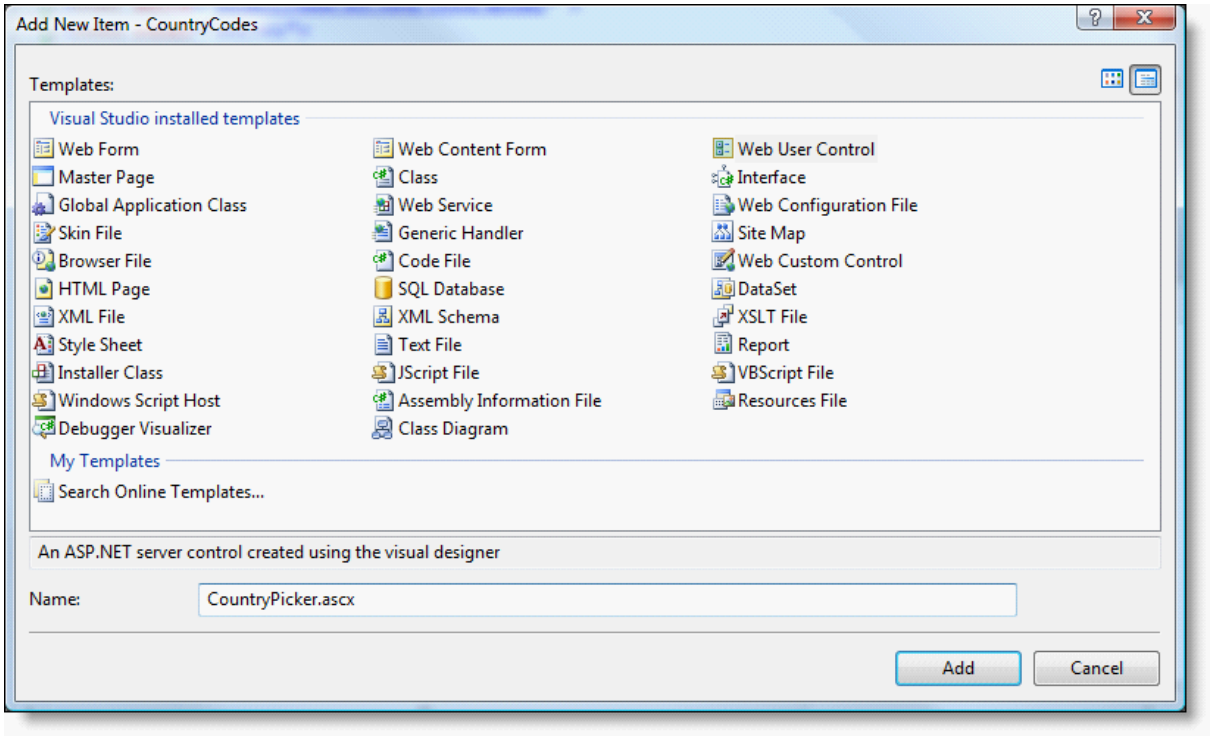
Lab: External Streamer Pages

With the advent of load on demand comboboxes, it is tempting to start writing some powerful, reusable components that provide various kinds of custom load on demand lookups. One easy way to do this is to place a load on demand RadComboBox in a user control (ascx), but this comes with a caveat: the OnItemsRequested handler can not be a method of a user control, it has to be a method of a page. One way to work around this limitation is to use what is called an External Streamer Page. That is the topic of this lab.

First, lets move our RadComboBox into a user control.

1) Choose Add New Item on the context menu of your CountryCodes project.

2) In the Add New Item dialog, select Web User Control. Name the control CountryPicker.ascx:



Creating the CountryPicker control

3) Cut and paste the RadComboBox HTML code from default.aspx into the CountryPicker.aspx markup:

4) Also cut and past the Tag Prefix code

```
<%@  Register  Assembly="RadComboBox.Net2"
   Namespace="Telerik.WebControls"  TagPrefix="radC"  %>
```

5) Add this attribute to the RadComboBox:

```
ExternalCallBackPage="~/CountryStreamer.aspx"
```

This tells the Ajax framework, that when the Ajax callback is performed, to fill the combo box, it is to be

directed to the web page CountryStreamer.aspx. The requirement is that there be a RadComboBox on that page with the same ID as on the calling page, and that the streamer page implements the ItemsRequested handler. Let's set this up:

6) Add a new Web Form to the project, called CountryStreamer.aspx

7) Copy paste the RadComboBox from CountryPicker.aspx to CountryStreamer.aspx (and the Tag Prefix). If you still have the client side java script that intercepts OnClientItemsRequesting, you can delete that code and attribute on the streamer page (that page will never be truly rendered in full to the browser, it is just there to server up the contents of the combo box). Also, delete the ExternalCallbackPage attribute from the copy on the streamer page. And delete the OnItemsRequested attribute from the CountryPicker.ascx version of the combo (as it is forwarding that call to the streamer page).

8) Cut and paste the server side event handler code from the Default.aspx code-behind to the streamer page (both the event handler `RadComboBox1_ItemsRequested` and the helper method `FetchMatchingCountries`, and the using/Imports statement at the top of the file).

At this point, the CountryPicker.ascx should declare the combo box something like this:

```
<radC:RadComboBox  ID="RadComboBox1"  runat="server"
     AllowCustomText="True"
     EnableLoadOnDemand="True"
     MarkFirstMatch="True"
      SkinsPath="~/RadControls/ComboBox/Skins"
    Width="150px"
     ItemRequestTimeout="200"
     HighlightTemplatedItems=True
      OnClientItemsRequesting="RequestingHandler"
      ExternalCallBackPage="~/CountryStreamer.aspx">
</radC:RadComboBox>
```

The CountryPicker code behind should be almost empty, just containing an empty Page_Load stub.

The CountryStreamer.aspx declaration should look like this:

```
<radC:RadComboBox  ID="RadComboBox1"  runat="server"
   AllowCustomText="True"
   EnableLoadOnDemand="True"
   MarkFirstMatch="True"
    SkinsPath="~/RadControls/ComboBox/Skins"
  Width="150px"
   ItemRequestTimeout="200"
   HighlightTemplatedItems=True
    OnItemsRequested="RadComboBox1_ItemsRequested">
</radC:RadComboBox>
```

With this code behind:

```
C# Example:
...
using  Telerik.WebControls;
using  System.Data.OleDb;

...

protected void  RadComboBox1_ItemsRequested(object o,
    Telerik.WebControls.RadComboBoxItemsRequestedEventArgs  e)
{
  if (e.Text.Length < 1)
    return;
   DataTable dt = FetchMatchingCountries(e.Text);
```

```
   RadComboBox combo = (RadComboBox)o;
    combo.Items.Clear();
  foreach (DataRow row in dt.Rows)
  {
     RadComboBoxItem item = new  RadComboBoxItem(row["Country"].ToString().ToLower());
     item.Value = row["Code"].ToString();
      combo.Items.Add(item);
  }
}

private  DataTable  FetchMatchingCountries(string  partialName)
{
  string  mdbPath = Server.MapPath("~/Combobox.mdb");
   OleDbConnection dbCon = new  OleDbConnection(
    "Provider=Microsoft.Jet.OLEDB.4.0;Data    Source=" + mdbPath);
   dbCon.Open();
  string sql = "SELECT * from CountryCodes WHERE Country LIKE '"
     + partialName.Replace("'", "''") + "%'";
   OleDbDataAdapter adapter = new  OleDbDataAdapter(sql,  dbCon);
   DataTable dt = new  DataTable();
    adapter.Fill(dt);
   dbCon.Close();
  return dt;
}
```

VB Example:

```
...
Imports   Telerik.WebControls
Imports   System.Data.OleDb

...

Protected Sub   RadComboBox1_ItemsRequested(ByVal o As Object, _
  ByVal e As    Telerik.WebControls.RadComboBoxItemsRequestedEventArgs)
  If e.Text.Length < 1 Then
    Return
  End If
  Dim dt As  DataTable  =  FetchMatchingCountries(e.Text)
  Dim combo As  RadComboBox  =  CType(o,  RadComboBox)
   combo.Items.Clear
  For Each row As  DataRow In  dt.Rows
    Dim item As  RadComboBoxItem  = New    RadComboBoxItem(row("Country").ToString.ToLower)
     item.Value  =  row("Code").ToString
     combo.Items.Add(item)
  Next
End Sub

Private Function  FetchMatchingCountries(ByVal  partialName As String) As  DataTable
  Dim  mdbPath As String  =  Server.MapPath("~/Combobox.mdb")
  Dim dbCon As  OleDbConnection  = New  OleDbConnection( _
      "Provider=Microsoft.Jet.OLEDB.4.0;Data  Source=" +  mdbPath)
   dbCon.Open
  Dim sql As String = "SELECT * from CountryCodes WHERE Country LIKE '" _
     + partialName.Replace("'", "''") + "%'"
  Dim  adapter As  OleDbDataAdapter  = New  OleDbDataAdapter(sql,  dbCon)
  Dim dt As DataTable = New  DataTable
   adapter.Fill(dt)
   dbCon.Close
  Return dt
End Function
```

9) Your user control is now ready to use, so lets make it available to the default.aspx page by registering a Tag Prefix in that page:

```
<%@ Register TagPrefix="uc1" TagName="CountryPicker" Src="~/CountryPicker.ascx" %>
```

10) And now, you can declare an instance of your user control:

```
<uc1:CountryPicker ID=cpPicker runat=server />
```

11) Try running the project. It should look exactly as before, but now you have a Country Picker User Control that you can re-use in you applications!

There are other scenarios that require an external streamer page, for instance if your load on demand RadComboBox is in a grid and is only created when you go to edit a row. Just follow the same technique - point the ExternalCallbackPage attribute of the combo to a streamer page and it will work.

1.4.5 Cascading combos

On many web sites you will see a series of cascading combo boxes. For instance, you might need to select a make and model of car. First, a combo box of all possible makes is presented, and you choose a make. This then refreshes a combo box of all models for that make (usually there is a year combo as well).

Obviously, you don't want to load the second model combo with all possible models, just the ones for the selected make, so it has to be dynamically populated when the make changes. This can be done in a number of ways using Ajax. You can use the Telerik RadAjaxManager or RadAjaxPanel to accomplish this, but I would like to show you another way that uses the intrinsic Ajax support of the RadComboBox. This is the topic of the next lab.

Lab: Cascading Combos

In this lab you will create a form that uses cascading combo boxes to select a make and model of car. We will be using an access database, with two tables - you guessed it: Make, and Model.

The Make table lists all of the Makes. Each make has a unique ID and a Name (Make):

Make

ID	Make
1	Acura
2	Alfa Romeo
3	AMC
4	Audi
5	Bertone
6	BMW
7	Buick
8	Cadillac
9	Chevrolet

The Make Table

The Model table lists all of the models. Each model has a unique ID, a Name (Model), and also a foreign key that links it to it's corresponding Make (MakeID):

Model

ID	Model	MakeID
1	CL	1
2	Integra	1
3	Legend	1
4	MDX	1
5	NSX	1
6	RL	1
7	RSX	1
8	SLX	1
9	TL	1
10	Vigor	1
11	164	2
12	GTV-6	2
13	Milano	2
14	Spider	2
15	Alliance	3
16	Eagle	3

The Model Table

Follow these steps to create the application:

1) Create a new ASP.NET project called "Cascade".

2) Add the file "ComboBox.mdb" to the project. ComboBox.mdb can be found in the \Rad ComboBox \Projects\Data directory. If the "Data Source Configuration Wizard" appears, cancel the dialog:

The CombBox.mdb added to the project

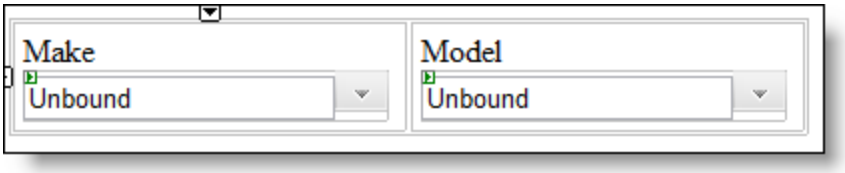
3) Add a table with one row and two columns to the form. This will be used to layout the combo boxes side by side:

```
<table  border=0  cellpadding=5  width=400px>
  <tr>
     <td  width=200px  align=left></td>
     <td  width=200px  align=left></td>
  </tr>
</table>
```

4) In the first cell, add a heading and a
 tag, repeat for the second cell:

```
<table  border=0  cellpadding=5  width=400px>
  <tr>
     <td width=200px  align=left>
       Make<br>
     </td>
     <td width=200px  align=left>
       Model<br>
     </td>
  </tr>
</table>
```

5) Switch to design view and drag one RadComboBox into each cell:



Layout of combos

7) Change the IDs of the combo boxes to rcbMake and rcbModel.

8) Before we can start hooking things up, we need to create a couple of methods to load data. We need to import the Telerik Webcontrols and OleDb namespaces first:

```
C#  Example:
using  System.Data.OleDb;
using  Telerik.WebControls;

VB  Example:
Imports  System.Data.OleDb
Imports  Telerik.WebControls
```

9) Next, create a method to load the list of Makes:

```
C#  Example:
private void  LoadMakes()
{
    string  path  =  Server.MapPath("~/Combobox.mdb");
     OleDbConnection  dbCon  =  new  OleDbConnection(
       "Provider=Microsoft.Jet.OLEDB.4.0;Data    Source="  +  path);
     dbCon.Open();
     OleDbDataAdapter  adapter  =  new  OleDbDataAdapter(
       "SELECT  *  FROM  Make",  dbCon);
    DataTable  dt  =  new  DataTable();
     adapter.Fill(dt);
     dbCon.Close();
     rcbMake.DataTextField  =  "Make";
     rcbMake.DataValueField  =  "ID";
     rcbMake.DataSource  =  dt;
     rcbMake.DataBind();
```

```
     rcbMake.Items.Insert(0, new  RadComboBoxItem("- Select a make -"));
    foreach (RadComboBoxItem item in rcbMake.Items)
         item.ToolTip = item.Text;
}

VB Example:
Private Sub LoadMakes()
  Dim path As String  =  Server.MapPath("~/Combobox.mdb")
  Dim dbCon As OleDbConnection = New OleDbConnection( _
      "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + path)
   dbCon.Open
  Dim adapter As OleDbDataAdapter = New OleDbDataAdapter( _
     "SELECT * FROM Make", dbCon)
  Dim dt As DataTable = New DataTable
   adapter.Fill(dt)
   dbCon.Close
   rcbMake.DataTextField = "Make"
   rcbMake.DataValueField = "ID"
   rcbMake.DataSource = dt
   rcbMake.DataBind
    rcbMake.Items.Insert(0, New RadComboBoxItem("- Select a make -"))
  For Each item As RadComboBoxItem In rcbMake.Items
    item.ToolTip = item.Text
  Next
End Sub
```

Note a couple of details here: we databind to the access table, and then manually insert a "- Select a make -" item at the top of the list. Also, we give each item a tooltip.

10) Now add a method that will load all of the models for a given MakeID:

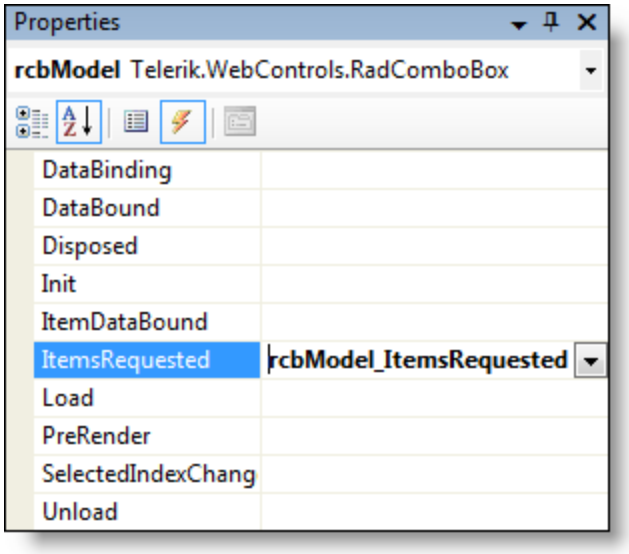
```
C# Example:
private void LoadModels( string makeID)
{
    string path = Server.MapPath("~/Combobox.mdb");
     OleDbConnection dbCon = new  OleDbConnection(
        "Provider=Microsoft.Jet.OLEDB.4.0;Data    Source=" + path);
     dbCon.Open();

     OleDbDataAdapter adapter = new  OleDbDataAdapter(
        "SELECT * FROM Model WHERE MakeID=" + makeID, dbCon);
    DataTable dt = new  DataTable();
     adapter.Fill(dt);
     dbCon.Close();

     rcbModel.DataTextField = "Model";
     rcbModel.DataValueField = "ID";
     rcbModel.DataSource = dt;
     rcbModel.DataBind();

     rcbModel.Items.Insert(0, new  RadComboBoxItem("- Select a model -"));

    foreach (RadComboBoxItem item in rcbModel.Items)
         item.ToolTip = item.Text;
}

VB Example:
Private Sub LoadModels(ByVal makeID As String)
   Dim path As String  =  Server.MapPath("~/Combobox.mdb")
```

```
    Dim dbCon As OleDbConnection = New OleDbConnection( _
         "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" + path)
     dbCon.Open
    Dim adapter As OleDbDataAdapter = New OleDbDataAdapter( _
        "SELECT * FROM Model WHERE MakeID=" + makeID, dbCon)
    Dim dt As DataTable = New DataTable
     adapter.Fill(dt)
     dbCon.Close
     rcbModel.DataTextField = "Model"
     rcbModel.DataValueField = "ID"
     rcbModel.DataSource = dt
     rcbModel.DataBind
      rcbModel.Items.Insert(0, New RadComboBoxItem("- Select a model -"))
    For Each item As RadComboBoxItem In rcbModel.Items
        item.ToolTip = item.Text
    Next
End Sub
```

11) We want the Makes to be loaded on the initial Page Load, so add this to the Page_Load event handler:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
     LoadMakes();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  If Not IsPostBack Then
     LoadMakes
  End If
End Sub
```

12) We want the models to be loaded when the model combo dynamically requests its items, so we need to add an event handler for ItemsRequested. Switch to Design View, select rcbModel, and double click the ItemsRequested event handler in the property inspector:

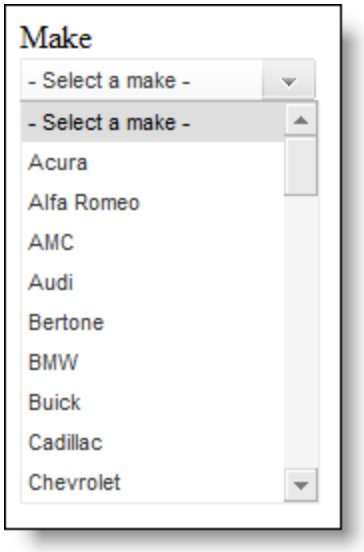


Adding an event handler

13) In the event handler, add this code to load the models for the chosen make. Note: we are assuming that e.Text contains a MakeID, you will soon see how that is the case:

```
C# Example:
protected void  rcbModel_ItemsRequested(object o,
    RadComboBoxItemsRequestedEventArgs  e)
{
   LoadModels(e.Text);
}

VB Example:
Protected Sub  rcbModel_ItemsRequested(ByVal o As Object, _
  ByVal e As   RadComboBoxItemsRequestedEventArgs)
   LoadModels(e.Text)
End Sub
```

14) Run the application to verify that the Make combo is indeed being populated:



The makes are populated

Now we are done with the server side logic. What remains is to write some client side javascript that will chain the events together. First of all, we need to react to the client side event that fires when an item is selected in the Make combobox. This is the OnClientSelectedIndexChanging client event.

15) Switch to source view and add an attribute to rcbMake to hook this event and direct it to a javascript method called LoadModels:

```
<radc:radcombobox  id="rcbMake"  runat="server"  height="200px"
    OnClientSelectedIndexChanging="LoadModels">
</radc:radcombobox>
```

16) Further down in the aspx page, after the closing table tag, implement the LoadModels function:

```
<script   type="text/javascript">
     var  modelCombo  =  <%=  rcbModel.ClientID  %>;

     function  LoadModels(item)
    {
          modelCombo.SetText("Loading...");
          modelCombo.ClearSelection();

         if (item.Index > 0)
        {
            modelCombo.RequestItems(item.Value, false);
        }
    }
</script>
```

Some things to note about this piece of code:

- The first line is the standard way to get a client side handle to a RadComboBox
- The LoadModels function receives a **item** parameter, which represents the selected item in the combo

that fired the event. So this is the selected Make. Now remember, when we bound the Make combo to it's DataTable, we set the DataTextField to the Make column, and the DataValueField to the ID column, so in order to get the ID, we need to use item.Value (not item.Text)

- The client side API for the combobox is used to clear the current selection in the model combo, and set the text of the combo to "Loading....".
- What is the if (item.Index > o) check for? Well, looking at the loading of the Make combo again, remember that we added an extra item to the beginning of the combo, with the text "- Select a make -" ? That is at index 0, and we don't want to try to fetch the models if that item is chosen.
- The call to modelCombo.RequestItems will cause an Ajax callback to be made, and the server side ItemsRequested handler will be invoked, returning the items for the model combo.

We now need one final step to the client side logic. When the Ajax call above returns, we want to drop down the list of models. To do that, we need to hook the OnClientItemsRequested event on the model combo.

17) Add this attribute to the Model combo:

```
<radc:radcombobox  id="rcbModel"  runat="server"  Height=200px
      onitemsrequested="rcbModel_ItemsRequested"
       OnClientItemsRequested="ItemsLoaded"></radc:radcombobox>
```

Don't confuse the two attributes OnItemsRequested and On**Client**ItemsRequested! OnItemsRequested is the server side handler. On ClientItemsRequested is the client side handler that fires after the ajax callback to the server side handler has returned.

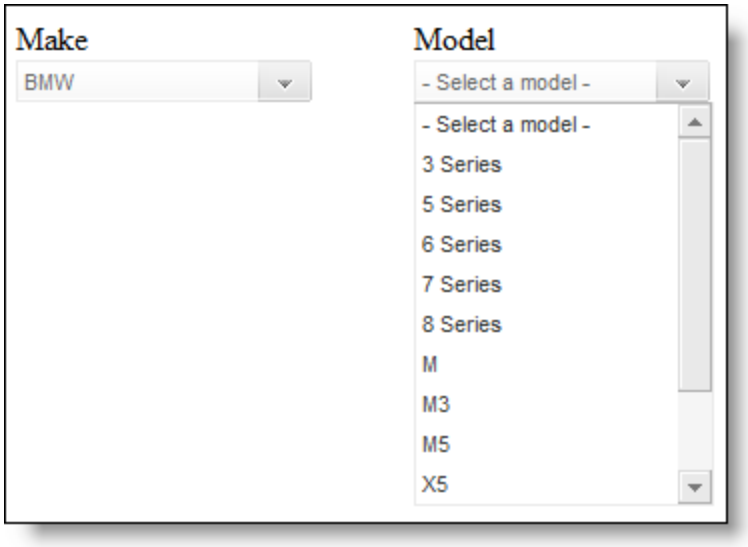
18) Define the ItemsLoaded function in your javascript block:

```
function  ItemsLoaded(combo)
{
     if (combo.Items.length > 0)
    {
          combo.SetText(combo.Items[0].Text);
    }
     combo.ShowDropDown();
}
```

Some details worth pointing out in the code above:

- The function is passed a **combo** parameter. This is a handle to the client side instance of the combo box that has just been loaded
- The check of combo.Items.length is just to make sure that at least one item was returned, so we don't access a non existing item on the following line of code
- The combo.SetText is used to set the text in the textbox to be the first item in the list (Items[0].Text)
- combo.ShowDropDown() drops the list down.

19) Run the application and select a make. You should see the matching models being populated!



Final output of lab

This pattern can of course be extended to several levels, for instance there could be a Year combo between Make and Model. Or you could have three cascading dropdowns with Continent, Country and City. You could also have the first combo be a load on demand, etc...

1.4.6 Summary

You have now learned a few of the many modes the RadComboBox can operate in. You have seen how to build multi column drop downs, using templates, and how to use the load on demand feature with regular pages or external callback pages. You have also seen how to use the intrinsic Ajax support to build cascading comboboxes, and some of the client side javascript capabilities of the component.

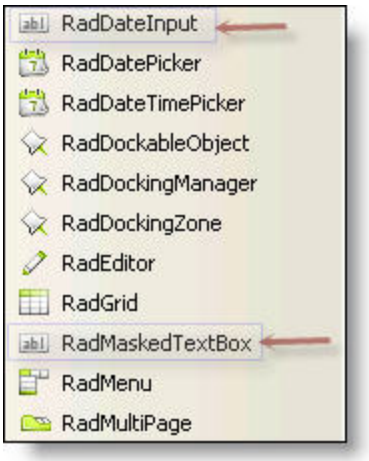
1.5 RadInput

1.5.1 Getting Started

To avoid excess heavy-lifting validation code on the backend you need thorough format and reasonableness checking on the front end. Telerik RadInput controls help ensure that data entry conforms to basic format rules so that social security numbers never contain letters, month dates never contain 39 days and that Discover credit card numbers can never be entered where a Visa is expected. RadInput controls also help the user by speeding up the process of data entry and providing visual cues.

The Telerik RadInput consists of the RadMaskedTextBox and the RadDateInput. Some possible uses for this control set are:

- Display format and take input for common data entry tasks like phone, social security, dates, and zip code.
- Provide an enumerated set of choices that can be selected from a text box hint.
- Display format and take input for custom data entry like promotion codes, order numbers, and product validation codes.



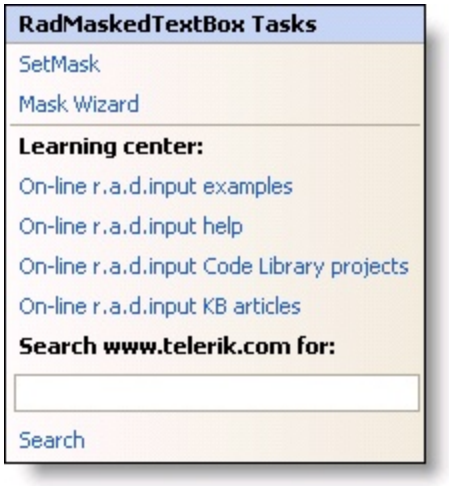
RadInput controls in the toolbox

1.5.2 Using RadInput in the Designer

Using RadMaskedTextBox at design time

Input Mask

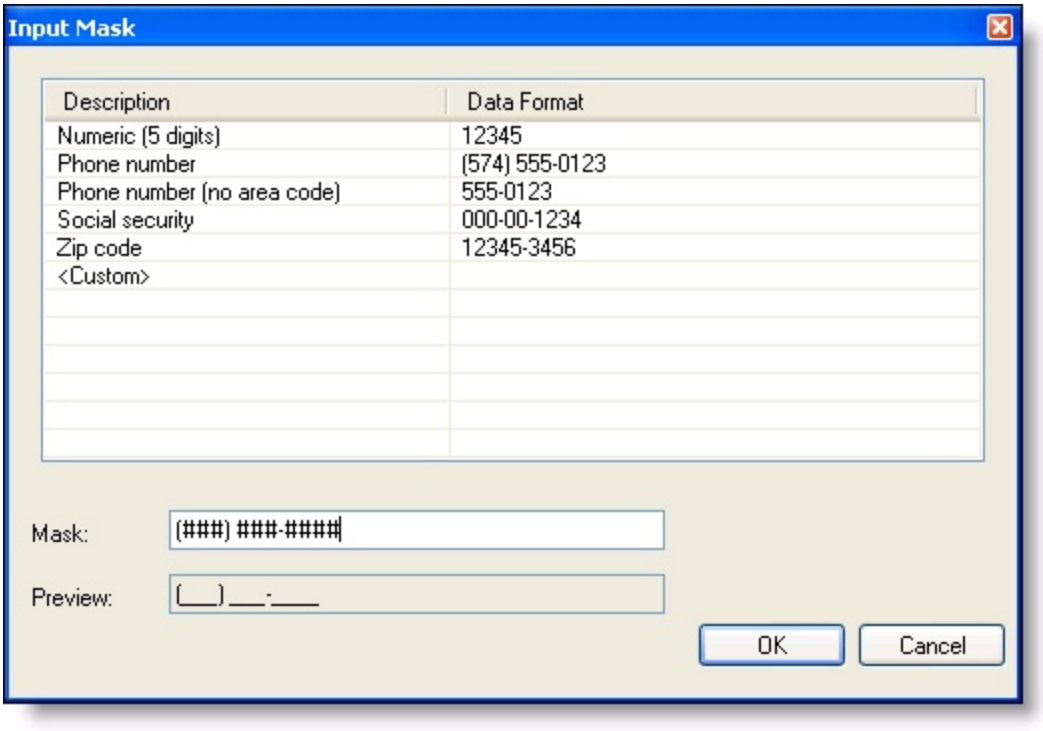
The RadMaskedTextBox Smart Tag has two options for working with the input mask: Set Mask and Mask Wizard.



SetMask and Mask Wizard options

Set Mask

The Set Mask, Input Mask dialog that allows you to choose from well known format patterns like phone number and zip code, or to construct your own.



Using the provided phone number mask in the Input Mask dialog

The table below describes the possible mask characters. A string of these characters are assigned to the Mask property of the control. The mask differentiates between literal characters and the characters the user will enter. The cursor will skip over the literal characters to the actual entry.

Note: Later we will programmatically build the mask using the MaskCollection of the control by adding instances of MaskPart classes.

Mask Character	**MaskPart Class**	**Description**
#	DigitMaskPart	Digit or space, optional. If this position is blank in the mask, it will be rendered as a PromptChar.
L	UpperMaskPart	Uppercase letter, required. Restricts input to the ASCII letters A-Z.
l	LowerMaskPar	Lowercase letter, required. Restricts input to the ASCII letters a-z.
a	FreeMaskPart	Accepts any character.
<n..m>	NumericRangeMaskPart	Restricts the user input to the declared numeric range e.g. <0..255>. The numeric range mask part may occupy multiple characters of the mask.
<option1|option2|option3>	EnumerationPart	Restricts the user input to one of the set options e.g. <Mon|Tue|Wed|Thu|Fri|Sat|Sun>. The pipe("|") serves as a separator between the option values.
\	N/A	Escapes a mask character, turning it into a literal. "\\" is the escape sequence for a back slash.
All other characters	LiteralPart	All non-mask elements will appear as themselves within **MaskedTextBox**. Literals always occupy a static position in the mask at run time, and cannot be moved or deleted by the user

Table: Mask Characters

To get a better idea how to build masks in common scenarios, here are more examples:

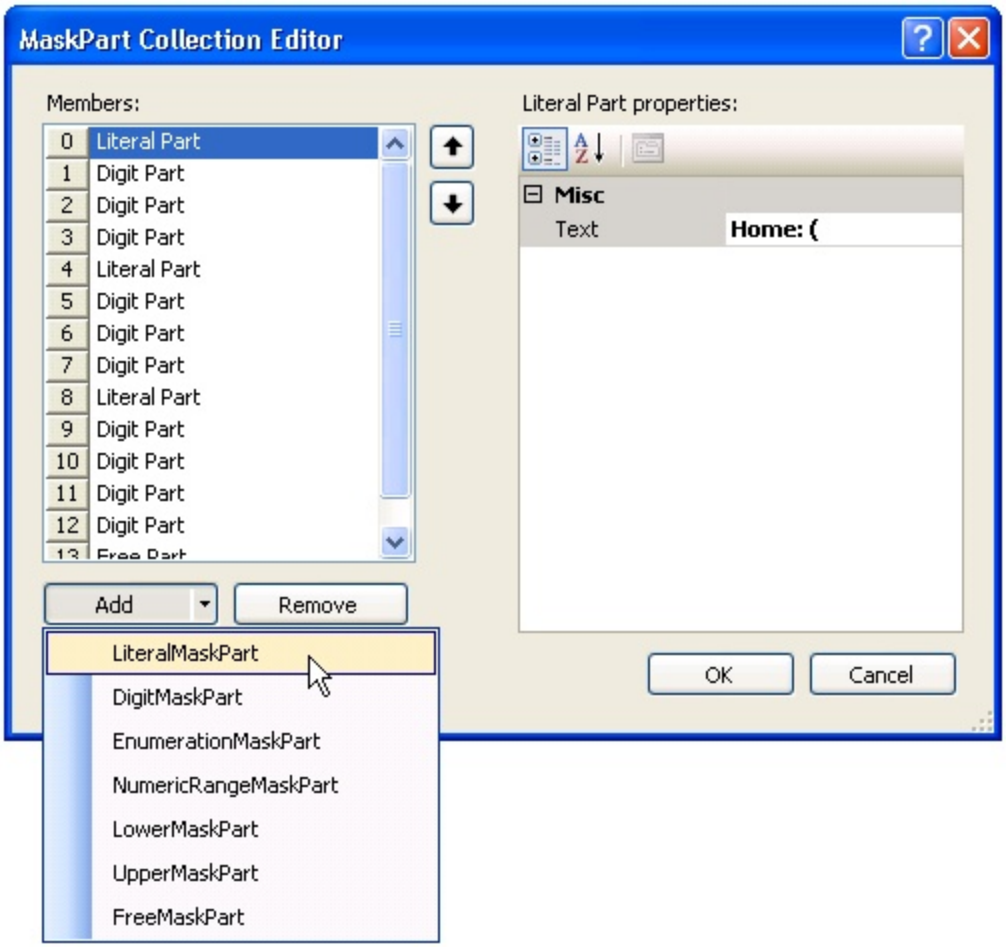
Example	**Mask**	**Comments**
First name	`"Lllllllll"`	Automatically capitalizes the first character.
Delivery Type	`"<Standard|2 day|1 day>"`	Allows one of three elements in an enumeration. If you set AllowHint to true, the enumeration can be selected from a hint window using the mouse. The user can also use arrow keys to navigate between enumerated items.
Notes	`"Notes for 1/1/2007aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa"`	Contains the literal string "Notes for 1/1/2007" and a number of spaces for free entry.
Money amount	`$######.##`	This has a literal "$", accepts a numeric amount with six places and two places to the right of the decimal: $123456.56.

Example	Mask	Comments
A lot number that starts with an escaped literal	`\Lot number:#####`	The backslash preceding the "L" prevents the prompt from showing as "ot number:"
A numeric range	`<1..10>`	This displays the prompt "01" and accepts the values 1 through 10.

Table: Example masks

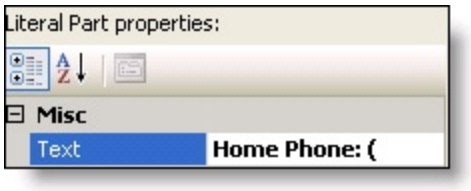
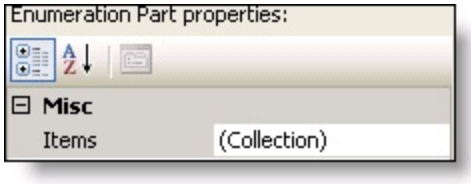
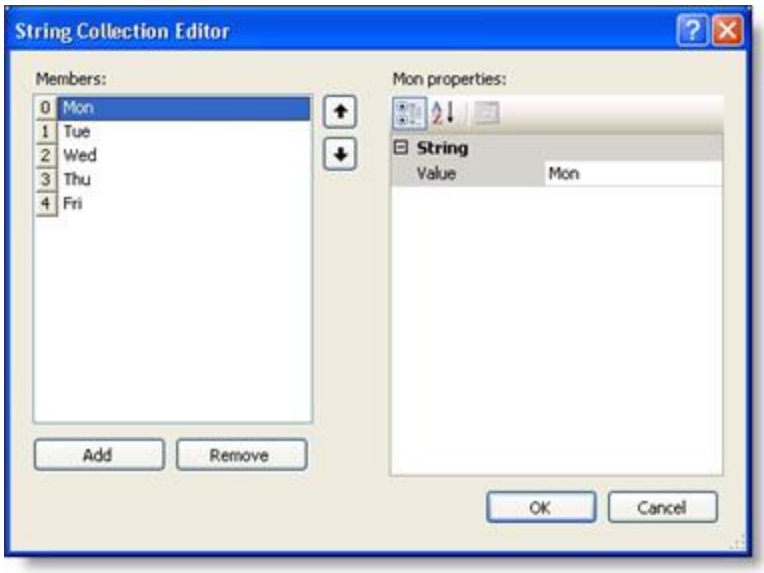
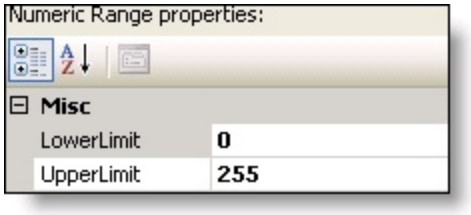
Mask Wizard

The Mask Wizard is a collection editor that provides a way to build your mask one piece at a time using a set of "MaskPart" classes. This dialog can also be helpful in modeling how you will programmatically build a mask.



Adding a literal mask part in the Mask Wizard

The properties on the right hand side of the wizard will change depending on the mask part class selected:

Mask Part Class	Property Editor
LiteralMaskPart	
EnumerationMaskPart	 
NumericRangeMaskPart	
LowerMaskPart	No displayed properties
UpperMaskPart	No displayed properties
FreeMaskPart	No displayed properties

Lab: Build a validation code entry

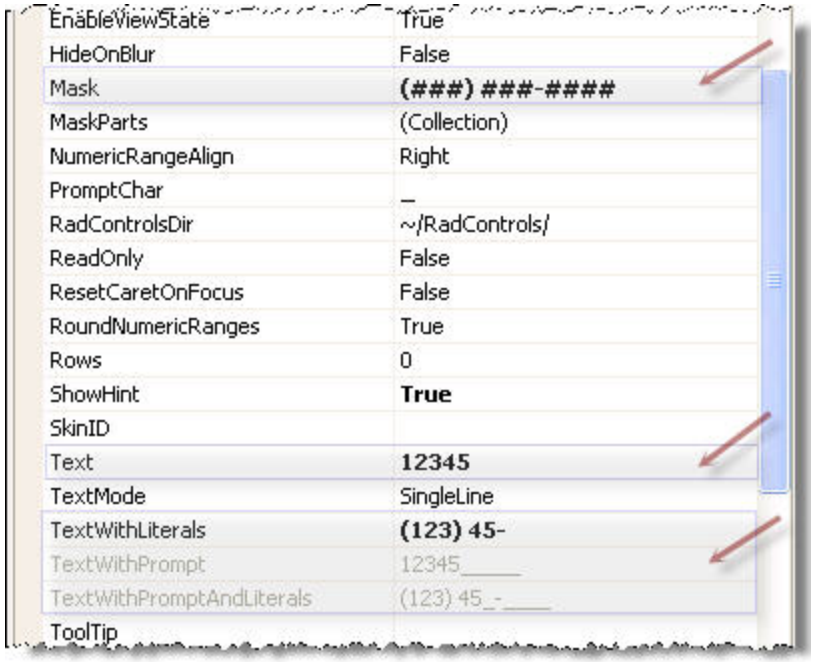
In this lab you will learn how to build a mask using the Set Mask Smart Tag. We will build an entry for a validation code similar to one you might enter for a Microsoft Office installation. This project will validate for format only.

1. Create a new web project "GettingStarted". We will reuse this project later as a testbed for dropping RadInput controls and testing their functionality.
2. Drag a RadMaskedTextBox onto the page.
3. Select the "SetMask" Smart Tag.
4. Enter the following characters to the Mask entry: LLLLL-LLLLL-LLLLL-LLLLL
5. Run the project and test the input. It should accept upper case alpha characters only. Numeric characters will not be accepted. Also notice that the caret jumps over the literal "-" character.

Lab: Output properties

Of course you want the text the user puts in but you may also need the literal parts as well. RadInput controls have four significant properties that represent the output of the control. *Text*, TextWithLiterals, *TextWithPrompt* and *TextWithPromptAndLiterals*. The best way to see how these properties behave is to try an example:

1. Reopen your "GettingStarted" application.
2. In the RadMaskedTextBox control, select the "SetMask" item from the Smart Tag.
3. Select the "Phone Number" mask and click "OK" to close the window.
4. In the Properties window enter "12345" to the *Text* property.
5. Now view the *Mask*, *TextWithLiterals*, *TextWithPrompt*, and *TextWithPromptAndLiterals* properties. Note: At the time of this writing these properties only refresh when clicked on.



Viewing the RadMaskTextBox control output

Using the RadDateInput at design time

The *RadDateInput* control restricts date and time input to valid values and provides keyboard navigation assistance. RadDateInput has some properties in common with RadMaskedTextBox because they both descend from a common ancestor, RadMaskedInputControl. Minimally you use the *DateFormat* property to control the layout (e.g. short date or long date and time) and the *SelectedDate* property to set or retrieve the input date. Some of the unique RadDateInput properties that are significant:

Property	Description
Label and LabelCssClass	A string property for a text label automatically placed to the left of the date input area. The *LabelCssClass* property lets you tailor the look of the label.
DateFormat	A one character code that describes the date format, i.e. long date, short date and time, etc.
SelectedDate	The date that displays the input date in the control itself. You can get or set this at design time or runtime.
MinDate	The input date can not be less than this value. If the user attempts to put in a date less than MinDate, the date will automatically correct itself up to this value.
MaxDate	The input date can not be more than this value. If the user attempts to put in a date greater than the MaxDate, the date is automatically corrected down to this value.

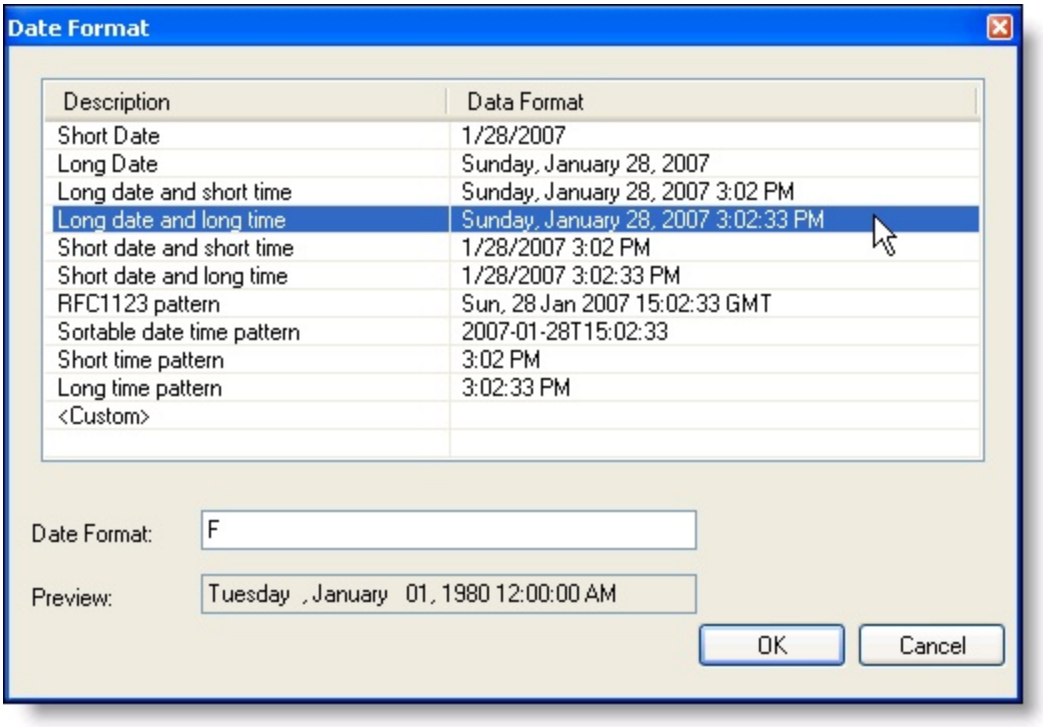
Property	Description
DisplayDateFormat	Provides a different format when the user is not focused on the control. For instance, you can display the long date format when the focus is not on the control, but allow the user to use the short date format for actual entry. You could also use this property as a prompt, i.e. "Enter Expiration Date Here".

To begin working with the RadDateInput, use the Smart Tag option "Set Date Format".

Set Date Format Smart Tag option

Set Date Format

Set the date format by selecting one of the descriptions in the Date Format dialog, or enter the Date Format code in the entry at the bottom of the dialog.



Setting the date format

The standard Date Format codes are listed in the table below. Note that the format characters are case sensitive.

Format Character	Associated Property/ 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

Standard Date Format codes

The standard date patterns will work for many common uses, but where these are too restrictive, you can create your own custom formats using the format patterns listed below. For example, a DateFormat of "ddd, MMMM yyyy - dd" displays "Tue, January 1980 - 01" in the prompt.

Format Pattern	Description
dd	The day of the month. Single-digit days will have a leading zero.
ddd	The abbreviated name of the day of the week, as defined in AbbreviatedDayNames.
M	The full name of the day of the week, as defined in DayNames.
MM	The numeric month. Single-digit months will have a leading zero.
MMM	The abbreviated name of the month, as defined in AbbreviatedMonthNames.
MMMM	The full name of the month, as defined in MonthNames.
y	The year without the century. If the year without the century is less than 10, the year is displayed with no leading zero.
yy	The year without the century. If the year without the century is less than 10, the year is displayed with a leading zero.
yyy	The year in four digits, including the century.
gg	The period or era. This pattern is ignored if the date to be formatted does not have an associated period or era string.
h	The hour in a 12-hour clock. Single-digit hours will not have a leading zero.
hh	The hour in a 12-hour clock. Single-digit hours will have a leading zero.
H	The hour in a 24-hour clock. Single-digit hours will not have a leading zero.
HH	The hour in a 24-hour clock. Single-digit hours will have a leading zero.
m	The minute. Single-digit minutes will not have a leading zero.
mm	The minute. Single-digit minutes will have a leading zero.
s	The second. Single-digit seconds will not have a leading zero.
ss	The second. Single-digit seconds will have a leading zero.
t	The first character in the AM/PM designator defined in AMDesignator or PMDesignator, if any.
tt	The AM/PM designator defined in AMDesignator or PMDesignator, if any.

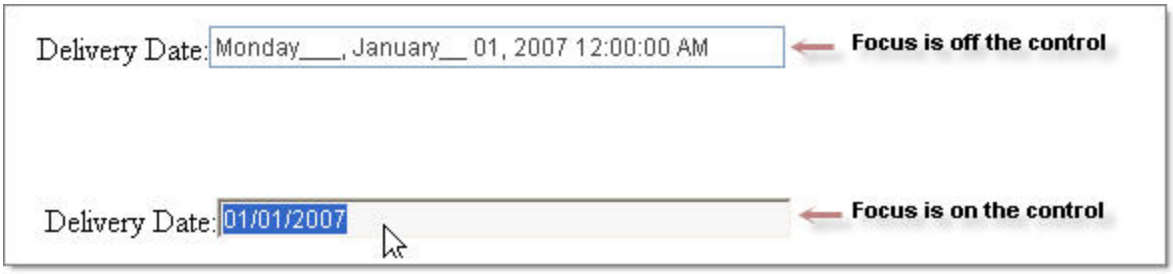
Lab: Explore RadDateInput properties

In this lab you will learn to use significant RadDateInput properties *DateFormat, SelectedDate*, *MinDate*, *MaxDate, and DisplayDateFormat* .

1. Create a new project "InputBasics"
2. Drop a RadDateInput control on the default page. Leave the *ID* property at the default value.
3. Set the following properties for the RadDateInput:
   - *DateFormat*: "d"
   - *DisplayDateFormat*: "F"
   - *Label*: "Delivery Date:"

- *MinDate*: Set the date for the first day of the current month
- *MaxDate*: Set the date for the last day of the current month

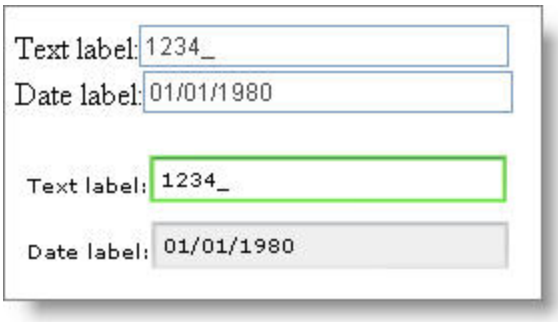
4. Run the application.
5. Notice the behavior of the RadDateInput with these property settings:
   - When the focus is off the control the DisplayDateFormat is in force. When the user selects the RadDateInput to begin entry the DateFormat is used.
   - The label "Delivery Date" appears to the left of the date input entry area. Note: the style for both the input area and the label can be controlled by the *CssClass* and LabelCssClass properties respectively.
   - You can use the up and down arrow keys to navigate dates. If the cursor is in the area showing the year "2007", then the up arrow will navigate to "2008" and the down arrow to "2006".
   - The RadDateInput dates can only be navigated to within the bounds you set in *MinDate* and *MaxDate* properties. Attempts to roll the month or year past the bounds are ignored except that the day may be changed to the upper or lower limits. The behavior is the same using direct keyboard entry or the arrow keys.



Control behavior with and without focus

Skins

It's worth looking specifically at the skinning for RadInput because the label for the input and the input itself are controlled by two different css classes. Currently the css available in the provided skins isn't really suitable for the label part so you need to be prepared to edit the css. In the figure below we have RadDateInput and RadMaskedTextBox with default appearance and using the "Macromedia" skin. The style sheet for the skinned example has been modified to provide a style suitable for the label.

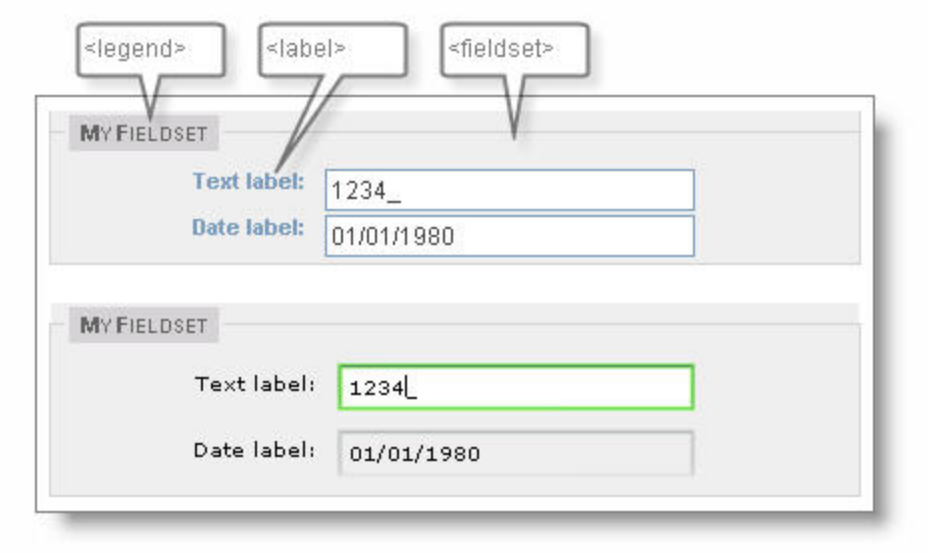


RadDateInput, RadMaskedEdit without skin and with Macromedia style

Taking the above example a step further, the figure below shows HTML fieldset tags surrounding the input controls. You can wrap each set of controls with fieldset tags to give the page a more Windows panel like appearance. This HTML tag also helps differentiate areas of the page involved with different types of entry, e.g. an order header input area and order detail input area. The html looks something like:

```
<fieldset>
   <legend>My Fieldset</legend>
     <radI:RadMaskedTextBox ... Label="Text Label:">
```

We can add styles for field set, legend and field label elements to achieve a more cohesive order form look.



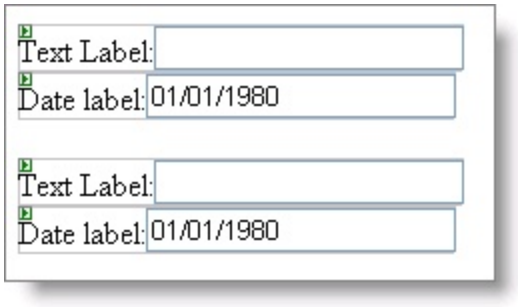
Styles applied to the field set

Notice in the top inputs and labels have a blue color that comes from the new styles for the fieldset. The bottom example demonstrates how the styles for the skin overrides the style for the text label. The text label is not the blue color of the fieldset label, but the style we defined in the stylesheet for the skin.

Lab: Setting RadInput styles

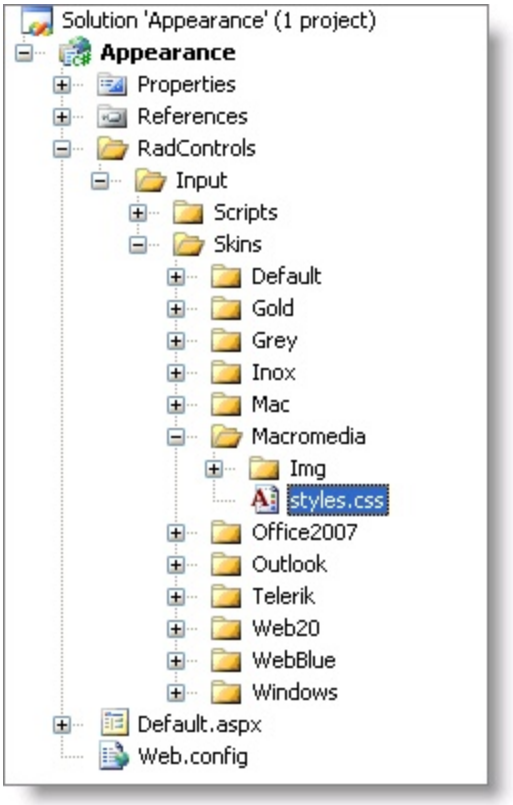
In this lab you will learn how to apply skins to the RadInput controls, manipulate the style sheet for custom effects, and handle style for the page as a whole.

1. Create a new project "Appearance".
2. Drop a RadMaskedTextBox and a RadDateInput on the page. Arrange them with the text box above the date input. Set the Label property of the RadMaskedTextBox to "Text label" and the RadDateInput to "Date label".
3. Copy both controls and paste them below the first set. Put a carriage return between the two sets of controls. The project should now look like the figure below:



The layout of controls on the page in the designer

4. In the Solution Explorer, create a new folder for the project "RadControls". Copy the "Input" folder from the Telerik installation RadControls directory; the directory is for most installs located at "\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls". Your project folders should look similar to the figure below.



The project with RadControls folders added

5. Double click "styles.css" found under the Macromedia skin to edit the style sheet. Add the following style to the end of the file. Note: This style was obtained by copying the .Default_Macromedia style and removing references to background images and hard coded widths.

```
.Label_Macromedia
{
	border: 0px;
	height: 24px;
	color: #000000;
	font-family: Verdana, Arial, Tahoma, Sans-Serif;
	font-size: 10px;
```

```
        font-weight:  normal;
        padding-top:  6px;
        padding-left:  6px;
        background-color:   transparent;
        vertical-align:   middle;
}
```

6. Now that the skins and styles are properly set up we can apply them to the RadInput controls. The first set of RadMaskedTextBox and RadDateInput controls we leave alone to demonstrate how they look with their default appearance.

7. For both the RadDateInput and RadMaskedTextBox in the second set of controls set the *Skin* property to "Macromedia" and *LabelCssClass* property to "Label_Macromedia". Note: setting the *Skin* property automatically sets the CssClass property to "Default_Macromedia". The project design view now looks like the figure below:

The second set of inputs with Skin property set

8. Before we add the fieldset tags to the page we need to prepare a few more styles. Add the following to styles.css in your project RadControls\Input\Skins\Macromedia folder:

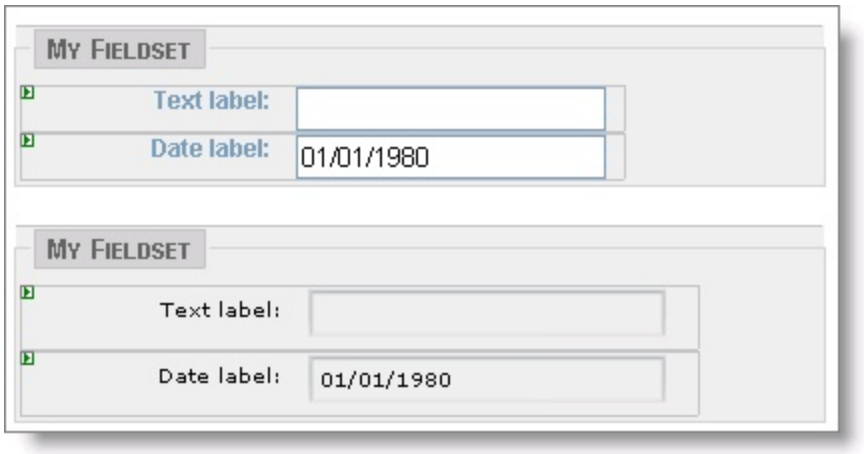
```
fieldset  {
     font:  0.7em  "Helvetica  Neue",  helvetica,  arial,  sans-serif;
    color: #666;
     background-color: #eeeeee;
    padding: 2px;
    border: solid 1px #d3d3d3;
    width: 400px;
    }
legend  {
    color: #666;
     font-weight: bold;
     font-variant: small-caps;
     background-color: #d3d3d3;
    padding: 2px 6px;
     margin-bottom: 8px;
    }
label   {
     font: normal 12px;
     color: #7f9db9;
     font-weight: bold;
     line-height: normal;
     text-align: right;
     margin-right: 10px;
     position: relative;
     display: block;
    float: left;
     width: 125px;
    }
```

9. The label style applies to any label elements within a fieldset tag, including the RadInput controls labels. The style for the fieldset label handles lining up the labels in a neat column.

10. Now we want to wrap each set of controls with fieldset tags. You can edit this directly in the HTML code. Edit the aspx page to add fieldset and legend tags as shown bold in the example below.

```
<fieldset  class="legend">
     <legend class="legend">My Fieldset</legend>
      <radI:RadMaskedTextBox ID="RadMaskedTextBox1" runat="server" Label="Text label:">
     </radI:RadMaskedTextBox><br />
      <radI:RadDateInput ID="RadDateInput2" runat="server" Label="Date label:">
     </radI:RadDateInput>
</fieldset>
<br />
<br />
<fieldset  class="legend">
     <legend class="legend">My Fieldset</legend>
      <radI:RadMaskedTextBox ID="RadMaskedTextBox2" runat="server" Label="Text label:"
           LabelCssClass="Label_Macromedia" Skin="Macromedia"></radI:RadMaskedTextBox><br />
     <radI:RadDateInput ID="RadDateInput1" runat="server" Label="Date label:"
        LabelCssClass="Label_Macromedia"
         Skin="Macromedia"></radI:RadDateInput>
</fieldset>
```

11. The page in the designer should now look like the figure below:



12. Run the project to test the behavior of the RadInput appearance as each control is selected.

1.5.3 Using RadInput at runtime

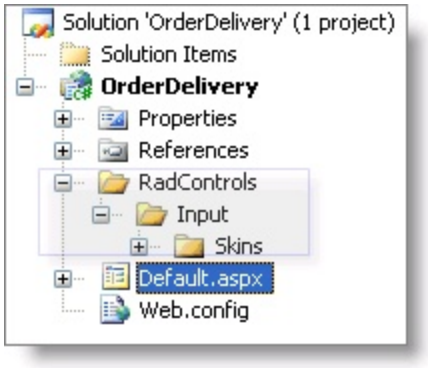
Lab: Order Delivery Entry with RadInput controls

In this lab we will build an order delivery entry form to demonstrate using the RadInput controls at runtime:

- Setting and retrieving values from text and date input controls.
- Building literal, enumeration, and free text masks programmatically.
- Using the RadMaskedTextBox in multiline mode.
- Using validation with RadInput controls.

Setting up the project

1. Create an new application "OrderDelivery".
2. Create a new folder in the project and name it "RadControls".
3. In the Solution Explorer, create a new folder for the project "RadControls". Copy the "Input" folder from the Telerik installation RadControls directory; the directory is for most installs located at "\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls". Your project folders should look similar to the figure below. Note: This project only requires the Office_2007 skin directory, so you need not copy over the other skins and javascript.



The project with RadControls added

Setting up the page design

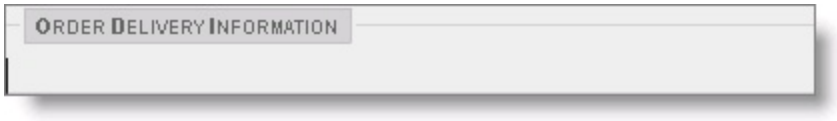
1. In the source for the aspx for the default page add the line that registers the RadInput assembly.

```
<%@ Register Assembly="RadInput.Net2" Namespace="Telerik.WebControls" TagPrefix="radI" %>
```

2. Enter the following HTML after the form tag.

```
<fieldset>
   <legend>Order Delivery Information</legend>

</fieldset>
```

Now the page will should like the following figure:



Page with just the fieldset

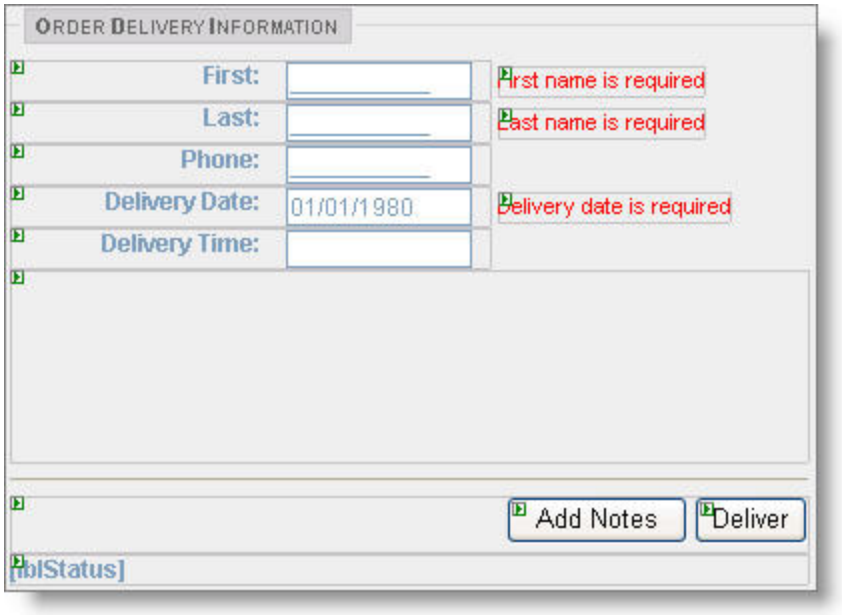
3. Populate the page with controls. Each control should be followed with a carriage return so that each control is on its own line. Because of the number of controls we're going to take a short cut here and paste the controls in as HTML in the aspx file. Copy the code below, leaving out the fieldset and legend tags (shown in bold) and paste it after the legend tag.

```
<fieldset>
     <legend>Order Delivery Information</legend>
      <radI:RadMaskedTextBox ID="rmtbFirst" runat="server" CatalogIconImageUrl=""
          Description="" DisplayPromptChar="_" Label="First:" Mask="Lllllllll"
        PromptChar="_" Skin="Office2007" LabelCssClass="Label_Office2007">
     </radI:RadMaskedTextBox>
      <asp:RequiredFieldValidator ID="valFirst" runat="server"
         ErrorMessage="First name is required"
           ControlToValidate="rmtbFirst"></asp:RequiredFieldValidator>
```

```
    <br />
      <radI:RadMaskedTextBox ID="rmtbLast" runat="server" DisplayPromptChar="_" Label="Last:"
           Mask="Lllllllll" PromptChar="_" Skin="Office2007" LabelCssClass="Label_Office2007">
     </radI:RadMaskedTextBox>
      <asp:RequiredFieldValidator ID="valLast" runat="server" ControlToValidate="rmtbLast"
          ErrorMessage="Last name is required"></asp:RequiredFieldValidator>
    <br />
      <radI:RadMaskedTextBox ID="rmtbPhone" runat="server" CatalogIconImageUrl=""
           Description="" DisplayPromptChar="_" Label="Phone:" Mask="##########"
          PromptChar="_" Skin="Office2007" DisplayMask="(###) ###-####"
         LabelCssClass="Label_Office2007">
     </radI:RadMaskedTextBox>
    <br />
      <radI:RadDateInput ID="rdiDate" runat="server" Description="" Label="Delivery Date:"
           Skin="Office2007" AllowEmpty="True" EmptyMessage="Enter date" HideOnBlur="True"
            SelectedDate="1980-01-01" LabelCssClass="Label_Office2007"></radI:RadDateInput>
      <asp:RequiredFieldValidator ID="valDeliveryDate" runat="server"
          ControlToValidate="rdiDate" ErrorMessage="Delivery date is required">
      </asp:RequiredFieldValidator>
    <br />
      <radI:RadMaskedTextBox ID="rmtbTime" runat="server" DisplayPromptChar="_"
         Label="Delivery Time:"
           PromptChar="_" ShowHint="True" Skin="Office2007" LabelCssClass="Label_Office2007">
     </radI:RadMaskedTextBox>
    <br />
      <asp:Panel ID="pnlNotes" runat="server" Width="100%" HorizontalAlign="Center"
       ScrollBars="Auto">
    </asp:Panel>
    <hr />
      <asp:Panel ID="pnlButtons" runat="server" HorizontalAlign="Right" Width="100%">
           <asp:Button ID="btnAddNotes" runat="server" Text="Add Notes" />
          <asp:Button ID="btnDeliver" runat="server" Text="Deliver" /></asp:Panel>
      <asp:Label ID="lblStatus" runat="server" CssClass="Label_Office2007" Width="100%">
    </asp:Label>
</fieldset>
```

The preceding code should set basic properties and styles of the controls. Review the control properties paying particular attention to the settings for Skin, Mask and DisplayMask. Also pay particular attention to the Delivery Time control and notice that the *ShowHint* property is true; the mask for this control will later contain an enumeration, the choices of which will be displayed in the hint. Three RequiredFieldValidators have been added to the page that ensure the First Name, Last Name and Delivery Date will have something entered.

The page should now look like the figure below:



Page populated with controls and styles set

The Code Behind

1. Add using statements for System.Collections.Generic and Telerik.WebControls. We will use the generic List<> type later to support persisting a series of delivery notes.

2. Each RadInput control has a MaskParts property which is actually a MaskPartCollection. The MaskParts collection can have any MaskPart descendant added to it. We need several methods to build masks for the MaskPartCollection. The most simple mask part to create is for literal characters. Here you create a LiteralMaskPart object for each literal string, set its *Text* property and add it to the MaskParts collection for the control. The AddLiteralMaskPart() method wraps that functionality. Add this method to the code behind for the default page.

```
C# Example:
private void AddLiteralMaskPart(MaskPartCollection maskParts, string text)
{
     LiteralMaskPart literalMaskPart = new LiteralMaskPart();
     literalMaskPart.Text = text;
      maskParts.Add(literalMaskPart);
}

VB Example:
Private Sub AddLiteralMaskPart(ByVal maskParts As MaskPartCollection, ByVal Text As String)
 Dim literalMaskPart As LiteralMaskPart = New LiteralMaskPart
  literalMaskPart.Text = Text
   maskParts.Add(literalMaskPart)
End Sub
```

3. This next method creates a series of mask parts for free entry. We're using FreeMaskPart, but any MaskPart descendant would be suitable here. A FreeMaskPart object is created for each character prompt. Add this method to the code behind for the default page.

```
C# Example:
private void AddFreeMaskPart(MaskPartCollection maskParts, int length)
{
```

```
    for (int i = 0; i < length; i++)
    {
         maskParts.Add(new  FreeMaskPart());
    }
}

VB  Example:
Private Sub  AddFreeMaskPart(ByVal  maskParts As  MaskPartCollection, ByVal  length As Integer)
 Dim i As Integer = 0
 While i < length
    maskParts.Add(New  FreeMaskPart)
       System.Math.Min(System.Threading.Interlocked.Increment(i),i-1)
 End While
End Sub
```

4. This last method builds an enumeration mask which is expressed in the control as a series of strings shown in the hint. Each element of the enumeration in the hint can be selected with the mouse or arrow keys. This method uses the System Enum class static GetNames() method to iterate the elements of enumeration types passed to it. You only need one EnumerationMaskPart instance. You can add strings representing the enumerated choices to the Items array of the EnumerationMaskPart. Add this method to the code behind for the default page.

```
C#  Example:
private void  AddEnumerationMaskPart(MaskPartCollection  maskParts,  Type  enumerationType)
{
     EnumerationMaskPart  enumerationMaskPart  = new  EnumerationMaskPart();
    foreach (string  element in   Enum.GetNames(enumerationType))
    {
          enumerationMaskPart.Items.Add(element);
    }
      maskParts.Add(enumerationMaskPart);
}

VB  Example:
Private Sub  AddEnumerationMaskPart(ByVal  maskParts As  MaskPartCollection,  _
  ByVal  enumerationType As Type)
 Dim  enumerationMaskPart As  EnumerationMaskPart  = New  EnumerationMaskPart
 For Each  element As String In   Enum.GetNames(enumerationType)
     enumerationMaskPart.Items.Add(element)
 Next
   maskParts.Add(enumerationMaskPart)
End Sub
```

5. At the top of the page class declare a DeliveryPeriod enumeration that will be used to populate the "Delivery Time" mask. Later when we call the AddEnumerationMaskPart() method from the Page_Load we pass the MaskParts collection for the control we want to populate and the enumeration type.

```
C#  Example:
enum  DeliveryPeriod  {  Morning,  Afternoon,  Evening  };

VB  Example:
Enum  DeliveryPeriod
 Private  Morning
 Private  Afternoon
 Private  Evening
End Enum
```

6. The next few methods involve creating multi-line RadMaskedTextBox controls dynamically at runtime. A few particulars to watch here are how to set up the carriage return/line feed so the control understands and parses everything properly and how to persist the control through page refreshes.

7. To persist the control and its contents after creation, we will use a property that holds an array of RadMaskedTextBox controls.

Note: depending on how your Session object is configured, you may need to serialize this list to another medium. That in turn may need additional work to support serializing the controls or reframing the problem so that saving the controls is not necessary.

```
C# Example:
public List<RadMaskedTextBox> Notes
{
    get
    {
        List<RadMaskedTextBox> list = Session["Notes"] as List<RadMaskedTextBox>;
        if (list == null)
        {
            Session["Notes"] = new List<RadMaskedTextBox>();
        }
        return Session["Notes"] as List<RadMaskedTextBox>;
    }
    set
    {
        Session["Notes"] = value;
    }
}

VB Example:
Public Property Notes() As List(Of RadMaskedTextBox)
    Get
        Dim myList As List(Of RadMaskedTextBox) = CType(Session("Notes"), _
           List(Of RadMaskedTextBox))
        If myList is Nothing Then
            Session("Notes") = New List(Of RadMaskedTextBox)()
        End If
        Return CType(Session("Notes"), List(Of RadMaskedTextBox))
    End Get
    Set(ByVal value As List(Of RadMaskedTextBox))
        Session("Notes") = value
    End Set
End Property
```

8. Now add a button click event handler to "btnAddNotes". Add the following code to the event handler.

```
C# Example:
protected void btnAddNotes_Click(object sender, EventArgs e)
{
    const string NEWLINE = "\r\n";

    RadMaskedTextBox rmtbNotes = new RadMaskedTextBox();
    rmtbNotes.Skin = "Office2007";
    rmtbNotes.TextMode = RadInputTextBoxMode.MultiLine;
    rmtbNotes.Width = new Unit("90%");
    rmtbNotes.Rows = 3; // show three rows

    AddLiteralMaskPart(rmtbNotes.MaskParts, string.Format("Notes ({0}) : ",
```

```
       DateTime.Now));
      AddLiteralMaskPart(rmtbNotes.MaskParts,  NEWLINE);
      AddFreeMaskPart(rmtbNotes.MaskParts, 40);
      AddLiteralMaskPart(rmtbNotes.MaskParts,  NEWLINE);
      AddFreeMaskPart(rmtbNotes.MaskParts, 40);

      pnlNotes.Controls.Add(rmtbNotes);
    this.Notes.Add(rmtbNotes);
     rmtbNotes.Focus();
}
```

VB Example:

```
Protected Sub  btnAddNotes_Click(ByVal sender As Object, ByVal e As  EventArgs)
 Const  NEWLINE As String = "" & Microsoft.VisualBasic.Chr(13) & "" & _
   Microsoft.VisualBasic.Chr(10) & ""
 Dim  rmtbNotes As  RadMaskedTextBox  = New  RadMaskedTextBox
  rmtbNotes.Skin  =  "Office2007"
  rmtbNotes.TextMode  =  RadInputTextBoxMode.MultiLine
  rmtbNotes.Width  = New  Unit("90%")
  rmtbNotes.Rows  = 3
  AddLiteralMaskPart(rmtbNotes.MaskParts,  String.Format("Notes  ({0})  :  ",  _
    DateTime.Now))
   AddLiteralMaskPart(rmtbNotes.MaskParts,   NEWLINE)
   AddFreeMaskPart(rmtbNotes.MaskParts,  40)
   AddLiteralMaskPart(rmtbNotes.MaskParts,   NEWLINE)
   AddFreeMaskPart(rmtbNotes.MaskParts,  40)
    pnlNotes.Controls.Add(rmtbNotes)
   Me.Notes.Add(rmtbNotes)
  rmtbNotes.Focus
End Sub
```

9. The btnAddNotes_Click handler creates a new RadMaskedTextBox, sets it basic properties, builds a mask using the private methods we created earlier and adds the control to the Notes property for later reuse on Page_Load. Some other points to note in the btnAddNotes_Click handler:

- The *Skin* property is set to "Office2007", matching the skin on the other controls.
- The *TextMode* property is set to MultiLine. Later when building the MaskParts collection we will add a literal that contains a carriage return and new line characters. **Important note**: Currently the control only accepts "\r\n" which remains as part of the mask when emitted to HTML. If you use "\n" a new line will appear in the emitted html causing an unterminated string error.
- The *Rows* property is "3". One row for the prompt and two rows of 40 FreeMaskPart characters each.

10. The pieces are put together in the Page_load. If this is the first time through we set the *MinDate* and *MaxDate* properties of the "Delivery Date" RadDateInput control, build and populate the "Delivery Time" enumeration mask, and set the focus on the first name entry. If this is a postback we re-populate any RadMaskedTextBox controls that were created to contain notes.

C# Example:

```
protected void  Page_Load(object  sender,  EventArgs  e)
{
    if  (IsPostBack)
    {
        foreach  (RadMaskedTextBox  textBox in this.Notes)
        {
              pnlNotes.Controls.Add(textBox);
```

```
        }
    }
    else
    {
        int year = DateTime.Today.Year;
        int month = DateTime.Today.Month;
        int lastDay = DateTime.DaysInMonth(year, month);
         rdiDate.MinDate = DateTime.Today;
        rdiDate.MaxDate = new DateTime(year, month, lastDay);
         rdiDate.SelectedDate = DateTime.Today;

          AddEnumerationMaskPart(rmtbTime.MaskParts, typeof(DeliveryPeriod));

         rmtbFirst.Focus();
    }
}
```

VB Example:

```
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If IsPostBack Then
   For Each textBox As RadMaskedTextBox In Me.Notes
      pnlNotes.Controls.Add(textBox)
   Next
 Else
   Dim year As Integer = DateTime.Today.Year
   Dim month As Integer = DateTime.Today.Month
   Dim lastDay As Integer = DateTime.DaysInMonth(year, month)
    rdiDate.MinDate = New DateTime(year, month, 1)
    rdiDate.MaxDate = New DateTime(year, month, lastDay)
      AddEnumerationMaskPart(rmtbTime.MaskParts, GetType(DeliveryPeriod))
    rmtbFirst.Focus
 End If
End Sub
```

11. The last piece of the application is to populate the "btnDeliver" OnClick event handler. In this method we retrieve the text properties from the RadMaskedTextBox controls, the SelectedDate from the RadDateInput and display the results in the status label.

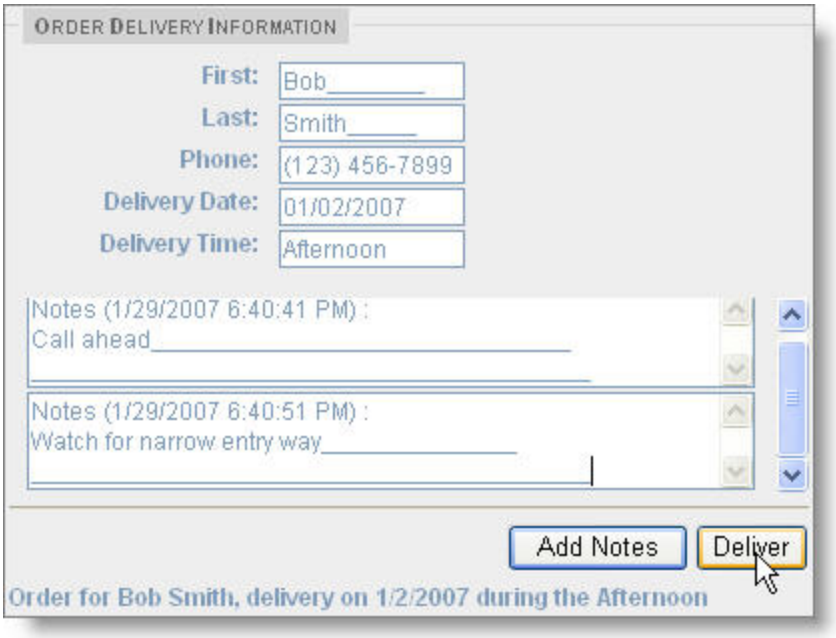
C# Example:

```
protected void btnDeliver_Click(object sender, EventArgs e)
{
    const string messageFormat = "Order for {0} {1}, delivery on {2} during the {3}";
     lblStatus.Text = string.Format(messageFormat,   rmtbFirst.Text,
          rmtbLast.Text, rdiDate.SelectedDate.ToString("d"), rmtbTime.Text);
}
```

VB Example:

```
protected Sub btnDeliver_Click(ByVal sender As Object, ByVal e As EventArgs)
 Const messageFormat As String = "Order for {0} {1}, delivery on {2} during the {3}"
  lblStatus.Text = String.Format(messageFormat, rmtbFirst.Text, rmtbLast.Text, _
    rdiDate.SelectedDate.ToString("d"), rmtbTime.Text)
End Sub
```

12. Run the application to test the functionality. Test the Delivery Time input to see how the hint with the DeliveryPeriod enumeration can be selected. Add multiple notes and enter into them. Click the delivery button to see the values entered.



The running project

Using RadMaskedTextBox at runtime

Lab: IP address

In the "Using RadInput in the Designer" section earlier we discussed the RadMaskedTextBox output properties *Text*, *TextWithLiterals*, *TextWithPrompt* and *TextWithPromptAndLiterals.* You can often use these properties right out of the box, but in other situations you may need to massage the output. IP addresses for example can be prompted, for example, "127.000.000.001". But you can't use the leading zeros when actually connecting to an IP address. You may be looking for "127.0.0.1" instead. In this lab you'll learn one way to setup an IP address prompt and format the results to a usable IP:

1) Create a new project "IPAddress".

2) On the default page place a RadMaskedTextBox, a standard ASP.NET button and label. Leave the *ID* properties at their default values.

3) The RadMaskedTextBox properties should be set as follows:

   a) *Mask*: `"<0..255>.<0..255>.<0..255>.<0..255>"`

   b) *ZeroPadNumericRanges*: `True`

   c) *PromptChar*: `"_"`

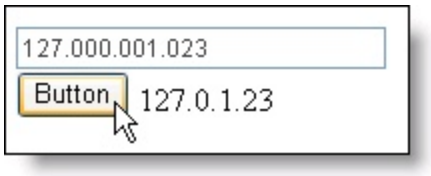
   Note: the HTML should be similar to this:

```
<radI:RadMaskedTextBox
     ID="RadMaskedTextBox1"
    runat="server"
    PromptChar="_"
      Mask="<0..255>.<0..255>.<0..255>.<0..255>"
     ZeroPadNumericRanges="true">
  </radI:RadMaskedTextBox>
```

4) Create a button click event handler and enter the following code:

```
C# Example:
protected void Button1_Click(object sender, EventArgs e)
{
    string result = "";
    //  "127.000.001.022"
    string[] parts = RadMaskedTextBox1.TextWithLiterals.Split('.');
    // "127" "000" "001" "022"
    foreach (string part in parts)
    {
        string trimmed = part.TrimStart('0');
        // "127" "" "1" "22"
        string corrected = trimmed.Length == 0 ? "0" : trimmed;
        // "127" "0" "1" "22"
         result += result.Length == 0 ? corrected : "." + corrected;
        //  "127.0.1.22"
    }
     Label1.Text = result;
}

VB Example:
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
 Dim result As String = ""
 Dim parts As String() = RadMaskedTextBox1.TextWithLiterals.Split("."C)
 For Each part As String In parts
   Dim trimmed As String = part.TrimStart("0"C)
   Dim corrected As String = Microsoft.VisualBasic.IIf(trimmed.Length = 0,"0",trimmed)
    result += Microsoft.VisualBasic.IIf(result.Length = 0,corrected,"." + corrected)
 Next
 Label1.Text = result
End Sub
```

5) Run the project and test it against varying input using 1, 2, 3 and no digits of the IP address. Notice that it correctly process when all digits of an IP address element are zero.



The running project

A second look at the code

The IP address "127.0.1.22" tests against having zero or more digits and so makes a good example. The *TextWithLiterals* property would read: "127.000.001.022". First we turn the TextWithLiterals into an array so we can iterate the parts of the IP address. Now we can think of the address as "127", "000", "001", "022".

```
C# Example:
string[] parts = RadMaskedTextBox1.TextWithLiterals.Split('.');

VB Example:
Dim parts As String() = RadMaskedTextBox1.TextWithLiterals.Split("."C)
```

Iterate the elements of the IP address, first trimming any leading zeros:

```
C# Example:
string trimmed = part.TrimStart('0');
```

```
VB Example:
Dim trimmed As String = part.TrimStart("0"C)
```

Now the elements would be: "127" "", "1" "22". That works nicely for most cases except where the element is a zero. To correct that problem we add our zero back in if the trimmed element is blank (length == 0):

```
C# Example:
string corrected = trimmed.Length == 0 ? "0" : trimmed;

VB Example:
Dim corrected As String = Microsoft.VisualBasic.IIf(trimmed.Length = 0,"0",trimmed)
```

The elements should now be "127" "0" "1" "22" -- close to what we want. To finish formatting the IP address, concatenate the results with periods.

```
C# Example:
result += result.Length == 0 ? corrected : "." + corrected;

VB Example:
result += Microsoft.VisualBasic.IIf(result.Length = 0,corrected,"." + corrected)
```

Using RadDateInput at runtime

Lab: Setting the culture

The RadDateInput control has culture properties that cause the control to display date in a given language and culture variant. In this lab you will learn how to create a CultureInfo object and assign it to the date input control. We will use a RadComboBox to select from available cultures.

1. Create a new project "DateInputCulture".
2. On the default page drop a RadComboBox and RadDateInput . Leave the *ID* properties at their defaults.
3. Set the RadComboBox properties:
   - *AutoPostBack*: true
   - *Height*: 100px
   - *Sort*: Ascending
4. Set the *DateFormat* property of the RadDateInput to "F" (long date and long time format).
5. Create a new page load event handler and input the following code:

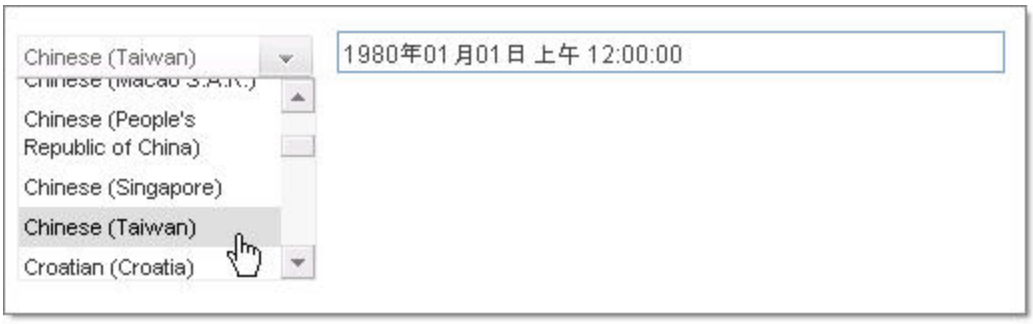
```
C# Example:
if (!IsPostBack)
{
      CultureInfo[] cultures = CultureInfo.GetCultures(CultureTypes.FrameworkCultures);
    foreach (CultureInfo culture in cultures)
    {
        if ((!culture.IsNeutralCulture))
        {
            RadComboBox1.Items.Add(new RadComboBoxItem(culture.DisplayName, culture.Name));
        }
    }

     RadComboBox1.FindItemByValue("en-US").Selected = true;
}
```

```
VB Example:
If Not IsPostBack Then
 Dim cultures As CultureInfo() = CultureInfo.GetCultures(CultureTypes.FrameworkCultures)
 For Each culture As CultureInfo In cultures
   If (Not culture.IsNeutralCulture) Then
      RadComboBox1.Items.Add(New RadComboBoxItem(culture.DisplayName, culture.Name))
   End If
 Next
  RadComboBox1.FindItemByValue("en-US").Selected = True
End If
```

Here we get a list of all the cultures supported by the .NET framework, iterate them, and populate the RadComboBox with those cultures that are associated with a specific region. The visible text in the combo is the culture *DisplayName,* i.e. "Spanish (Mexico)", and the value portion is the culture *Name,* i.e. "es-MX".

6. In the SelectedIndexChanged event handler for the RadComboBox create a new CultureInfo object using the culture name in the constructor and assign it to the RadDateInput *Culture* property.

```
C# Example:
protected void RadComboBox1_SelectedIndexChanged
    (object o, Telerik.WebControls.RadComboBoxSelectedIndexChangedEventArgs e)
{
     CultureInfo cultureInfo = new CultureInfo(RadComboBox1.SelectedValue);
     RadDateInput1.Culture = cultureInfo;
}

VB Example:
protected Sub RadComboBox1_SelectedIndexChanged(ByVal o As Object, _
  ByVal e As Telerik.WebControls.RadComboBoxSelectedIndexChangedEventArgs)
   Dim cultureInfo As New CultureInfo(RadComboBox1.SelectedValue)
    RadDateInput1.Culture = cultureInfo
End Sub
```



Running the project

7. Run the project. Notice that the date formatting has changed to reflect the culture.

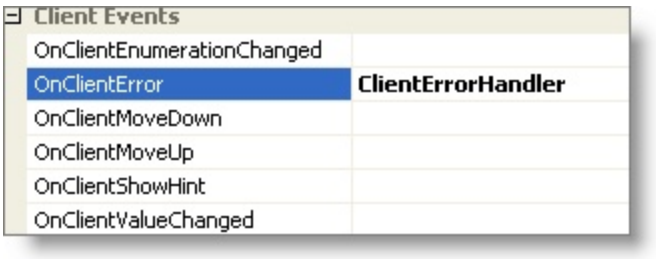
1.5.4 Client Scripting with RadInput

RadDateInput has a single client event "OnClientDateChange" that takes two parameters, a reference to the input control itself and "args" which contains references to javascript dates "NewDate" and "OldDate". The example below compares to see if this is a new date and if so, performs some slicing and dicing to format the date for display.

```
function ClientDateChangedHandler(input, args)
```

```
{
   if (args.OldDate != args.NewDate)
  {
     var date = args.NewDate;
     var mmddyy = date.getMonth() + "/" + date.getDay() + "/" + date.getYear();
     DisplayStatus("Your appointment is rescheduled to " + mmddyy);
  }
}
```

RadMaskedTextBox has a series of client events shown in the figure below.



RadMaskedTextBox client side events

Each of the RadMaskedTextBox events take two parameters, a reference to the input control itself and an event arguments object which contains references to the "NewValue and "OldValue". The semantics for NewValue and OldValue vary depending on the event.

Client Event	Description
OnClientEnumerationChange	This event fires when the value of any enumeration mask part changes. OldValue and NewValue refer to the old and new values of the enumeration mask part.
OnClientError	This event fires every time the user attempts to type a disallowed character NewValue is the invalid character.
OnClientMoveDown	This event fires when the user increases the value of an enumeration or numeric range mask part. OldValue and NewValue refer to the old and new values of the current mask part.
OnClientMoveUp	This event fires when the user decreases the value of an enumeration or numeric range mask part. OldValue and NewValue refer to the old and new values of the current mask part.
OnClientShowHint	This event fires after the hint is shown. OldValue and NewValue refer to the old and values of the mask part.
OnClientValueChanged	This fires when the value of the control changes. OldValue and NewValue refer to the old and new value of the control.

Lab: Adding client side feedback

1. We can add client side event handling to the previous Order Delivery Entry example to provide instantaneous feedback during erroneous input, when the delivery time enumeration changes, and when the delivery date changes.

2. Add the script tag to the head tag:

```
<script   type="text/javascript">
    //<!--

    //-->
</script>
```

3. Add a helper function to display status messages in the javascript tag. This just gets a reference to status label control "lblStatus" and sets the innerHTML to display passed in text..

```
function   DisplayStatus(message)
{
     var  lblStatus  =  document.getElementById('lblStatus');
    if (lblStatus)
    {
         lblStatus.innerHTML = message;
    }
}
```

4. Add a OnClientErrorHandler to the javascript. This uses the DisplayStatus() function just defined and displays the invalid keypress.

```
function  ClientErrorHandler(input,  args)
{
     DisplayStatus("Invalid  character:"  +  args.NewValue);
}
```

5. Add a OnClientEnumerationChange event handler to the JavaScript. This function detects if a new enumeration values was selected from the hint and maps a display message to the new value selected.

```
function   ClientEnumerationChangedHandler(input,   args)
{
   if  (args.OldValue  !=  args.NewValue)
  {
     switch  (args.NewValue){
         case "Morning":
              DisplayStatus("Delivery will be between 9am and 12pm");
             break;
         case "Afternoon":
              DisplayStatus("Delivery will be between 12pm and 5pm");
             break;
         case "Evening":
              DisplayStatus("Delivery will be between 5pm and 9pm");
             break;
    }
  }
}
```

6. Add the OnClientDateChanged event handler described earlier to the JavaScript. This function displays the latest input date.

```
function  ClientDateChangedHandler(input,  args)
{
     var month = args.NewDate.getMonth() + 1;
     var mmddyy = month + "/" + args.NewDate.getDate() + "/" + args.NewDate.getFullYear();
     DisplayStatus("Your appointment is rescheduled to " + mmddyy);
}
```

7. In the designer, set the client side event handler properties:

- Set the OnClientError property of all RadMaskedTextBox to "ClientErrorHandler".
- Set the OnClientEnumerationChanged property for rmtbTime to "ClientEnumerationChangedHandler".
- Set the OnClientDateChanged property for rdiDate to "ClientDateChangedHandler".
- Run the application and test by entering invalid input. Also try changing the delivery time enumeration and the delivery date.

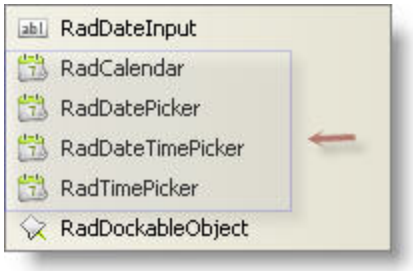
1.5.5 Summary

This section covered using RadDateInput and RadMaskedTextBox to retrieve valid, properly formed input from the user. You saw how the RadInput controls assist the user in making correct input choices. The section demonstrated both design time and programmatic usage and explained the most significant properties.

1.6 RadCalendar

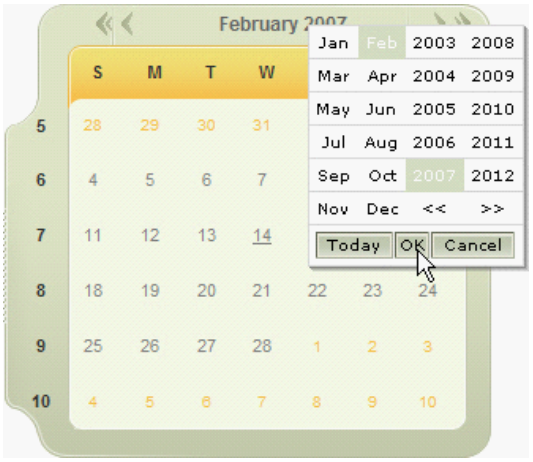
1.6.1 Getting Started

RadCalendar consists of date input controls for quick, intuitive date, time and date/time selection as well as a stand alone calendar control. Although these controls are considered lightweight in respect to payload traveling to the client, the feature set and ability to customize are anything but lightweight. The challenge to using this control suite isn't "can I control XYZ aspect" but organizing and locating specific functionality.



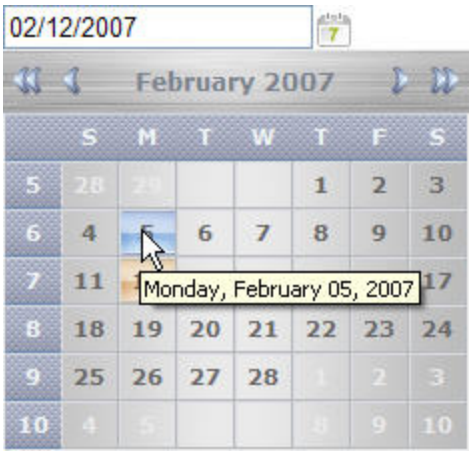
RadCalendar controls in the toolbox

RadCalendar is used stand-alone and is also a part of the composite picker controls shown below. Some of the ways you can customize:

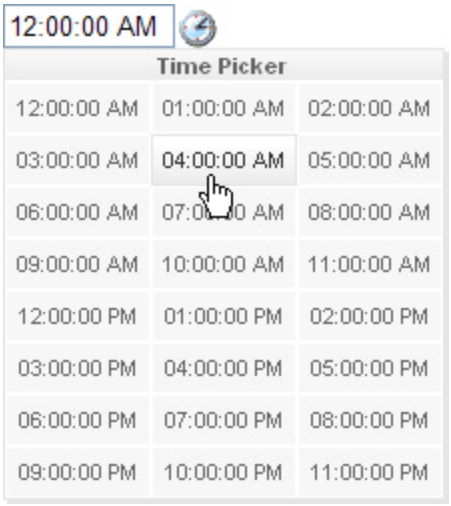


- You have access to events that render both cell and header cell areas. Here you can access information about the cell, change properties of the cell and add your own controls to the cell.
- Skins are provided for a number of common styles, such as Office2007 or Outlook. You can also build your own skins. The example here is using the "Modern" skin.
- Style properties cover most areas of the calendar. All aspects of the day appearance are covered: default, disabled, mouse over, out-of-range days, weekends, special days, selected days. There are also styles for the header, months, title area, "fast navigator" area, and the calendar as a whole.
- Templates allow you to drop images or other controls into the header, footer and any day area.
- Layout: You have control over the number of months shown at one time, number of rows/columns and the manner they are arranged, how the start day of the week is presented,

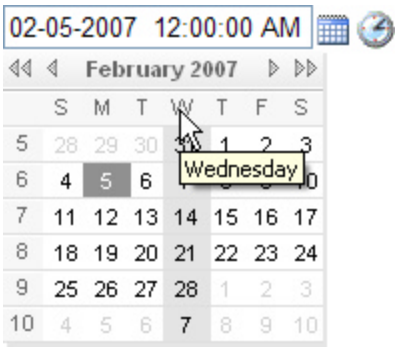
RadDatePicker is a composite control consisting of RadDateInput, a RadCalendar and a popup button control. Each element of the control is customizable in respects to behavior and appearance. The example shown here shows the calendar with the "Inox" skin. You have direct access to the underlying RadDateInput and so may use techniques described in the RadInput section of this course. RadCalendar is also accessible as a property of RadDatePicker, or you may share a RadCalendar among several inputs for better performance.



RadTimePicker is also a composite control consisting of a RadDateInput and popup button control. Again, each of the pieces of the control are customizable. The example shown here is the default as dropped from the palette and shows the default one hour interval in the popup.

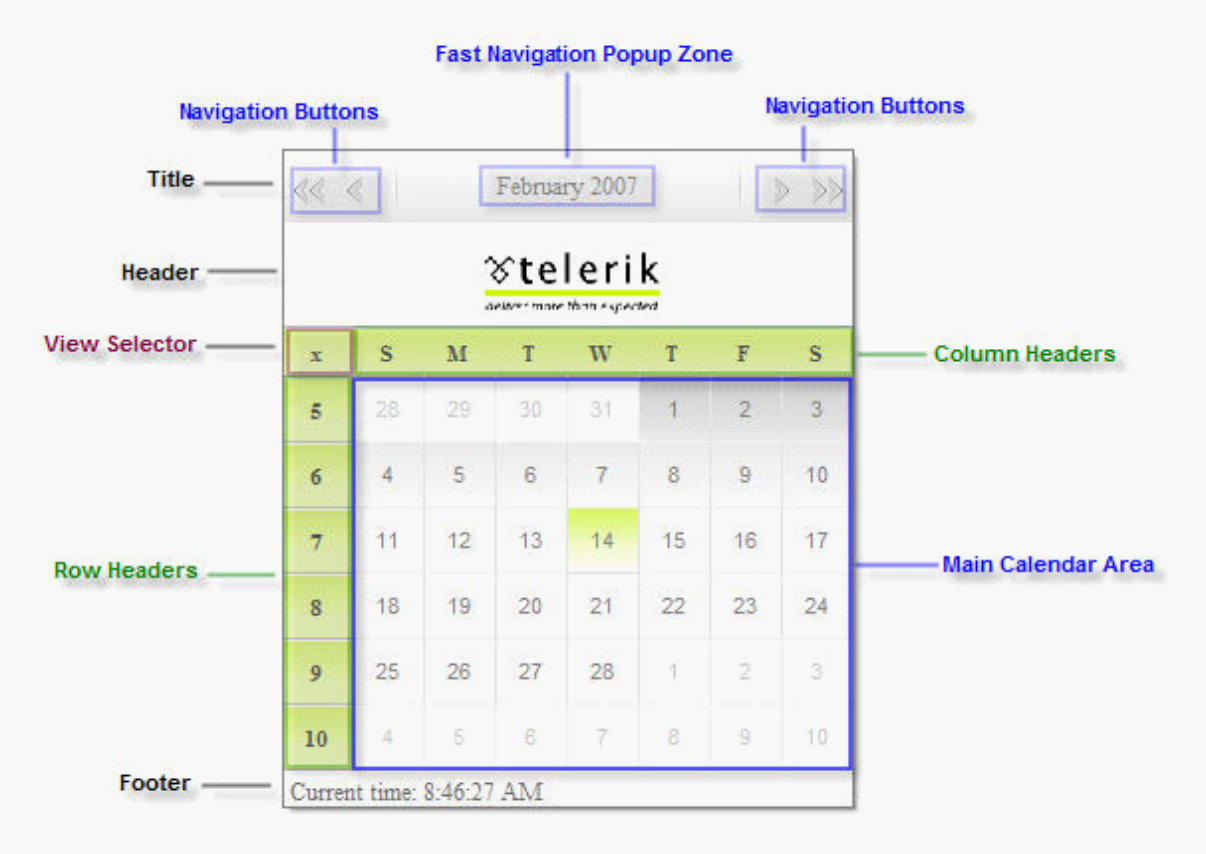


RadDateTimePicker combines the functionality of the date and time pickers shown above. The example here uses the default appearance.



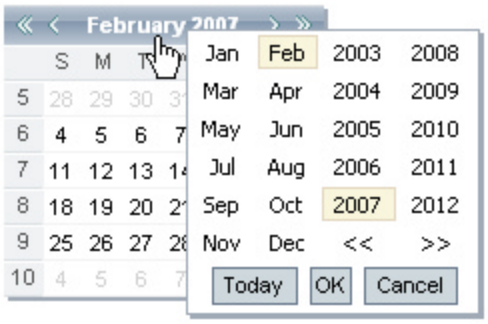
Lab: RadCalendar Quick Tour

The RadCalendar is arranged in logical areas that have corresponding properties, methods and styles. The figure below shows the general layout and then describes each of the elements in the Calendar. Review the descriptions and proceed to the lab below.



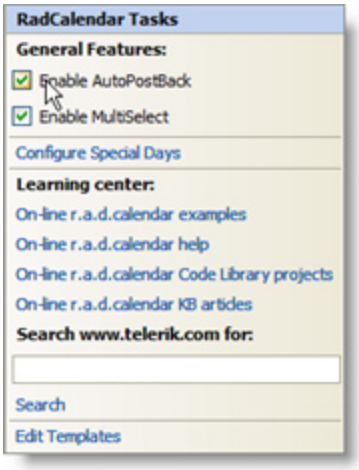
Calendar layout

Calendar Element	Description
Title	The title displays the selected month and year for one or more months and the right and left navigation buttons.
Header	Header is a template area that you can drop images or other controls into at design time. You can obtain a reference to objects in the template using the FindControl() method of RadCalendar.
View Selector	The View Selector allows the user 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.
RowHeaders, ColumnHeaders	Headers are the cells that bound the top and left of the main calendar area.
Footer	Like the Header, the Footer is also a template area where you can drop images, text or other controls at design time. You can obtain a reference to objects in the template using the FindControl() method of RadCalendar.

Main Calendar Area	This area contains cells that represent days. Properties, methods and style for this area handle formatting, display, style, special days collection and rendering of individual cells.
Navigation Buttons	These buttons provide the user an easy way to quickly page through days and months. The double arrows are "Fast Navigation" buttons.
Fast Navigation Popup Zone	Clicking on the Fast Navigation Zone displays a popup with abbreviated navigation through months and years and a shortcut to "Today".  **Fast Navigation Popup**

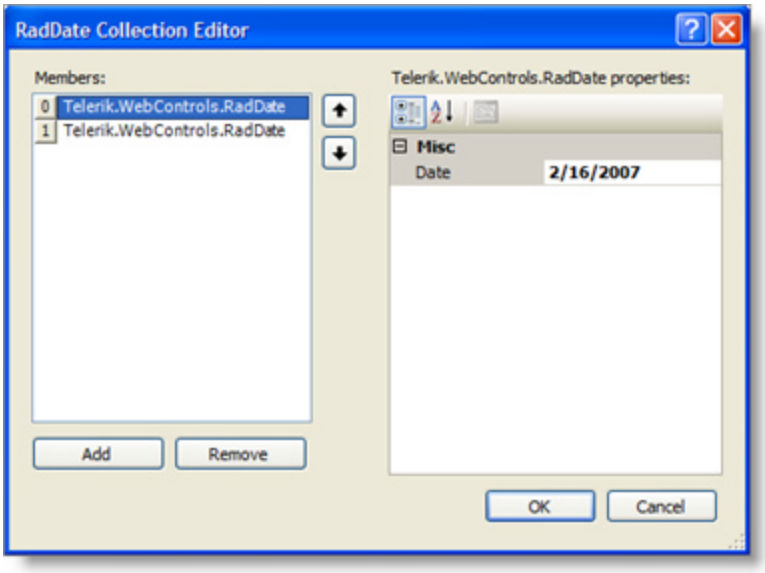
The end product you're typically trying to get from RadCalendar are the *SelectedDate* and *SelectedDates* property values. By default the user is able to set multiple dates in the calendar thus populating the SelectedDates collection. You can also populate SelectedDates at design time. This lab demonstrates setting SelectedDates and several formatting properties on RadCalendar.

1. Create a new web application "GettingStarted".
2. Add a RadCalendar control to the default page.
3. From the RadCalendar Smart Tag enable AutoPostBack.



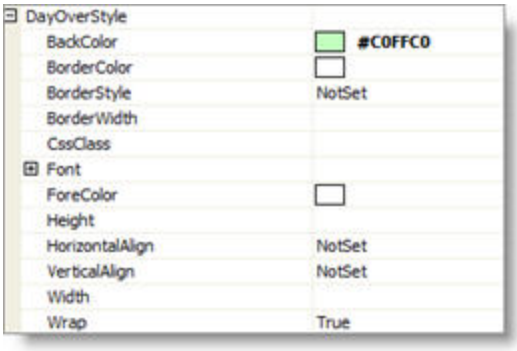
Enabling AutoPostBack

4. Set the *TitleFormat* property to "D" in the Properties Window. This is the long date format.
5. Set the *EnableViewSelector* property to "True".
6. Set the *DayNameFormat* to "Short".
7. Set the *FirstDayOfWeek* to "Monday".
8. Click the ellipses on *SelectedDates* in the Properties Window.
9. Click the Add button twice. Set the dates for the two days following today and click OK to close the dialog.



Editing SelectedDates in the RadDate Collection Editor

10. In the Properties Window open the *DayOverStyle* and set *BackColor* to light green.



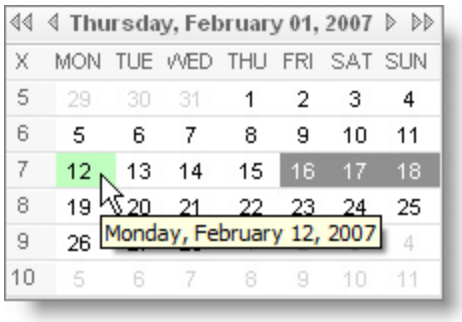
Setting the DayOverStyle BackColor

11. Run the application and test the functionality:

- Row and column selectors
- The view selector in the upper left corner
- Move the mouse over the days and notice the DayOverStyle effect.
- Click the Fast Navigation Popup Zone (by clicking the date in the title area) and use that to navigate

dates.

- Use the navigation buttons to move through days and months. Notice that the date in the title area changes as you navigate.

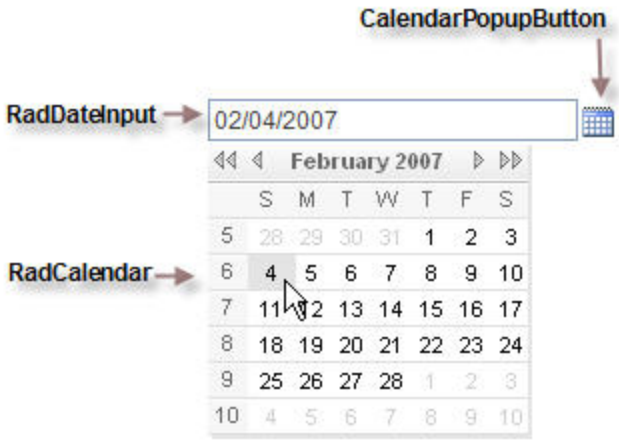


The Running Application

Lab: Date, Time Picker Quick Tour

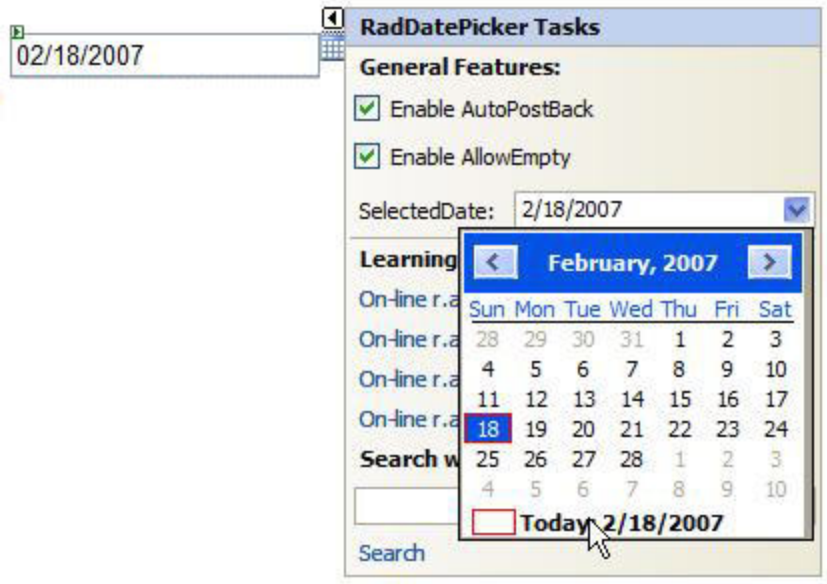
Take the quick tour of RadDatePicker and RadTimePicker then proceed to the lab below.

RadDatePicker is a composite of the RadCalendar, RadDateInput and an image button PopupButton. Properties that address the functionality of the overall control have been surfaced to the parent control, e.g. MinDate, MaxDate, and FocusedDate. More specific elements can be accessed by using the underlying control itself, e.g. DateFormat.



RadDatePicker Layout

Use the RadDatePicker Smart Tag to enable AutoPostBack, AllowEmpty and to quickly set the SelectedDate.



RadDatePicker SelectedDate SmartTag

There are two approaches to handling the calendar portion of the control:

- Leave the baked-in RadCalendar and simply access its properties.
- Use the SharedCalendarID property. In this approach you drop a separate stand alone RadCalendar control and any number of RadDatePicker controls on the page. In each RadDatePicker you point the SharedCalendarID property to the stand alone instance of RadCalendar. The advantage is that you set up the properties of the calendar once, and more importantly improve performance by cutting down on the page payload.

Note: Be aware that at the time of this writing you can't directly access the MultiView aspects of the calendar from the RadDatePicker, but you can use a shared calendar to achieve the same effect.

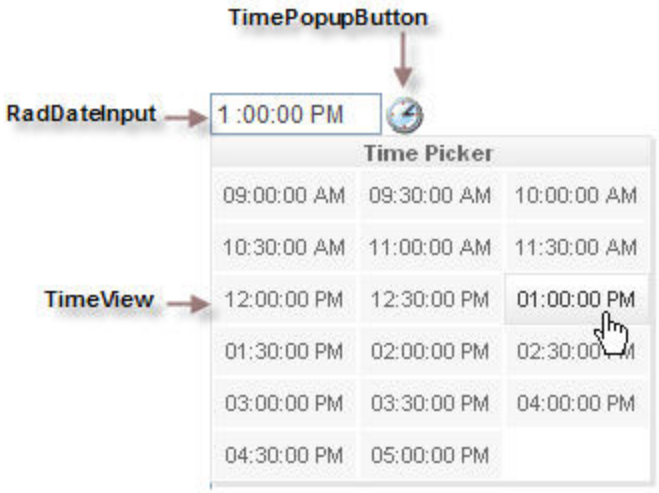
RadTimePicker is a composite of a RadDateInput where the *DateFormat* is set to "T" for time format, an image button *TimePopupButton* property and a *TimeView* property. Some important *TimeView* properties are:

StartTime, EndTime: By default these values are 00:00:00 and 23:59:59 respectively. Only times within these bounds show in the popup window.

Interval: The interval between times shown in the popup window. By default this value is 1 and displays every hour between StartTime and EndTime.

TimeFormat: By default this value is "hh:mm:ss tt". See the RadInput section for formatting information.

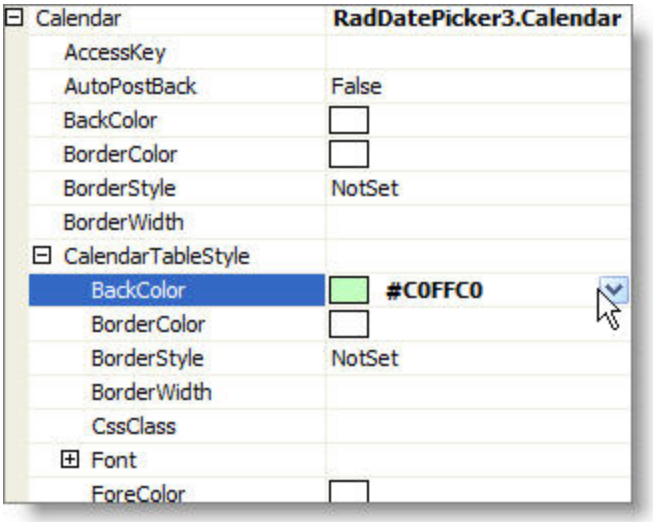
Skin: Controls the look of the popup area.



RadTimePicker with StartTime, EndTime and Interval

RadDateTimePicker is essentially the sum of the RadDatePicker and RadTimePicker. One unique property worth noting is the *AutoPostBackControl* that determines which control triggers an automatic postback. This property can be set to "None", "TimeView", "Calendar" or "Both.

1. Re-open the "GettingStarted" application created in the RadCalendar Quick Tour.
2. Drop a RadDatePicker on the page.
3. Use the Smart Tag to enable AutoPostBack and AllowEmpty. Also set the SelectedDate to be "Todays" date. In the Properties Window set the *SharedCalendarID* property to "RadCalendar1". This RadCalendar instance should be left over from the previous lab.
4. Drop another RadDatePicker control on the page. Set the *SharedCalendarID* property to "RadCalendar1".
5. Set the RadCalendar1 *CalendarTableStyle BackColor* to a light red to visually differentiate it.
6. Drop a third RadDatePicker control on the page. Set the *CalendarTableStyle BackColor* to light green for the internal Calendar instance.



Setting the CalendarTableStyle for the internal Calendar

7. Run the application. Notice the first two date pickers display the shared calendar with the red background. The third calendar main calendar area is light green.

8. Stop the application.

9. Drop a RadTimePicker control on the page.

10. Set the AutoPostBack property to "True".

11. In the Properties Window set *TimeView* sub properties:

- *StartTime*: "09:00:00"
- *EndTime*: "17:30:00"
- *Interval*: "00:30:00"

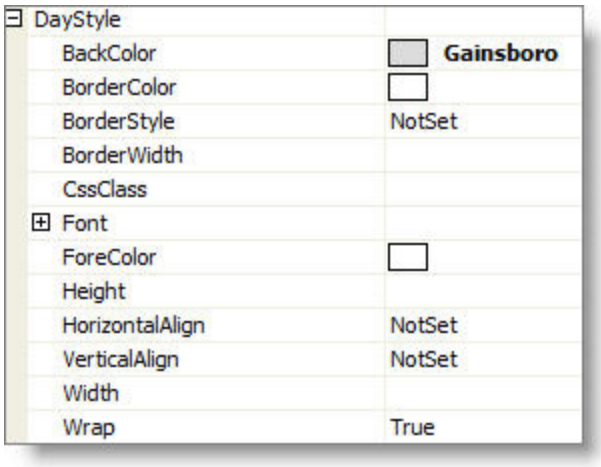
12. Run the application. Click the time picker popup button and notice the half hour intervals and that only hours between 9 and five are displayed.

13.

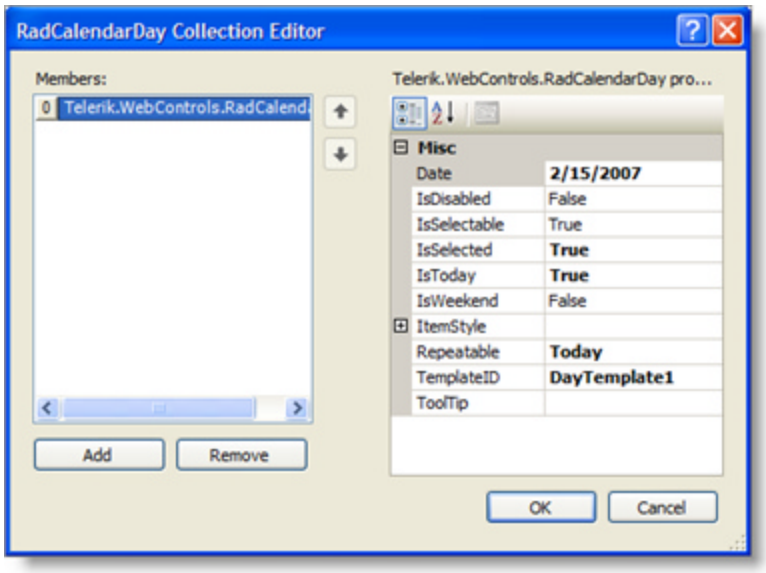
1.6.2 Using RadCalendar in the Designer

RadCalendar Properties

This section groups important properties based on the calendar layout diagram in the previous lab. Try to familiarize yourself with the general grouping of properties here so you know what properties affect particular aspects of the calendar control.

Calendar Element	Description
Title	The title displays the selected month and year for one or more months and the right and left navigation buttons. Some properties that directly control this area are: *TitleAlign:* Governs horizontal alignment within the title area. *TitleFormat:* The date format pattern as described in the RadInput section "Set Date Format". This can also include literal text. *DateRangeSeparator:* The character(s) that go between months when viewing multiple months. By default a dash "-" character. *TitleStyle*: The CSS style that governs the appearance of the title area. The various style properties for the entire control have the same format:  **Sample Style Property**
Header	Header is a template area that you can drop images or other controls into at design time. You can obtain a reference to objects in the template using the FindControl() method of RadCalendar. *HeaderStyle:* The CSS style that governs the appearance of the header.
View Selector	The View Selector allows the user 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. *EnableViewSelector*: True if the selector is to display and allow selection. *ViewSelectorImage*: A path to the image that displays in the view selector. *ViewSelectorText*: Text that appears in the view selector.

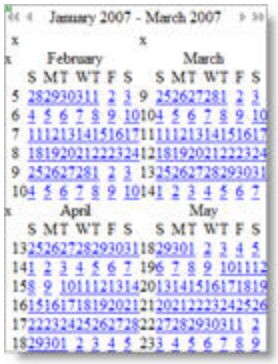
Calendar Element	Description
RowHeaders, ColumnHeaders	Headers are the cells that bound the top and left of the main calendar area. *RowHeaderImage*, *ColumnHeaderImage*: An image path for an image that appears in all row and column headings, respectively. *RowHeaderText, ColumnHeaderText*: Text that appears in all row/column headings. *UseRowHeadersAsSelectors, UseColumnHeadersAsSelectors*: If true allows the user to click on the row/column and select the entire row/ column. *ShowRowHeaders, ShowColumnHeaders*: If true displays the row/column headings. *HeaderCellRender*: This event fires non-day cells, i.e. column header cells, row header cells and the view selector cell. It returns the cell object and the HeaderType which may be View, Row or Column. *DayNameFormat*: The format for the column headers showing the day names. Defaults to first letter, i.e. "S M T W T F S" but can be "Full", "Short" FirstTwoLetters" and "Shortest". *HeaderStyle*: Contains the CssClass and other properties related to appearance.
Footer	Like the Header, the Footer is also a template area where you can drop images, text or other controls at design time. You can obtain a reference to objects in the template using the FindControl() method of RadCalendar.

Calendar Element	Description
Main Calendar Area	This area contains the cells that represent individual days. *CellAlign, CellVAlign*: Governs horizontal and vertical alignment within the cell. *CellDayFormat*: Defaults to the numeric day of the month but can use any valid date format. *DayStyle, DayOverStyle, DisabledDayStyle, WeekendDayStyle*: Style properties that govern the appearance of day cells in various states: default, when the mouse hovers over the day, when the day is disabled and weekend days respectively. *CalendarDayTemplates*: Templates for individual days *ShowOtherMonthDays*: If true displays days from leading and trailing months. *SpecialDays*: A collection of days that need to be handled in a specific way. Each day in the collection can be marked with several flags (e.g. IsDisabled, IsSelectable, IsWeekend) and can be populated by a template. Templates are configured in the CalendarDayTemplates collection.  **Editing special days in the collection editor**

Calendar Element	Description
Navigation Buttons	Navigate buttons by default are the arrow buttons pointing left and right in the title area of the calendar. These can be replaced with text or images using these properties: *EnableNavigation*: Displays the navigation buttons if true. *EnableNavigationAnimation*: If true, motion between months becomes visible. *NavigationCellPadding, NavigationCellSpacing*: Sets the table padding and spacing that the navigation buttons display in. *NavigationNextImage, NavigationNextText, NavigationPrevImage, NavigationPrevText*: Instead of the default next and previous arrow buttons you can replace them with text or your own images. *FastNavigationNextImage, FastNavigationNextText, FastNavigationPrevImage, FastNavigationPrevText*: Instead of the default next and previous double arrow buttons you can replace them with text or your own images. *FastNavigationStep*: Choose the number of months to travel forward or back when clicking the double arrow "Fast Navigation" buttons.
Fast Navigation Popup Zone	*FastNavigationSettings*: Set the text for sub properties *CancelButtonCaption, DateIsOutOfRangeMessage, OkButtonCaption* and *TodayButtonCaption*. FastNavigationNextImage: fastNavRight.gif FastNavigationNextText: ++ FastNavigationPrevImage: fastNavLeft.gif FastNavigationPrevText: -- FastNavigationSettings CancelButtonCaption: Cancel DateIsOutOfRangeMessage: Date is out of range. OkButtonCaption: OK TodayButtonCaption: Today FastNavigationStep: 1 **Fast Navigation Popup Settings**

These settings affect the look and layout of RadCalendar as a whole:

Property	Description
CalendarTableStyle	This governs the display of the overall style for the main calendar area *and* column/row headers unless a more specific style is defined, e.g. the *HeadersStyle* properties are changed from their defaults.

Property	Description
CssClass	Css class for the entire RadCalendar. Defaults to the calendarWrapper class defined in each of the skin Calendar.css definitions.
DefaultCellPadding, DefaultCellSpacing	table cell padding and spacing for the calendar if nothing more specific is defined.
MonthLayout, *SingleViewColumns*, *SingleViewRows*	These properties govern the column/row arrangement within the month. Use the *MonthLayout* property for predefined arrangements: 7x6, 14x3, 21x2, 7x6,14x3 or 21x2. Or use the *SingleViewColumns* and *SingleViewRows* properties to exactly tailor the configuration. The figure below shows the 14x3 configuration.  **MonthLayout 14x3 option**
MultiViewColumns, MultiViewRows	Use these properties to show more than one month.  **MultiViewColumns/Rows set to 2 x 2**
FirstDayOfWeek	Can be default (Sunday) or any day of the week.
Orientation	Set to *RenderInColumns* this property causes the day headings to be displayed to the left and the days of the month along the top.
PresentationType	Interactive or Preview. Set to Preview for a read only view of the calendar.

Property	Description
FocusedDate, *FocusedDateColumn*, *FocusedDateRow*:	The *FocusedDate* property determines what viewable area will be displayed. For example, if the selected date is 1/1/07 and the FocusedDate is 3/1/07, then March will be the displayed month. FocusedDateRow/Column set where the FocusedDate will be positioned in a multi view area.
RangeMaxDate, *RangeMinDate*	The minimum and maximum dates that can be selected.

Lab: Use RadCalendar Properties

Using the previous section "RadCalendar Properties" as a reference, set up a calendar to the following specifications:

1. The view selector will be displayed and allow the user to select all visible days within a month view.
2. Days that are not part of the month (leading and trailing days from previous and next month) will not be visible.
3. Each week will start on a Monday.
4. Six months will be displayed at one time where months will be displayed in a single row.

Try implementing these suggestions without looking at the directions below. Check your results against directions.

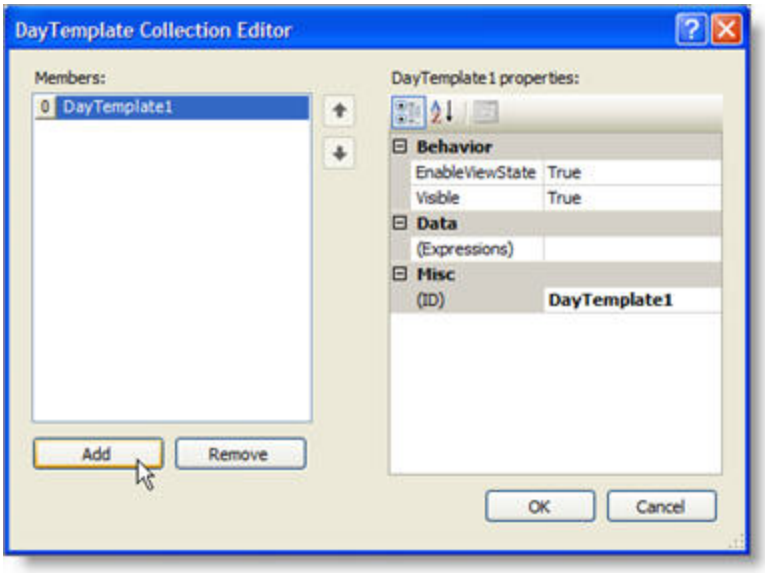
1. Set the *EnableViewSelector* property to "True".
2. Set the *ShowOtherMonthDays* property to "False".
3. Set the *FirstDayOfWeek* property to "Monday".
4. Set the *MultiViewColumns* property to "6".
5. Run the application. It should look like the figure below.

RadCalendar specifications running example

Lab: Special Days

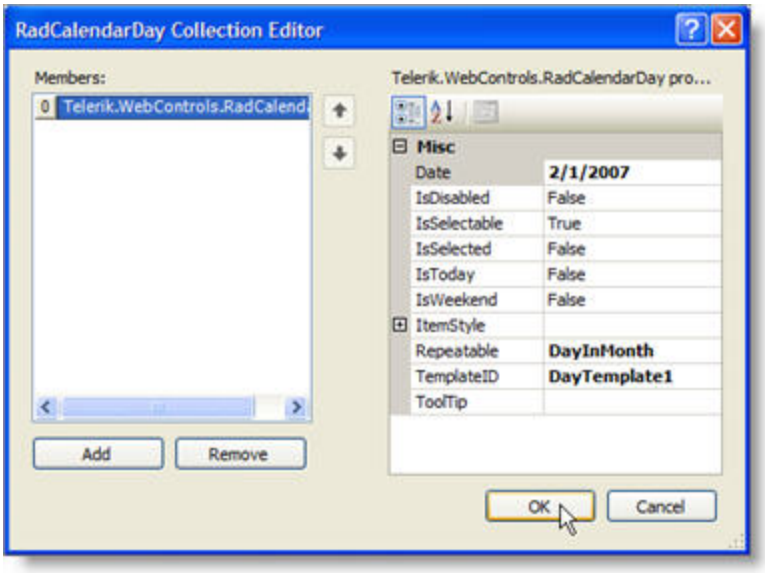
A likely scenario for using the Special Days collection is that you will want to display an image or in some way show the uniqueness of the day. There are two collections that work together to achieve the effect: CalendarDayTemplates and SpecialDays collections. The first step is to create a template, edit the template to provide the visual effect you want, then add to the special days collection and for each day specify a template.

1. Create a new web application "SpecialDays".
2. Drop a RadCalendar control from the toolbox to the default web page.
3. In the Properties Window set the *DefaultCellPadding* and *DefaultCellSpacing* to "5". This will allow room for the text displayed in the special day.
4. In the Properties Window click the ellipses for the *CalendarDayTemplates* property. In the DayTemplate Collection Editor click the Add button to create a single day template. Leave the default values and click the OK button to close the dialog. See figure below.



Adding a day template

5. In the Properties Window click the ellipses for the *SpecialDays* property. Enter the first day of the current month in the *Date* property. Set the *Repeatable* property to "DayInMonth". Enter "DayTemplate1" for the *TemplateID* property.



Adding a special day

6. Add another special day to the list. Leave the *Date* property blank. Set the *IsToday* property to "True" and the *Repeatable* property to "Today". Open the *ItemStyle* property and set *BorderColor* to "Black", *BorderStyle* to "Solid" and *BorderWidth* to "1px".
7. Click the OK button to close the dialog.
8. Click the Smart Tag "EditTemplates" link. From the drop down combo in the Smart Tag select "DayTemplate1". In the template type "Holiday!". Click the Smart Tag "End Template Editing" link.
9. Run the application. Notice that the template displays "Holiday!" on the first day of the month. Navigate to the following and previous months to show the same display on the first day of each month. Also notice that "Today" has a solid border.

The running application displaying the special day

1.6.3 Using RadCalendar at Runtime

RadCalendar Server Events

The first RadCalendar server side event you may need is *SelectionChanged*. It provides an object Sender that represents the RadCalendar and *SelectedDatesEventArgs* that contain a *SelectedDates* property. SelectedDates is not just a DateTime type, but a Telerik DateTimeCollection that provides access to the parent calendar and has methods for manipulating ranges of dates. Use the *Date* member to access the actual DateTime type. For example to iterate and display all the dates selected in the calendar:

```
C# Example:
protected void RadCalendar1_SelectionChanged(object sender,
       Telerik.WebControls.Base.Calendar.Events.SelectedDatesEventArgs e)
{
     StringBuilder stringBuilder = new StringBuilder();
    foreach(RadDate selectedDate in e.SelectedDates)
    {
          stringBuilder.Append(selectedDate.Date.ToString() + "<br />");
    }
     Label1.Text = stringBuilder.ToString();
}

VB Example:
Protected Sub RadCalendar1_SelectionChanged(ByVal sender As Object, _
  ByVal e As     Telerik.WebControls.Base.Calendar.Events.SelectedDatesEventArgs)
 Dim stringBuilder As StringBuilder = New StringBuilder
 For Each selectedDate As RadDate In e.SelectedDates
    stringBuilder.Append(selectedDate.Date.ToString + "<br />")
 Next
 Label1.Text = stringBuilder.ToString
End Sub
```

Or to select a six day range of dates in the calendar:

```
C# Example:
protected void RadCalendar1_SelectionChanged(object sender,
       Telerik.WebControls.Base.Calendar.Events.SelectedDatesEventArgs e)
{
       e.SelectedDates.SelectRange(DateTime.Now, DateTime.Now.AddDays(5));
}

VB Example:
Protected Sub RadCalendar1_SelectionChanged(ByVal sender As Object, _
  ByVal e As     Telerik.WebControls.Base.Calendar.Events.SelectedDatesEventArgs)
   e.SelectedDates.SelectRange(DateTime.Now,   DateTime.Now.AddDays(5))
End Sub
```

Rendering for column headings and individual cells are performed with HeaderCellRender and DayRender events.

HeaderCellRender occurs once for each non-day cell and returns the cell object and the HeaderType. For example, if the HeaderType is output on every HeaderCellRender as in the code sample below, the portions of the calendar that normally display the selector "X", day names across the top and month numbers along the left side will display as the figure below:

```
C# Example:
protected void RadCalendar2_HeaderCellRender(object sender,
      Telerik.WebControls.Base.Calendar.Events.HeaderCellRenderEventArgs e)
{
```

```
    e.Cell.Controls.Add(new  LiteralControl("<b>"  +  e.HeaderType.ToString()  + "</b>"));
}

VB  Example:
Protected Sub   RadCalendar2_HeaderCellRender(ByVal  sender As Object, _
  ByVal e As     Telerik.WebControls.Base.Calendar.Events.HeaderCellRenderEventArgs)
  e.Cell.Controls.Add(New  LiteralControl("<b>"  +  e.HeaderType.ToString  +  "</b>"))
End Sub
```

Results of HeaderCellRender ouput of HeaderType

For complete control of output for each day displayed in the main calendar area, use the DayRender event. It passes both Cell, Day and View objects in the DayRenderEventArgs parameter. You can assign the Cell Text or Controls collection properties to populate the cell. For example, you can use the Day property and assign it to the cell's text property (this approximates the behavior that already occurs).

```
C#  Example:
protected void  RadCalendar1_DayRender(object  sender,
       Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs  e)
{
   e.Cell.Text  =  e.Day.Date.Day.ToString();
}

VB  Example:
Protected Sub   RadCalendar1_DayRender(ByVal  sender As Object, _
  ByVal e As     Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs)
  e.Cell.Text  =  e.Day.Date.Day.ToString
End Sub
```

You can get the same effect adding a literal control to the cells Controls collection. With the Cell Controls property you can go further and add images or other controls. This example adds a "check" image to any selected dates.

```
C#  Example:
protected void  RadCalendar1_DayRender(object  sender,
       Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs  e)
{
    if  (e.Day.IsSelected)
    {
         Image checkImage = new  Image();
         checkImage.ImageUrl = "~/Check.gif";
          e.Cell.Controls.Add(checkImage);
         e.Cell.Controls.Add(new    LiteralControl(e.Day.Date.Day.ToString()));
    }
}

VB  Example:
```

```
Protected Sub  RadCalendar1_DayRender(ByVal sender As Object, _
  ByVal e As    Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs)
 If  e.Day.IsSelected Then
   Dim  checkImage As Image = New Image
    checkImage.ImageUrl = "~/Check.gif"
     e.Cell.Controls.Add(checkImage)
    e.Cell.Controls.Add(New   LiteralControl(e.Day.Date.Day.ToString))
 End If
End Sub
```

Note: If both Cell Controls and Text properties are defined, the last one assigned is displayed.

"MoonPhases" DayRender Example

RadCalendar doesn't currently support integrated data binding, but the effect can be achieved through dynamically creating special days or by using the DayRender event handler. This example consumes a web service that returns the phase of the moon based on a passed in date. Because the longevity of the service isn't known at this time, the example will not be a lab. The service is a REST style web service that is accessed with a direct HTTPRequest call and retrieves a HTTPResponse stream containing an xml file with the data we need to populate the grid. The xml looks similar to this:

```
<?xml  version="1.0"  encoding="utf-8"  ?>
<trynt>
   <moon-phase>
     <date>February  17,  2007</date>
      <date-timestamp>1171763313</date-timestamp>
      <moon-phase-name>Waxing  Crescent</moon-phase-name>
       <moon-phase-position>0.54034641379823</moon-phase-position>
      <lunar-illumination>1.6%%</lunar-illumination>
      <days-til-full-moon>13.57</days-til-full-moon>
      <days-til-new-moon>28.3</days-til-new-moon>
       <days-til-first-quarter-moon>6.2</days-til-first-quarter-moon>
       <days-til-last-quarter-moon>21</days-til-last-quarter-moon>
    <moon-phases>
      <data>
         <day>Saturday</day>
          <moon-phase-name>Waxing  Crescent</moon-phase-name>
      </data>
      <data>
         <day>Sunday</day>
          <moon-phase-name>Waxing  Crescent</moon-phase-name>
      </data>
      <data>

       . . .

    </moon-phases>
   </moon-phase>
</trynt>
```

1. Phases is a generic list of strings property that contains text for phases of the moon, one entry per day. This list is populated whenever its accessed and is null. _phaseIndex is a private member that lets us track the current position with the list of phases defined in the next step. When the list is repopulated, _phaseIndex is reset to zero.

```
C#  Example:
private int  _phaseIndex;
```

```
public List<string> Phases
{
    get
    {
        List<string> phases = ViewState["Phases"] as List<string>;
        if (phases == null)
        {
            _phaseIndex = 0;
              phases = GetMoonPhases(RadCalendar1.FocusedDate.Date);
        }
        ViewState["Phases"] = phases;
        return phases;
    }
    set
    {
        ViewState["Phases"] = value;
    }
}

VB Example:
Private _phaseIndex As Integer

Public Property Phases() As List(of String)
 Get
   Dim phases As List = _
        CType(ConversionHelpers.AsWorkaround(ViewState("Phases"), _
        GetType(List(of String))), List(of String))
   If phases is Nothing Then
      _phaseIndex = 0
       phases = GetMoonPhases(RadCalendar1.FocusedDate.Date)
   End If
    ViewState("Phases") = phases
   Return phases
 End Get
 Set
    ViewState("Phases") = value
 End Set
End Property
```

2. The first page load provides a little elbow room for the text not to look cramped by setting default padding and spacing. *AutoPostBack* is set to true so that navigation to other months or years will still retrieve the moon phase information. This example is designed to work with seven day weeks and so *MonthLayout* is set to 7 X 6. Finally a generic list of strings containing the moon phase text is loaded.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
          RadCalendar1.DefaultCellPadding = 5;
          RadCalendar1.DefaultCellSpacing = 5;
          RadCalendar1.AutoPostBack = true;
           RadCalendar1.MonthLayout = MonthLayout.Layout_7columns_x_6rows;
    }
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
```

```
If Not IsPostBack Then
    RadCalendar1.DefaultCellPadding = 5
    RadCalendar1.DefaultCellSpacing = 5
    RadCalendar1.AutoPostBack = True
     RadCalendar1.MonthLayout = MonthLayout.Layout_7columns_x_6rows
End If
End Sub
```

3. The LoadOneWeek() method adds one weeks worth of moon phase text. An HttpRequest object makes a REST style web service call and uses the HttpResponse to retrieve a stream. The format of the url is taken directly from the web service vendor website example. The stream is resolved to an XmlTextReader and the relevant portions are extracted to the list.

```
C# Example:
private void LoadOneWeek(List<string> phases, DateTime startDate)
{
    const string urlFormat =
      @"http://www.trynt.com/moon-phase-api/v1/?d={0}&fo=xml&f=0";
    string url = string.Format(urlFormat,   startDate.ToString("yyyy-MM-dd"));

     HttpWebRequest request = WebRequest.Create(url) as HttpWebRequest;

    using (HttpWebResponse response = request.GetResponse() as HttpWebResponse)
    {
         StreamReader reader = new   StreamReader(response.GetResponseStream());
         XmlTextReader xmlreader = new  XmlTextReader(reader);
        if  (xmlreader.ReadToFollowing("moon-phases"))
        {
            // load one week worth of moon phase data
            for (int i = 0; i < RadCalendar1.SingleViewColumns; i++)
            {
                if  (xmlreader.ReadToFollowing("moon-phase-name"))
                {
                      phases.Add(xmlreader.ReadString());
                }
            }
        }
    }
}

VB Example:
Private Sub LoadOneWeek(ByVal phases As List(of String), _
  ByVal startDate As DateTime)
 Const urlFormat As String = _
      "http://www.trynt.com/moon-phase-api/v1/?d={0}&fo=xml&f=0"
 Dim url As String  =  String.Format(urlFormat,  startDate.ToString("yyyy-MM-dd"))
 Dim request As HttpWebRequest = _
     CType(ConversionHelpers.AsWorkaround(WebRequest.Create(url), _
    GetType(HttpWebRequest)), HttpWebRequest)
 ' Using
 Dim response As HttpWebResponse = _
     CType(ConversionHelpers.AsWorkaround(request.GetResponse, _
    GetType(HttpWebResponse)), HttpWebResponse)
 try
   Dim reader As StreamReader = New   StreamReader(response.GetResponseStream)
   Dim xmlreader As XmlTextReader = New  XmlTextReader(reader)
   If   xmlreader.ReadToFollowing("moon-phases") Then
     Dim i As Integer = 0
     While i < RadCalendar1.SingleViewColumns
       If   xmlreader.ReadToFollowing("moon-phase-name") Then
           phases.Add(xmlreader.ReadString)
```

```
      End If
          System.Math.Min(System.Threading.Interlocked.Increment(i),i-1)
    End While
  End If
 finally
    CType(response, IDisposable).Dispose()
 End try
End Sub
```

4. This method sets the initial start date and walks through the number of rows showing in the calendar, populating each row worth of data and bumping the start date by a week as it goes.

```
C# Example:
private List<string> GetMoonPhases(DateTime startDate)
{
    List<string> phases = new List<string>();
     DateTime currentDate = startDate;
    for (int i = 0; i < RadCalendar1.SingleViewRows; i++)
    {
          LoadOneWeek(phases, currentDate);
           currentDate = currentDate.AddDays(RadCalendar1.SingleViewColumns);
    }
    return phases;
}

VB Example:
Private Function GetMoonPhases(ByVal startDate As DateTime) As List(of String)
 Dim phases As List(of String) = New List(of String)
 Dim currentDate As DateTime = startDate
 Dim i As Integer = 0
 While i < RadCalendar1.SingleViewRows
    LoadOneWeek(phases, currentDate)
     currentDate = currentDate.AddDays(RadCalendar1.SingleViewColumns)
      System.Math.Min(System.Threading.Interlocked.Increment(i),i-1)
 End While
 Return phases
End Function
```

5. A private helper method GetNextPhase() returns the element in the Phases generic list and bumps the index.

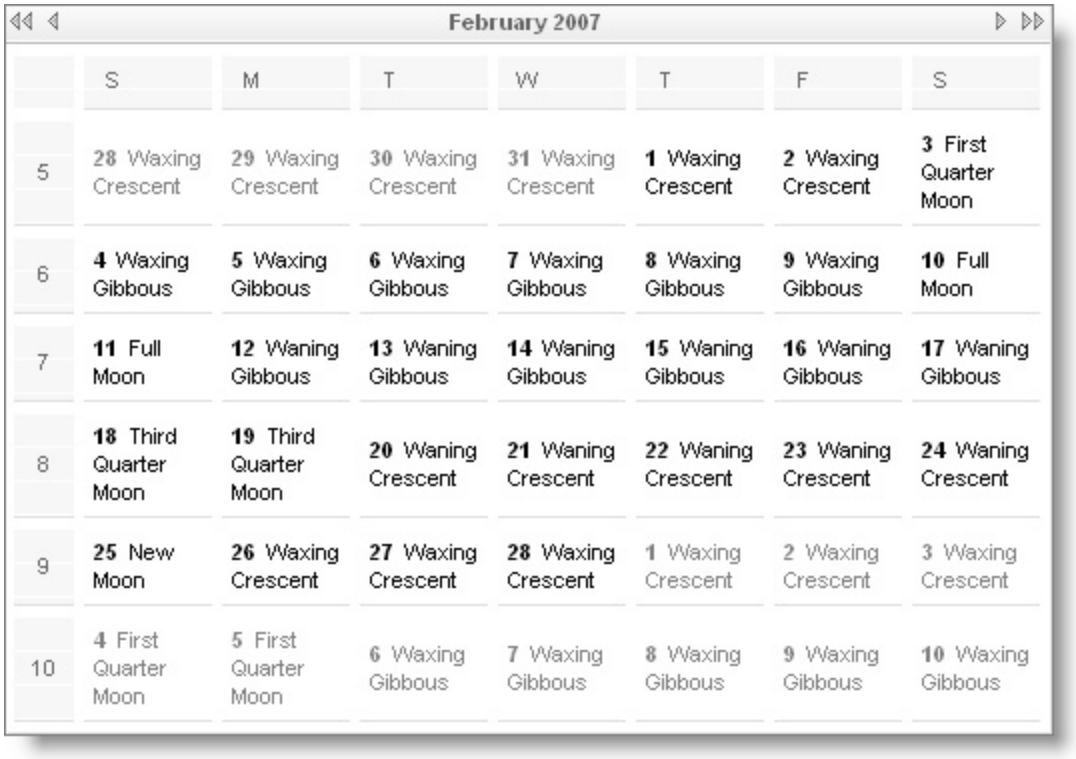
```
C# Example:
private string GetNextPhase()
{
    string result = this.Phases[_phaseIndex];
    _phaseIndex += 1;
    return result;
}

VB Example:
Private Function GetNextPhase() As String
 Dim result As String = Me.Phases(_phaseIndex)
 _phaseIndex += 1
 Return result
End Function
```

6. The RadCalendar *DefaultViewChanged* event handler resets the list.

```
C# Example:
protected void  RadCalendar1_DefaultViewChanged(object sender,
       Telerik.WebControls.Base.Calendar.Events.DefaultViewChangedEventArgs e)
{
    this.Phases = null;
}

VB Example:
Protected Sub  RadCalendar1_DefaultViewChanged(ByVal sender As Object, _
  ByVal e As    Telerik.WebControls.Base.Calendar.Events.DefaultViewChangedEventArgs)
 Me.Phases = Nothing
End Sub
```

7. RadCalendar *DayRender* sets up a string format containing the day number and the moon phase text. The font color is also soft coded here so we can gray out days that aren't part of the current month. The e.Day month is compared with the RadCalendar.FocusedDate month. FocusedDate is used to position the visible, "current" month. Finally the day number and moon phase text are assigned to the cell text property and formatted to indicate if the day is in the current month.

```
C# Example:
protected void  RadCalendar1_DayRender(object sender,
       Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs e)
{
    const string cellFormat = "<font  style=\"color:{0}\"><b>{1}</b>  {2}</font>";

    bool isInMonth = e.Day.Date.Month == RadCalendar1.FocusedDate.Month;
    string color = isInMonth ? "Black" : "Gray";
     e.Cell.Text = string.Format(cellFormat,  color,  e.Day.Date.Day,  GetNextPhase());
}

VB Example:
Protected Sub  RadCalendar1_DayRender(ByVal sender As Object, _
  ByVal e As    Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs)
 Const cellFormat As String = _
    "<font  style=""color:{0}""><b>{1}</b>  {2}</font>"
 Dim isInMonth As Boolean = e.Day.Date.Month = RadCalendar1.FocusedDate.Month
 Dim color As String = Microsoft.VisualBasic.IIf(isInMonth,"Black","Gray")
 e.Cell.Text = String.Format(cellFormat, color, e.Day.Date.Day, GetNextPhase)
End Sub
```



The running application

Lab: RadCalendar Server Events

The lab demonstrates usage of RadCalendar *SelectionChanged*, *DefaultViewChanged*, *HeaderCellRender*, and *DayRender* events. The example will show how to :

- Display the old Norse day names in the header of the calendar (not something you normally see handled by internationalization).
- List the *SelectedDate* and *SelectedDates* property values.
- Override the user selected date and select a range of dates instead.
- Display the start and end dates of the current month view when the user navigates to a new month.
- Display a check mark on selected dates.

1. Create a new web application "CalendarEvents".
2. On the default page drop a RadCalendar control.
3. Copy "Check.gif" from the RadCalendar \projects\images folder to your project. Note: if the file is not actually included in the project, not just available in the project directory, the image will not be found.
4. Add statements to reference Text (supports StringBuilder), Telerik.WebControls, and Telerik.WebControls.RadCalendarHeaders. Note: A RadCalendarHeaders enumeration will be used to check the header type during HeaderCellRender.

C# Example:

```
using  System.Text;
using   Telerik.WebControls;
using    Telerik.WebControls.RadCalendarHeaders;

VB  Example:
Imports   System.Text
Imports    Telerik.WebControls
Imports     Telerik.WebControls.RadCalendarHeaders
```

5. In the page load set the default cell padding and spacing to provide enough space for long string names to display without visually running together. Also set the AutoPostBack property to "True" so the server side events have a chance to process.

```
C#  Example:
protected void  Page_Load(object  sender,  EventArgs  e)
{
    if  (!IsPostBack)
    {
          RadCalendar1.DefaultCellPadding = 5;
          RadCalendar1.DefaultCellSpacing = 5;
          RadCalendar1.AutoPostBack = true;
    }
}

VB  Example:
protected Sub  Page_Load(ByVal  sender As Object, ByVal e As  EventArgs)
 If Not  IsPostBack Then
    RadCalendar1.DefaultCellPadding  =  5
    RadCalendar1.DefaultCellSpacing  =  5
    RadCalendar1.AutoPostBack  = True
 End If
End Sub
```

6. Define an array of strings containing the old Norse day names:

```
C#  Example:
private string[]  _oldNorseDays  = new string[7] {
    "Sunnudagr",
    "Mánadagr",
    "Týsdagr",
    "Óðinsdagr",
    "Þórsdagr",
    "Frjádagr",
    "Laugardagr"
};

VB  Example:
Private  _oldNorseDays As String() = New String(7)  {"Sunnudagr",  "Mánadagr",  _
  "Týsdagr",  "Óðinsdagr",  "Þórsdagr",  "Frjádagr",  "Laugardagr"}
```

7. Define an indexer to the Norse day names array.

```
C#  Example:
private int  _columnIndex  = 0;

VB  Example:
Private  _columnIndex As Integer = 0
```

8. Define a handler for the HeaderCellRender event that only provides new behavior when the HeaderType is a column. The column index is used to access the correct Norse day name and populate the header cells text.

```
C# Example:
protected void RadCalendar1_HeaderCellRender(object sender,
       Telerik.WebControls.Base.Calendar.Events.HeaderCellRenderEventArgs e)
{
    if (e.HeaderType == HeaderType.Column)
    {
          e.Cell.Text = _oldNorseDays[_columnIndex];
          e.Cell.BackColor = System.Drawing.Color.WhiteSmoke;
          e.Cell.ForeColor = System.Drawing.Color.Blue;

         _columnIndex += 1;
    }
}

VB Example:
Protected Sub RadCalendar1_HeaderCellRender(ByVal sender As Object, _
  ByVal e As     Telerik.WebControls.Base.Calendar.Events.HeaderCellRenderEventArgs)
 If e.HeaderType = HeaderType.Column Then
    e.Cell.Text = _oldNorseDays(_columnIndex)
    e.Cell.BackColor = System.Drawing.Color.WhiteSmoke
    e.Cell.ForeColor = System.Drawing.Color.Blue
   _columnIndex += 1
 End If
End Sub
```

9. Define a handler for the DayRender event. The code here determines if the day being rendered is within the current month by comparing the "Day" argument with the FocusedDate of the RadCalendar. If the day is selected and within the current month a Image object is created to display a check mark and added to the cells Controls collection. The day number should also be displayed so it's added here as a LiteralControl. If the day is not selected then the day number is simply assigned to the cells Text property. Note; Remember that the cell text and controls assignment cancel one another out depending on the last one defined.

```
C# Example:
protected void RadCalendar1_DayRender(object sender,
       Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs e)
{
    bool isInMonth = e.Day.Date.Month == RadCalendar1.FocusedDate.Month;

    if (e.Day.IsSelected & isInMonth)
    {
         Image checkImage = new Image();
         checkImage.ImageUrl = "~/Check.gif";
        e.Cell.Text = "stuff";
         e.Cell.Controls.Add(checkImage);
        e.Cell.Controls.Add(new    LiteralControl(e.Day.Date.Day.ToString()));
    }
    else
    {
         e.Cell.Text = e.Day.Date.Day.ToString();
    }
}
```

```
VB Example:
Protected Sub RadCalendar1_DayRender(ByVal sender As Object, _
 ByVal e As Telerik.WebControls.Base.Calendar.Events.DayRenderEventArgs)
 Dim isInMonth As Boolean = e.Day.Date.Month = RadCalendar1.FocusedDate.Month
 If e.Day.IsSelected And isInMonth Then
   Dim checkImage As Image = New Image
    checkImage.ImageUrl = "~/Check.gif"
   e.Cell.Text = "stuff"
    e.Cell.Controls.Add(checkImage)
    e.Cell.Controls.Add(New LiteralControl(e.Day.Date.Day.ToString))
 Else
    e.Cell.Text = e.Day.Date.Day.ToString
 End If
End Sub
```

10.Define a handler for the SelectionChanged event. The handler does three things:

- Ignores the user selection and defines a range of selection dates, starting with "today" and extending five days.
- Displays the SelectedDate property value.
- Lists the SelectedDates collection.

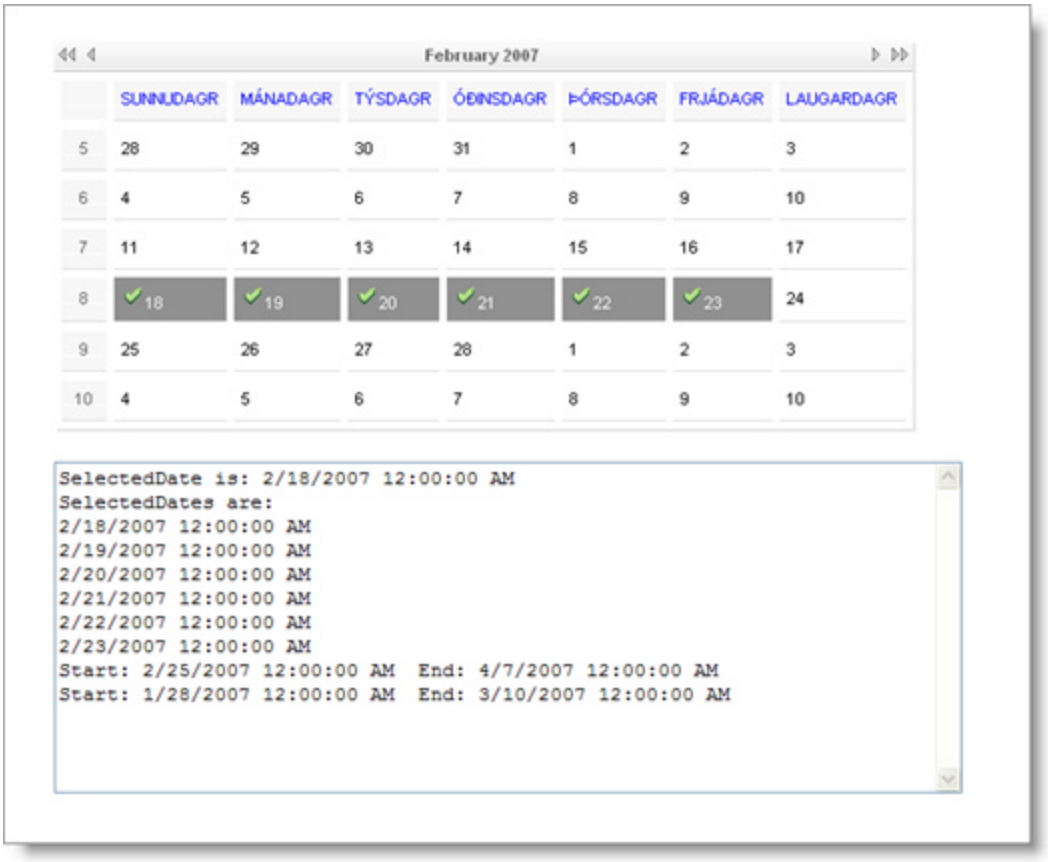
```
C# Example:
protected void RadCalendar1_SelectionChanged(object sender,
       Telerik.WebControls.Base.Calendar.Events.SelectedDatesEventArgs e)
{
       e.SelectedDates.SelectRange(DateTime.Now, DateTime.Now.AddDays(5));

     StringBuilder stringBuilder = new StringBuilder();
     stringBuilder.Append("SelectedDate is: " +
        (sender as RadCalendar).SelectedDate + "\n");

     stringBuilder.Append("SelectedDates are: \n");
    foreach (RadDate selectedDate in e.SelectedDates)
    {
          stringBuilder.Append(selectedDate.Date.ToString() + "\n");
    }

      TextBox1.Text += stringBuilder.ToString();
}

VB Example:
Protected Sub RadCalendar1_SelectionChanged(ByVal sender As Object, _
  ByVal e As Telerik.WebControls.Base.Calendar.Events.SelectedDatesEventArgs)
   e.SelectedDates.SelectRange(DateTime.Now, DateTime.Now.AddDays(5))
 Dim stringBuilder As StringBuilder = New StringBuilder
 stringBuilder.Append("SelectedDate is: " + _
    (CType(ConversionHelpers.AsWorkaround(sender, GetType(RadCalendar)), _
   RadCalendar)).SelectedDate + "" & Microsoft.VisualBasic.Chr(10) & "")
 stringBuilder.Append("SelectedDates are: " & Microsoft.VisualBasic.Chr(10) & "")
 For Each selectedDate As RadDate In e.SelectedDates
    stringBuilder.Append(selectedDate.Date.ToString + "" & _
   Microsoft.VisualBasic.Chr(10) & "")
 Next
 TextBox1.Text += stringBuilder.ToString
End Sub
```

11. Define an event handler for the DefaultViewChanged event. The DefaultViewChangedEventArgs passed in contain the NewView and OldView objects. This example displays the NewView start and end dates.

```
C# Example:
protected void  RadCalendar1_DefaultViewChanged(object sender,
       Telerik.WebControls.Base.Calendar.Events.DefaultViewChangedEventArgs e)
{
    const string message = "Start: {0}  End: {1}\n";
     TextBox1.Text += string.Format(message,   e.NewView.ViewStartDate,
         e.NewView.ViewEndDate);
}

VB Example:
Protected Sub  RadCalendar1_DefaultViewChanged(ByVal sender As Object, _
  ByVal e As     Telerik.WebControls.Base.Calendar.Events.DefaultViewChangedEventArgs)
 Const message As String = "Start: {0} End: {1}" & Microsoft.VisualBasic.Chr(10) & ""
  TextBox1.Text  +=  String.Format(message,  e.NewView.ViewStartDate,  _
    e.NewView.ViewEndDate)
End Sub
```

12. Run the application. It should be similar to the example below. Experiment with the selection and navigation. Notice that selecting any day will be overridden and the "today" date plus five days will be selected.

13. Stop the application. Comment out the line of code that calls SelectRange() and run the application again. Notice the order of the SelectedDates when the leading, ending and middle items are selected and de-selected.



The running CalendarEvents application

Picker Events

RadDatePicker has a SelectionDateChanged event surfaced in the designer, but the same effect can be achieved for RadTimePicker and RadDateTimePicker in code as shown in the example below.

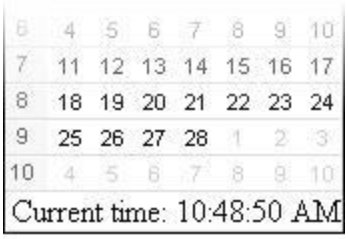
```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
      RadTimePicker1.SelectedDateChanged +=
        new     SelectedDateChangedEventHandler(RadTimePicker1_SelectedDateChanged);
      RadDateTimePicker1.SelectedDateChanged +=
        new     SelectedDateChangedEventHandler(RadDateTimePicker1_SelectedDateChanged);
}

void   RadDateTimePicker1_SelectedDateChanged(object sender,
     SelectedDateChangedEventArgs e)
{
     Log((sender as RadDateTimePicker).ID, e.OldDate, e.NewDate);
}

void   RadTimePicker1_SelectedDateChanged(object sender,
     SelectedDateChangedEventArgs e)
{
     Log((sender as RadTimePicker).ID, e.OldDate, e.NewDate);
}

private void Log(string controlName, DateTime oldDate, DateTime newDate)
```

```
{
    TextBox1.Text += "RadTimePicker -- Old date: " +
        oldDate.ToString() +
       "  New date: " +
        newDate.ToString();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  AddHandler RadTimePicker1.SelectedDateChanged, _
  AddressOf  RadTimePicker1_SelectedDateChanged
  AddHandler RadDateTimePicker1.SelectedDateChanged, _
  AddressOf  RadDateTimePicker1_SelectedDateChanged
End Sub

Sub  RadDateTimePicker1_SelectedDateChanged(ByVal sender As Object, _
  ByVal e As  SelectedDateChangedEventArgs)
   Log((CType(ConversionHelpers.AsWorkaround(sender,  GetType(RadDateTimePicker)),  _
    RadDateTimePicker)).ID, e.OldDate, e.NewDate)
End Sub

Sub  RadTimePicker1_SelectedDateChanged(ByVal sender As Object, _
  ByVal e As  SelectedDateChangedEventArgs)
   Log((CType(ConversionHelpers.AsWorkaround(sender,  GetType(RadTimePicker)),  _
    RadTimePicker)).ID, e.OldDate, e.NewDate)
End Sub

Private Sub Log(ByVal controlName As String, ByVal oldDate As DateTime, _
  ByVal newDate As DateTime)
 TextBox1.Text += "RadTimePicker -- Old date: " + oldDate.ToString + _
   " New date: " + newDate.ToString
End Sub
```

RadCalendar Templates

The objects in the header and footer templates can be accessed by using the FindControl() method of the RadCalendar.

```
C# Example:
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);

     Label lblFooter =  RadCalendar2.FindControl("lblFooter") as Label;
     lblFooter.Text = "Current time: "  +  DateTime.Now.ToLongTimeString();
}

VB Example:
Protected Overloads Overrides Sub OnPreRender(ByVal e As EventArgs)
  MyBase.OnPreRender(e)
 Dim lblFooter As Label = _
      CType(ConversionHelpers.AsWorkaround(RadCalendar2.FindControl("lblFooter"), _
    GetType(Label)), Label)
 lblFooter.Text = "Current time: "  +  DateTime.Now.ToLongTimeString
End Sub
```



Footer template area accessed at runtime

1.6.4 Client Scripting with RadCalendar

Client API

RadCalendar

The RadCalendar client api has methods for selecting and unselecting dates. This may be just a single date or a group of dates. You can optionally navigate to the selected date.

- SelectDate(date, navigate)
- SelectDates(dates, navigate)
- UnselectDate(date)
- UnselectDates(dates)
- GetSelectedDates()

Each day parameter is an array containing [Year][Month][Day]. To prepare one of these parameters you can declare:

```
var date = new Array(2007, 11, 28);
```

For methods that allow a "navigate" parameter this is simply a Boolean that navigates visually to the date if true. As with the other RadControls the calendar is accessed through the client id:

```
 <%= RadCalendar1.ClientID %>
```

Putting this all together in a call to set the date to 11/28/07 and show it in the calendar:

```
var date = new Array(2007, 11, 28);
 <%= RadCalendar1.ClientID %>.SelectDate(date, true);
```

To get all the dates that have been selected so far use GetSelectedDates(). This returns an array of three element arrays. Here is how you might iterate the results:

```
var dates = <%= RadCalendar1.ClientID %>.GetSelectedDates();
document.getElementById('TextBox1').value = "";
for(i=0; i<dates.length; i++)
{
   document.getElementById('TextBox1').value += dates[i][0] +
    "/" +
    dates[i][1] +
    "/" +
    dates[i][2] +
    "\n";
}
```

RadDatePicker

The RadDatePicker client api has methods to get and set the date, a IsEmpty property and methods to show and hide the popup:

- GetDate()
- SetDate(date)
- IsEmpty
- ShowPopup(), ShowPopup(x, y)
- HidePopup()

The SetDate() date is a JavaScript date object. The following code sets the date to Dec. 28, 2007 (The JavaScript date object month is zero based).

```
var newDate = new Date();
newDate.setFullYear(2007,11,28)
<%= RadDatePicker1.ClientID %>.SetDate(newDate);
```

RadTimePicker

The RadTimePicker client api has methods for getting and setting the time through the GetTimeView() function. Using the result of GetTimeView() you can call GetTime() and SetTIme() functions. The api also handles showing and hiding the popup.

- GetTimeView()
- ShowTimePopup()
- HideTimePopup()

Here is an example of accessing the time view:

```
var time = <%= RadTimePicker1.ClientID %>.GetTimeView().GetTime();
document.getElementById('TextBox1').value += time;
```

To show the popup use the ClientID property to access the control and call ShowTimePopup() :

```
<%= RadTimePicker1.ClientID %>.ShowTimePopup()
```

Client Events

RadCalendar

The main events for RadCalendar are:

- OnInit: Fires when the calendar has been constructed.
- OnLoad: When the page is loaded.
- OnDateSelecting: Before the selected date is added or removed from the dates collection. Cancel this event by returning false.
- OnDateSelected: After the selected date is added or removed from the dates collection.
- OnDateClick: Fired whenever a day cell is clicked, even if disabled.

- OnCalendarViewChanging: Before the viewable area rendered by the calendar is changed. Cancel this event by returning false.
- OnCalendarViewChanged: After the viewable area rendered by the calendar is changed.
- OnRowHeaderClick, OnColumnHeaderClick, OnViewSelectorClick: Fired when a calendar row header, column header or selector is clicked. Each of these events can be canceled by returning false.
- OnDayRender: This is the counterpart to the server side event that allows you to customize each day cells rendering.

Here is an example of the OnDateSelected event handler that passes an instance of the calendar and an object representing the date selected. This example sets a date picker date to the date selected in the calendar.

```
function Calendar_OnDateSelected(calendarInstance, renderDay)
{
   var year = renderDay.Date[0];
   var month = renderDay.Date[1] - 1;
   var day = renderDay.Date[2];
   date = new Date (year, month, day);

   if (renderDay.IsSelected)
  {
      <%= RadDatePicker1.DateInput.ClientID %>.SetDate(date);
  }
}
```

RadDatePicker, RadTimePicker

The events for RadDatePicker and RadTimePicker are:

- OnDateSelected: Fires whenever the selected date is changed.
- OnPopupUpdating: This fires just prior to synchronizing the date in the internal calendar and date input.
- TypingTimeOut: The time in milliseconds to expire after the last keystroke to the date input prior to firing the OnDateSelected event.

Here is an example of handling OnDateSelected. The "e" argument parameter has properties for NewDate and OldDate where NewDate is the newly selected date. Here we create a date array and use it to select the date in a calendar.

```
function DatePicker_OnDateSelected(sender, e)
{
   var tripletDate = new Array(e.NewDate.getYear(), e.NewDate.getMonth() + 1,
     e.NewDate.getDate());
    <%= RadCalendar1.ClientID %>.SelectDate(tripletDate, true);
}
```

Lab: Client Scripting RadCalendar and RadDatePicker

For responsive access to actions that don't require a trip back to the server, the client event OnDateSelected provides instant feedback and control. OnDateSelected is defined for both calendar and date input controls.

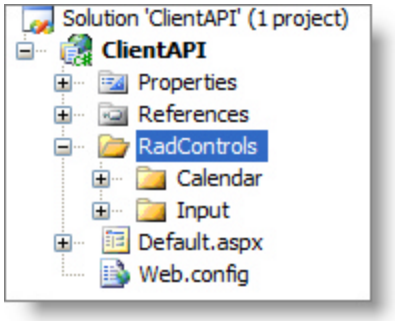
This lab will demonstrate:

- Handling calendar and date picker events. The event handlers show selecting and clearing dates.
- Converting between JavaScript and the RadCalendar internal "triplet" date array type.
- How to disconnect events dynamically to prevent recursion.
- While we have different UI elements available on the screen the lab will also show how to apply skins to different parts of the calendar and date picker.

The user will be able to click a month on the stand alone calendar and the selected day will show up in the date picker and label below. If the day has already been

1. Create a new web application project "ClientAPI".
2. Create a RadControls directory in the project and copy the Calendar and Input folders from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls



The project with RadControls setup.

3. Copy the html found below to just Inside the form tag for the default page. Look at the RadCalendar declaration and notice:
   - The calendar *Skin* property is "Inox"
   - The calendar ClientEvents property defines the OnDateSelected event.
   - The RadDatePicker actually has two skins defined. The DateInput portion has a "Macromedia" skin and the internal calendar uses "WebBlue".
   - The date picker client events defines OnDateSelected.
   - The date picker defines an *EmptyMessage* and sets the *HideOnBlur* property to "True". When the project runs the *EmptyMessage* will display when the date has no value.

```
. . .
 <radCln:RadCalendar ID="RadCalendar1" runat="server"
  Font-Names="Arial, Verdana, Tahoma"
  ForeColor="Black"
   Style="border-left-color: #ececec; border-bottom-color: #ececec;
   border-top-color: #ececec; border-right-color: #ececec"
  Skin="Inox">
   <ClientEvents OnDateSelected="Calendar_OnDateSelected" />
 </radCln:RadCalendar>
 <br />
 <br />
 <radCln:RadDatePicker ID="RadDatePicker1" runat="server">
```

```
        <DateInput CatalogIconImageUrl="" Description=""
          DisplayPromptChar="_" PromptChar=" "
          Title="" TitleIconImageUrl="" TitleUrl=""
         Skin="Macromedia"
          EmptyMessage="Input Date Here"
          HideOnBlur="True">
       </DateInput>
         <ClientEvents OnDateSelected="DatePicker_OnClientDateChanged" />
        <Calendar Skin="WebBlue">
       </Calendar>
      </radCln:RadDatePicker>
      <br />
      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>
  </div>
</form>
```

4. Inside the form and div tags place the beginning and ending JavaScript tags:

```
<script   type="text/javascript">
//<!--

// your code here...

//-->
</script>
```

5. Inside the JavaScript tags place the event handler for the calendar OnDateSelected event:

```
function   Calendar_OnDateSelected(calendarInstance,   renderDay)
{
   var year = renderDay.Date[0];
   var month = renderDay.Date[1] - 1;
   var day = renderDay.Date[2];
   date = new Date (year, month, day);

   <%= RadDatePicker1.ClientID %>.OnDateSelected = null;
   if (renderDay.IsSelected)
  {
      <%= RadDatePicker1.DateInput.ClientID %>.SetDate(date);
  }
  else
  {
      <%= RadDatePicker1.DateInput.ClientID %>.Clear();
  }
    <%= RadDatePicker1.ClientID %>.OnDateSelected = DatePicker_OnDateSelected;
}
```

6. Examine the JavaScript for the following:

- The selected date is in a three element array holding [year][month][day]. The first lines of code convert that to a JavaScript date object usable by a date input SetDate() function.
- Notice how the OnDateSelected is set to null, operations are performed on the date picker internal DateInput and finally the OnDateSelected is reassigned. This step prevents the OnDateSelected from firing while the value is being changed. This is important because the OnDateSelected in turn makes date selection changes to the calendar.
- The renderDay argument has a Boolean IsSelected property that reflects the calendar selection in the UI. If IsSelected is true then we set the date input to the calendar date, otherwise the date is cleared.

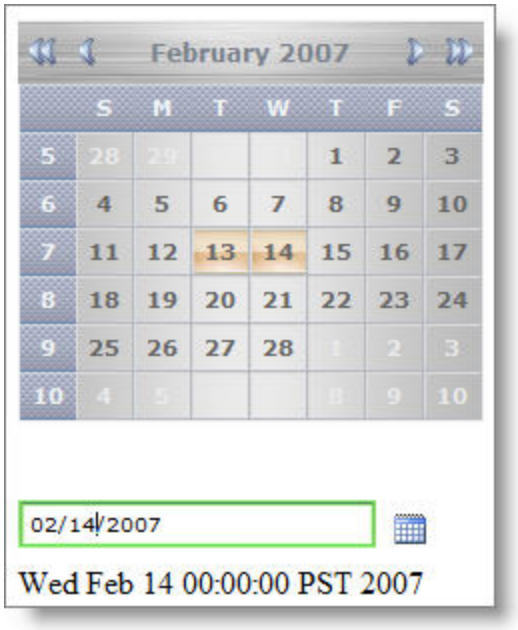
7. Add a second event handler for the date picker OnDateSelected

```
function   DatePicker_OnDateSelected(sender,   e)
```

```
{
    document.getElementById('Label1').innerText  =  e.NewDate;

   <%=  RadCalendar1.ClientID  %>.OnDateSelected  =  null;
  var  tripletDate  =
      new  Array(e.NewDate.getYear(),  e.NewDate.getMonth()  +  1,  e.NewDate.getDate());
   <%=  RadCalendar1.ClientID  %>.SelectDate(tripletDate,  true);
   <%=  RadCalendar1.ClientID  %>.OnDateSelected  =  Calendar_OnDateSelected;
}
```

The pattern is similar to the previous event handler but this time the calendar OnDateSelected is momentarily unhooked and the JavaScript date is converted back to an array.

8. Run the Application. Notice the two second time lag between entry to the date input and display to label and calendar. This is controlled by the date picker ClientEvents.TypingTimeOut property that by default is 2000 milliseconds. Try selecting and un-selecting dates in the calendar. Also try entering to the date picker in the input and also using the popup calendar.



The application showing selected dates

1.6.5 Summary

In this section you became familiar with how the RadCalendar, RadDatePicker, RadTimePicker and RadDateTimePicker are layed out and how the properties are organized in each control. You learned to customize the RadCalendar by defining special days, adding templates and handling rendering events, You also worked with some of the important client side events and methods.

Day

2 Day 2

2.1 RadGrid

2.1.1 Getting Started

To say that the number of property settings available in the RadGrid is amazing would be an understatement - there are around 1200 settings in the properties that can be set, which control the myriad of ways the RadGrid control can be configured to display your data (and that's **before** you get into the settings for any detail tables you might want to embed into your grid's display). Here's a quick peek at a sample master table with two detail tables from the Telerik help files:

RadGrid with two detail levels displayed

One of the first things that should strike you in looking at this image is that the detail tables are embedded right inside the RadGrid - no more trying to juggle your screen real estate to find a reasonable way to lay out your tables. Plus, with the detail expansion features of the RadGrid, you can drill right in to the data that interests you, always keeping the context of the detail data displayed immediately above. The next thing you'll notice is that the master and detail tables all have a different color scheme available to them, so not only do you get a visual cue from the indentation level of the detail tables, but you also have a different appearance - and this extends to every different facet of the appearance, from header, item and alternating item styles, to the font used to display the text or data.

The good news is, a large portion of the 1200 settings are related to various facets of the grid's appearance, so mastering just one set of appearance features covers a large amount of territory. The more involved parts relate to the behavior of the grid and how to configure data, so that's where we'll spend most of our time in this section.

2.1.2 Using RadGrid in the Designer

Let's get started, then. The easiest way (and the way that you will almost certainly use most often) to use the RadGrid component is in the Visual Studio designer. There, the properties are readily available for setting, and there are several wizards available to guide you. Another advantage of developing in the Visual Studio designer environment is that your changes are reflected immediately on the design surface, giving you instant feedback about how your settings will appear.

You'll find the RadGrid component in the Telerik controls section of your Visual Studio toolbox like this:

RadGrid in the toolbox

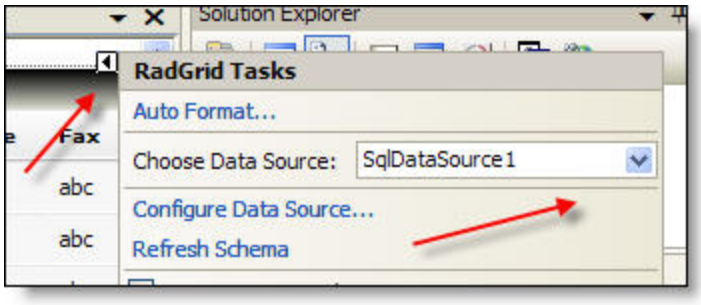
Lab: Connecting to a Data Source

In this lab you will learn to connect to a database and select the columns you want to display from a single table.

1. Open up a new file-system based website named RadGrid_Lab_Exercises. Delete the default.aspx page, add an aspx page named DataSourceConnection, and place a RadGrid component onto your default.aspx page.

2. Next, add SqlDataSource with a connection to the Northwind database, selecting all columns from the "Customers" table. Order the columns by CustomerId.

3. Connect the grid to the data source. You may do this either by clicking on the grid's smart tag found by default in the grid's upper-right corner (pictured below) and then selecting the data source you just configured; or by locating the "DataSourceID" property in the grid's property sheet, and selecting it there.



Grid DataSource connection

4. Run your project. This is as basic as it gets - you should have the entire Northwind customer table with all columns displayed on one scrollable page.

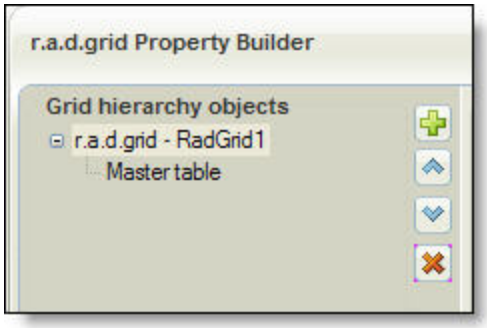
There's nothing fancy here, and you can imagine how awkward it would be to manage if there were dozens of columns or hundreds of rows of data, so in the next lab we're going to take a little more control over what's displayed to us and how.

Lab: Controlling paging and item style

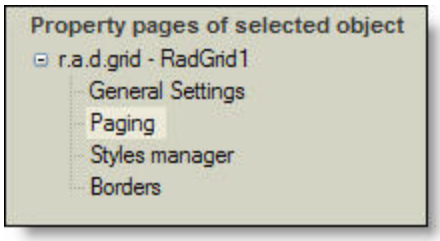
In the last lab, we saw how easy it was to establish the basic display of data in a RadGrid component, but it was about as exciting as reading a telephone directory. Most of the time, the tables you're dealing with will contain a lot more information than the Northwind customers table. Not only will you want to limit the columns of data displayed, and the number of rows displayed at one time, but you'll also want to present the data in a more visually appealing way - at the very least, one that will make for easier

reading of the data. That's just what we'll do in the next lab.

1. Open your RadGrid's smart tag again. This time, select the property builder. The first thing we'll do is control the number of items we display on each page. When you first open the property builder, you should see a display similar to the image on the left, with the top entry (for the grid) selected. Below it, you'll see an area where you can select one of four different property pages for the grid (pictured on the right). Select the "Paging" property page.



Property Builder Object Selection



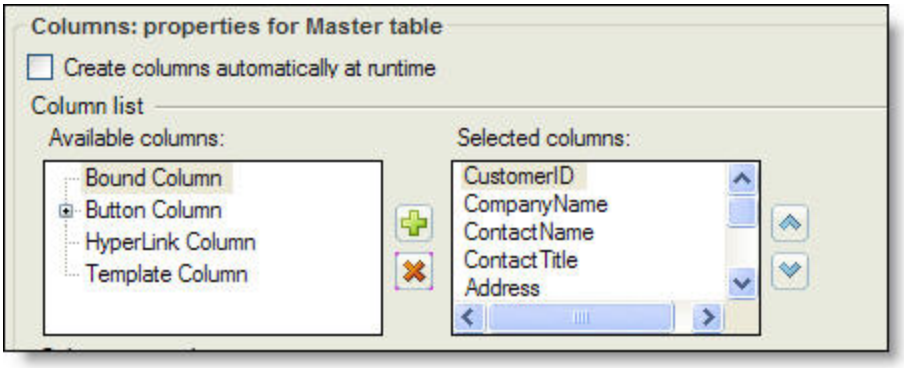
Property Page selector

2. On the "Paging properties" sheet, check the "Allow Paging" box, and accept the default of 10 rows. In the "Navigation Mode" dropdown, select the "Next, Previous and Pages" entry.

3. Select the "Styles Manager" property page. Notice that as you navigate away from the "Paging Properties" page, you'll be asked if you want to apply the changes you've made so far. This isn't really a choice - if you click "Cancel", you'll stay on the same property page. The only way to leave this property page is to apply the changes, or to cancel your changes altogether (by clicking "Cancel" at the bottom of the screen).

4. The styles manager page is where you'll be able to make the items in your grid a little more readable, by setting a different style for alternating rows. In the Styles Manager tree view, expand the "Items" node, and select the "Alternating Items" entry. One of the easiest ways to distinguish your entries from one another is to vary the background of alternating rows, so click on the button with the ellipses ("...") next to the BackColor dropdown, and select a distinctive color that is light enough to still allow you to see the main (black) text (of course, you have a lot of flexibility here to do any imaginable color scheme, but we're going to keep it simple for the purposes of the lab). I chose "Pale Turquoise", at the bottom of the third column from the left on the "Named Colors" tab.. Click "OK" to apply your changes.

5. The final adjustment we'll make in this section of the lab is to limit the columns that will be displayed in the grid. Select the "Master table" entry in the object selection area, and the "Columns" property page (which is not visible when only the grid is selected). You will see a columns manager which is very similar to the same manager in the out-of-the-box Visual Studio GridView property builder. It looks like this:



Grid Column Properties

Immediately below the columns manager is a "Column properties" setting sheet. We mention it here only to bring it to your attention, and we'll return to it later.

On our grid, we'd like to display only the CustomerId, CompanyName, ContactName, ContactTitle, Country, and Phone fields. To remove the other fields, highlight each one in turn, and click on the red "X" between the two boxes. (If you decide that you've mistakenly eliminated displayed column, you can select the "Bound Column" entry in the left-hand list, click the green "+" icon in the middle, and then enter the appropriate DataField and HeaderText in the property sheet).

TIP: If you're just experimenting with how you want things to appear, before actually deleting the columns, go to the "behaviors" section and set the Visible property to false. This will cause the column not to be displayed, but makes it a whole lot easier to restore the column if you decide to do so.

6. Run your project again. Notice how much easier it is to distinguish between the rows of data from the table. Use the paging to navigate around the table data.

The only thing that's really awkward here is the postback click-and-flash - but the RadGrid component can address that quite handily.

7. Stop your project from running by closing the browser. Open up the RadGrid smart tag again, and check the "Enable AJAX" box located at the top of the "General Features" section. Run your project again, and click your navigation buttons again.

It is so very doggone cool that you can get javascript paging with the click of a mouse, (At this point, insert your own word of choice to express wonder and amazement. My own personal favorite is "Shezam!")

Lab: Scrolling, Grouping, Sorting and Filtering

One powerful method of presenting and viewing data is the ability to group, sort or filter the information; another important capability is to control the user experience and interaction with the data display using a scrolling area. The RadGrid control makes an easy provision for each of these. In this lab, you'll discover just how easy it is to add these features to your application.

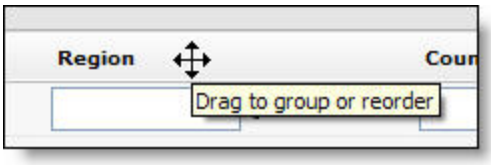
1. If your browser is open, close it now.

2. On the RadGrid smart tag, make sure that the grouping, sorting and filtering boxes are checked. These properties are less conveniently available on the property sheet - AllowFilteringByColumn and

AllowSorting are in the "Behavior" section, while GroupingEnabled is in the Grouping section.

3. Run your project.

4. Selection of the column used to group your data can be completely dynamic. Hover your mouse over the column to group by (in this exercise, the only column that really makes sense to group on is the "Country" column). Notice as you are moving your mouse that there are two separate hot-spot areas available for selection, one for grouping and one for sorting (the sorting area is basically the area of the column heading text, while the area surrounding the text can be dragged to group the data, as explained below). You will also notice that the cursor displayed is different for each of the areas:



Grouping HotSpot

Sorting HotSpot

With the cursor in the country column's drag-to-group area, click and hold your left mouse button. This will spawn a semi-transparent label with the "Country" legend (pictured below on the left), which can be dragged and placed onto the grouping bar, which in the default configuration is the dark grey bar at the top of the grid.

TIP: Having a good aim is important here - the point of your cursor needs to be between the top and bottom of the grouping panel. If you're even just slightly outside of those bounds, no grouping will be added.

Dragging a Column Header to Group Information

Grouped Columns

When the grouping has been dropped successfully, the opaque label with the name of the column selected for grouping will be displayed at the top left corner of the grid, as displayed in the right-hand figure above. As soon as the grouping parameter is established, the grid display will be altered, and an expand/collapse control will be visible in the left-hand column of the grid adjacent to the beginning of each new grouping.

5. RadGrid also permits you to establish multiple grouping criteria. Drag the "Region" column onto the grouping bar. If you're using the standard Northwind database with the settings we have suggested, you'll need to scroll down to Canada (on page 2) to see a good example of this multi-level grouping. There, you'll something similar to this:

Grid Multi-Level Grouping

As you can see, each level of grouping has its own expansion control. Another nice feature is that when the grid is displaying grouped items that have group siblings displayed "off the edge" of the grid (either top or bottom), there is an advisory to that effect displayed on the grouping control line.

6. Another feature of the grouping control is the ability to specify sort direction for the column. Click on the direction arrow at the right-hand side of the "Country" label on the group, and you'll see the grid's contents reorder themselves on the corresponding column data. Sorts can be specified as either ascending or descending.

TIP: One nice aspect to using the grouping feature for sorting is that you can sort on multiple columns simultaneously, whereas using the column headers alone only permits you to select a single column for sorting at a time. You should also know that when there are grouping items in the grouping panel, the sort controls at the top of each column are still active for any column not contained in the grouping control.

7. Remove all grouping by dragging any items in the group panel back onto the column header area. Now, click on the text of the column header to activate sorting from there. Notice that you are able to select only one column at a time for sorting.

The last part of this exercise deals with using the RadGrid filtering feature. There are two parts to filtering using the RadGrid, namely the **filter function** and the **filter value**. The filter value merely specifies what it is that you are looking for (or not looking for, in the case negative functions like "Does Not Contain"). The function is specified by clicking on the filter icon adjacent to the filter field.

8. In the "Country" filter (at the top of the Country column), enter one of the countries from the table - Brazil, for instance.

9. Click on the filter icon. While the selection list is open, inspect the different options that are available to you here. Select the "EqualTo" option. The display will change, displaying the information for Northwind's nine Brazilian customers.

10. Change the contents of the "Country" filter text box to Germany. Nothing happens. That's because the client-side event that triggers the filtering action is the selection of a filter function from the functions dropdown list. Even if all you do is open the list and re-select an element of the list, you'll need to do this to get your display refreshed.

Each of the filter value selections for the various columns is available for setting a filter. When multiple filter conditions are specified. each and every condition must be met for the data to be displayed.

11. In the "Contact Name" filter, enter "Pe" and select the "Starts With" filter function. You should see

one entry in your grid, for "Peter Franken".

The final feature we'll explore in this section is Scrolling. There are circumstances in which the most convenient way to approach data display is to have the table rows shown in a designated area of the screen, and to permit the user to scroll through the data while keeping other screen components visible in a static display.

12. If your application is still running in the browser, close it.

13. Reopen the grid's smart tag, check "Enable Scrolling" and un-check the "Enable paging" box. (Paging and scrolling are mutually exclusive notions, to the extent that the number of rows displayed on the grid will fit inside the grid's allocated display area. For instance, setting the page size to 10 rows will permit all rows in a page to be displayed without the need for scrolling, so no scroll bars will be visible. Setting the page size to 40 rows, on the other hand, does display scroll bars for the 40 items, and to see the next 40 items, you'd need to navigate to the next page. For this exercise, we'll simply disable paging).

14. Apply the settings and run the project. You should now have a grid with nicely-behaved scrolling, which keeps your header columns, filters and grouping panel displayed in a static position at the top of the grid, while the data is capable of being scrolled, grouped, sorted and filtered in an AJAX enabled fashion.

That concludes this lab exercise. You should now be familiar with the essentials of connecting to a data source, and managing basic data display and navigation control using the RadGrid component.

2.1.3 Using the RadGrid at Runtime

Lab: Adding detail information

One of the bread and butter uses of grids is a parent-child or master-detail view of data. Naturally, Telerik grids have this same capability, but the RadGrid control takes things a step further by embedding the detail table(s) right inside the main grid. We'll take a look at how to accomplish that in this section by adding the Northwind orders and order details to our existing display of customer information.

1. Close your browser if it's running.

2. Add a SqlDataSource to your project. You'll also connect this to the Northwind database, but to the Orders table. Select all columns from the table.

3. In the properties editor for the grid, expand the "Layout" section (if you're sorted by category).

4. Expand the "MasterTableView" entry, and open the "DetailTables" collection editor. Click on "Add" to add a new detail table.

5. In the "Data" section, enter "SqlDataSource2" as the DataSourceID, and "OrderID" as the DataKeyName.

6. Scroll down a bit, and in the Hierarchy section, open the "ParentTableRelations" editor. Add a new relation - the MasterKeyField and DetailKeyField should both be set to "CustomerId".

7. Close the DetailTables collection editor, and select SqlDataSource2. Open the SelectQuery command editor.

8. The select command needs to be limited to returning just the customer we have selected in the RadGrid control, so change the select command so that it reads as follows: "SELECT * FROM [Orders] where CustomerId = @CustomerID".

9. Click the "Add a Parameter" button. When the new entry is created, set the name of the parameter to "@CustomerID". For the parameter source, select "Session", and for the SessionField entry, enter "CustomerID". Close the editor.

10. Back on the main design surface, select the RadGrid control. In the properties window, switch to the Events pane, and double-click on the "ItemCommand" event. Add the following code to the prototype event:

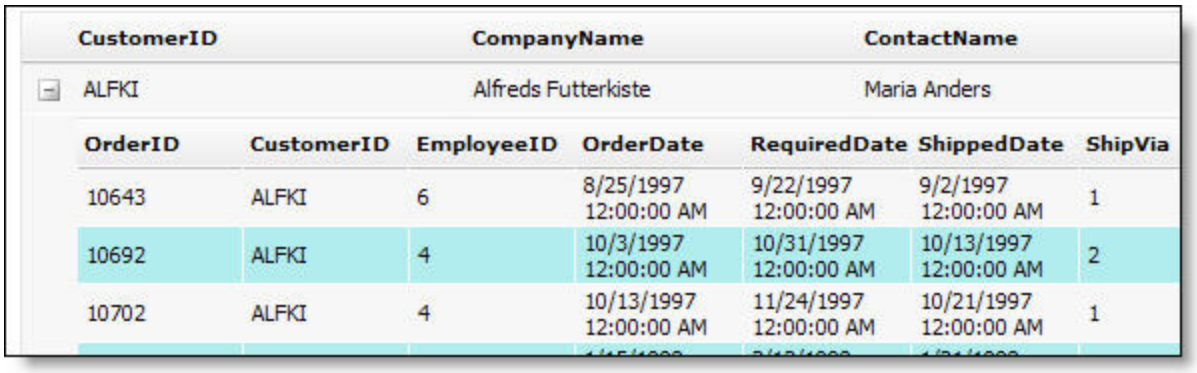
```
C# Code:
if ((string)e.CommandName == "ExpandCollapse")
   Session["CustomerID"] = e.Item.Cells[2].Text;

VB Code:
If e.CommandName = "ExpandCollapse" Then
    Session("CustomerID") = e.Item.Cells(2).Text
End If
```

Note: There is an absolute reliance in the code snippet above on the CustomerId being available in the second cell. This is not advocated as good programming practice, it merely fulfills the lab objective. Or in other words, don't drink and program - uh, I mean, please program responsibly. <hic>

11. In the RadGrid's smart tag, enable paging, and disable filtering, scrolling, sorting and grouping. Run your project. What you should have now is a grid which still displays the customer data - but now, there is an expansion control in the far left column:

Grid with Expansion Control

When you click the expansion button for the selected customer, the ItemCommand is sent back (via Ajax - you've still got Ajax enabled, don't you?) and determining that you've triggered the ExpandCollapse event, the application sets the value of the session variable "CustomerId" to that of the customer just selected. Next, the DetailTable containing the Orders information is instantiated, and it makes use of the Session variable to select only those orders for the selected customer. The result is a RadGrid displaying a slightly indented detail table with the orders for your customer, like this:



Grid expanded with order details

The next question you may have is "That's nice, but can I display the order details as well?" Thanks for asking.

12. Close your browser. Drop yet another SqlDataSource onto the design surface. Configure it to connect to the Northwind database, and specify the following SQL statement to retrieve the data: "Select * from [Order Details Extended] where OrderID = @OrderID". On the parameter definition page, set the Parameter source to "Session", using the "OrderID" SessionField.

13. Select the RadGrid on the design surface. In the property editor, select the "Layout -> Master Table View -> DetailTables" collection editor like you did above. This time, however, go to the "Hierarchy" section and select the "DetailTables" collection editor at that location, to open up a second GridTableView Collection editor.

14. Add a new member, setting the DataSourceID to "SqlDataSource3", and the DataKeyNames to "OrderID,ProductID" (if you use the editor, be sure to enter each field on a separate line). Close both collection editors.

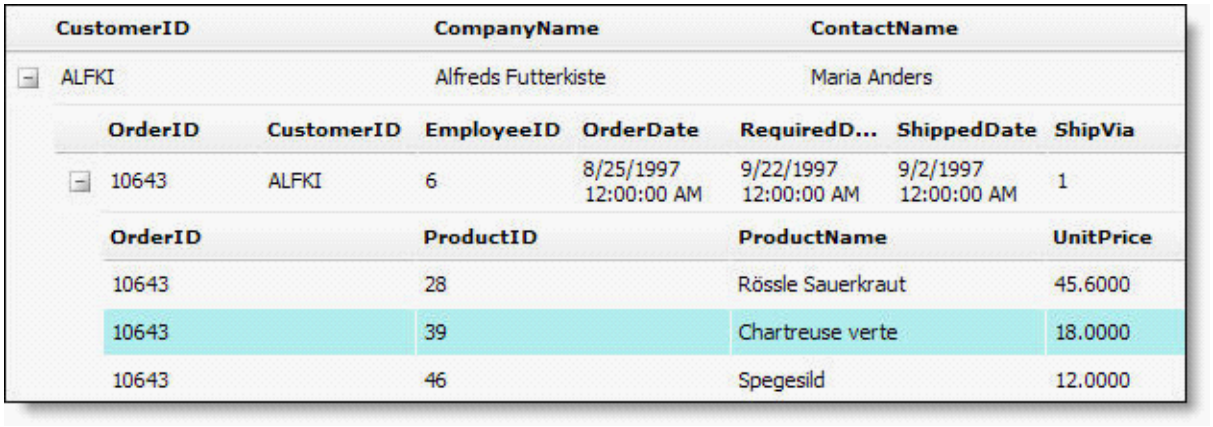
15. Because there is now a conflict between which table within the RadGrid is triggering the ItemCommand event, you need to get a bit more sophisticated in the event handler. Undoubtedly there are a number of ways to determine which of the tables was clicked by the user; the way we'll use in this example is to inspect the KeyValues of the grid item that was clicked, by using the following code, which you should use to replace what you've currently got in the ItemCommand event handler:

```
C# Example:
if ((string)e.CommandName == "ExpandCollapse")
{
  string keyValues = ((GridEditableItem)e.Item).KeyValues;
  if (keyValues.Contains("CustomerID"))
     Session["CustomerID"] = keyValues.Substring(13, keyValues.Length - 15);
  else
     Session["OrderID"] = keyValues.Substring(10, keyValues.Length - 12);
}

VB Example:
If e.CommandName = "ExpandCollapse" Then
 Dim keyValues As String = CType(e.Item, GridEditableItem).KeyValues
 If keyValues.Contains("CustomerID") Then
    Session("CustomerID") = keyValues.Substring(13, keyValues.Length - 15)
 Else
    Session("OrderID") = keyValues.Substring(10, keyValues.Length - 12)
 End If
End If
```

Also, notice that you are introducing a specific reference to a GridEditableItem in the code. For this to work properly, you must reference the appropriate namespace. Add the following near the top of the code:

```
C# Code:
using Telerik.WebControls;

VB Code:
Imports Telerik.WebControls
```

16.Run your project again. This time, you're able to display three different levels of detail:



Grid displaying three levels of detail

In this section of the lab, you've learned how to hook up and display multiple levels of tables. You probably noticed when you had the TableCollection editor open that it looked like you could probably keep on adding detail tables to detail tables, and you're correct - you can add detail to any reasonable level. You also probably noticed that there is the capacity to add more than one detail table to the data collection editor at every level. Correct again - other detail tables in the same collection will be displayed below their parent, at the same level of indentation as the sibling table (that's why there are up/down arrows provided in the collection editor, so that you can change the order of display of the tables if desired).

That's a tremendous amount of functionality - and so far, we've just displayed the data. Later, we'll take a look at editing what we see.

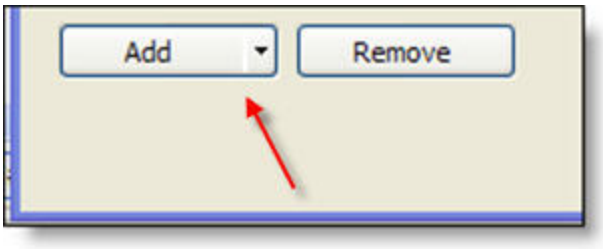
Lab: Displaying Totals in Grid Footers

Back in the last lab ("Adding Detail Information"), you may have noticed that although we got a nice display of all customers, orders, and details, there were no totals displayed. RadGrid provides an easy and convenient way to display the sum of a column of numbers in the grid footer.

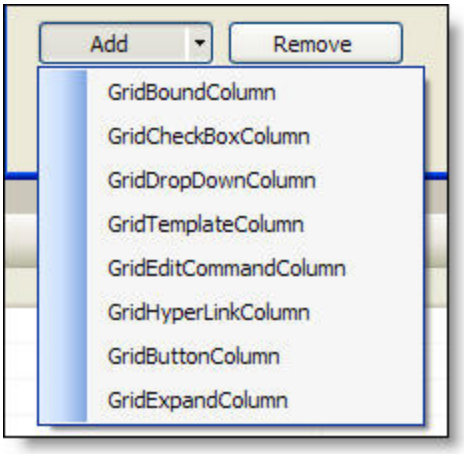
We'll start with the code where we left it in that lab.

1. Open the property sheet for the RadGrid component. In the MasterTableView section, select the lowest-level detail table (the one for the order details), and open the columns collection editor.

2. Again under the "MasterTableView" heading, open the "Columns" collection editor. Add a new GridTemplateColumn. **Notice the "Add" control:** there is a faint division line between the word "Add", and the down arrow on the button.



Grid ColumnCollectionEditor Add Button



New GridColumn Selection

Down arrow on a button? Click the arrow, and you'll get a list of eight different button types to select from. Each of the button types has a different set of properties that you can set, so the properties displayed in the property sheet will depend on the type of button you select here. The default is a GridBoundColumn. However, for the lab requirement, you'll need to select the GridTemplateColumn entry. All of the defaults for this column type will be appropriate for this exercise. Give it the unique name "DetailExtensionColumn", with header text "Extension".

3. Close the columns editor. On the third-level table, set the "ShowFooter" setting to true.

4. Close all popup editors. In the events section of the RadGrid property sheet, double-click the "ItemDataBound" event to create the prototype. Add the following code:

C# Code:

```
double  totalOrder;

protected  void  RadGrid1_ItemDataBound(object  sender,  GridItemEventArgs  e)
{
      if  (e.Item.OwnerTableView.DataSourceID  ==  "SqlDataSource3")
     {
    if  (e.Item  is  GridDataItem)
        {
              GridDataItem dataItem = e.Item as GridDataItem;
              double fieldValue = int.Parse(dataItem["Quantity"].Text) *
             double.Parse(dataItem["UnitPrice"].Text);
            dataItem["DetailExtensionColumn"].Text
           = fieldValue.ToString("c");
              totalOrder += fieldValue;
        }
         if (e.Item is GridFooterItem)
        {
              GridFooterItem footerItem = e.Item as GridFooterItem;
            footerItem["Quantity"].Text
          = "Total order: "  +  totalOrder.ToString("c");
        }
     }
}

VB Code:
Dim  totalOrder As Double

protected Sub   RadGrid1_ItemDataBound(sender As Object, e As  GridItemEventArgs)  _
    handles  RadGrid1_ItemDataBound
```

```
Begin
    If  (e.Item.OwnerTableView.DataSourceID  =  "SqlDataSource3") Then
    Begin
   If (e.Item is GridDataItem) Then
       Begin
            GridDataItem dataItem = e.Item As GridDataItem
           Dim fieldValue As Double = int.Parse(dataItem("Quantity").Text) * _
             double.Parse(dataItem("UnitPrice").Text)
             dataItem("DetailExtensionColumn").Text
          = fieldValue.ToString("c")
            totalOrder += fieldValue
       End
       If (e.Item is GridFooterItem) Then
       Begin
            GridFooterItem footerItem = e.Item As GridFooterItem
            footerItem("Quantity").Text
          = "Total order: " + totalOrder.ToString("c")
       End
    End
End
```

Because the ItemDataBound event gets raised at whatever level of the RadGrid's tables are getting bound, we need some kind of check to make sure that we're dealing with the third-level table; otherwise, we'll get an error when trying to locate the "Quantity" field in the data item. Notice also that we declare a variable to hold the order total OUTSIDE the method call, so that we can accumulate the total for all rows of the detail (since the ItemDataBound event will be raised once for each row).

5. Run your project. You should now see the footer displayed with the extended amount for each detail line displayed on the left-hand side, and the total of all detail lines for each order displayed in the footer.

Lab: Editing data in-place

One very convenient way to edit data is to view it in the context of all the data surrounding it - that is to say, right inside the RadGrid table. In this section, we'll take the steps necessary to put the table into a simple edit mode, update the data, and write the updated data back to the database.

Let's take the cheap ride first - and when I say the cheap ride, we mean that it costs you almost nothing to get from where you are to where you'd like to be: sitting in front of an application that lets you view, edit and update data.

1. In design mode, open the smart tag for SqlDataSource1, and select "Configure Data Source". You've already specified the connection string a long time ago, so click "Next" to move to Select statement configuration.

2. On the "Configure the Select Statement" screen, click the "Advanced" button, and in the helper screen that pops up, select the checkbox that generates insert, update and delete commands for you. In this very simple one-user lab application, there's really no need to check the "Use Optimistic Concurrency" box. In a production environment, you'll want to take additional precautions to avoid concurrency issues. Click "OK", "Next" and "Finish" to update your settings to the .aspx source page.

3. Next, you need to put the grid row you wish to edit into edit mode. In the RadGrid's property sheet, expand the "MasterTableView" entry. Set the "EditMode" entry to "InPlace". This setting tells RadGrid to provide very basic (but still entirely functional) editing capabilities for the data. Also, in order to actually accomplish the update, you should set the "AllowAutomaticUpdates" property to true.

TIP: You can set the "AllowAutomaticDeletes", "AllowAutomaticInserts", and "AllowAutomaticUpdates" properties in several different places. If you set them at the top-most (RadGrid) level, they will cascade to the lower levels. Alternatively, they may be set individually on the MasterTableView, or on any of the detail views.

4. Finally, you need to provide a means of activating the edit mode. Again under the "MasterTableView" heading, open the "Columns" collection editor. To activate the edit mode, you'll need to add a GridEditCommandColumn entry, so select that from the "Add" dropdown button. All of the defaults for this column type will be appropriate for this exercise; the only thing you'll probably want to do is to move the button to the left-hand side of the grid, so make sure the newly added button is selected, and use the arrow keys to move it to the top of the listing.

5. Run your project. Select a row for editing by clicking on the "Edit" button. Change the data in one of the fields, and click the "Update" (back where you found the "Edit" link to begin with.

Too easy! We told you this was the cheap ride - except for repositioning the edit column, that cost you 15 mouse clicks to add complete edit functionality to your application.

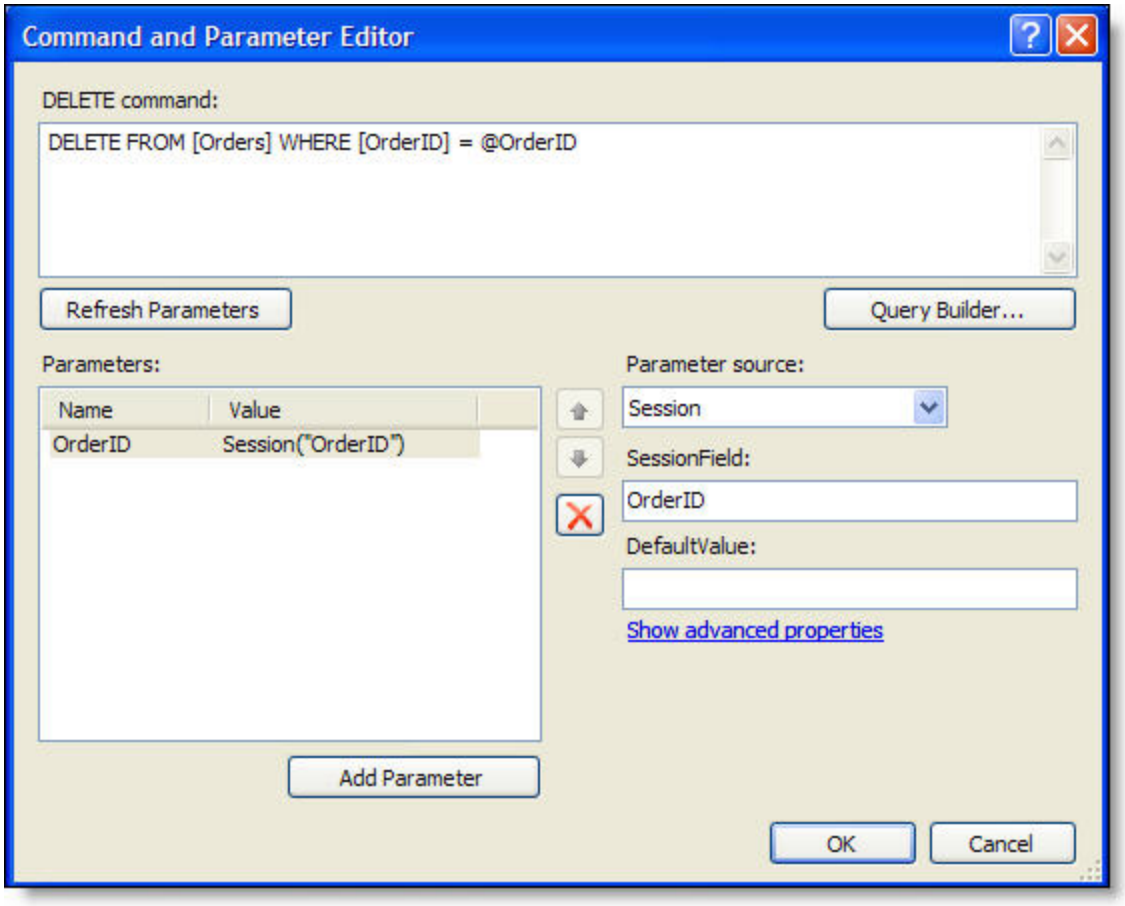
6. Configuring your table to delete data is only slightly more complicated. Since you already configured the delete statement along with the insert and update statements back in step 2, your first need is to provide a way to activate data deletion. To do this, you'll need to open up the Columns collection editor again. This time, add a GridButtonColumn. Set both the Text and the CommandName properties to "Delete", and close the column collection editor by clicking "OK".

7. Set the "AllowAutomaticDeletes" property to true.

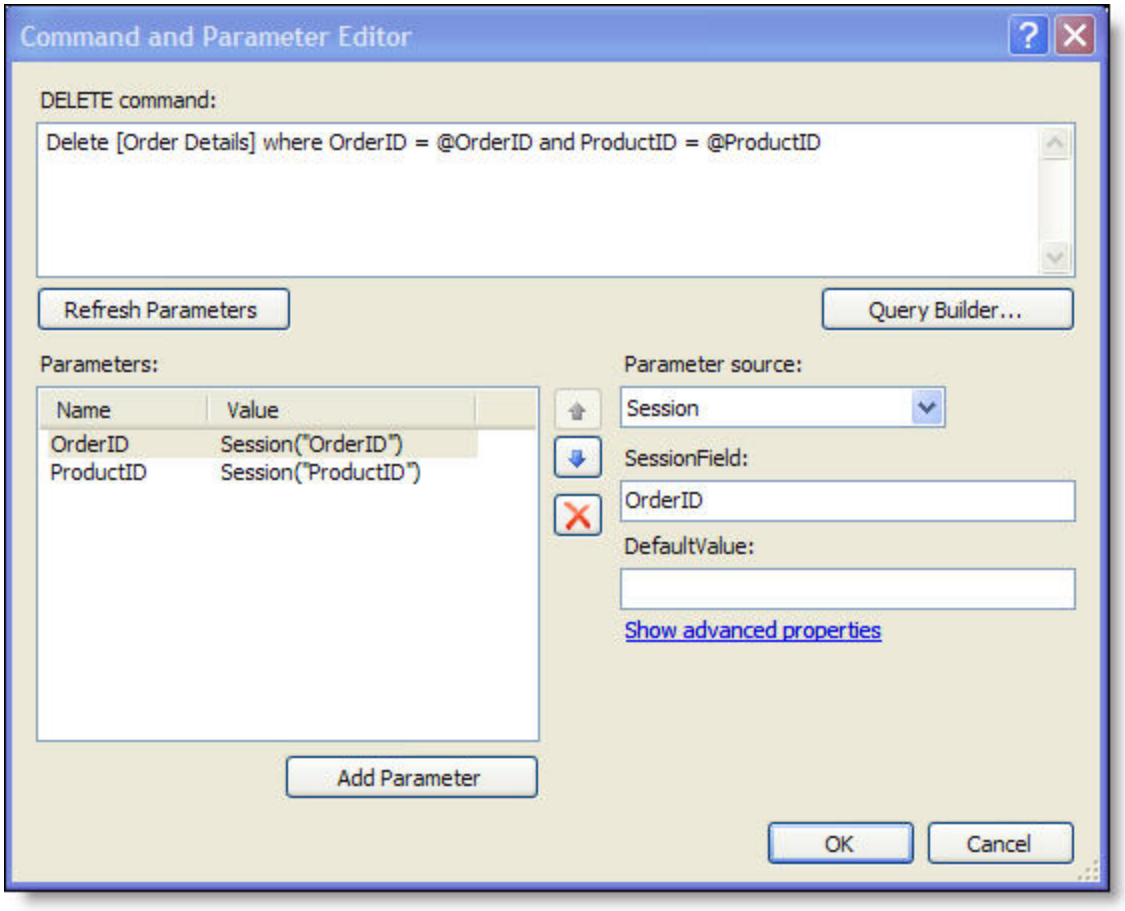
Nominally, that's all you need to do. If you run at this point and try to delete something, however, you'll encounter a SQL exception due to referential integrity constraints. There are a couple of ways around this. One is to alter the affected database tables, adding "ON DELETE CASCADE" provisions to the constraints. Another would be to write a stored procedure that you called on deletion that took care of the cascading deletions for you. Since this is a lab about the RadGrid control, however (and also because it's so easy), let's simply add the necessary controls at each level to do the deletions. It will be a little more work to get a customer deleted, but you'll be more familiar with the necessary steps.

8. Under the MasterTableView, open the DetailTables collection editor. At the first level, add a GridButton column, setting the CommandName and Text to "Delete". Edit the DetailTables collection at this level, and repeat (create a delete button).

9. You'll also need to create the delete statements in the SqlDataSource objects associated with the tables. You can use the "Configure Data Source" option of the smart tag to revisit the SQL structures to make the necessary adjustments by having the wizard generate all of the CRUD statements for you; or edit everything into the source view of the markup page. No matter which way you approach things, you'll probably need to do some hand-tuning to get the deletion logic just right. Probably the easiest way to get this right is to use the SQL editing wizards associated with the delete commands at each level. Here's what the wizards should look like when they're ready to do the job:



SQL Wizard with Order Deletion Logic



SQL Wizard with Order Detail Deletion Logic

Here's one snippet of code that works:

```
<asp:SqlDataSource   ID="SqlDataSource4"   runat="server"
    ConnectionString="<%$  ConnectionStrings:NorthwindConnectionString  %>"
     SelectCommand="SELECT  *  FROM  [Customers]">
</asp:SqlDataSource>
<asp:SqlDataSource   ID="SqlDataSource2"   runat="server"
    ConnectionString="<%$  ConnectionStrings:NorthwindConnectionString  %>"
     SelectCommand="select  *  from  orders  where  CustomerID  =  @CustomerID"
DeleteCommand="Delete  Orders  where  OrderID  =  @OrderID">
     <SelectParameters>
          <asp:SessionParameter  Name="CustomerID"  SessionField="CustomerID"  />
     </SelectParameters>
     <DeleteParameters>
         <asp:Parameter  Name="OrderID"  />
     </DeleteParameters>
</asp:SqlDataSource>
<asp:SqlDataSource   ID="SqlDataSource3"   runat="server"
    ConnectionString="<%$  ConnectionStrings:NorthwindConnectionString  %>"
     SelectCommand="Select  *  from  [Order  Details  Extended]  where  OrderID  =  @OrderID"
     DeleteCommand="Delete  [Order  Details]
       where  OrderID  =  @OrderID  and  ProductID  =  @ProductID">
     <SelectParameters>
          <asp:SessionParameter  Name="OrderID"  SessionField="OrderID"  />
     </SelectParameters>
     <DeleteParameters>
```

```
        <asp:Parameter Name="OrderID" />
        <asp:Parameter Name="ProductID" />
    </DeleteParameters>
</asp:SqlDataSource>
```

10. Run your project. You should now be able to expand your customer and an associated order, click on the "Delete" button next to an order detail line at the lowest hierarchical level, and delete it. Once all the detail lines for an order have been cleared, you'll be able to delete the order; and the customer can be deleted once all the associated orders have been deleted.

Again, remember: this is not intended as a production solution, but to demonstrate principles.

To complete the data cycle, you need to be able to add new items. Telerik makes that as easy as clicking a couple of buttons.

11. Expand the property sheet's "MasterTableView" section again. Set the "AllowAutomaticInserts" property to true, and select a value other than "None" for the "CommandItemDisplay" setting (this is what displays the "Add Record" link).

The final thing you need to do involves a little more technical programming. The issue is that you've got a primary key in the Customers table which is not system-generated, but rather is based on an abbreviated form of the customer's name. For new records, you'll probably need to be able to enter this truncated name, but you need the referential integrity of an unchanging key field where the customer has orders.

Fortunately, there are enough events available between the RadGrid and the SqlDataSource controls that we'll be able to accomplish this.

12. Start by editing the markup of the page the grid is displayed on, and remove the property setting ReadOnly="True" for the GridBoundColumn "CustomerID". In the events section of the SqlDataSource1 property sheet, double-click the Inserted event to create the prototype there.

13. In the code-behind, we need to make a couple of changes. The first will be to make sure that the ReadOnly property we just removed for the CustomerID column is re-established, so in the Page_Load event make it so - if we don't, any edit will permit the user to alter the CUstomerID, and this is a "bad thing" (please pardon the use of such a highly technical term).

14. Next, we'll intercept the InitInsert command (which takes place when you click the "Add New Record" button), and set the ReadOnly property false for the purpose of adding new records.

15. Also intercept the cancel command, re-establishing the readonly nature of the field if the record is not added after all.

16. Finally, we also need to re-establish the CustomerID field as ReadOnly when the record is inserted into the database, but ONLY AFTER the record has been inserted. If you set this field as read-only too early in the process, the change gets lost, and you'll raise a database error chastising you for attempting to insert a null value into the CustomerID field.

Code to accomplish all this is as follows:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        GridBoundColumn idColumn
        = (GridBoundColumn)RadGrid1.MasterTableView.GetColumnSafe("CustomerID");
```

```
        idColumn.ReadOnly = true;
    }

}
protected void RadGrid1_ItemCommand(object source,
     Telerik.WebControls.GridCommandEventArgs e)
{
     GridBoundColumn idColumn
     = (GridBoundColumn)e.Item.OwnerTableView.GetColumnSafe("CustomerID");
     if (e.CommandName == "ExpandCollapse")
     {
          string keyValues = ((GridEditableItem)e.Item).KeyValues;
          if (keyValues.Contains("CustomerID"))
              Session["CustomerID"] = keyValues.Substring(13, keyValues.Length - 15);
          else
              Session["OrderID"] = keyValues.Substring(10, keyValues.Length - 12);
     }
      else if (e.CommandName == RadGrid.InitInsertCommandName)
     {
          e.Canceled = true;
           if (e.Item.OwnerTableView.EditMode == GridEditMode.InPlace)
              idColumn.Display = true;
          idColumn.ReadOnly = false;
           e.Item.OwnerTableView.InsertItem();
            GridEditableItem insertedItem = RadGrid1.MasterTableView.GetInsertItem();
          ((TextBox)insertedItem["CustomerID"].Controls[0]).Text = String.Empty;
     }
      else if (e.CommandName == RadGrid.CancelCommandName)
          idColumn.ReadOnly = true;
}
protected void SqlDataSource1_Inserted(object sender, SqlDataSourceStatusEventArgs e)
{
     GridBoundColumn idColumn =
        (GridBoundColumn)RadGrid1.MasterTableView.GetColumnSafe("CustomerID");
     idColumn.ReadOnly = true;
}
```

VB Example:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
   Dim idColumn As GridBoundColumn = _
       CType(RadGrid1.MasterTableView.GetColumnSafe("CustomerID"), GridBoundColumn)
    idColumn.ReadOnly = True
 End If
End Sub

Protected Sub RadGrid1_ItemCommand(ByVal source As Object, _
  ByVal e As   Telerik.WebControls.GridCommandEventArgs)
 Dim idColumn As GridBoundColumn = _
    CType(e.Item.OwnerTableView.GetColumnSafe("CustomerID"), GridBoundColumn)
 If e.CommandName = "ExpandCollapse" Then
   Dim keyValues As String = CType(e.Item, GridEditableItem).KeyValues
   If  keyValues.Contains("CustomerID") Then
       Session("CustomerID") = keyValues.Substring(13, keyValues.Length - 15)
   Else
       Session("OrderID") = keyValues.Substring(10, keyValues.Length - 12)
   End If
 Else
   If e.CommandName = RadGrid.InitInsertCommandName Then
     e.Canceled = True
     If e.Item.OwnerTableView.EditMode = GridEditMode.InPlace Then
```

```
        idColumn.Display = True
      End If
       idColumn.ReadOnly = False
        e.Item.OwnerTableView.InsertItem
      Dim insertedItem As GridEditableItem = RadGrid1.MasterTableView.GetInsertItem
        CType(insertedItem("CustomerID").Controls(0), TextBox).Text = String.Empty
    Else
      If e.CommandName = RadGrid.CancelCommandName Then
        idColumn.ReadOnly = True
      End If
    End If
 End If
End Sub

Protected Sub SqlDataSource1_Inserted(ByVal sender As Object, _
  ByVal e As SqlDataSourceStatusEventArgs)
 Dim idColumn As GridBoundColumn = _
    CType(RadGrid1.MasterTableView.GetColumnSafe("CustomerID"), GridBoundColumn)
 idColumn.ReadOnly = True
End Sub
```

17. Run your project. Add a new record (you should now be able to enter the key field "CustomerID"). Save it. Edit the same or any other record - the "CustomerID" field should be displayed, but not editable.

It took a little extra work to accomplish this in-place editing when the item key is not system-generated. All in all, however, you've got to appreciate the tremendous amount of functionality you get automatically with the RadGrid control and In-place editing.

Lab: Creating Edit Templates

If the built-in forms generation capability is still not sufficient to meet your needs, there are still two more levels of edit form customization available to you as a developer: form template edit forms, and user control edit forms.

1. Add a new aspx page to your project. Name it FormTemplateEditForm. As in the previous lab exercise, add a SqlDataSource and a RadGrid. Hook the RadGrid's data source up to the Northwind employees table, format the birthdate and hiredate columns, and don't display the Notes column.

2. Add a GridEditCommandColumn to your table.

3. Under the MasterTableView property, make sure that the EditMode property is set to EditForms.

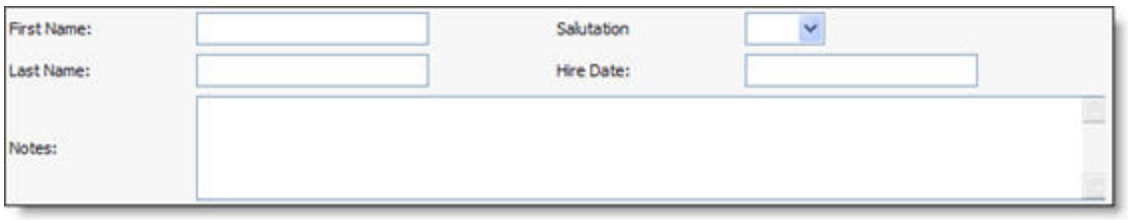
4. Expand the EditFormSettings section, and set the EditFormType to "Template".

5. Open the RadGrid's smart tag, and select the "Edit Templates" entry (at the bottom of the smart tag). In the RadGrid tasks selector, select the "MasterTableViewEditForm".

You are now positioned to use the template designer to create the edit template for your form. This template can be as simple or as complex as you'd like. For the purposes of this lab exercise, we'll add only a few fields to the template for editing, but there's no real limitation to what you can achieve with this method. However, you may find the template editor somewhat cumbersome to work with, and as you'll see shortly, you can also create your own user (ascx) controls for performing the editing task, so there's no point in getting carried away with the template editing capability unless it suites the purposes of your application better.

6. Add fields to collect the First name, Last Name, Salutation, HireDate and notes. Try to do this on

your own before looking at the sample solution, so that the edit form looks like this when displayed (before hooking up any data to your template):



Empty EditForm Template

7. The next task is to get the data displayed in the appropriate fields. Because this data is two-way (updateable) data, you'll want to use the "Bind" method rather than the "Eval" method.

8. The next task is to provide buttons for updating or cancelling your changes. This is a little tricky, because you'll be using the same form for inserting new records as well as updating existing ones. One possible approach is suggested in the sample code below.

9. Don't forget to enable updating by setting the "AllowAutomaticUpdates" property of the RadGrid to true.

10. Finally, if you look at the SQL statements in the markup source, you'll see that the automatically generated SQL took every field in the table into account - but we haven't done anything with many of the fields. Consequently, in order to run correctly and not lose a bunch of data for every record we edit, simply remove the portion of the SQL statement which updates the fields we haven't addressed, along with the associated parameters. Your final update command text should look something like this:

```
UpdateCommand="UPDATE [Employees] SET [LastName] = @LastName,
       [FirstName] = @FirstName, [TitleOfCourtesy] = @TitleOfCourtesy,
       [HireDate] = @HireDate, [Notes] = @Notes WHERE [EmployeeID] = @EmployeeID"

<UpdateParameters>
     <asp:Parameter Name="LastName" Type="String" />
     <asp:Parameter Name="FirstName" Type="String" />
     <asp:Parameter Name="TitleOfCourtesy" Type="String" />
     <asp:Parameter Name="HireDate" Type="DateTime" />
     <asp:Parameter Name="Notes" Type="String" />
     <asp:Parameter Name="EmployeeID" Type="Int32" />
</UpdateParameters>
```

11. Enable new record creation

If you need a little assistance, the markup code inside the template should look something like this:

```
<FormTemplate>
   <table width="60%" >
    <tr width="100%">
        <td width="10%"></td>
        <td width="20%"></td>
        <td width="10%"></td>
        <td width="20%"></td>
    </tr>
    <tr>
       <td>First Name:</td>
      <td>
           <asp:TextBox ID="txtFirstName" runat="server"
            Text='<%# Bind("FirstName") %>' />
      </td>
```

```
        <td>Salutation</td>
         <td><asp:DropDownList ID="ddlSalutation" runat="server"
              SelectedValue='<%#  Bind("TitleOfCourtesy")  %>'  >
             <asp:ListItem Text="" />
             <asp:ListItem Text="Dr." />
             <asp:ListItem Text="Mr." />
             <asp:ListItem Text="Mrs." />
             <asp:ListItem Text="Ms." />
        </asp:DropDownList></td>
    </tr>
    <tr>
        <td>Last Name:</td>
         <td><asp:TextBox ID="txtLastName" runat="server" /></td>
        <td>Hire Date:</td>
         <td><asp:TextBox ID="txtHireDate" runat="server" /></td>
    </tr>
    <tr>
         <td width="5%">Notes:</td>
          <td colspan=3 width="95%"><asp:TextBox ID="txtNotes" runat="server"
             Rows=4 TextMode="MultiLine" Width="100%" /></td>
    </tr>
    <tr>
       <td align="right" colspan="2">
         <asp:Button ID="btnUpdate"
            Text='<%# (Container as GridItem).OwnerTableView.IsItemInserted ?
          "Insert" : "Update" %>'
          runat="server"
            CommandName='<%# (Container as GridItem).OwnerTableView.IsItemInserted ?
             "PerformInsert" : "Update" %>'>
        </asp:Button> 
          <asp:Button ID="btnCancel" Text="Cancel" runat="server" CausesValidation="False"
           CommandName="Cancel"></asp:Button></td>
    </tr>
   </table>
</FormTemplate>
```

12. Run your project. Select an item to edit. Change some information for one of the employees, and save your changes. Try adding a new employee. Once again, with little effort on your part, you have a form that will allow you to maintain the employees table.

Lab: Editing with RadGrid-generated forms

In addition to editing items in-place, RadGrid also provides an automatic edit form generation facility. This feature is useful when there are many columns of data to be edited, or where one or more of the columns contains a long text field which would make editing on a single line awkward.

1. Start a new Web Application project. Add a RadGrid and a SqlDataSource.

2. Connect the SqlDataSource to the Northwind Employees table. Select the "*" entry for the table columns, click the "Advanced" button, and have the wizard generate all of the SQL for the CRUD statements.

3. Configure the RadGrid to connect to the SqlDataSource you just created, and you might as well enable AJAX just because it's so cool. In the RadGrid's property sheet, set AllowAutomaticUpdates to true. Finally, under the MasterTableView, make sure that the EditMode is set to "EditForms", and in the Columns collection, add a GridEditCommandColumn (which you may want to move to the top of the list

so that it displays on the far left side of the grid).

4. Run your project, and edit one of the rows. The elements of the row you selected for editing are displayed for editing.

You'll probably notice from inspecting the edit form that the automatically generated form leaves a lot to be desired in terms of usage of space, handling of text fields, and validation, among other things. The good news is that Telerik has provided a lot of hooks so that developers can address these issues. Let's start with the ugliest first, and do something about that "Notes" field.

5. Close your browser. In the RadGrid property sheet, expand the MasterTableView entry and open the Columns collection editor. Select the entry for the "Notes" column, and set the "Display" property to false. This eliminates the display of the notes column without making it impossible for us to edit and update the field contents.

6. Next, unless your application demands include unusually precise records, you probably don't need the hours - minutes - seconds display included with birthdate and hire date. With the columns collection editor still open, set the data format string for each of those columns to "{0 : MM/dd/yyyy}" (without the quote marks).

7. Even though we don't necessarily want to display the notes field in the grid, we still want to be able to access and edit it as part of the employee record. The default input form for this field would be a textbox a set number of pixels wide, enough to accommodate 20 - 25 characters. If that's sufficient for your needs, you can accept the default. But especially in the case of lengthy fields like the Notes field, something more is called for. There are two things you need to do. The first is to hook up an event handler to the RadGrid's "CreateColumnEditor" event that will set the column editor for the notes column to a custom editor. Switch to the events listing on the RadGrid's property sheet, and double-click the CreateColumnEditor event to create the method prototype. Add the following code to the event handler:

```
C# Example:
if (e.Column is GridBoundColumn)
{
    if ((e.Column as GridBoundColumn).DataField == "Notes")
    {
        e.ColumnEditor = new MultiLineTextBoxColumnEditor();
    }
}

VB Example:
If TypeOf e.Column Is GridBoundColumn Then
 If (CType(ConversionHelpers.AsWorkaround(e.Column, GetType(GridBoundColumn)),_
   GridBoundColumn)).DataField = "Notes" Then
   e.ColumnEditor = New MultiLineTextBoxColumnEditor
 End If
End If
```

8. Add the definition for the MultiLineTextBoxColumnEditor class using the following code:

```
C# Example:
using Telerik.WebControls;
public class MultiLineTextBoxColumnEditor : GridTextColumnEditor
{
    private TextBox textBox;

    protected override void LoadControlsFromContainer()
    {
        this.textBox = this.ContainerControl.Controls[0] as TextBox;
    }
```

```
    public override bool  IsInitialized
    {
        get
        {
            return this.textBox  != null;
        }
    }

    public override string Text
    {
        get
        {
            return this.textBox.Text;
        }
        set
        {
            this.textBox.Text  =  value;
        }
    }

    protected override void  AddControlsToContainer()
    {
        this.textBox  = new  TextBox();
        this.textBox.TextMode  =  TextBoxMode.MultiLine;
        this.textBox.Rows  = 4;
        this.textBox.Columns  = 80;
        this.ContainerControl.Controls.Add(this.textBox);
    }
}
```

VB Example:

```
imports  Telerik.WebControls

Public  class  MultiLineTextBoxColumnEditor
Inherits  GridTextColumnEditor
 Private textBox As TextBox

 protected  Overloads  Overrides Sub  LoadControlsFromContainer()
     Me.textBox  =  CType(ConversionHelpers.AsWorkaround(Me.ContainerControl.Controls(0),_
    GetType(TextBox)), TextBox)
 End Sub

 Public  Overloads  Overrides  readonly Property  IsInitialized() As Boolean
   Get
     Return Not (Me.textBox is Nothing)
   End Get
 End Property

 Public  Overloads  Overrides Property Text() As String
   Get
     Return  Me.textBox.Text
   End Get
   Set
      Me.textBox.Text = value
   End Set
 End Property

 protected  Overloads  Overrides Sub  AddControlsToContainer()
   Me.textBox = New TextBox
    Me.textBox.TextMode  =  TextBoxMode.MultiLine
```

```
    Me.textBox.Rows = 4
    Me.textBox.Columns = 80
      Me.ContainerControl.Controls.Add(Me.textBox)
 End Sub
End class
```

9. Run your project again. With the Notes field and time-of-day display gone, the initial RadGrid display is much cleaner. And, when you edit a record, the Notes field is much more manageable. Better, yes - but there's still more unused space than there needs to be. Close your browser and we can make some more adjustments

10. In the RadGrid property sheet, expand the MasterTableView once again, and expand the "EditFormSettings" section. Set the "Column Number" property to 3; this will cause the edit form to display in three sections. You can also add a formatted heading if you'd like (as pictured below) by placing a format string like "Edit details for employee with ID {0}" in the "CaptionFormatString" property, and the "EmployeeID" field name in the "CaptionDataField" property.

11. The other critical requirement for generating a columnar edit form is to assign the fields to one of the zero-based columns (If you don't make this assignment, everything gets placed into column zero by default, and you'll be right back where you started). You can do this a couple of different ways. One is to open the Columns collection editor, step through the individual columns, and set the "EditFormColumnIndex" property; or if you'd prefer, you can edit the page source, pasting in a property setting similar to

```
EditFormColumnIndex="2"
```

to indicate the column the data field should be displayed in.

While you're looking at entries in the EditFormSettings section of the property sheet, notice that there is a full compliment of settings available to configure the appearance of virtually any facet of the edit form's appearance.

12. Run the project again. click on one of the "Edit" links. Your screen should appear similar to the following:

Grid in Edit Form mode

You now know how to exercise a great deal of control over the display and editing of table data. Sometimes, however, your requirements might be even more sophisticated, and for that, RadGrid provides the ability to create completely custom forms, which we'll cover in the next section.

Lab: Custom Edit Forms

Although useful, the template editor can sometimes be difficult to work with. Another choice you have is to create a user control, and use that to edit your data instead.

1. Add a new aspx page to your project. Name it UserControlEditForm. As in the previous lab exercise, add a SqlDataSource and a RadGrid. Hook the RadGrid's data source up to the Northwind employees table, format the birthdate and hiredate columns, and don't display the Notes column.

2. Add a GridEditCommandColumn to your table.

3. Under the MasterTableView property, make sure that the EditMode property is set to EditForms.

4. Expand the EditFormSettings section. Set the EditFormType to "WebUserControl", and the UserControlName to "EmployeeEditForm.ascx".

5. Create a new user control named "EmployeeEditForm.ascx" in the project. Because the notion of using a user control as your edit form is so powerful, it would be good to include the entire Northwind employees table structure into the edit form. The markup code is set forth below; or, you can go to the **t**elerik site and copy the code from there. It's found at http://www.telerik.com/demos/aspnet/Grid/Examples/DataEditing/UserControlEditForm/DefaultCS.aspx. Do take the time to inspect and understand the code. For the most part, it's just the same pattern repeated many times, but it is important to understand the pattern.

```
<%@ Control Language="C#" AutoEventWireup="true" CodeFile="EmployeeEditForm.ascx.cs"
   Inherits="EmployeeEditForm" %>
<table id="Table2" cellspacing="2" cellpadding="1" width="100%" border="1" rules="none"
   style="border-collapse: collapse">
   <tr class="EditFormHeader">
      <td colspan="2"><b>Employee Details</b></td>
  </tr>
  <tr>
     <td colspan="2"><b>Personal Info:</b></td>
  </tr>
  <tr>
    <td>
       <table id="Table3" cellspacing="1" cellpadding="1"
         width="100%" border="0" class="module">
        <tr>
          <td></td>
          <td></td>
        </tr>
        <tr>
           <td>Country:</td>
          <td>
              <asp:TextBox ID="TextBox7" runat="server"
                 Text='<%# DataBinder.Eval( Container, "DataItem.Country" ) %>'>
             </asp:TextBox></td>
        </tr>
        <tr>
           <td>City:</td>
          <td>
              <asp:TextBox ID="TextBox8" runat="server"
                 Text='<%# DataBinder.Eval( Container, "DataItem.City") %>'
                 TabIndex="1">
             </asp:TextBox></td>
        </tr>
        <tr>
           <td>Region:</td>
          <td>
```

```
            <asp:TextBox ID="TextBox9" runat="server"
               Text='<%# DataBinder.Eval( Container, "DataItem.Region") %>'
             TabIndex="2">
           </asp:TextBox></td>
      </tr>
      <tr>
         <td>Home Phone:</td>
        <td>
            <asp:TextBox ID="TextBox10" runat="server"
               Text='<%# DataBinder.Eval( Container, "DataItem.HomePhone") %>'
             TabIndex="3">
           </asp:TextBox></td>
      </tr>
      <tr>
         <td>Birth Date:</td>
        <td>
            <asp:TextBox ID="TextBox11" runat="server"
               Text='<%# DataBinder.Eval( Container, "DataItem.BirthDate") %>'
             TabIndex="4">
           </asp:TextBox></td>
      </tr>
      <tr>
         <td>Title Of Courtesy</td>
        <td>
            <asp:DropDownList ID="ddlTOC" runat="server" TabIndex="7">
             <asp:ListItem Text="" />
             <asp:ListItem Text="Dr." />
             <asp:ListItem Text="Mr." />
             <asp:ListItem Text="Mrs." />
             <asp:ListItem Text="Ms." />
           </asp:DropDownList></td>
      </tr>
    </table>
  </td>
  <td>
     <table id="Table1" cellspacing="1" cellpadding="1"
       width="300" border="0" class="module">
      <tr>
         <td>Notes:</td>
      </tr>
      <tr>
        <td>
           <asp:TextBox ID="TextBox1"
              Text='<%# DataBinder.Eval( Container, "DataItem.Notes") %>'
               runat="server" TextMode="MultiLine" Rows="5" Columns="40" TabIndex="5">
           </asp:TextBox></td>
      </tr>
      <tr>
         <td>Address:</td>
      </tr>
      <tr>
        <td>
           <asp:TextBox ID="TextBox6"
              Text='<%# DataBinder.Eval( Container, "DataItem.Address") %>'
               runat="server" TextMode="MultiLine" Rows="2" Columns="40" TabIndex="6">
           </asp:TextBox></td>
      </tr>
    </table>
  </td>
</tr>
<tr>
```

```
    <td colspan="2"><b>Company Info:</b></td>
  </tr>
  <tr>
    <td>
        <table id="Table4" cellspacing="1" cellpadding="1" width="300" border="0"
          class="module">
        <tr>
           <td>FirstName:</td>
          <td>
             <asp:TextBox ID="TextBox2"
                 Text='<%# DataBinder.Eval( Container, "DataItem.FirstName") %>'
               runat="server" TabIndex="8">
             </asp:TextBox></td>
        </tr>
        <tr>
           <td>Last Name:</td>
          <td>
             <asp:TextBox ID="TextBox3"
                 Text='<%# DataBinder.Eval( Container, "DataItem.LastName") %>'
               runat="server" TabIndex="9">
             </asp:TextBox></td>
        </tr>
        <tr>
           <td>Hire Date:</td>
          <td>
             <asp:TextBox ID="Textbox5"
                 Text='<%# DataBinder.Eval(Container, "DataItem.HireDate") %>'
               runat="server" TabIndex="10">
             </asp:TextBox></td>
        </tr>
        <tr>
           <td>Title:</td>
          <td>
             <asp:TextBox ID="TextBox4"
                 Text='<%# DataBinder.Eval( Container, "DataItem.Title") %>'
               runat="server" TabIndex="11">
             </asp:TextBox></td>
        </tr>
      </table>
    </td>
    <td></td>
  </tr>
  <tr>
     <td align="right" colspan="2">
       <asp:Button ID="btnUpdate" Text="Update" runat="server"
         CommandName="Update"
          Visible='<%# !(DataBinder.Eval(Container, "DataItem") is
                     Telerik.WebControls.GridInsertionObject)    %>'></asp:Button>
       <asp:Button ID="btnInsert" Text="Insert" runat="server"
         CommandName="PerformInsert"
          Visible='<%# DataBinder.Eval(Container, "DataItem") is
                     Telerik.WebControls.GridInsertionObject    %>'></asp:Button>
       
         <asp:Button ID="btnCancel" Text="Cancel" runat="server" CausesValidation="False"
          CommandName="Cancel"></asp:Button></td>
  </tr>
</table>
```

Note: as of the time of this writing, the code on the Telerik website has an issue with the visibility setting of the insert and update buttons; you should conform your code to look like that shown above.

There is a significant tradeoff in the amount of work required to use a user control as your edit form. Up until now, we've had it relatively easy, letting the RadGrid control take care of all the database work for use with little more than a few mouse clicks. With the user control, all of that changes.

You may have noticed in the markup above that we're setting the contents of the controls by binding to the data contained in the container's DataItem property. Unlike the other forms of editing that we've used, which maintain a two-way connection to the data automatically, there is no automatic update when using a user control. Accordingly, we need to provide code for the insert and update events to convey any updated information back to the database. We do this by attaching event handlers to the insert and update commands, collecting the data from the user control, and updating the database.

6. Add code to accomplish the insert and update tasks. **NOTE:** This code goes into the form which contains the grid, and **not** in the code behind the user control. The reason for this is that the RadGrid picks up the button-click event, inspects the command name property of the button, and turns it into a RadGrid event - update, cancel, etc.

7. Your code should look something like this:

```
C# Example:
using System.Data.SqlTypes;
using System.Data.SqlClient;
using Telerik.WebControls;

private DataTable _employees;
private System.Data.SqlClient.SqlDataAdapter _adapter;
private DataSet _ds;

private DataTable Employees
{
  get
  {
    if (_employees != null)
      return _employees;
    else
    {
      _adapter = new SqlDataAdapter("SELECT * from Employees",
         Convert.ToString(
             ConfigurationManager.ConnectionStrings["NorthwindConnectionString"]));
      _ds = new DataSet();
       _adapter.Fill(_ds);
       _ds.Tables[0].TableName = "Employees";
       _employees = _ds.Tables[0];
      return _employees;
    }
  }
}

protected Hashtable collectFormData(Telerik.WebControls.GridCommandEventArgs e)
{
   UserControl userControl =
(UserControl)e.Item.FindControl(GridEditFormItem.EditFormUserControlID);
   Hashtable newValues = new Hashtable();

   newValues["Country"] = (userControl.FindControl("TextBox7") as TextBox).Text;
   newValues["City"] = (userControl.FindControl("TextBox8") as TextBox).Text;
   newValues["Region"] = (userControl.FindControl("TextBox9") as TextBox).Text;
   newValues["HomePhone"] = (userControl.FindControl("TextBox10") as TextBox).Text;
   newValues["BirthDate"] = (userControl.FindControl("TextBox11") as TextBox).Text;
   newValues["TitleOfCourtesy"] =
      (userControl.FindControl("ddlTOC") as DropDownList).SelectedItem.Value;
```

```
   newValues["Notes"]  =  (userControl.FindControl("TextBox1") as  TextBox).Text;
   newValues["Address"]  =  (userControl.FindControl("TextBox6") as  TextBox).Text;
   newValues["FirstName"]  =  (userControl.FindControl("TextBox2") as  TextBox).Text;
   newValues["LastName"]  =  (userControl.FindControl("TextBox3") as  TextBox).Text;
   newValues["HireDate"]  =  (userControl.FindControl("Textbox5") as  TextBox).Text;
   newValues["Title"]  =  (userControl.FindControl("TextBox4") as  TextBox).Text;
  return  newValues;
}

protected void  RadGrid1_InsertCommand(object  source,
    Telerik.WebControls.GridCommandEventArgs  e)
{
   GridEditableItem  editedItem  =  e.Item as  GridEditableItem;
   UserControl  userControl  =
         (UserControl)e.Item.FindControl(GridEditFormItem.EditFormUserControlID);

  //Create  new  row  in  the  DataSource
  DataRow  newRow = this.Employees.NewRow();

  //Insert  new  values
   Hashtable  newValues  =  collectFormData(e);

  //make  sure  that  unique  primary  key  value  is  generated  for  the  inserted  row
   newValues["EmployeeID"] =
    (int)this.Employees.Rows[this.Employees.Rows.Count  - 1]["EmployeeID"] + 1;
   _adapter.InsertCommand  = new   SqlCommand(SqlDataSource1.InsertCommand,
    new     SqlConnection(Convert.ToString(ConfigurationManager.ConnectionStrings[
  "NorthwindConnectionString"])));
  try
  {
    foreach  (DictionaryEntry  entry  in  newValues)
    {
       newRow[(string)entry.Key]  =  entry.Value;
         _adapter.InsertCommand.Parameters.Add(new
       SqlParameter((string)entry.Key,  entry.Value));
    }

    // Add  parameters  not  collected
      _adapter.InsertCommand.Parameters.Add(new  SqlParameter("ReportsTo",
       SqlInt32.Null));
      _adapter.InsertCommand.Parameters.Add(new  SqlParameter("PhotoPath",
       SqlString.Null));
      _adapter.InsertCommand.Parameters.Add(new  SqlParameter("Extension",
       SqlString.Null));
      _adapter.InsertCommand.Parameters.Add(new  SqlParameter("PostalCode",
       SqlString.Null));

    this.Employees.Rows.Add(newRow);

     _adapter.InsertCommand.CommandType  =  CommandType.Text;
    _adapter.Update(_ds, "Employees");
  }
  catch  (Exception  ex)
  {
     RadGrid1.Controls.Add(new  LiteralControl(
      "Unable  to  update/insert  Employees.  Reason:  " + ex.Message));
    e.Canceled = true;
  }
}
protected void  RadGrid1_UpdateCommand(object  source,
```

```
   Telerik.WebControls.GridCommandEventArgs  e)
{
   GridEditableItem editedItem = e.Item as  GridEditableItem;
   UserControl userControl =
         (UserControl)e.Item.FindControl(GridEditFormItem.EditFormUserControlID);

  //Find the row to be updated in the DataSource
   DataRow[] changedRows = this.Employees.Select("EmployeeID = " +
        editedItem.OwnerTableView.DataKeyValues[editedItem.ItemIndex]["EmployeeID"]);

  if (changedRows.Length != 1)
  {
     RadGrid1.Controls.Add(new  LiteralControl(
      "Unable to locate the Employee for updating."));
    e.Canceled = true;
    return;
  }

  //Update new values
   Hashtable newValues = collectFormData(e);
   newValues["EmployeeID"] = (int)editedItem.OwnerTableView.DataKeyValues[
      editedItem.ItemIndex]["EmployeeID"];
   _adapter.UpdateCommand = new   SqlCommand(SqlDataSource1.UpdateCommand,
    new  SqlConnection(
     Convert.ToString(
        ConfigurationManager.ConnectionStrings["NorthwindConnectionString"])));
   changedRows[0].BeginEdit();
  try
  {
    foreach (DictionaryEntry entry in newValues)
    {
       changedRows[0][(string)entry.Key] = entry.Value;
        _adapter.UpdateCommand.Parameters.Add(new
         SqlParameter((string)entry.Key,  entry.Value));
    }
    // Add parameters not collected
      _adapter.UpdateCommand.Parameters.Add(new  SqlParameter("ReportsTo",
      editedItem["ReportsTo"].Text));
      _adapter.UpdateCommand.Parameters.Add(new  SqlParameter("PhotoPath",
      editedItem["PhotoPath"].Text));
      _adapter.UpdateCommand.Parameters.Add(new  SqlParameter("Extension",
      editedItem["Extension"].Text));
      _adapter.UpdateCommand.Parameters.Add(new  SqlParameter("PostalCode",
      editedItem["PostalCode"].Text));
    changedRows[0].EndEdit();
     _adapter.UpdateCommand.CommandType = CommandType.Text;
    _adapter.Update(_ds, "Employees");
    this.Employees.AcceptChanges();
  }
  catch (Exception ex)
  {
     changedRows[0].CancelEdit();
     RadGrid1.Controls.Add(
      new  LiteralControl("Unable to update/insert Employees. Reason: "
        + ex.Message));
    e.Canceled = true;
  }
}

VB Example:
imports   System.Data.SqlDataTypes;
```

```
imports   System.Data.SqlCLient;
imports   Telerik.WebControls

Private _employees As DataTable
Private _adapter As   System.Data.SqlClient.SqlDataAdapter
Private _ds As DataSet

Private ReadOnly Property Employees() As DataTable
 Get
   If Not (_employees Is Nothing) Then
     Return _employees
   Else
      _adapter = New SqlDataAdapter("SELECT * from Employees", _
         Convert.ToString(_
              ConfigurationManager.ConnectionStrings("NorthwindConnectionString")))
     _ds = New DataSet
      _adapter.Fill(_ds)
      _ds.Tables(0).TableName = "Employees"
     _employees = _ds.Tables(0)
     Return _employees
   End If
 End Get
End Property

Protected Function collectFormData(ByVal e As  Telerik.WebControls.GridCommandEventArgs) _
  As Hashtable
 Dim userControl As UserControl = _
      CType(e.Item.FindControl(GridEditFormItem.EditFormUserControlID), UserControl)
 Dim newValues As Hashtable = New Hashtable
 newValues("Country") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox7"), _
     GetType(TextBox)), TextBox)).Text
 newValues("City") =
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox8"), _
     GetType(TextBox)), TextBox)).Text
 newValues("Region") =
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox9"), _
     GetType(TextBox)), TextBox)).Text
 newValues("HomePhone") =
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox10"), _
     GetType(TextBox)), TextBox)).Text
 newValues("BirthDate") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox11"), _
     GetType(TextBox)), TextBox)).Text
 newValues("TitleOfCourtesy") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("ddlTOC"), _
      GetType(DropDownList)), DropDownList)).SelectedItem.Value
 newValues("Notes") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox1"), _
     GetType(TextBox)), TextBox)).Text
 newValues("Address") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox6"), _
     GetType(TextBox)), TextBox)).Text
 newValues("FirstName") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox2"), _
     GetType(TextBox)), TextBox)).Text
 newValues("LastName") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox3"), _
     GetType(TextBox)), TextBox)).Text
 newValues("HireDate") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("Textbox5"), _
```

```
     GetType(TextBox)), TextBox)).Text
 newValues("Title") = _
    (CType(ConversionHelpers.AsWorkaround(userControl.FindControl("TextBox4"), _
     GetType(TextBox)), TextBox)).Text
 Return newValues
End Function

Protected Sub RadGrid1_InsertCommand(ByVal source As Object, _
  ByVal e As Telerik.WebControls.GridCommandEventArgs)
 Dim editedItem As GridEditableItem = _
     CType(ConversionHelpers.AsWorkaround(e.Item, GetType(GridEditableItem)),_
     GridEditableItem)
 Dim userControl As UserControl = _
     CType(e.Item.FindControl(GridEditFormItem.EditFormUserControlID), UserControl)
 Dim newRow As DataRow = Me.Employees.NewRow
 Dim newValues As Hashtable = collectFormData(e)
 newValues("EmployeeID") = _
    CType(Me.Employees.Rows(Me.Employees.Rows.Count - 1)("EmployeeID"), Integer) + 1
 _adapter.InsertCommand = _
  New SqlCommand(SqlDataSource1.InsertCommand, _
  New SqlConnection(Convert.ToString(_
      ConfigurationManager.ConnectionStrings("NorthwindConnectionString"))))
 Try
  For Each entry As DictionaryEntry In newValues
     newRow(CType(entry.Key, String)) = entry.Value
      _adapter.InsertCommand.Parameters.Add(_
      New SqlParameter(CType(entry.Key, String), entry.Value))
  Next
    _adapter.InsertCommand.Parameters.Add(_
    New SqlParameter("ReportsTo", SqlInt32.Null))
    _adapter.InsertCommand.Parameters.Add(_
    New SqlParameter("PhotoPath", SqlString.Null))
    _adapter.InsertCommand.Parameters.Add(_
    New SqlParameter("Extension", SqlString.Null))
    _adapter.InsertCommand.Parameters.Add(_
    New SqlParameter("PostalCode", SqlString.Null))
   Me.Employees.Rows.Add(newRow)
   _adapter.InsertCommand.CommandType = CommandType.Text
   _adapter.Update(_ds, "Employees")
 Catch ex As Exception
   RadGrid1.Controls.Add(_
    New LiteralControl("Unable to update/insert Employees. Reason: " + ex.Message))
  e.Canceled = True
 End Try
End Sub

Protected Sub RadGrid1_UpdateCommand(ByVal source As Object, _
  ByVal e As Telerik.WebControls.GridCommandEventArgs)
 Dim editedItem As GridEditableItem = _
    CType(ConversionHelpers.AsWorkaround(_
      e.Item, GetType(GridEditableItem)), GridEditableItem)
 Dim userControl As UserControl = _
     CType(e.Item.FindControl(GridEditFormItem.EditFormUserControlID), UserControl)
 Dim changedRows As DataRow() = _
   Me.Employees.Select("EmployeeID = " + _
       editedItem.OwnerTableView.DataKeyValues(editedItem.ItemIndex)("EmployeeID"))
 If Not (changedRows.Length = 1) Then
   RadGrid1.Controls.Add(_
    New LiteralControl("Unable to locate the Employee for updating."))
  e.Canceled = True
  Return
```

```
End If
Dim newValues As Hashtable = collectFormData(e)
newValues("EmployeeID") = _
    CType(editedItem.OwnerTableView.DataKeyValues(_
     editedItem.ItemIndex)("EmployeeID"), Integer)
_adapter.UpdateCommand = _
  New SqlCommand(SqlDataSource1.UpdateCommand, _
  New SqlConnection(_
     Convert.ToString(_
          ConfigurationManager.ConnectionStrings("NorthwindConnectionString"))))
 changedRows(0).BeginEdit
Try
  For Each entry As DictionaryEntry In newValues
      changedRows(0)(CType(entry.Key, String)) = entry.Value
       _adapter.UpdateCommand.Parameters.Add(_
       New SqlParameter(CType(entry.Key, String), entry.Value))
  Next
    _adapter.UpdateCommand.Parameters.Add(_
    New SqlParameter("ReportsTo", editedItem("ReportsTo").Text))
    _adapter.UpdateCommand.Parameters.Add(_
    New SqlParameter("PhotoPath", editedItem("PhotoPath").Text))
    _adapter.UpdateCommand.Parameters.Add(_
    New SqlParameter("Extension", editedItem("Extension").Text))
    _adapter.UpdateCommand.Parameters.Add(_
    New SqlParameter("PostalCode", editedItem("PostalCode").Text))
   changedRows(0).EndEdit
    _adapter.UpdateCommand.CommandType = CommandType.Text
   _adapter.Update(_ds, "Employees")
    Me.Employees.AcceptChanges
Catch ex As Exception
   changedRows(0).CancelEdit
   RadGrid1.Controls.Add(_
    New LiteralControl("Unable to update/insert Employees. Reason: " + ex.Message))
  e.Canceled = True
End Try
End Sub
```

Let's look a little bit at what's going on in this code.

- We declare a DataTable property named Employees. That's because we used a SqlDataSource to populate the RadGrid component, and there's no convenient way to access the underlying data for insert and update operations. We also declare the associated elements we'll need to support the property, like a dataset and a data adapter.
- Except for the EmployeeID, the data collected for insert and update is identical, so the "CollectFormData" method is written to accomplish that.
- We need SQL to do both the insert and the update - but that SQL was already generated for us as a byproduct of creating the SQLDataSource control, so there's no need to reinvent it (that's another reason to pass the events back up to the form which contains the RadGrid control)..
- Not all of the columns are represented in the edit form. For new records, we place null values into the table rows; for tows being updated, we use the original contents.
- Note the positioning of the "AcceptChanges" command on the employee table - if this command is executed any time before the call to Update the adapter, there will be no changed or updated table rows to be acted upon.

8. The last adjustment you'll need to make before you can run your code successfully is to make some allowance for the "Photo" field, which is contained in the database table and which had code

automatically generated for it. Since we aren't making any provision for collecting an image on the edit form, the most expedient route to getting the code to run is to remove references to the "Photo" field in the Bound columns, sql parameters, and the insert/update sql itself.

9. Run your project. Use the form to add a new employee record, and to edit the record after adding it.

You should now have a pretty good handle on the various ways to display and update data. The user control editor form, in particular, is a little more work, but extremely powerful and flexible. Not all applications will need that much flexibility, but for those that do, it's an extremely handy feature.

Lab: Defining RadGrid at Runtime

Why would you want to do this?

Seriously, while it's POSSIBLE to instantiate and manipulate a RadGrid component programmatically, it's difficult to imagine a circumstance under which you'd want to do so. There's simply too much connection to the underlying data, too much richness in the connection to templated components and user controls, and too many available features that it wouldn't make sense to disable. In point of fact, the subject of run-time use is covered in depth in nearly every other RadControl help file published by Telerik - but not in the RadGrid help file.

For the sake of completeness, however, we'll go through a brief example for the lab.

1. Start a new web page in your project, and name it "ProgrammaticGrid". As in previous lab exercises, add a SqlDataSource connected to the Northwind Employees table. Also add a placeholder.

2. In order for a control's ViewState to be restored once values have been set, it needs to exist in the page's controls collection prior to the Page_Load event; in other words, the grid should be instantiated in the Page_Init event, because the ViewState tree is traversed and values are restored between the Init and the Load events.

```
C# Example:
protected void Page_Init(object source, System.EventArgs e)
{
   defineGridStructure();
}

VB Example:
protected Sub Page_Init(ByVal source As Object, ByVal e As System.EventArgs)
  defineGridStructure
End Sub
```

3. Assigning properties to the grid itself is mostly an exercise in knowing which properties you need and want to set. You can do anything here that you normally would in the property sheets or the RadGrid wizard, including adding detail tables to create a hierarchical grid, add templates for display and editing, or assign a user web control as your editing vehicle. Here's the code for about as simple a grid as you could design - all it displays is the EmployeeID, LastName and FirstName. The principles involved for complex grids are no different, just more detailed.

```
C# Example:
using Telerik.WebControls;

private void DefineGridStructure()
{
   RadGrid RadGrid1 = new RadGrid();
   RadGrid1.DataSourceID = "SqlDataSource1";
    RadGrid1.MasterTableView.DataKeyNames = new string[] { "EmployeeID" };
```

```
   RadGrid1.Skin = "Default";

   RadGrid1.Width = Unit.Percentage(100);
   RadGrid1.PageSize = 5;
   RadGrid1.AllowPaging = true;
    RadGrid1.AutoGenerateColumns = false;

    RadGrid1.MasterTableView.PageSize = 15;

  //Add columns
   GridBoundColumn boundColumn;
   boundColumn = new GridBoundColumn();
   boundColumn.DataField = "EmployeeID";
   boundColumn.HeaderText = "Employee ID";
     RadGrid1.MasterTableView.Columns.Add(boundColumn);

   boundColumn = new GridBoundColumn();
   boundColumn.DataField = "LastName";
   boundColumn.HeaderText = "Last Name";
     RadGrid1.MasterTableView.Columns.Add(boundColumn);

   boundColumn = new GridBoundColumn();
   boundColumn.DataField = "FirstName";
   boundColumn.HeaderText = "First Name";
     RadGrid1.MasterTableView.Columns.Add(boundColumn);

  //Add the RadGrid instance to the controls
  this.PlaceHolder1.Controls.Add(RadGrid1);
}

VB Example:
imports  Telerik.WebControls

Private Sub  DefineGridStructure()
 Dim RadGrid1 As RadGrid = New RadGrid
  RadGrid1.DataSourceID = "SqlDataSource1"
   RadGrid1.MasterTableView.DataKeyNames  = New String()  {"EmployeeID"}
 RadGrid1.Skin = "Default"
 RadGrid1.Width = Unit.Percentage(100)
 RadGrid1.PageSize = 5
 RadGrid1.AllowPaging = True
 RadGrid1.AutoGenerateColumns = False
 RadGrid1.MasterTableView.PageSize = 15
Dim boundColumn As GridBoundColumn
 boundColumn = New GridBoundColumn
 boundColumn.DataField = "EmployeeID"
 boundColumn.HeaderText = "Employee ID"
    RadGrid1.MasterTableView.Columns.Add(boundColumn)
 boundColumn = New GridBoundColumn
 boundColumn.DataField = "LastName"
 boundColumn.HeaderText = "Last Name"
    RadGrid1.MasterTableView.Columns.Add(boundColumn)
 boundColumn = New GridBoundColumn
 boundColumn.DataField = "FirstName"
 boundColumn.HeaderText = "First Name"
    RadGrid1.MasterTableView.Columns.Add(boundColumn)
   Me.PlaceHolder1.Controls.Add(RadGrid1)
End Sub
```

4. Run your project and inspect your results.

That's an awful lot of work to display three simple columns of data, with no formatting whatever, and no editing functionality. The point here is that it's available, if it's what you need.

2.1.4 Miscellaneous topics

While the meat of what there is to know about the RadGrid component is in the retrieval, display, and editing of data, there are several other features that make using the grid convenient and versatile. In this section, we'll walk through several niche topics.

Lab: Changing the Skin

The RadGrid component ships with a number of skins which are available to dress up the appearance of your application. Many of these skins are available across the entire suite of RadControls, so a consistent look is readily available where ever you use the Telerik components.

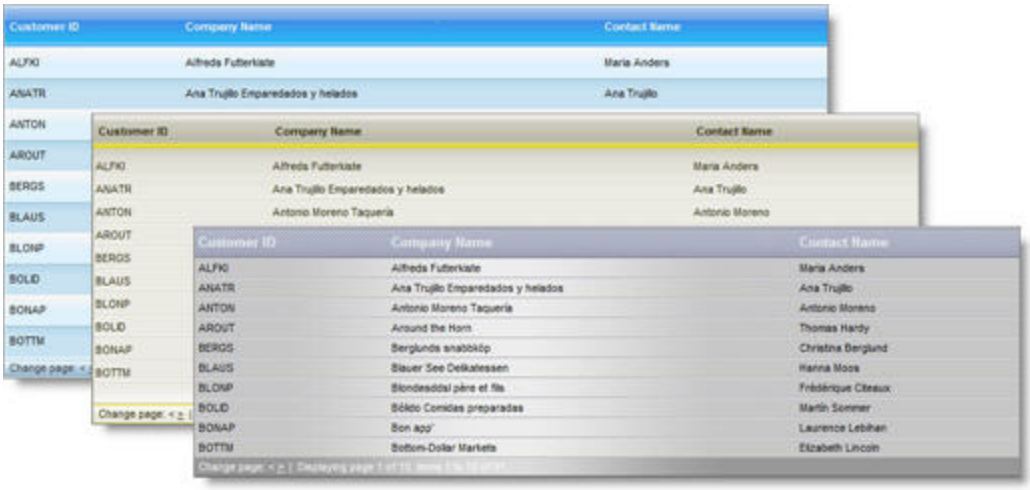
(There is no sample code in the accompanying materials for this lab exercise)

To use a skin, take the following steps:

1. Create a RadControls/Grid/Skins directory under your project's root directory.

2. Copy all of the contents of the corresponding directory from your Telerik installation directory to the directory you just created. Note that if you only want to use one or two of the skins, all you need to do is copy those folders.

3. Set the skin to use by entering the name of the skin folder in the "Skin" property in the property sheet. You don't even need to run the project to see what it will look like - the skin is applied in the design surface immediately when you tab out of the property field. If you've copied all of the skin folders over, some skins you might sample are: Grey, Green, Ice, and WebBlue.



Glassy, 3D and Inox skins

Lab: No Records to Display

Whenever you're displaying records in a RadGrid, and there are no records to be displayed, it is useful to convey that information to the user as feedback, so that they don't wonder if there was perhaps some malfunction in the display of the data. RadGrid provides two properties at every table level which you can set to convey this information.

(There is no sample code in the accompanying materials for this lab exercise)

1. Expand the "MasterTableView" property for your grid. Scroll down and locate the properties named "NoMasterRecordsText" and "NoDetailRecordsText". This is where you would set the text for any different message that you would like to display.

2. Open the "DetailTables" collection editor. As with the master table view, scroll down again. Note that there are settings here as well for tables at both the parent and detail levels.

Note here that the message displayed may not necessarily be what you'd think from the property names. For instance, if you set this text on the Customer-Order-Details table collection, then delete all of the order details for an order, the message that will be displayed is the "NoDetailRecordsText" for the 3rd-level table. In other words, at least at the lower levels of the hierarchy, you may never see the "NoMasterRecordsText" message.

3. To complicate things just a bit further, Telerik also documents the "NoRecordsTemplate" property in their help file, which can be used in the markup of a GridTableView (master or detail) element. Setting this value overrides the "NoDetailRecordsText" setting.

Lab: Exporting data to Excel and Word

Telerik also provides a means of taking the data displayed in your grid, and exporting it to a Microsoft Excel or Word format.

(There is no sample code in the accompanying materials for this lab exercise)

1. Using the same three-level grid structure we have used previously (with the Northwind Customer/Order/Details tables), add a button to the form.

2. In the code-behind for the button's click event, add the following:

```
C# Example:
protected void Button1_Click(object sender, EventArgs e)
{
      RadGrid1.ExportSettings.ExportOnlyData = true;
      RadGrid1.ExportSettings.IgnorePaging = true;
      RadGrid1.ExportSettings.OpenInNewWindow = true;
       RadGrid1.MasterTableView.ExportToExcel();
}

VB Example:
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
  RadGrid1.ExportSettings.ExportOnlyData = True
  RadGrid1.ExportSettings.IgnorePaging = True
  RadGrid1.ExportSettings.OpenInNewWindow = True
   RadGrid1.MasterTableView.ExportToExcel
End Sub
```

The first setting, "ExportOnlyData", eliminates the export of non-data items such as command buttons, expansion buttons, and the like. Well, almost - you do get the column headers (which is a good thing), but if you're displaying things like "Add New Record" or "Refresh" in the top and bottom margins, you'll get those as well. They also get exported with their associated javascript, which will raise an error if you try to click it inside the spreadsheet.

The second setting, "IgnorePaging", instructs the export routine to export only the data from the current page (if set false), or all data (if true).

The third setting, "OpenInNewWindow", determines whether the output will be to a new window. If you set this option false, the data will be sent to the Excel browser plugin. As a word of caution, if you

choose to open the file in this mode (rather than save it to disk), and have any thoughts of going back to the previously displayed RadGrid page, you're hosed (to use the technical term) if you've enabled AJAX. The Excel plugin seems to do just that - plugs itself in, in place of the page you had open.

There are some other export idiosyncrasies, as well. If you have detail tables, don't even open them prior to export. If you do, you'll get row after row of the detail of whatever item you opened, repeated for each master table row. If you opened more than one set of detail records, the detail that will be repeated is whichever one you opened last. If you open a third-level table (like the Order Details table), you won't see any of that data displayed. And if you open up a detail record, then close it back down, it still gets displayed in the spreadsheet.

In brief, then, as of the time of this writing, Telerik still has a bit of work to do on the RadGrid export feature, at least as far as the Office 2003 product goes. Still in all, if you're aware of the limitations, and can work within them, the export feature is pretty slick.

2.1.5 Summary

This chapter covered the things you need to know to be productive with the RadGrid component:

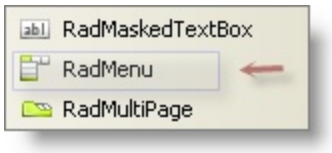
- Connecting to a data source
- Enabling AJAX
- Paging
- Scrolling, grouping, sorting and filtering
- Displaying hierarchical data tables
- Data editing using in-place editing, automatically generated forms, templates, and user controls
- Programmatic instantiation of a RadGrid component.

There's plenty of ways to use this workhorse component, and you'll find that there is control and to spare over all the different facets of the grid appearance and data presentation. The great news here is if you learn the basics, you can expand on them virtually without limit - all you need is a vision of what you'd like to accomplish, and RadGrid will provide the vehicle to get you there.

2.2 RadMenu

2.2.1 Getting Started

The Telerik RadMenu allows you to build just about any type of drop-down or context menu for your ASP.NET applications. With RadMenu you can build multi-row, vertical, hierarchical or scrollable menus. You can change numerous aspects of the menu's appearance and behavior. Additionally, the RadMenu fully supports binding to various data sources.



RadMenu in the toolbox

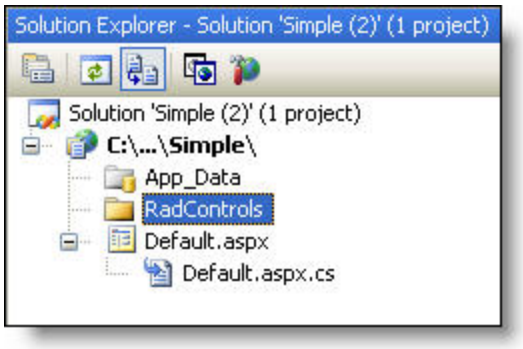
2.2.2 Using RadMenu in the Designer

This section will teach you how to create and configure a simple web page using the RadMenu. It will also illustrate how to modify various properties within the designer and how to change the look of the menu using skins. You will also learn how to construct a context level menu.

Lab: Create a simple menubar

This lab will teach you to build a simple RadMenu using the designer. You will also set other properties such as the AccessKey and ImageUrl properties.

1. Create a new ASP.NET Project named Simple.
2. Create a folder in this project named \RadControls as shown in the figure below.



Create a RadControls Directory

3. Copy the contents of the \Menu folder form the Telerik installation directory. This step supplies the Smart Tag with the option to apply visual styles to the entire RadMenu using AutoFormat.
   - **Note**: Assuming you selected the default installation path, the directory location would be:

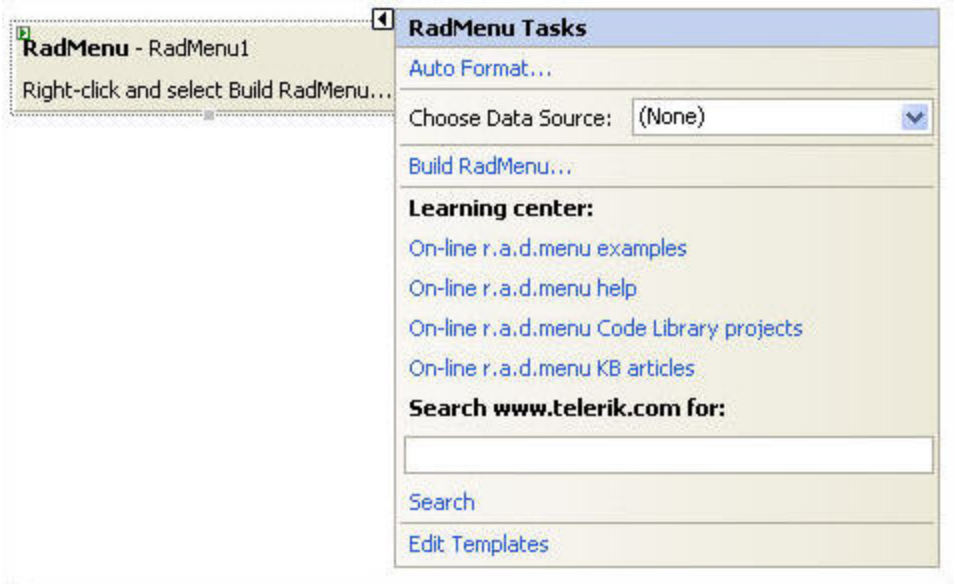
   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls\Menu
4. Copy the contents of an image directory from the Telerik installation directory to your project. This

directory should be a sub-directory of the root.

Note: Assuming you selected the default installation path, the directory location would be:

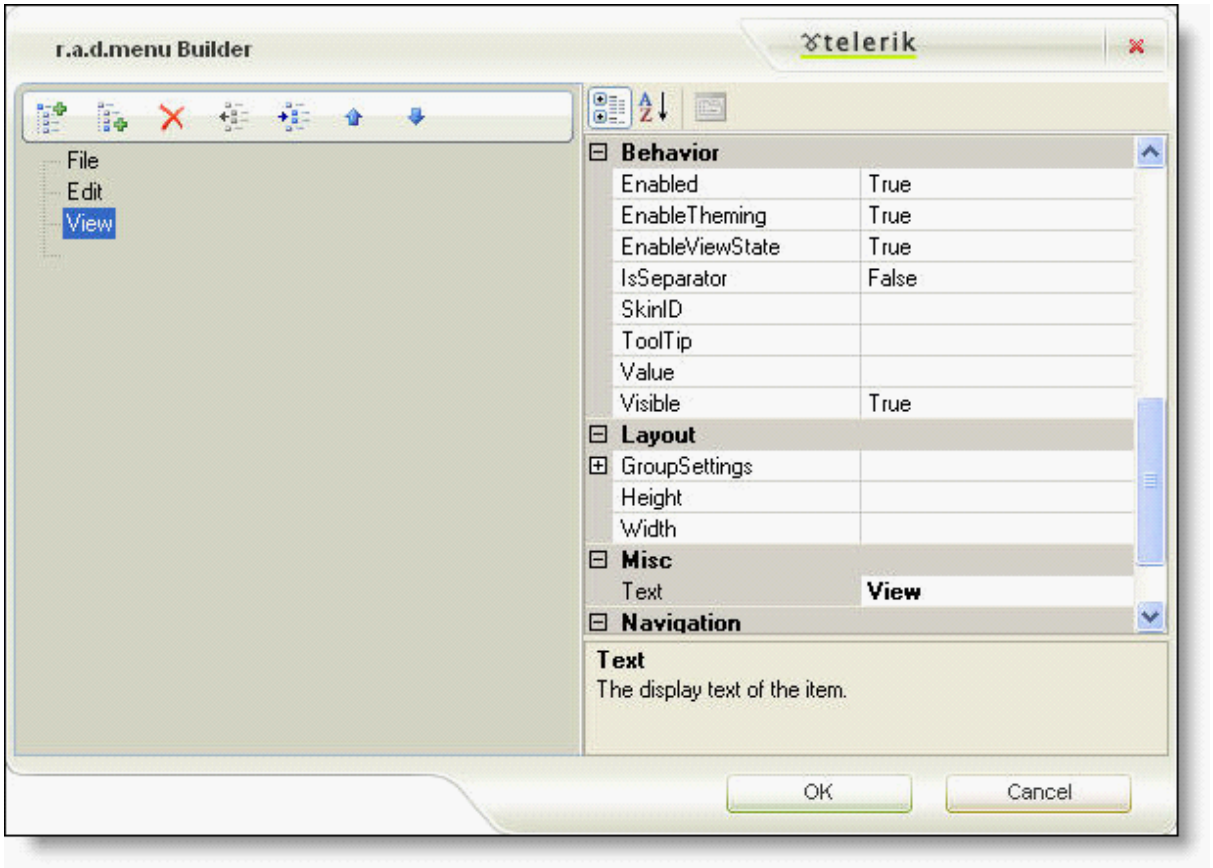
C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\Menu\Examples\Functionality\FirstLook\Images

5. Add a RadMenu control to the default page.
6. Click the Smart Tag button to invoke RadMenu Tasks as shown in the figure below.



RadMenu Tasks Window

7. Start Building the menu by selecting "Build RadMenu..." link which will invoke the r.a.d menu Builder dialog
8. Add the three root menu items as depicted in the Figure below. To change the title of the menu items you must change the Text property accordingly.



Adding Menu Items

9. Underneath the File menu, add the following menu items and set their properties specified accordingly.

Menu	Property	Value
New	Text	"New"
	ImageUrl	"Images/11new.gif"
	AccessKey	"w"
Open	Text	"Open"
	ImageUrl	"Images/12open.gif"
	AccessKey	"o"
Close	Text	"Close"
	AccessKey	"c"
Separator	Text	""
Save	Text	"Save"

Menu	Property	Value
	ImageUrl	"Images/13Save.gif"
	AccessKey	"S"
Save As	Text	"Save As"
	AccessKey	"A"

10.Underneath the Edit Menu, add the following menu items and set their properties accordingly.

Menu	Property	Value
Undo	Text	"Undo"
	ImageUrl	"Images/21undo.gif"
	AccessKey	"U"
Separator	Text	""
	IsSeparator	True
Copy	Text	"Copy"
	ImageUrl	"Images/12open.gif"
	AccessKey	"c"

11.Underneath the View Menu, add the following menu items and set their properties accordingly.

Menu	Property	Value
Normal	Text	"Normal"
	ImageUrl	"Images/31normal.gif"
	AccessKey	"n"
Print	Text	"Print"
	ImageUrl	"Images/33print.gif"
	AccessKey	"p"

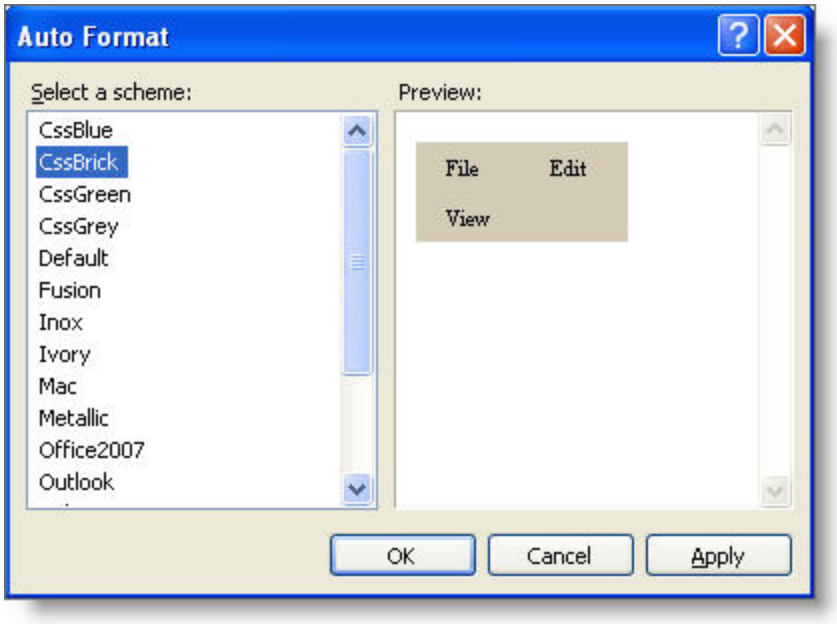
Run the program. Note how the AccessKey property denotes a keyboard shortcut for the control. Also, when running the application, you'll see that the image supplied as the ImageUrl property is displayed to the left of the menu.

Lab: Modify the RadMenu skin

This Lab will illustrate how to change the RadMenu skin using the Auto Format dialog in the designer.

1. Using the project from the previous Lab, you can change the Skin of the RadMenu by clicking on the

Smart Tag and then selecting "Auto Format..." link. This will invoke the Auto Format dialog shown in Figure 1.



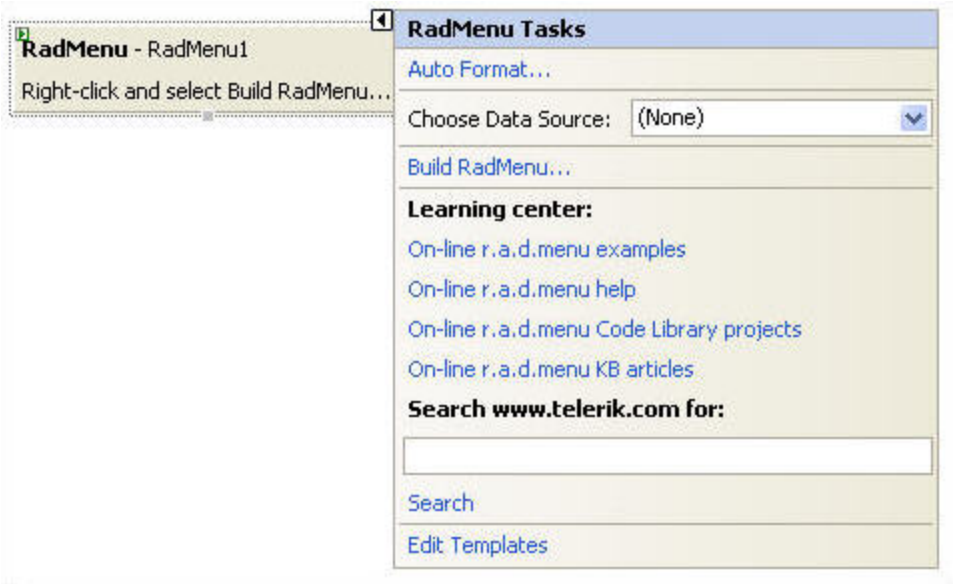
Auto Format Dialog

2. Select the "CssBrick" skin and press OK.
3. This completes this lab. Execute the project to examine the modified skin.

Lab: Create a Simple Context Menu

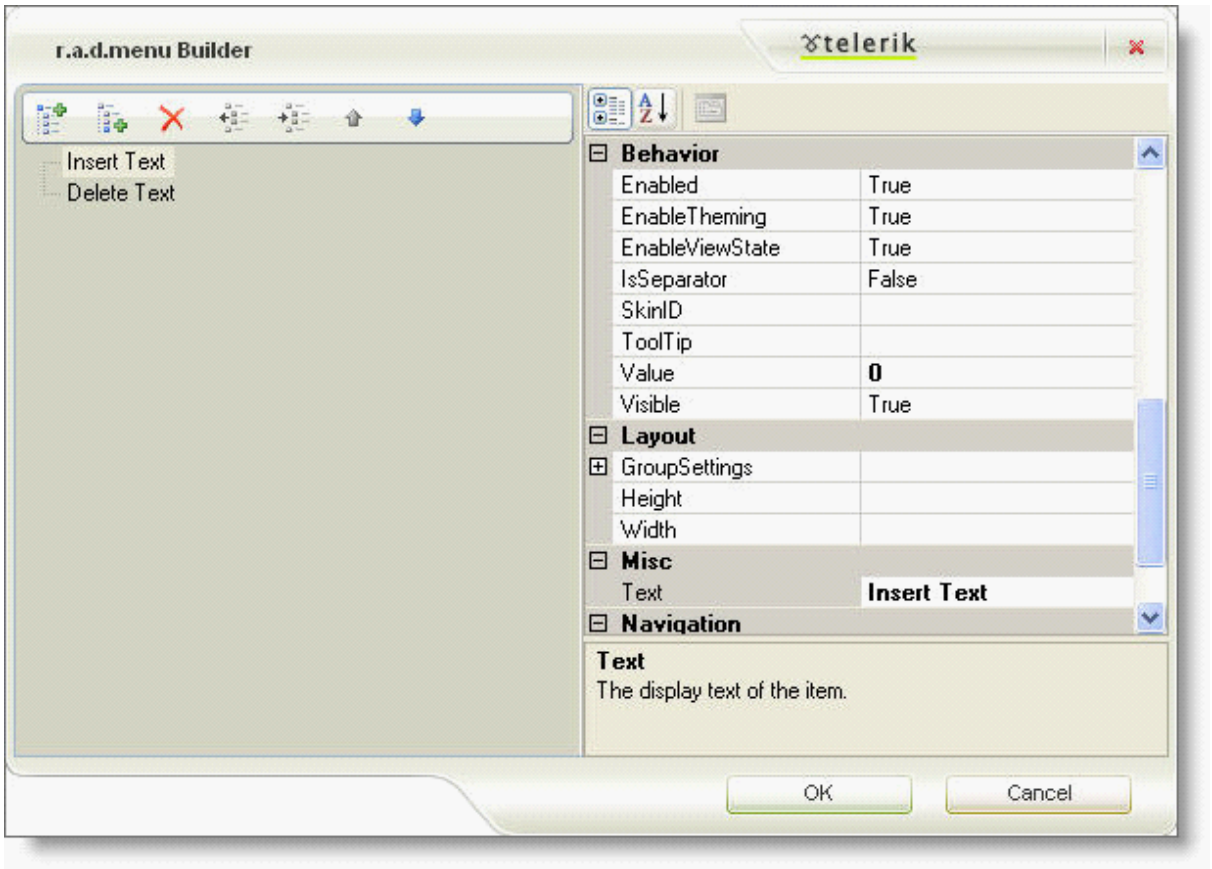
This lab demonstrates building a simple context RadMenu using the designer.

1. Create a new ASP.NET Project named ContextMenu.
2. Drop a TextBox on to the Web Form.
3. Drop a RadMenu on to the Web Form.
4. Click the Smart Tag button to invoke RadMenu Tasks as shown in the figure below.



RadMenu Tasks Window

5. Start Building the menu by selecting "Build RadMenu..." link which will invoke the r.a.d menu Builder dialog
6. Add the two root menu items as depicted in the Figure below. To change the title of the menu items you must change the Text property accordingly.



Adding two menus

7. Change the Value property for the "Insert Text" menu item to "0". Change the Value property for the "Delete Text" menu item to "1".

8. Press OK to close the dialog.

9. Set the RadMenu1.IsContext property to True in the Property Editor.

10. Set the RadMenu1.ContextMenuElementID property to "TextBox1" in the Property Editor.

11. From the Property Editor for RadMenu1, double-click on the ItemClick event to go to the Code Editor within the RadMenu1_ItemClick() event. Type the code as shown below:

```
C# Example:
protected void  RadMenu1_ItemClick( object  sender,
   Telerik.WebControls.RadMenuEventArgs  e  )
{
   RadMenuItem  ItemClicked  =  e.Item;

  if(ItemClicked.Value  ==   "0")
     TextBox1.Text = "Inserted  Text";
  else
     TextBox1.Text = "";
}

VB  Example:
Protected Sub  RadMenu1_ItemClick(ByVal  sender As Object,
 ByVal e As   Telerik.WebControls.RadMenuEventArgs)
```

```
 Dim  ItemClicked As RadMenuItem = e.Item
 If  ItemClicked.Value = "0" Then
    TextBox1.Text = "Inserted Text"
 Else
    TextBox1.Text = ""
 End If
End Sub
```

12. In this example, using the value held by the value property, you can determine the menu item that was invoked and perform the appropriate logic.

13. You can now run the program. Note how the AccessKey property denotes a keyboard shortcut for the control. Also, when running the application, you'll see that the image supplied as the ImageUrl property is displayed to the left of the menu.

2.2.3 Databinding with RadMenu

The RadMenu supports databinding to IListSource implementations such as DataSet, DataTable and DataView. It can also bind to IEnumerable and ICollection implementations such as ArrayLists.

To support hierarchical databinding, RadMenu can bind to a self-referencing table from which it obtains its hierarchical structure. The RadMenu automatically realizes the inherent hierarchy of hierarchical declarative data sources such as XmlDataSource and SiteMapDataSource.

For flat declarative data sources such as SqlDataSource, you specify the hierarchy using the ID -> ParentID relationship model. These are specified in the RadMenu.DataFieldID and RadMenu. DataFieldParentID properties.

Lab: Binding to xml Data

This lab shows how a RadMenu is bound to an XML data source. It also illustrates how the XML file sets properties such as the *NavigateUrl* property.

The RadMenu can easily bind to xml formatted data. This is particularly useful when the structure of the menu is relatively static, yet when change is required you need only modify the xml file.

This lab will use the xml file "menu.xml" shown in the listing below:

```
<!--menu.xml-->

<?xml  version="1.0"  encoding="utf-8"  ?>
<Menu>
   <Group  Flow="Horizontal">
     <Item  Text="Vendors"  Key="V" >
        <Group  OffsetX="-2"  OffsetY="2">
          <Item  Text="Telerik">
             <Group  Width="140"  Flow="Vertical"  OffsetX="10" >
                <Item  Text="Home"  NavigateUrl="http://www.telerik.com"/>
              <Item  Text="Products"
                   NaviageUrl="http://www.telerik.com/products/aspnet/overview.aspx"/>
              <Item  Text="Support"
                  NavigateUrl="http://www.telerik.com/support/home.aspx"/>
           </Group>
          </Item>
          <Item  Text="Microsoft" >
             <Group  Width="140"  Flow="Vertical"  OffsetX="10" >
                <Item  Text="Home"  NavigateUrl="http://www.microsoft.com" />
                 <Item  Text="MSDN"  NavigateUrl="http://msdn2.microsoft.com"/>
```

```
          </Group>
        </Item>
         <Item Text="PostBack Item" PostBack="True"/>
        <Item IsSeparator="True" />
         <Item Text="ToolTip Item" ToolTip="This is a tooltip" />
      </Group>
    </Item>
     <Item Text="Search" AccessKey="S" >
       <Group Flow="Vertical" OffsetY="10" >
          <Item Text="Google" NavigateUrl="http://www.google.com"/>
          <Item Text="Yahoo" NavigateUrl="http://www.yahoo.com" />
      </Group>
    </Item>
     <Item Text="News" AccessKey="N" >
       <Group Flow="Vertical" OffsetY="2">
          <Item Text="Fox News" NavigateUrl="http://www.foxnews.com" />
          <Item Text="CNN" NavigateUrl="http://www.cnn.com" />
          <Item Text="MSNBC" NavigateUrl="http://www.msnbc.com" />
      </Group>
    </Item>
  </Group>
</Menu>
```

1) Create a new project named XMLFile.

2) Add an XML file to the project and paste the menu.xml code from the above listing into this file.

3) Add a RadMenu from the ToolBox to the Web Form.

4) In the code behind, modify the Page_Load() method to contain the code shown below.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  if  (!Page.IsPostBack)
      RadMenu1.LoadContentFile( "~/menu.xml" );
}

VB Example:
protected Sub Page_Load(ByVal sender as object, ByVal e as EventArgs)
 if Not Page.IsPostBack Then
     RadMenu1.LoadContentFile("~/menu.xml")
 End if
End Sub
```

5) Run the application.

A few high points to notice in reviewing menu.xml:

- **Group Settings**: You can specify settings for a group if menu items by using the <Group> tag in the XML file. This actually corresponds to the GroupSettings property for the RadMenuItem (the RadMenuItemGroupSettings class). Other settings can be specified to change the appearance or behaviour of a group of items. For instance, the OffsetY and OffsetX properties are used to offset child menu items from their parent. You may explore other properties for groups by looking at the members for the RadMenuItemGroupSettings class.
- **Navigation**: The NavigateUrl property allows using the Menu to navigate to web pages or external URLS. Alternatively, you can navigate programmatically which is demonstrated in the "Programmatic Navigation" lab.

- **Postback**: You can enforce a postback operation by setting the Postback property to **true**. The default value is **false** unless the menu item has subscribed to an Item_Click event.

Lab: Binding to hierarchical database data

This lab demonstrates binding RadMenu to a database data source programmatically.

RadMenu has a series of properties that govern the associations between menu properties and data columns:

Property Name	Description
DataTextField	Used to bind the Text property to a DataColumn in the data source.
DataValueField	Used to bind the Value property to a DataColumn in the data source.
DataFieldID	Set this optional property to that of a DataColumn containing an ID.
DataFieldParentID	Set this optional property to that of a DataColumn containing the ParentID. When used, the DataFieldID and DataFieldParentID enforce the self-referencing model for hierarchical menu items.
DataSource	Set this mandatory field to the data source to which to bind.
DataNavigateUrlField	When used, binds the NavigateUrl property to a DataColumn in the data source.

1. Prepare data to be consumed by a data source. This lab uses a database within SQL Server or SQL Server Express (MSDE). The script in the listing below creates the required table for this lab. Run the script below to create a table "radmenu".

```
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [dbo].[radmenu](
       [Text] [varchar](50) COLLATE SQL_Latin1_General_CP1_CI_AS NOT NULL,
       [ID] [int] NOT NULL,
       [ParentID] [int] NULL,
       [NavigateUrl] [varchar] (50) COLLATE SQL_Latin1_General_CP1_CI_AS NULL
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
```

2. Run a second script to populate the table:

```
insert into radmenu (Text, ID) values ('Politics', 1)
insert into radmenu (Text, ID, ParentID, NavigateUrl)
```

```
   values ('CNN', 2, 1, 'http://www.cnn.com')
insert into radmenu (Text, ID, ParentID, NavigateUrl)
   values ('NBC', 3, 1, 'http://www.nbc.com')
insert into radmenu (Text, ID, ParentID, NavigateUrl)
   values ('ABC', 4, 1, 'http://www.abc.com')
insert into radmenu (Text, ID) values ('Sports', 5)
insert into radmenu (Text, ID, ParentID, NavigateUrl)
   values ('ESPN', 6, 5, 'http://www.espn.com')
insert into radmenu (Text, ID, ParentID, NavigateUrl)
   values ('USA Hockey', 7, 5, 'http://www.usahockey.com')
insert into radmenu (Text, ID) values ('Events', 8)
insert into radmenu (Text, ID, ParentID, NavigateUrl)
   values ('Oscar Awards', 9, 8, 'http://www.oscar.com')
insert into radmenu (Text, ID, ParentID, NavigateUrl)
   values ('MTV Movie Awards', 10, 8, 'http://www.mtv.com/ontv/movieawards')
```

3. Create a new Web Application project and name it DBDatabind.
4. Drop a RadMenu onto the Web Page Designer
5. Add the following using statement

```
using   System.Data.SqlClient;
```

6. Go to the Code View for the Page and add the code shown in the listing below to the Page_Load event

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  if(!Page.IsPostBack)
  {
     SqlConnection sqlCon = new SqlConnection( @"Data    Source=EBF_VELOCITY\SQLEXPRESS;Initial
     Catalog=telerik;User  ID=sa;Password=F@1afe1" );
     SqlDataAdapter sqlDa = new  SqlDataAdapter( "select * from radmenu", sqlCon );
     DataSet dsradMenu = new DataSet();
     sqlDa.Fill( dsradMenu );
     RadMenu1.DataTextField = "Text";
     RadMenu1.DataFieldID = "ID";
     RadMenu1.DataFieldParentID = "ParentID";
     RadMenu1.DataNavigateUrlField = "NavigateUrl";
     RadMenu1.DataSource = dsradMenu;
     RadMenu1.DataBind();
  }
}

VB Example:
protected Sub Page_Load(ByVal sender as object, ByVal e as EventArgs)
 if Not Page.IsPostBack Then
    Dim sqlCon as SqlConnection = new SqlConnection("Data Source=EBF_VELOCITY\SQLEXPRESS;Initial_
     Catalog=telerik;User  ID=sa;Password=F@1afe1")
   Dim sqlDa as SqlDataAdapter = new SqlDataAdapter("select * from radmenu", sqlCon)
   Dim dsradMenu as DataSet = new DataSet
    sqlDa.Fill(dsradMenu)
    RadMenu1.DataTextField = "Text"
    RadMenu1.DataFieldID = "ID"
    RadMenu1.DataFieldParentID = "ParentID"
    RadMenu1.DataNavigateUrlField = "NavigateUrl"
    RadMenu1.DataSource = dsradMenu
```

```
    RadMenu1.DataBind
 End if
End Sub
```

7. When you run the application, you will see the RadMenu populated with hierarchical data as shown in the figure below.

RadMenu populated from database

2.2.4 Using RadMenu at Runtime

This section shows working with RadMenu at runtime. You will learn how to populate the RadMenu using some of the classes and API functions specific to RadMenu and how to respond to server side events.

Lab: Populating the RadMenu

This lab demonstrates populating RadMenu on-the-fly for those instances where RadMenu contents are not static and are determined at runtime.

The RadMenuItem class represents a single item in the RadMenu. The entire RadMenu is composed of a hierarchical list of RadMenuItems. The lab application will dynamically populate a RadMenu in code-behind using the constructor and properties of the RadMenuItem class.

1. Create a new web application project named RuntimeProp.

2. Add a RadMenu from the ToolBox to the Web Form.

3. Add a using/Imports statement to make RadMenu and RadMenuItem classes accessible in code:

```
C# Example:
using Telerik.WebControls;

VB Example:
Imports Telerik.WebControls
```

4. Also in the code behind, add the following code to the Page_Load event handler.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  // Create first root item
   RadMenuItem rmItem = new RadMenuItem();
   rmItem.Text = "Politics";
   rmItem.CssClass = "MainItem";
```

```
    RadMenu1.Items.Add( rmItem );

    // Add sub-items
    RadMenuItem rmSubItem = new RadMenuItem();
    rmSubItem.Text = "CNN";
    rmSubItem.NavigateUrl = "http://www.cnn.com";
    rmItem.Items.Add( rmSubItem );

    rmSubItem = new RadMenuItem();
    rmSubItem.Text = "NBC";
    rmSubItem.NavigateUrl = "http://www.nbc.com";
    rmItem.Items.Add( rmSubItem );

    rmSubItem = new RadMenuItem();
    rmSubItem.Text = "ABC";
    rmSubItem.NavigateUrl = "http://www.abc.com";
    rmItem.Items.Add( rmSubItem );

    // Create second root item
    rmItem = new RadMenuItem();
    rmItem.Text = "Sports";
    rmItem.CssClass = "MainItem";

    RadMenu1.Items.Add( rmItem );

    // Add sub-items
    rmSubItem = new RadMenuItem();
    rmSubItem.Text = "ESPN";
    rmSubItem.NavigateUrl = "http://www.espn.com";
    rmItem.Items.Add( rmSubItem );

    rmSubItem = new RadMenuItem();
    rmSubItem.Text = "USA Hockey";
    rmSubItem.NavigateUrl = "http://www.usahockey.com";
    rmItem.Items.Add( rmSubItem );

    // Create third root item
    rmItem = new RadMenuItem();
    rmItem.Text = "Events";
    rmItem.CssClass = "MainItem";

    RadMenu1.Items.Add( rmItem );

    // Add sub-items
    rmSubItem = new RadMenuItem();
    rmSubItem.Text = "Oscar Awards";
    rmSubItem.NavigateUrl = "http://www.oscar.com";
    rmItem.Items.Add( rmSubItem );

    rmSubItem = new RadMenuItem();
    rmSubItem.Text = "MTV Movie Awards";
    rmSubItem.NavigateUrl = "http://www.mtv.com";
    rmSubItem.Enabled = false;  // Disable this item.
    rmItem.Items.Add( rmSubItem );
}
```

VB: Example:

```
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 Dim rmItem As RadMenuItem = New RadMenuItem
 rmItem.Text = "Politics"
 rmItem.CssClass = "MainItem"
```

```
  RadMenu1.Items.Add(rmItem)
Dim  rmSubItem As  RadMenuItem  = New  RadMenuItem
 rmSubItem.Text  =  "CNN"
  rmSubItem.NavigateUrl  =  "http://www.cnn.com"
   rmItem.Items.Add(rmSubItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "NBC"
  rmSubItem.NavigateUrl  =  "http://www.nbc.com"
   rmItem.Items.Add(rmSubItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "ABC"
  rmSubItem.NavigateUrl  =  "http://www.abc.com"
   rmItem.Items.Add(rmSubItem)
 rmItem  = New  RadMenuItem
 rmItem.Text  =  "Sports"
 rmItem.CssClass  =  "MainItem"
  RadMenu1.Items.Add(rmItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "ESPN"
  rmSubItem.NavigateUrl  =  "http://www.espn.com"
   rmItem.Items.Add(rmSubItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "USA  Hockey"
  rmSubItem.NavigateUrl  =  "http://www.usahockey.com"
   rmItem.Items.Add(rmSubItem)
 rmItem  = New  RadMenuItem
 rmItem.Text  =  "Events"
 rmItem.CssClass  =  "MainItem"
  RadMenu1.Items.Add(rmItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "Oscar  Awards"
  rmSubItem.NavigateUrl  =  "http://www.oscar.com"
   rmItem.Items.Add(rmSubItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "MTV  Movie  Awards"
  rmSubItem.NavigateUrl  =  "http://www.mtv.com"
  rmSubItem.Enabled  = False
   rmItem.Items.Add(rmSubItem)
End Sub
```

5. The listing above illustrates the process of dynamically populating a RadMenu. Using this approach you can deterministically set properties to values based on some external criteria. For example, you can determine whether or not to add a particular item to the RadMenuItem based on a user's access rights.

Lab: Handling Server Events

This lab explores three server side events available to the RadMenu control:

Event	Description
ItemClick	This event is fired when the user clicks on a menu item (a menu item postback).
ItemCreated	This event is fired whenever a new menu item is added to the RadMenu.
ItemDataBound	This event is fired whenever the RadMenu is being bound to a datasource.

Handling the ItemClick Event

1. Create a new project and name it ServerEvents.
2. Drop a RadMenu control on to the WebForm for this project.
3. Drop a standard Label control on to the WebForm for this project.
4. Add the CreateTable() method shown in the listing below to the code-behind Page class for this Web Form

```
C# Example:
private DataTable CreateTable()
{
   DataTable tbl = new DataTable();
    tbl.Columns.Add( "ID" );
    tbl.Columns.Add( "Text" );
    tbl.Columns.Add( "Value" );
    tbl.Columns.Add( "ParentID" );
    tbl.Columns.Add( "NavigateUrl" );

    tbl.Rows.Add( "1", "News", "You clicked News", null );
    tbl.Rows.Add( "2", "Sports", "You clicked Sports", null );
    tbl.Rows.Add( "3", "Events", "You clicked Events", null );

    tbl.Rows.Add( "4", "CNN", "You clicked CNN", 1, "http://www.cnn.com" );
    tbl.Rows.Add( "5", "NBC", "You clicked NBC", 1, "http://www.nbc.com" );
    tbl.Rows.Add( "6", "Fox", "You clicked Fox", 1, "http://www.fox.com" );

    tbl.Rows.Add( "7", "ESPN", "You clicked ESPN", 2, "http://www.espn.com" );
    tbl.Rows.Add( "8", "USA Hockey", "You clicked USA Hockey", 2, "http://www.usahockey.com" );

    tbl.Rows.Add( "9", "Oscar Awards", "You clicked Oscar Awards", 3, "http://www.oscars.com" );
    tbl.Rows.Add( "10", "MTV Movie Awards", "You clicked MTV Movie Awards", 3, "http://www.mtv.com"

   return tbl;
}

VB Example:
Private Function CreateTable() As DataTable
 Dim tbl As DataTable = New DataTable
   tbl.Columns.Add("ID")
   tbl.Columns.Add("Text")
   tbl.Columns.Add("Value")
   tbl.Columns.Add("ParentID")
   tbl.Columns.Add("NavigateUrl")
 tbl.Rows.Add("1", "News", "You clicked News", Nothing)
 tbl.Rows.Add("2", "Sports", "You clicked Sports", Nothing)
 tbl.Rows.Add("3", "Events", "You clicked Events", Nothing)
 tbl.Rows.Add("4", "CNN", "You clicked CNN", 1, "http://www.cnn.com")
 tbl.Rows.Add("5", "NBC", "You clicked NBC", 1, "http://www.nbc.com")
 tbl.Rows.Add("6", "Fox", "You clicked Fox", 1, "http://www.fox.com")
 tbl.Rows.Add("7", "ESPN", "You clicked ESPN", 2, "http://www.espn.com")
 tbl.Rows.Add("8", "USA Hockey", "You clicked USA Hockey", 2, "http://www.usahockey.com")
 tbl.Rows.Add("9", "Oscar Awards", "You clicked Oscar Awards", 3, "http://www.oscars.com")
 tbl.Rows.Add("10", "MTV Movie Awards", "You clicked MTV Movie Awards", 3, "http://www.mtv.com")
 Return tbl
End Function
```

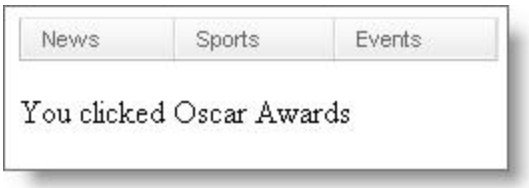
5. Add the code to the Page_Load() event shown in the listing below. Notice that the line assigning a value to the DataNavigateUrlField property is commented out.

```
C# Example:
protected void Page_Load( object sender, EventArgs e )
{
   RadMenu1.DataSource = CreateTable();
   RadMenu1.DataFieldID = "ID";
   RadMenu1.DataTextField = "Text";
   RadMenu1.DataValueField = "Value";
   RadMenu1.DataFieldParentID = "ParentID";
//      RadMenu1.DataNavigateUrlField = "NavigateUrl";
   RadMenu1.DataBind();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  RadMenu1.DataSource = CreateTable
  RadMenu1.DataFieldID = "ID"
  RadMenu1.DataTextField = "Text"
  RadMenu1.DataValueField = "Value"
  RadMenu1.DataFieldParentID = "ParentID"
  RadMenu1.DataBind
End Sub
```

6. Add an ItemClick event to the RadMenu. You can do this through the Properties Window in Visual Studio. Add the code shown in the listing below to the `RadMenu1_ItemClick()` event that is created for you.

```
C# Example:
protected void RadMenu1_ItemClick( object sender,
   Telerik.WebControls.RadMenuEventArgs e )
{
   Label1.Text = e.Item.Value;
}

VB Example:
Protected Sub RadMenu1_ItemClick(ByVal sender As Object,
ByVal e As   Telerik.WebControls.RadMenuEventArgs)
 Label1.Text = e.Item.Value
End Sub
```

7. Run the application. When you select a menu item, the label will display the value of the DataValueField to which the e.Item.Value property refers as shown in the figure below.



Value of DataValueField

8. Now uncomment the line in the Page_Load() method shown below.

```
//      RadMenu1.DataNavigateUrlField = "NavigateUrl";
```

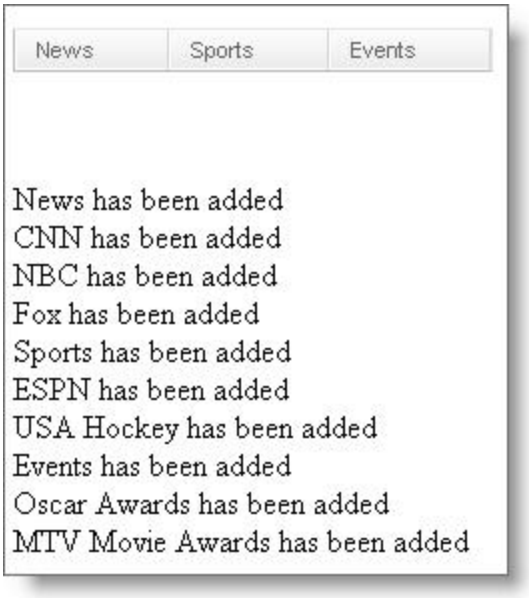
This will cause the url stored in the NavigateUrl field in the table to be assigned to the NavigateUrl property of the RadMenuItem. When you run the application, the Item_Click() will not fire. Instead, the browser will navigate to the respective url.

Handling the ItemCreated Event

1. In the same project, add another Label control to the WebForm and clear its Text property. Place this label somewhere underneath the previous Label.
2. Add the ItemCreated event to the RadMenu. You can do this through the Properties Window in Visual Studio. Add the code shown in the listing below to the RadMenu1_ItemCreated() method that was created for you.

```
C# Example:
protected void  RadMenu1_ItemCreated( object  sender,
   Telerik.WebControls.RadMenuEventArgs  e  )
{
   Label2.Text  +=  e.Item.Text  +  " has been added" + "<br/>";
}

VB Example:
Protected Sub  RadMenu1_ItemCreated(ByVal  sender As Object,
ByVal e As   Telerik.WebControls.RadMenuEventArgs)
 Label2.Text  +=  e.Item.Text  +  " has been added" + "<br/>"
End Sub
```

3. When you run the application you should see something like that shown in the figure below



Results of Item_Created() event

Handling the ItemDataBound Event

The ItemDataBound event is fired for each item in a data source being bound to the RadMenu. This event gives you the opportunity to handle any processing prior to the individual items. The item being bound is accessible through the DataItem property of the RadMenuEventArgs argument. You must cast DataItem to the appropriate type, in this case a DataRowView since this example illustrates binding to a

DataTable.

1. Add the ItemDataBound event to the RadMenu through the Properties Window in Visual Studio. Add the code shown in the listing below to the RadMenu1_ItemDataBound method that was created for you.

```
C# Example:
protected void RadMenu1_ItemDataBound( object sender,
   Telerik.WebControls.RadMenuEventArgs e )
{
   DataRowView row = (DataRowView)e.Item.DataItem;
   e.Item.Text = row["Text"].ToString() + " from IDB";
}

VB Example:
Protected Sub RadMenu1_ItemDataBound(ByVal sender As Object,
ByVal e As Telerik.WebControls.RadMenuEventArgs)
 Dim row As DataRowView = CType(e.Item.DataItem, DataRowView)
 e.Item.Text = row("Text").ToString + " from IDB"
End Sub
```

2. When you run the application, you will see that the text for the menu items is change that specified in the event shown in the figure below.

Result of ItemDataBound event

Lab: Programmatic Navigation

This lab will teach you how to programmatically effect navigation using the RadMenu control. There will be times when not every menu will navigate to a specific url and you will have to perform operations based on the menu selected. In this lab, you will see how to bind the DataValueField to a field in the database. You will then be able to query this field to determine the appropriate action to take. This lab will also illustrate how to prevent postback on a specific menu item.

1. Create a new project and name it ProgNav.
2. Drop a RadMenu control on to the WebForm for this project.
3. Drop a standard Label control on to the WebForm for this project.
4. Add the CreateTable() method shown in the listing below to the code-behind Page class for this Web Form:

```
C# Example:
private DataTable CreateTable()
{
   DataTable tbl = new DataTable();
    tbl.Columns.Add( "ID" );
    tbl.Columns.Add( "Text" );
    tbl.Columns.Add( "Value" );
```

```
   tbl.Columns.Add( "ParentID" );

   tbl.Rows.Add( "1", "News", null, null );
   tbl.Rows.Add( "2", "Sports", null, null );
   tbl.Rows.Add( "3", "Events", null, null );

   tbl.Rows.Add( "4", "CNN", "http://www.cnn.com", 1 );
   tbl.Rows.Add( "5", "NBC", "http://www.nbc.com", 1 );
   tbl.Rows.Add( "6", "Fox", "http://www.fox.com", 1 );

   tbl.Rows.Add( "7", "ESPN", "http://www.espn.com", 2 );
   tbl.Rows.Add( "8", "USA  Hockey", "http://www.usahockey.com", 2 );

   tbl.Rows.Add( "9", "Oscar  Awards", "http://www.oscars.com", 3 );
   tbl.Rows.Add( "10", "MTV  Movie  Awards", "http://www.mtv.com", 3 );

  return tbl;
}
```

VB Example:

```
Private Function  CreateTable() As  DataTable
 Dim tbl As DataTable = New  DataTable
  tbl.Columns.Add("ID")
  tbl.Columns.Add("Text")
  tbl.Columns.Add("Value")
  tbl.Columns.Add("ParentID")
 tbl.Rows.Add("1",  "News", Nothing, Nothing)
 tbl.Rows.Add("2",  "Sports", Nothing, Nothing)
 tbl.Rows.Add("3",  "Events", Nothing, Nothing)
 tbl.Rows.Add("4",  "CNN",  "http://www.cnn.com",  1)
 tbl.Rows.Add("5",  "NBC",  "http://www.nbc.com",  1)
 tbl.Rows.Add("6",  "Fox",  "http://www.fox.com",  1)
 tbl.Rows.Add("7",  "ESPN",  "http://www.espn.com",  2)
 tbl.Rows.Add("8",  "USA  Hockey",  "http://www.usahockey.com",  2)
 tbl.Rows.Add("9",  "Oscar  Awards",  "http://www.oscars.com",  3)
 tbl.Rows.Add("10",  "MTV  Movie  Awards",  "http://www.mtv.com",  3)
 Return tbl
End Function
```

5. Add the code to the Page_Load() event shown in the listing below.

C# Example:

```
protected void  Page_Load( object  sender,  EventArgs  e  )
{
   RadMenu1.DataSource  =  CreateTable();
   RadMenu1.DataFieldID  = "ID";
   RadMenu1.DataTextField  = "Text";
   RadMenu1.DataValueField  = "Value";
   RadMenu1.DataFieldParentID  = "ParentID";
   RadMenu1.DataBind();
}
```

VB Example:

```
Protected Sub  Page_Load(ByVal  sender As Object, ByVal e As  EventArgs)
  RadMenu1.DataSource  =  CreateTable
  RadMenu1.DataFieldID  =  "ID"
  RadMenu1.DataTextField  =  "Text"
  RadMenu1.DataValueField  =  "Value"
  RadMenu1.DataFieldParentID  =  "ParentID"
  RadMenu1.DataBind
```

```
End Sub
```

In page load you assign the url to the Value field which will be bound to the DataValueField. This field has no effect on the navigation of the menu. Therefore, you will have to force navigation using the Response.Redirect() method within the ItemClick event handler.

6. Add an ItemClick event to the RadMenu. You can do this through the Properties Window in Visual Studio. Add the code shown in the listing below to the RadMenu1_ItemClick() event that is created for you.

```
C# Example:
protected void  RadMenu1_ItemClick( object  sender,
   Telerik.WebControls.RadMenuEventArgs  e  )
{
  if(e.Item.Value  != null)
  {
    string  url  =  e.Item.Value;
     Response.Redirect( url );
  }
}

VB Example:
Protected Sub  RadMenu1_ItemClick(ByVal  sender As Object,
ByVal e As   Telerik.WebControls.RadMenuEventArgs)
 If Not  (e.Item.Value  Is Nothing) Then
   Dim url As String  =  e.Item.Value
    Response.Redirect(url)
 End If
End Sub
```

The RadMenu ItemClick evaluates the contents of the RadmenuEventArgs.Item.Value property to determine whether it contains a url. If true, it invokes the Response.Redirect() method.

7. Run the application to test the functionality.

Preventing Postback

You may have noticed that when you clicked on one of the top level menu items, a postback (round trip) occurs, even though there is no url. To prevent a postback to specific menu items, set the PostBack property to false for those RadMenuItems. You can do this within the ItemDataBound event:

```
C# Example:
protected void  RadMenu1_ItemDataBound( object  sender,
   Telerik.WebControls.RadMenuEventArgs  e  )
{
   DataRowView  row  =  (DataRowView)e.Item.DataItem;
  if( row["Value"] is  System.DBNull  )
    e.Item.PostBack = false;
}

VB Example:
Protected Sub  RadMenu1_ItemDataBound(ByVal  sender As Object,
ByVal e As   Telerik.WebControls.RadMenuEventArgs)
 Dim row As  DataRowView  =  CType(e.Item.DataItem,  DataRowView)
 If TypeOf  row("Value")  Is  System.DBNull Then
   e.Item.PostBack = False
 End If
End Sub
```

The ItemDataBound handler evaluates the bindable content to determine which properties to effect on the resulting menu item. In this case, if there is no url, the PostBack property is set to false to prevent a round-trip for the specific menu item in question.

Lab: Customized Menu Construction

This lab demonstrates programmatically creating a menu, given a real world scenario where the menu definition is acquired through an external source. This definition may be in a format not compatible with the RadMenu. Therefore, you must write code to read and transform the menu definition to that which the RadMenu can use. In this example, we will map our non-compatible menu format to valid xml format for the RadMenu rather than calling the RadMenu API functions. This is simply to illustrate an alternative means to constructing the menu. This lab also demonstrates:

- Navigating the xml menu files using XPATH search.
- Using the originating menu definition as a super-set to be evaluated against user rights to determine the resulting menu. This will likewise be simulated in this exercise.

1. Create a new web solution and name it CustomCreate.
2. Drop a RadMenu control onto the web form.
3. Add a new item to your project and select XML File from within the Add New Item dialog. Save it as SuperMenu.xml.
4. Copy and Paste the following XML content into the new XML file.

```
<?xml version="1.0" encoding="utf-8" ?>
<Modules>
   <Contacts>
      <Item DisplayName="Add Contact" FunctionID="10"/>
      <Item DisplayName="Delete Contact" FunctionID="11"/>
      <Item DisplayName="Edit Contact" FunctionID="12"/>
   </Contacts>
   <Calendar>
      <Item DisplayName="Add Event" FunctionID="20"/>
      <Item DisplayName="Delete Event" FunctionID="21"/>
      <Item DisplayName="Edit Event" FunctionID="22"/>
   </Calendar>
   <Journal>
      <Item DisplayName="Add Entry" FunctionID="30"/>
      <Item DisplayName="Delete Entry" FunctionID="31"/>
      <Item DisplayName="Edit Entry" FunctionID="32"/>
   </Journal>
   <Tasks>
      <Item DisplayName="Add Task" FunctionID="40"/>
      <Item DisplayName="Delete Task" FunctionID="41"/>
      <Item DisplayName="Edit Task" FunctionID="42"/>
      <Item DisplayName="Task 01" FunctionID="43"/>
      <Item DisplayName="Task 02" FunctionID="44"/>
      <Item DisplayName="Task 03" FunctionID="45"/>
      <Item DisplayName="Task 04" FunctionID="46"/>
      <Item DisplayName="Task 05" FunctionID="47"/>
      <Item DisplayName="Task 06" FunctionID="48"/>
      <Item DisplayName="Task 07" FunctionID="49"/>
      <Item DisplayName="Task 08" FunctionID="401"/>
```

```
      <Item DisplayName="Task 09" FunctionID="402"/>
      <Item DisplayName="Task 10" FunctionID="403"/>
      <Item DisplayName="Task 11" FunctionID="404"/>
      <Item DisplayName="Task 12" FunctionID="405"/>
      <Item DisplayName="Task 13" FunctionID="406"/>
      <Item DisplayName="Task 14" FunctionID="407"/>
      <Item DisplayName="Task 15" FunctionID="408"/>
      <Item DisplayName="Task 16" FunctionID="409"/>
      <Item DisplayName="Task 17" FunctionID="410"/>
      <Item DisplayName="Task 18" FunctionID="411"/>
   </Tasks>
</Modules>
```

The above XML content represents the definition of a menu but it does not conform to the format expected by RadMenu. The 2nd level elements (Contacts, Calendar, etc) represent what will be top level menu items. Their children represent the sub-level menu for each top-level menu item. The DisplayName attribute represents what is displayed on the menu. The FunctionID represents a identifying value for a application level function. A typical usage of this is where the FunctionID here corresponds to some system FunctionID list (perhaps from a database). You would then determine whether the current user has access to this function. If not, the menu is not displayed. We will simulate this scenario in this example.

5. Now go to the code-behind and add the following using declaration:

```
C# Example:
using System.Xml;

VB Example:
Imports System.Xml
```

6. To simulate the process of determining whether a user has access to a function (menu item), create the following method for the web page. This function simply performs a modulus operation to determine which functions not to display.

```
C# Example:
private bool HasAccess( int FunctionID )
{
   return ( FunctionID % 3 != 0 );
}

VB Example:
Private Function HasAccess(ByVal FunctionID As Integer) As Boolean
 Return (Not (FunctionID Mod 3 = 0))
End Function
```

7. Create another method named ConstructMenu(). It's contents are displayed in the listing below.

```
C# Example
private void ConstructMenu()
{

  // Create and initialize xml representation of the menu.
  xmlMenu = new XmlDocument();
   XmlDeclaration xmlDeclaration = xmlMenu.CreateXmlDeclaration( "1.0", "utf-8", null );
   XmlElement rootNode = xmlMenu.CreateElement( "Menu" );
   XmlElement grNode = xmlMenu.CreateElement( "Group" );
   grNode.SetAttribute( "Flow", "Horizontal" );
```

```
   rootNode.AppendChild( grNode );
    xmlMenu.InsertBefore( xmlDeclaration, xmlMenu.DocumentElement );
   xmlMenu.AppendChild( rootNode );

  // Load the xml file which defines the menu, but is not suitable for the RadMenu.
  // This file simulates a menu definition obtained from an external source.
   XmlDocument xmlDoc = new XmlDocument();
   xmlDoc.Load( MapPath("SuperMenu.xml") );
   XmlNodeList xnl = xmlDoc.SelectNodes( "Modules/*" );

  // Add the top level menu items. Then create a group to contain the sublevel items.
  foreach(XmlNode xn in xnl)
  {

      XmlElement parentNode = xmlMenu.CreateElement( "Item" );
      parentNode.SetAttribute( "PostBack", "False" );
      parentNode.SetAttribute( "Text", xn.Name );
       xmlMenu.DocumentElement.AppendChild( parentNode );

      XmlElement groupNode = xmlMenu.CreateElement( "Group" );
      groupNode.SetAttribute( "OffsetX", "10" );
      groupNode.SetAttribute( "Height", "100" );
      parentNode.AppendChild( groupNode );


    // Now add the children to this menu
    foreach(XmlNode xnc in xn.ChildNodes)
    {
      int FunctionID = Convert.ToInt32 (xnc.Attributes["FunctionID"].Value) ;
      if(HasAccess( FunctionID ))
      {
          XmlElement childNode = xmlMenu.CreateElement( "Item" );
         childNode.SetAttribute( "Text", xnc.Attributes["DisplayName"].Value );
          groupNode.AppendChild( childNode );
      }
    }

  }
  // Assign the resulting Xml formatted string to the RadMenu
     RadMenu1.LoadXmlString(xmlMenu.InnerXml);
}

VB Example:
Private Sub ConstructMenu()
 xmlMenu = New XmlDocument
 Dim xmlDeclaration As XmlDeclaration = _
     xmlMenu.CreateXmlDeclaration("1.0", "utf-8", Nothing)
 Dim rootNode As XmlElement = xmlMenu.CreateElement("Menu")
 Dim grNode As XmlElement = xmlMenu.CreateElement("Group")
  grNode.SetAttribute("Flow", "Horizontal")
   rootNode.AppendChild(grNode)
   xmlMenu.InsertBefore(xmlDeclaration, xmlMenu.DocumentElement)
   xmlMenu.AppendChild(rootNode)
 Dim xmlDoc As XmlDocument = New XmlDocument
    xmlDoc.Load(MapPath("SuperMenu.xml"))
 Dim xnl As XmlNodeList = xmlDoc.SelectNodes("Modules/*")
 For Each xn As XmlNode In xnl
   Dim parentNode As XmlElement = xmlMenu.CreateElement("Item")
     parentNode.SetAttribute("PostBack", "False")
     parentNode.SetAttribute("Text", xn.Name)
      xmlMenu.DocumentElement.AppendChild(parentNode)
```

```
    Dim groupNode As XmlElement = xmlMenu.CreateElement("Group")
      groupNode.SetAttribute("OffsetX", "10")
      groupNode.SetAttribute("Height", "100")
      parentNode.AppendChild(groupNode)
    For Each xnc As XmlNode In xn.ChildNodes
      Dim FunctionID As Integer = Convert.ToInt32(xnc.Attributes("FunctionID").Value)
      If HasAccess(FunctionID) Then
        Dim childNode As XmlElement = xmlMenu.CreateElement("Item")
           childNode.SetAttribute("Text", xnc.Attributes("DisplayName").Value)
          groupNode.AppendChild(childNode)
      End If
    Next
  Next
    RadMenu1.LoadXmlString(xmlMenu.InnerXml)
End Sub
```

ConstructMenu first creates and initializes an XmlDocument (xmlmenu) that will store an XML representation of the menu understood by the RadMenu control. Next it loads a second XmlDocument with the contents of SuperMenu.xml. SuperMenu.xml represents xml obtained from an external source. After loading SuperMenu.xml the XmlDocument nodes are iterated to build top-level and child-level menu items. Note that the inner foreach loop invokes the HasAccess() method to determine whether or not to include a child item. The final operation is to assign the resulting XML string to the RadMenu and this is done by invoking the RadMenu.LoadXmlString() function and passing the InnerXml property of the newly created XmlDocument.

8. Finally, call ConstructMenu() from within the PageLoad event handler:

```
C# Example:
protected void Page_Load( object sender, EventArgs e )
{
   ConstructMenu();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  ConstructMenu
End Sub
```

9. Run the Application.

2.2.5 Client Scripting with RadMenu

With the RadMenu client API you can use client side functions to avoid round trips to the server. This section demonstrates some of the client side features of the RadMenu control.

Lab: Responding to Client Events

This lab shows how to setup and to respond to events on the client. The RadMenu contains nine events to which you may respond. It contains three additional events specific to context menus:

Event	Description
OnClientLoad	This event is fired after the menu is fully loaded on the client.
OnClientMouseOver	This event is fired when the mouse moves over the menu item.

Event	Description
OnClientMouseOut	This event is fired when the mouse moves out of the menu item.
OnClientItemOpen	This event is fired when a group of child items is opened.
OnClientItemFocus	This event is fired when a menu item is selected using either the keyboard (the [TAB] or arrow keys) or by clicking it
OnClientItemBlur	This event is fired when a menu item loses focus as a result of the user pressing a key or clicking elsewhere on the page
OnClientItemClicking	This event is fired just before a menu item is clicked upon.
OnClientItemClicked	This event is fired just after a menu item is clicked upon.
OnClientItemClose	This event is fired just before a menu item is closed.
OnClientContextShowing	This event is fired before the context menu is displayed
OnClientContextShown	This event is fired after the context menu id displayed
OnClientContextHidden	This event is fired after the context menu is hidden.

1. Create a new project and name it ClientSide.
2. Add a RadMenu control to the WebPage.
3. Add a TextBox control to the WebPage.
4. Set the TextBox TextMode property to MultiLine.
5. Adjust the TextBox size to so that it looks similar to that shown in the figure below.

WebForm layout

6. Add the method shown below to the page class in the code-behind. This method initializes client events with client-side functions to be written later.

```
C# Example:
private void InitializeClientEvents()
{
   RadMenu1.OnClientMouseOut = "OnMouseOut";
   RadMenu1.OnClientMouseOver = "OnMouseOver";
   RadMenu1.OnClientItemOpen = "OnItemOpen";
   RadMenu1.OnClientItemClicking = "OnItemClicking";
   RadMenu1.OnClientItemClicked = "OnItemClicked";
   RadMenu1.OnClientItemClose = "OnItemClose";
   RadMenu1.OnClientItemFocus = "OnItemFocus";
   RadMenu1.OnClientItemBlur = "OnItemBlur";
}

VB Example:
Private Sub InitializeClientEvents()
  RadMenu1.OnClientMouseOut = "OnMouseOut"
  RadMenu1.OnClientMouseOver = "OnMouseOver"
  RadMenu1.OnClientItemOpen = "OnItemOpen"
  RadMenu1.OnClientItemClicking = "OnItemClicking"
  RadMenu1.OnClientItemClicked = "OnItemClicked"
  RadMenu1.OnClientItemClose = "OnItemClose"
  RadMenu1.OnClientItemFocus = "OnItemFocus"
  RadMenu1.OnClientItemBlur = "OnItemBlur"
End Sub
```

7. In the code-behind add the following using statement which will make the RadMenu and RadMenuItem classes accessible in your code :

```
C# Example:
using Telerik.WebControls;

VB Example:
Imports Telerik.WebControls
```

8. Also in the code behind, add the following code to the Page_Load event handler:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  // Create first root item
   RadMenuItem rmItem = new RadMenuItem();
   rmItem.Text = "Politics";
   rmItem.CssClass = "MainItem";
   RadMenu1.Items.Add( rmItem );

  // Add sub-items
   RadMenuItem rmSubItem = new RadMenuItem();
   rmSubItem.Text = "CNN";
   rmSubItem.NavigateUrl = "http://www.cnn.com";
   rmItem.Items.Add( rmSubItem );

   rmSubItem = new RadMenuItem();
   rmSubItem.Text = "NBC";
   rmSubItem.NavigateUrl = "http://www.nbc.com";
   rmItem.Items.Add( rmSubItem );
```

```
   rmSubItem = new  RadMenuItem();
   rmSubItem.Text  = "ABC";
    rmSubItem.NavigateUrl  = "http://www.abc.com";
    rmItem.Items.Add(  rmSubItem  );

  // Create  second  root  item
   rmItem = new  RadMenuItem();
   rmItem.Text  = "Sports";
    rmItem.CssClass  = "MainItem";

    RadMenu1.Items.Add(  rmItem  );

  // Add  sub-items
   rmSubItem = new  RadMenuItem();
   rmSubItem.Text  = "ESPN";
    rmSubItem.NavigateUrl  = "http://www.espn.com";
    rmItem.Items.Add(  rmSubItem  );

   rmSubItem = new  RadMenuItem();
   rmSubItem.Text  = "USA  Hockey";
    rmSubItem.NavigateUrl  = "http://www.usahockey.com";
    rmItem.Items.Add(  rmSubItem  );

  // Create  third  root  item
   rmItem = new  RadMenuItem();
   rmItem.Text  = "Events";
    rmItem.CssClass  = "MainItem";

    RadMenu1.Items.Add(  rmItem  );

  // Add  sub-items
   rmSubItem = new  RadMenuItem();
   rmSubItem.Text  = "Oscar  Awards";
    rmSubItem.NavigateUrl  = "http://www.oscar.com";
    rmItem.Items.Add(  rmSubItem  );

   rmSubItem = new  RadMenuItem();
   rmSubItem.Text  = "MTV  Movie  Awards";
    rmSubItem.NavigateUrl  = "http://www.mtv.com";
    rmSubItem.Enabled  = false;  // Disable  this  item.
    rmItem.Items.Add(  rmSubItem  );

     InitializeClientEvents();
}

VB  Example:
protected Sub  Page_Load(ByVal  sender As Object, ByVal e As  EventArgs)
 Dim  rmItem As  RadMenuItem  = New  RadMenuItem
 rmItem.Text  =  "Politics"
 rmItem.CssClass  =  "MainItem"
   RadMenu1.Items.Add(rmItem)
 Dim  rmSubItem As  RadMenuItem  = New  RadMenuItem
 rmSubItem.Text  =  "CNN"
  rmSubItem.NavigateUrl  =  "http://www.cnn.com"
   rmItem.Items.Add(rmSubItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "NBC"
  rmSubItem.NavigateUrl  =  "http://www.nbc.com"
   rmItem.Items.Add(rmSubItem)
 rmSubItem  = New  RadMenuItem
 rmSubItem.Text  =  "ABC"
```

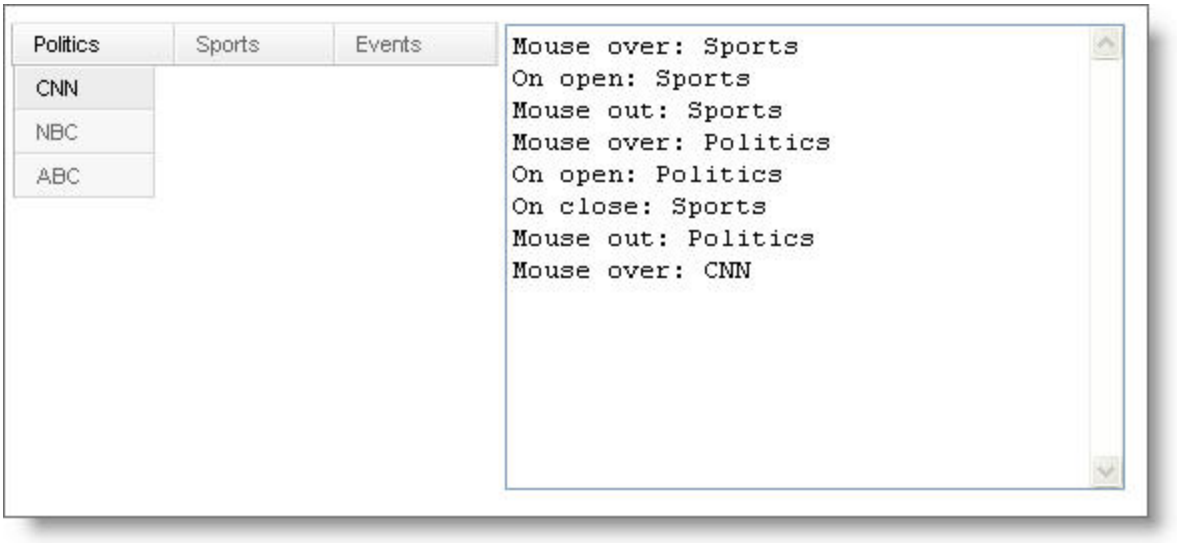
```
 rmSubItem.NavigateUrl  =  "http://www.abc.com"
  rmItem.Items.Add(rmSubItem)
 rmItem = New  RadMenuItem
 rmItem.Text  =  "Sports"
 rmItem.CssClass  =  "MainItem"
  RadMenu1.Items.Add(rmItem)
 rmSubItem = New  RadMenuItem
 rmSubItem.Text  =  "ESPN"
 rmSubItem.NavigateUrl  =  "http://www.espn.com"
  rmItem.Items.Add(rmSubItem)
 rmSubItem = New  RadMenuItem
 rmSubItem.Text  =  "USA  Hockey"
 rmSubItem.NavigateUrl  =  "http://www.usahockey.com"
  rmItem.Items.Add(rmSubItem)
 rmItem = New  RadMenuItem
 rmItem.Text  =  "Events"
 rmItem.CssClass  =  "MainItem"
  RadMenu1.Items.Add(rmItem)
 rmSubItem = New  RadMenuItem
 rmSubItem.Text  =  "Oscar  Awards"
 rmSubItem.NavigateUrl  =  "http://www.oscar.com"
  rmItem.Items.Add(rmSubItem)
 rmSubItem = New  RadMenuItem
 rmSubItem.Text  =  "MTV  Movie  Awards"
 rmSubItem.NavigateUrl  =  "http://www.mtv.com"
 rmSubItem.Enabled  = False
  rmItem.Items.Add(rmSubItem)
  InitializeClientEvents
End Sub
```

9. Add the <script> block shown in the listing below to the <head> section of the WebForm as shown. This listing contains a function for each client-side event. When the event is fired, it invokes a LogEvent() function which writes information about that event to the TextBox.

```
<head  runat="server">
   <script  type="text/javascript">
     function  OnMouseOver(sender,  eventArgs)
    {
         LogEvent("Mouse  over:  " + eventArgs.Item.Text);
    }

     function  OnMouseOut(sender,  eventArgs)
    {
         LogEvent("Mouse  out:  " + eventArgs.Item.Text);
    }

     function  OnItemClicking(sender,  eventArgs)
    {
         LogEvent("On  clicking:  " + eventArgs.Item.Text);

    }

     function  OnItemClicked(sender,  eventArgs)
    {
         LogEvent("On  clicked:  " + eventArgs.Item.Text);
    }

     function  OnItemOpen(sender,  eventArgs)
    {
         LogEvent("On  open:  " + eventArgs.Item.Text);
```

```
    }

     function OnItemClose(sender, eventArgs)
    {
         LogEvent("On close: " + eventArgs.Item.Text);
    }

     function OnItemFocus(sender, eventArgs)
    {
         LogEvent("On focus: " + eventArgs.Item.Text);
    }

     function OnItemBlur(sender, eventArgs)
    {
         LogEvent("On blur: " + eventArgs.Item.Text);
    }

    function ClearLog()
    {
          var log = document.getElementById("Textbox1");

        log.value = "";
    }

    function LogEvent(text)
    {
          var tb = document.getElementById("Textbox1");

        if (tb)
        {
            tb.value += text + "\n";
             tb.scrollTop = tb.scrollHeight;
        }
    }

 </script>
</head>
```

10.Run the application to examine the client-side functionality.

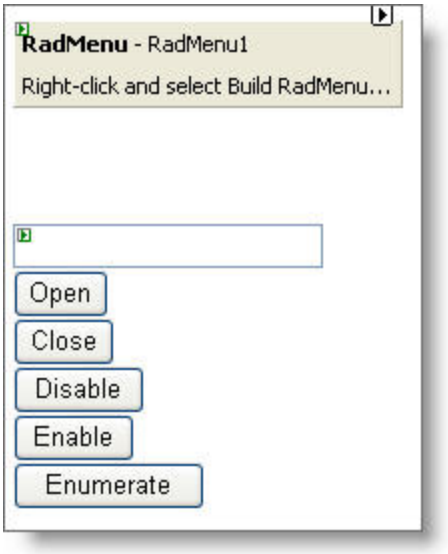


Running application

Lab: Using the Client API

This lab shows how to interact with the client API for the RadMenu. In the first part of this example, you find a specific menu item on the form, then use client code to interact with it. Additionally, you will learn how to iterate through all items on the menu.

1. Create a new project and name it ClientAPI.
2. Drop a RadMenu control on to the WebForm for this project.
3. Add a TextBox control on to WebForm.
4. Add four HTML Buttons to the second column and change their value properties to: "Expand" "Collapse", "Enable" and "Disable"
5. Your form should appear as that shown in the Figure below



WebForm in the designer

6. In the code-behind add the following using statement which will make the RadMenu and RadMenuItem classes accessible in your code:

```
C# Example:
using Telerik.WebControls;

VB Example:
Imports Telerik.WebControls
```

7. Add the CreateTable() method shown in the listing below to the code-behind Page class for this Web Form

```
C# Example:
private DataTable CreateTable()
{
   DataTable tbl = new DataTable();
    tbl.Columns.Add( "ID" );
    tbl.Columns.Add( "Text" );
    tbl.Columns.Add( "Value" );
    tbl.Columns.Add( "ParentID" );

    tbl.Rows.Add( "1", "News", null, null );
    tbl.Rows.Add( "2", "Sports", null, null );
    tbl.Rows.Add( "3", "Events", null, null );

    tbl.Rows.Add( "4", "CNN", "http://www.cnn.com", 1 );
    tbl.Rows.Add( "5", "NBC", "http://www.nbc.com", 1 );
    tbl.Rows.Add( "6", "Fox", "http://www.fox.com", 1 );

    tbl.Rows.Add( "7", "ESPN", "http://www.espn.com", 2 );
    tbl.Rows.Add( "8", "USA Hockey", "http://www.usahockey.com", 2 );

    tbl.Rows.Add( "9", "Oscar Awards", "http://www.oscars.com", 3 );
    tbl.Rows.Add( "10", "MTV Movie Awards", "http://www.mtv.com", 3 );

   return tbl;
}
```

```
VB Example:
Private Function CreateTable() As DataTable
 Dim tbl As DataTable = New DataTable
  tbl.Columns.Add("ID")
  tbl.Columns.Add("Text")
  tbl.Columns.Add("Value")
  tbl.Columns.Add("ParentID")
 tbl.Rows.Add("1", "News", Nothing, Nothing)
 tbl.Rows.Add("2", "Sports", Nothing, Nothing)
 tbl.Rows.Add("3", "Events", Nothing, Nothing)
 tbl.Rows.Add("4", "CNN", "http://www.cnn.com", 1)
 tbl.Rows.Add("5", "NBC", "http://www.nbc.com", 1)
 tbl.Rows.Add("6", "Fox", "http://www.fox.com", 1)
 tbl.Rows.Add("7", "ESPN", "http://www.espn.com", 2)
 tbl.Rows.Add("8", "USA Hockey", "http://www.usahockey.com", 2)
 tbl.Rows.Add("9", "Oscar Awards", "http://www.oscars.com", 3)
 tbl.Rows.Add("10", "MTV Movie Awards", "http://www.mtv.com", 3)
 Return tbl
End Function
```

8. Add the following code to the Page_Load event handler. The code here simply populates the RadMenu with data.

```
C# Example:
protected void Page_Load( object sender, EventArgs e )
{
   RadMenu1.DataSource = CreateTable();
   RadMenu1.DataFieldID = "ID";
   RadMenu1.DataTextField = "Text";
   RadMenu1.DataValueField = "Value";
   RadMenu1.DataFieldParentID = "ParentID";
   RadMenu1.DataBind();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 RadMenu1.DataSource = CreateTable
 RadMenu1.DataFieldID = "ID"
 RadMenu1.DataTextField = "Text"
 RadMenu1.DataValueField = "Value"
 RadMenu1.DataFieldParentID = "ParentID"
 RadMenu1.DataBind
End Sub
```

9. Add the javascript functions in the listing below to the WebForm. You can place these within the <Form> block. The listing contains three client-side functions

```
<script   type="text/javascript">
     function OpenItem( OpenIt )
    {
         var menu = <%= RadMenu1.ClientID %>;
          var text = document.getElementById("TextBox1").value;

         var item = menu.FindItemByText(text);
        if (item)
        {
            if ( OpenIt )
                item.Open();
            else
```

```
                item.Close();
        }
        else
        {
             alert("Item with text '" + text + "' not found.");
        }
    }

     function EnableItem( EnableIt )
    {
         var menu = <%= RadMenu1.ClientID %>;
          var text = document.getElementById("TextBox1").value;

         var item = menu.FindItemByText(text);
        if (item)
        {
            if ( EnableIt )
                item.Enable();
            else
                item.Disable();
        }
        else
        {
             alert("Item with text '" + text + "' not found.");
        }
    }

     function EnumerateAllItems()
    {
         var menu = <%= RadMenu1.ClientID %>;
         for (var i=0; i<menu.AllItems.length; i++)
           {
                alert(menu.AllItems[i].Text);
           }
     }
</script>
```

The RadMenuItem instance is retrieved with the line:

```
var menu = <%= RadMenu1.ClientID %>;
```

OpenItem() opens or closes the menu item specified in the Textbox using the menu FindItemByText() function to locate the item and calling the items Open()/Close() functions. EnableItem() enables or disables the item specified in the Textbox using the Enable() and Disable() client-side functions. Finally, EnumerateAllItems() uses the menu client AllItems array to enumerate menu items and display an alert for each.

10.Modify the Button tags to contain onclick events that will reference client-side functions:

```
<input id="Button1" type="button" value="Open" onclick="OpenItem( true );return false;"/><br />
<input id="Button2" type="button" value="Close" onclick="OpenItem( false );return false;"/><br />
<input id="Button3" type="button" value="Disable" onclick="EnableItem( false );return false;"/><br
<input id="Button4" type="button" value="Enable" onclick="EnableItem( true );return false;"/><br /
<input id="Button5" type="button" value="Enumerate" onclick="EnumerateAllItems();return false;"/>
```

11.Run the application to test the functionality.

2.2.6 Summary

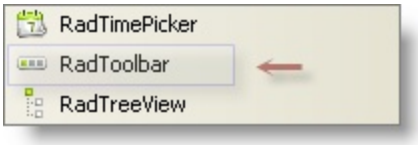
This chapter demonstrated how to build drop down menus and context menus in the designer without the need for code. You also saw how to modify the appearance in the designer using skins and how to trigger menu items using keyboard shortcuts. Multiple methods for binding to data were presented using XML files, hierarchical database data and by transforming arbitrary XML data from some external source to a RadMenu friendly XML format.

You also saw how to build menu items programmatically on the server, how the menu can navigate to internal web pages or external urls and how to respond to events originating from the menu items. Using RadMenu client API function calls and events you were able to locate individual menu items, modify them and to detect when changes in the menu and menu items had occurred.

2.3 RadToolBar

2.3.1 Getting Started

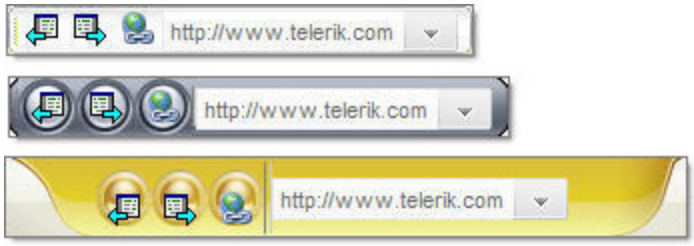
RadMenu is used to build tool and button strips and adapts well to many web application scenarios. Paired with RadDock you can achieve a slick "floating toolbar" look for your application. Or use menu, tabstrip, toolbar, and combobox controls together to display a Office 2007 style ribbon bar. RadMenu consists of a single RadToolBar control found in the toolbox (see figure below).



RadToolbar in the toolbox

The RadToolBar control contains a collection of button elements:

- Each button in the collection can be image, text or a combination.
- Each button type in the collection can be a standard push button, radio button or can be a template. Template buttons can contain essentially anything in the ASP.NET and Telerik arsenal of controls, giving you maximum flexibility.
- Buttons can be displayed both vertically or horizontally.
- A full set of pre-built skins (many of which coordinate with all RadControls) provide a polished appearance and blend in with your web application look and feel.



RadToolBar showing skins, buttons and templated button

Data binding support includes control of the button text, image, and button command. Toolbar buttons can be created and managed:

- At design time using the built-in editor
- Created programmatically
- By binding to an XML "ContentFile"
- By binding to any data source that implements IListSource or IEnumerable.

2.3.2 Using RadToolBar in the Designer

The labs in this section describe the basics of configuring the tool bar at design time. During these labs you will learn how to add regular push buttons, toggle buttons, separators and template button items in the designer as well as assign images, commands, and hot keys to each button. You will also learn how to populate RadToolBar from a static XML file.

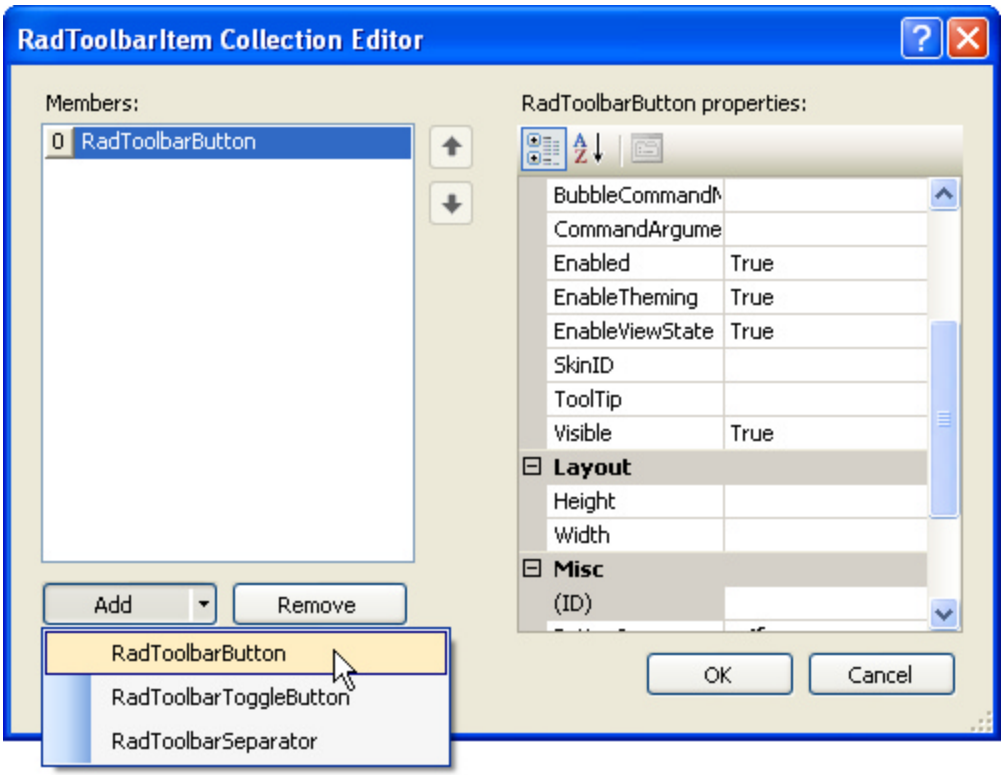
Lab: Exploring RadToolbar

This lab demonstrates the general pattern for adding and editing items in the toolbar at design time and also covers:

- The difference between tool bar button types.
- How to create a template button.
- How to define hot keys for your buttons.

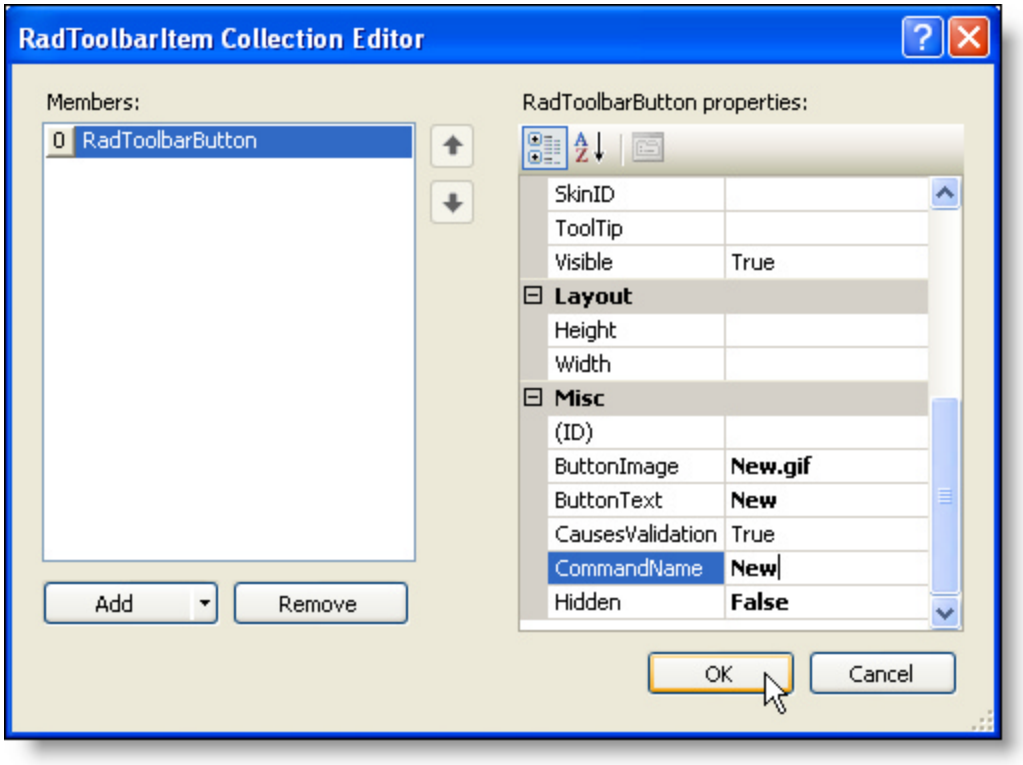
1. Create a new application "GettingStarted"
2. Create a RadControls directory in the project and copy the Toolbar folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls
3. Drop a RadToolBar on the default page.
4. Edit the *Items* property using the the ellipses button in the Properties window. This will display the RadToolBarItem Collection Editor.



Adding a new button to the collection

5. Click the "Add" button and select RadToolBarButton. This item will show as a standard push button in the tool bar.
6. In the properties for the button, set the CommandName to "New". Notice that ButtonText and ButtonImage properties follow suit.



Setting the button command

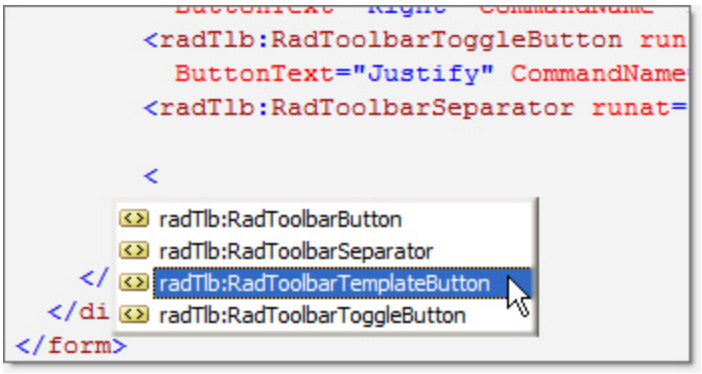
7. Create two more new RadToolBarButton items with CommandNames "Open" and "Save".
8. Using the Add button to create a new RadToolBarSeparator. This tool bar button item will not have command or image properties associated with it, but will just visually separate groups of items.
9. Use the Add button to create a RadToolBarToggleButton. Set the CommandName property of the button to "Bold". Set the AccessKey property to "b".
10. Add two more RadToolBarToggleButtons with CommandName properties "Italic" and "Underline". Set the AccessKey properties to "i" and "u".
11. Add another RadToolBarSeparator.
12. The three "Bold", "Italic" and "Underline" buttons are intended to work separately. The next group of text alignment buttons work together as a unit, similar to a group of radio buttons. If any of the four buttons are pressed the remaining buttons return to their normal, un-toggled positions. This is accomplished by setting the ButtonGroup property of each button in a group to the same value.
13. Add a RadToolBarToggleButton with the CommandName property "AlignLeft" and ButtonGroup property to "Align".
14. Add three more RadToolBarToggleButton items with CommandName properties "Center", "AlignRight" and "Justify". Set the ButtonGroup property of each to "Align".
15. Run the application and notice the behavior of each type of button.

Three types of buttons at runtime

16.Stop the application

These next steps show how to add any kind of control or group of controls to the toolbar using templates. A RadToolbarTemplateButton object provides the container for other controls. RadToolbarTemplateButton does not get added using the designer. Rather its added using the code view for the aspx page.

17.Navigate to the aspx code for the page and just before the closing RadToolBar "Items" tag type a "<" begin tag bracket. From the IntelliSense drop down select RadToolBarTemplateButton.



Creating a RadToolbarTemplateButton tag

18.Continue to use IntelliSense and create a ButtonTemplate tag within the RadToolBarTemplateButton. The fragment should look like the fragment below:

```
<radTlb:RadToolbarTemplateButton>
   <ButtonTemplate>
   </ButtonTemplate>
</radTlb:RadToolbarTemplateButton>
```

19.Now that you have the button template set up you can add any control of set of controls to it.

20.From the Toolbox, drag a RadComboBox and place it just after the ButtonTemplate start tag.

21.Within the RadComboBox tag create an "Items" tag.

22.Within the Items tag create three RadComboBoxItem tags. Set the Text properties of each item to "Red", "Green" and "Blue" respectively. The fragment should now look like the example below.

```
<radTlb:RadToolbarTemplateButton>
   <ButtonTemplate>
      <radC:RadComboBox  ID="RadComboBox1"  runat="server">
       <Items>
            <radC:RadComboBoxItem Text="Red" />
            <radC:RadComboBoxItem Text="Green" />
           <radC:RadComboBoxItem Text="Blue" />
       </Items>
     </radC:RadComboBox>
   </ButtonTemplate>
</radTlb:RadToolbarTemplateButton>
```

23.Run the application.

The running application with templated combobox

Lab: Binding to static XML data

The RadToolBar is designed to populate its buttons and properties based on properties set in an xml file. This lab shows how the xml file is structured and accessed by the control.

1. Create a new web application "ContentFile".
2. Create a RadControls directory in the project and copy the ToolBar folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls

3. Drop a RadToolBar on the default page. Set the ContentFile property to "MyToolbarContent.xml".
4. In the Solution Explorer add an XML file item to the project by right clicking the project, Add | New Item | Xml File. Name the file "MyToolbarContent.xml". Copy the following xml markup to the MyToolbarContent.xml.

```
<?xml version="1.0" encoding="utf-8" ?>
<radToolbar>
   <Button ToolTip="New Document" CommandName="New" AutoPostBack="False"
           ButtonImage="new.gif" DisplayType="ImageOnly" />
   <Button ToolTip="Open Document" CommandName="Open" AutoPostBack="False"
           ButtonImage="Open.gif" DisplayType="ImageOnly" />
   <Button ToolTip="Save Document" CommandName="Save" AutoPostBack="False"
           ButtonImage="Save.gif" DisplayType="ImageOnly" />
  <Separator />
    <ToggleButton CommandName="Bold" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="Bold.gif" ButtonText="Bold" Visible="True" />
    <ToggleButton CommandName="Italic" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="Italic.gif" ButtonText="Italic" Visible="True" />
    <ToggleButton CommandName="Underline" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="Underline.gif" ButtonText="Underline" Visible="True" />
  <Separator />
    <ToggleButton CommandName="AlignLeft" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="AlignLeft.gif" ButtonText="Align Left" ButtonGroup="Align"
     Visible="True" />
    <ToggleButton CommandName="Center" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="Center.gif" ButtonText="Align Center" ButtonGroup="Align"
                Visible="True" />
    <ToggleButton CommandName="AlignRight" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="AlignRight.gif" ButtonText="Align Right" ButtonGroup="Align"
                Visible="True" />
    <ToggleButton CommandName="Justify" AutoPostBack="False" DisplayType="ImageOnly"
                  ButtonImage="Justify.gif" ButtonText="Justify" ButtonGroup="Align"
                Visible="True" />
</radToolbar>
```

Notice that the content file contains button types for standard push buttons, separators and toggle buttons. The attributes of each button tag correspond to the properties of the button.

5. Run the application.

Important notes:

- Currently, the parent tag "radToolbar" must be spelled this way and appears to be case sensitive. If spelled any other way the toolbar does not appear, although no error is generated.
- Template buttons are not supported in the content file at the time of this writing.

2.3.3 Using RadToolBar at Runtime

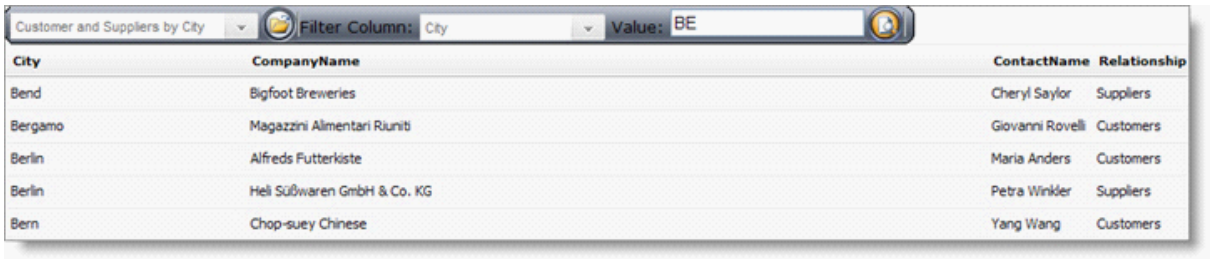
The labs in this section demonstrate building and responding to RadToolBar programmatically at runtime:

- Create tool bar buttons, toggle buttons, separators and template items.
- Implement ITemplate.
- Respond to button click events.
- Respond to events for controls in a template.
- Bind RadToolBar to an ObjectDataSource.

Lab: Respond to RadToolbar Events

This lab demonstrates responding to tool bar click events, using skins on the toolbar, finding buttons in the button items collection, and finding individual controls in button templates. The project uses the RadToolBar to display table data from an MSDE instance containing the Microsoft sample Northwind database. The project assumes you have the MSDE installed and that the Northwind database install scripts have been run. See Installing MSDE in the appendix for steps on setting up the data for this project.

The user can drop down a list of tables from the leftmost combo box. After selecting a table the user can click the "Open" button to see the table data. As each table is opened a second combo box will populate with the names of the columns containing character data. The user can select a column name and enter a few characters to filter by rows starting with those characters. The rightmost button is a toggle button that turns filtering on when toggled.



The running project, filtering data

1. Create a new application "DatabaseExplorer"
2. Create a RadControls directory in the project and copy the ToolBar folder from installation RadControls directory. **Note:** You need only copy the "winAmpModern" skin for this project. For a default installation you find the RadControls directory in:

C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls

3. Register the controls by entering this markup under the Page element:

```
<%@ Register Assembly="RadComboBox.Net2" Namespace="Telerik.WebControls"
    TagPrefix="radC" %>
<%@ Register Assembly="RadToolBar.Net2" Namespace="Telerik.WebControls"
    TagPrefix="radTlb" %>
<%@ Register Assembly="RadGrid.Net2" Namespace="Telerik.WebControls"
    TagPrefix="radG" %>
```

4. Copy the following HTML into the form div element:

```
<form id="form1" runat="server">
    <div>
         <radTlb:RadToolBar ID="RadToolbar1" runat="server" OnOnClick="RadToolbar1_OnClick"
             AutoPostBack="True" Skin="winAmpModern">
            <Items>
                  <radTlb:RadToolbarTemplateButton ID="rttbTables" runat="server">
                    <ButtonTemplate>
                          <radC:RadComboBox ID="rcbTables" runat="server" >
                         </radC:RadComboBox>
                    </ButtonTemplate>
                 </radTlb:RadToolbarTemplateButton>
                  <radTlb:RadToolbarButton ID="RadToolbarButton1" runat="server"
                      ButtonImage="Open.gif" ButtonText="Open" CommandName="Open"
                    Hidden="False"
                     ToolTip="Click this button to open the selected table " />
                 <radTlb:RadToolbarSeparator runat="server" />
                  <radTlb:RadToolbarTemplateButton ID="rttbFilter" runat="server">
                    <ButtonTemplate>
                        Filter Column:
                          <radC:RadComboBox ID="rcbFilter" runat="server">
                         </radC:RadComboBox>
                        Value:
                           <asp:TextBox ID="tbFilter" runat="server"></asp:TextBox>
                    </ButtonTemplate>
                 </radTlb:RadToolbarTemplateButton>
                  <radTlb:RadToolbarToggleButton ID="rtbFilter" runat="server"
                      ButtonText="Filter" ToolTip="Filter results"
                       CommandName="Filter" Hidden="False" ButtonImage="Find.gif" />
                 <radTlb:RadToolbarSeparator runat="server" />
            </Items>
          </radTlb:RadToolBar>
            <radG:RadGrid ID="RadGrid1" runat="server" GridLines="None"></radG:RadGrid>
    </div>
</form>
```

Previous sections covered basic design time setup including toggle and template buttons, so we won't go over that ground again. But do notice the template buttons and controls within the templates. Later these controls are accessed in code. Also notice the RadToolbarToggleButton for the same reason.

```
<radTlb:RadToolbar ID="RadToolbar1" runat="server" OnOnClick="RadToolbar1_OnClick"
    AutoPostBack="True" Skin="winAmpModern">
    <Items>
        <radTlb:RadToolbarTemplateButton ID="rttbTables" runat="server">
            <ButtonTemplate>
                <radC:RadComboBox ID="rcbTables" runat="server" ></radC:RadComboBox>
            </ButtonTemplate>
        </radTlb:RadToolbarTemplateButton>
        <radTlb:RadToolbarButton ID="RadToolbarButton1" runat="server"
            ButtonImage="Open.gif" ButtonText="Open" CommandName="Open"
            Hidden="False"
            ToolTip="Click this button to open the selected table " />
        <radTlb:RadToolbarSeparator runat="server" />
        <radTlb:RadToolbarTemplateButton ID="rttbFilter" runat="server">
            <ButtonTemplate>
                Filter Column:
                <radC:RadComboBox ID="rcbFilter" runat="server">
                </radC:RadComboBox>
                Value:
                <asp:TextBox ID="tbFilter" runat="server"></asp:TextBox>
            </ButtonTemplate>
        </radTlb:RadToolbarTemplateButton>
        <radTlb:RadToolbarToggleButton ID="rtbFilter" runat="server" ButtonText="Filter"
            ToolTip="Filter results" CommandName="Filter" Hidden="False" ButtonImage="Find.gif" />
        <radTlb:RadToolbarSeparator runat="server" />
    </Items>
</radTlb:RadToolbar>
<radG:RadGrid ID="RadGrid1" runat="server" GridLines="None"></radG:RadGrid>
```

Button templates and toggle buttons to notice

5. In the code behind at the top of the page class enter a constant to hold the connection string to the MSDE Northwind database:

```
const string ConnectionString =
   @"Data Source=localhost\Telerik;Initial Catalog=Northwind;" +
    "Persist Security Info=True;User ID=sa;Password=sa";
```

6. In the code behind enter helper properties to access controls embedded in templates. For example, "rcbTables" is a RadComboBox located on the "rttbTables" template. In the rcbTables property use the FindControl() method to locate the control, then cast the control to its actual type. Setting up the properties this way allows you to easily work with the controls as if they were directly available on the page. Also notice how the Filtered property retrieves the "Filter" toggle button using the GetButtonByCommandName() method.

Note: At the time of this writing the toggle button object does not correctly report the "Toggled" property state. Instead, use the button items array (see the "Filtered" property below) or get the state from the e.Button argument in the OnClick event handler.

```
C# Example:
private RadComboBox rcbTables
{
    get
    {
        return   (RadComboBox)rttbTables.FindControl("rcbTables");
    }
}

private TextBox tbFilter
{
    get
    {
```

```
        return   (TextBox)rttbFilter.FindControl("tbFilter");
    }
}

private  RadComboBox  rcbFilter
{
    get
    {
        return   (RadComboBox)rttbFilter.FindControl("rcbFilter");
    }
}

private bool  Filtered
{
    get
    {
        return     ((RadToolbarToggleButton)RadToolbar1.Items.GetButtonByCommandName(
      "Filter")).Toggled;
    }
    set
    {
            ((RadToolbarToggleButton)RadToolbar1.Items.GetButtonByCommandName(
      "Filter")).Toggled  =  value;
    }
}
```

VB Example:

```
Private  readonly Property  rcbTables() As  RadComboBox
 Get
   Return   CType(rttbTables.FindControl("rcbTables"),   RadComboBox)
 End Get
End Property

Private  readonly Property  tbFilter() As  TextBox
 Get
   Return   CType(rttbFilter.FindControl("tbFilter"),   TextBox)
 End Get
End Property

Private  readonly Property  rcbFilter() As  RadComboBox
 Get
   Return   CType(rttbFilter.FindControl("rcbFilter"),   RadComboBox)
 End Get
End Property

Private Property  Filtered() As Boolean
 Get
   Return   CType(RadToolbar1.Items.GetButtonByCommandName("Filter"),   _
     RadToolbarToggleButton).Toggled
 End Get
 Set
     CType(RadToolbar1.Items.GetButtonByCommandName("Filter"),  _
    RadToolbarToggleButton).Toggled  =  value
 End Set
End Property
```

7. Enter a helper method to fill the first combo with names of tables from the Northwind database. The BindTableCombo() method uses the SqlConnection GetSchema method to retrieve the table names and passes "Tables" as the collection to retrieve and an array of string "restrictions" to show only user created tables. Also notice the use of the rcbTables helper property created in the last step. See

the MSDN for additional information on SqlConnection.GetSchema.

```
C# Example:
private void BindTableCombo()
{
     SqlConnection connection = new SqlConnection(ConnectionString);
     connection.Open();

    string[] restrictions = new string[4];
     restrictions[0] = null; // catalogs
     restrictions[1] = null; // all owners
     restrictions[2] = null; // all tables
     restrictions[3] = "BASE TABLE"; // User created tables only, no views or system
      DataTable schemaTable = connection.GetSchema("Tables", restrictions);

      rcbTables.DataSource = schemaTable;
      rcbTables.DataTextField = "TABLE_NAME";
      rcbTables.DataBind();
}

VB Example:
Private Sub BindTableCombo()
 Dim connection As SqlConnection = New SqlConnection(ConnectionString)
  connection.Open
 Dim restrictions(4) As String
 restrictions(0) = Nothing
 restrictions(1) = Nothing
 restrictions(2) = Nothing
 restrictions(3) = "BASE TABLE"
 Dim schemaTable As DataTable = connection.GetSchema("Tables", restrictions)
 rcbTables.DataSource = schemaTable
 rcbTables.DataTextField = "TABLE_NAME"
  rcbTables.DataBind
End Sub
```

8. In the page load event handler call BindTableCombo() to fill the combo box.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
     if (!IsPostBack)
     {
          BindTableCombo();
     }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
    BindTableCombo
 End If
End Sub
```

9. Create an event handler for the RadToolBar OnClick event defined in the HTML markup. This will be a shell for logic added later that responds to the Open and Filter buttons:

```
C# Example:
protected void RadToolbar1_OnClick(object sender,
  Telerik.WebControls.RadToolbarClickEventArgs e)
{
    switch (e.Button.CommandName)
    {
        case "Open":
```

```
            {
                break;
            }
        case "Filter":
            {
                break;
            }
    }
}

VB Example:
Protected Sub  RadToolbar1_OnClick(ByVal sender As Object,_
  ByVal e As   Telerik.WebControls.RadToolbarClickEventArgs)
 Select  e.Button.CommandName
 Case "Open"
   ' break
 Case "Filter"
   ' break
 End Select
End Sub
```

10. Run the application to observe the combo box is filled from the Northwind schema.

11. Stop the application.

12. Create new helper method to select all records from the currently selected table and return the result as a DataTable:

```
C# Example:
private DataTable GetSelectedTable(string table)
{
    string sql = "SELECT * FROM [" + table + "]";
     SqlConnection connection = new  SqlConnection(ConnectionString);
     SqlDataAdapter adapter = new SqlDataAdapter(sql, connection);
     DataSet dataSet = new DataSet();
     adapter.Fill(dataSet);
    return dataSet.Tables[0];
}

VB Example:
Private Function GetSelectedTable(ByVal table As String) As DataTable
 Dim sql As String = "SELECT * FROM [" + table + "]"
 Dim connection As SqlConnection = New  SqlConnection(ConnectionString)
 Dim adapter As SqlDataAdapter = New SqlDataAdapter(sql, connection)
 Dim dataSet As DataSet = New DataSet
  adapter.Fill(dataSet)
 Return dataSet.Tables(0)
End Function
```

13. Create a matching property that returns the result of the GetSelectedTable() method passing the currently selected table name.

```
C# Example:
private DataTable SelectedTable
{
    get
    {
        return  GetSelectedTable(rcbTables.Text);
    }
}

VB Example:
Private ReadOnly Property SelectedTable() As DataTable
```

```
 Get
   Return   GetSelectedTable(rcbTables.Text)
 End Get
End Property
```

14. Create a method that iterates the columns of the selected table, populates a generic list with column names and binds the list to the second combo box. Notice that only string fields are added to the list.

```
C# Example:
private void BindColumnCombo()
{
    List<string> TextColumns = new List<string>();
    foreach (DataColumn column in SelectedTable.Columns)
    {
        if (column.DataType == typeof(System.String))
        {
              TextColumns.Add(column.Caption);
        }
    }

     rcbFilter.DataSource = TextColumns;
     rcbFilter.DataBind();
}

VB Example:
Private Sub BindColumnCombo()
    Dim TextColumns As New List(of String)
    Dim column As DataColumn
    For Each column In SelectedTable.Columns
        If column.DataType = Type.GetType(System.String) Then
             TextColumns.Add(column.Caption)
        End If
    Next

     rcbFilter.DataSource = TextColumns
     rcbFilter.DataBind()
End Sub
```

15. Create a method to populate the grid based on the currently selected table and any filtering the user may have entered. BindGrid() gets a reference to the default view object for the selected table. If the user has entered any filter criteria and the Filtered toggle button is pressed, then a filter string "<column name> like <user input>%" is used as the view RowFilter. The view is then bound to the grid to display the table data.

```
C# Example:
private void BindGrid()
{
    string sqlFormat = "{0} like '{1}%'";

     DataView currentView = SelectedTable.DefaultView;

    if ((tbFilter.Text.Length > 0) && this.Filtered)
    {
        string sql = string.Format(sqlFormat, rcbFilter.Text, tbFilter.Text);
         currentView.RowFilter = sql;
    }
     RadGrid1.DataSource = currentView;
     RadGrid1.DataBind();
}

VB Example:
```

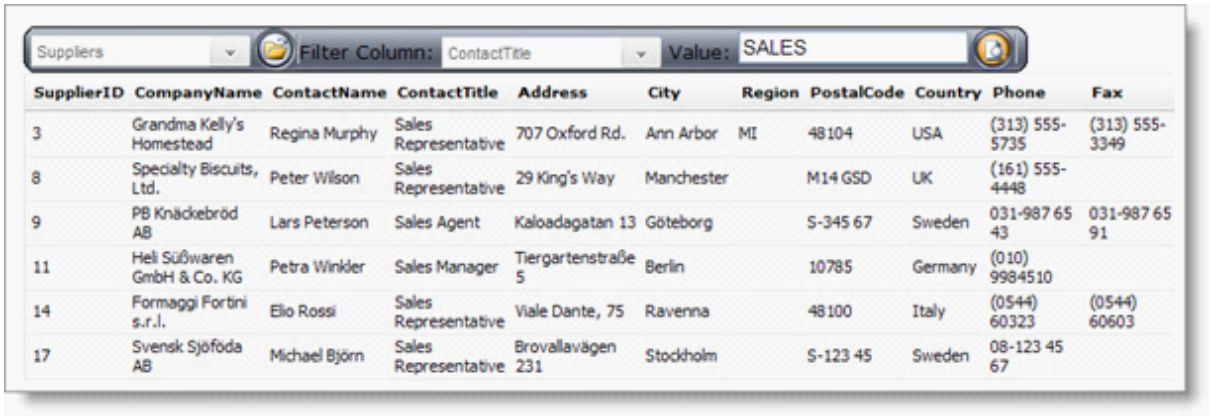
```
Private Sub BindGrid()
 Dim sqlFormat As String = "{0} like '{1}%'"
 Dim currentView As DataView = SelectedTable.DefaultView
 If (tbFilter.Text.Length > 0) AndAlso Me.Filtered Then
   Dim sql As String = String.Format(sqlFormat, rcbFilter.Text, tbFilter.Text)
    currentView.RowFilter = sql
 End If
  RadGrid1.DataSource = currentView
  RadGrid1.DataBind
End Sub
```

16.To pull the pieces together, finish coding the OnClick event handler for the toolbar with the code in bold below.

- When the Open button is clicked, filtering is turned off and the filter button is returned to its un-toggled state and the filter text is cleared. Data for the currently selected table is displayed in the grid and the columns for that table are loaded to the second combo box.
- When the Filter button is clicked, the BindGrid() method is run again and any filter criteria the user has entered can be taken into account.

```
C# Example:
protected void RadToolbar1_OnClick(object sender,
    Telerik.WebControls.RadToolbarClickEventArgs e)
{
    switch (e.Button.CommandName)
    {
        case "Open":
            {
                this.Filtered = false;
                 tbFilter.Text = "";
                 BindGrid();
                  BindColumnCombo();
                break;
            }
        case "Filter":
            {
                 BindGrid();
                break;
            }
    }
}

VB Example:
Protected Sub RadToolbar1_OnClick(ByVal sender As Object,_
  ByVal e As    Telerik.WebControls.RadToolbarClickEventArgs)
 Select  e.Button.CommandName
 Case "Open"
   Me.Filtered = False
   tbFilter.Text = ""
   BindGrid
    BindColumnCombo
   ' break
 Case "Filter"
   BindGrid
   ' break
 End Select
End Sub
```

17.Run the application. Try selecting the Suppliers table and clicking the Open Button. Set the Filter Column drop down combo to ContactTitle and enter a value of SALES.



SupplierID	CompanyName	ContactName	ContactTitle	Address	City	Region	PostalCode	Country	Phone	Fax
3	Grandma Kelly's Homestead	Regina Murphy	Sales Representative	707 Oxford Rd.	Ann Arbor	MI	48104	USA	(313) 555-5735	(313) 555-3349
8	Specialty Biscuits, Ltd.	Peter Wilson	Sales Representative	29 King's Way	Manchester		M14 GSD	UK	(161) 555-4448	
9	PB Knäckebröd AB	Lars Peterson	Sales Agent	Kaloadagatan 13	Göteborg		S-345 67	Sweden	031-987 65 43	031-987 65 91
11	Heli Süßwaren GmbH & Co. KG	Petra Winkler	Sales Manager	Tiergartenstraße 5	Berlin		10785	Germany	(010) 9984510	
14	Formaggi Fortini s.r.l.	Elio Rossi	Sales Representative	Viale Dante, 75	Ravenna		48100	Italy	(0544) 60323	(0544) 60603
17	Svensk Sjöföda AB	Michael Björn	Sales Representative	Brovallavägen 231	Stockholm		S-123 45	Sweden	08-123 45 67	

Suppliers filtered on ContactTitle like "Sales%:

Lab: Create Tool Bar Items at Runtime

This lab demonstrates creating tool bar buttons, toggle buttons, separators and template items programmatically at runtime. You will also see how to respond to button commands and template item events.

1. Create a new web application "Programmatic".
2. Create a RadControls directory in the project and copy the ToolBar folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls
3. Drop a RadToolBar on the default page. Set the *AutoPostBack* property to True.
4. Add a Label from the Standard items in the toolbox. Set the *ID* property to "lblStatus".
5. Enter a helper function to build and return a button. This creates a standard RadToolbarButton push button and populates it with command, tool tip and image.

```
C# Example:
private RadToolbarButton GetButton(string command, string tip, string image)
{
     RadToolbarButton radToolBarButton = new RadToolbarButton();
     radToolBarButton.CommandName = command;
     radToolBarButton.ToolTip = tip;
     radToolBarButton.ButtonImage = image;
    return radToolBarButton;
}

VB Example:
Private Function GetButton(ByVal command As String,_
  ByVal tip As String, ByVal image As String) As RadToolbarButton
 Dim radToolBarButton As RadToolbarButton = New RadToolbarButton
  radToolBarButton.CommandName = command
  radToolBarButton.ToolTip = tip
  radToolBarButton.ButtonImage = image
 Return radToolBarButton
End Function
```

6. In the page load use the GetButton() method to create button instances and add them to the RadToolBarItem collection:

```
C Example:
if (!IsPostBack)
```

```
{
      RadToolbar1.Items.Add(GetButton("New", "New  Document",
        "New.gif"));
      RadToolbar1.Items.Add(GetButton("Open", "Open  Document",
        "Open.gif"));
      RadToolbar1.Items.Add(GetButton("Save", "Save  Document",
        "Save.gif"));
}

VB  Example:
If Not  IsPostBack Then
  RadToolbar1.Items.Add(GetButton("New",  "New  Document",  "New.gif"))
  RadToolbar1.Items.Add(GetButton("Open",  "Open  Document",  "Open.gif"))
  RadToolbar1.Items.Add(GetButton("Save",  "Save  Document",  "Save.gif"))
End If
```

7. In the properties window events create an OnClick event handler for the RadToolBar. Add the following code to the event handler:

```
C#  Example:
protected void  RadToolbar1_OnClick(object  sender,
    Telerik.WebControls.RadToolbarClickEventArgs  e)
{
     lblStatus.Text = "You clicked on "  +  e.Button.CommandName;
}

VB  Example:
Protected Sub  RadToolbar1_OnClick(ByVal  sender As Object,_
  ByVal e As    Telerik.WebControls.RadToolbarClickEventArgs)
 lblStatus.Text  =  "You  clicked  on  "  +  e.Button.CommandName
End Sub
```

8. Run the application. Notice that the tool tip displays when the mouse is over the button and the label changes to respond to the clicked tool bar item command.

9. Stop the application.

10. Add a new helper function to create toggle buttons.

```
C#  Example:
private  RadToolbarToggleButton  GetToggleButton(
   string  command, string tip, string  image, string  group)
{
      RadToolbarToggleButton  radToolBarToggleButton  = new   RadToolbarToggleButton();
      radToolBarToggleButton.CommandName  =  command;
     radToolBarToggleButton.ToolTip  =  tip;
      radToolBarToggleButton.ButtonImage  =  image;
      radToolBarToggleButton.ButtonGroup  =  group;
    return   radToolBarToggleButton;
}

VB  Example:
Private Function  GetToggleButton(ByVal  command As String,_
  ByVal tip As String, ByVal image As String, _
  ByVal group As String) As  RadToolbarToggleButton
 Dim  radToolBarToggleButton As  RadToolbarToggleButton  = New  RadToolbarToggleButton
  radToolBarToggleButton.CommandName  =  command
  radToolBarToggleButton.ToolTip  =  tip
  radToolBarToggleButton.ButtonImage  =  image
  radToolBarToggleButton.ButtonGroup  =  group
 Return  radToolBarToggleButton
End Function
```

11.Add an overload of the GetToggleButton method for buttons that can be toggled independently:

```
C# Example:
private RadToolbarToggleButton GetToggleButton(
  string command, string tip, string image)
{
    return GetToggleButton(command, tip, image, "");
}

VB Example:
Private Function GetToggleButton(ByVal command As String,_
  ByVal tip As String, ByVal image As String) As RadToolbarToggleButton
 Return GetToggleButton(command, tip, image, "")
End Function
```

12.Add the code in bold below to the page load method to create separator and toggle buttons. The first set of buttons have a *ButtonGroup* property of "Align". When one is pressed, the other three pop back up. The second set of buttons for Bold, Italic and Underline leave the *ButtonGroup* property empty and so function independently.

```
C# Example:
if (!IsPostBack)
{
. . .

     RadToolbar1.Items.Add(new RadToolbarSeparator());
      RadToolbar1.Items.Add(GetToggleButton("Left", "Align Left",
        "AlignLeft.gif", "Align"));
      RadToolbar1.Items.Add(GetToggleButton("Center", "Align Center",
        "Center.gif", "Align"));
      RadToolbar1.Items.Add(GetToggleButton("Right", "Align Right",
        "AlignRight.gif", "Align"));
      RadToolbar1.Items.Add(GetToggleButton("Justify", "Align Justified",
        "Justify.gif", "Align"));

     RadToolbar1.Items.Add(new RadToolbarSeparator());
      RadToolbar1.Items.Add(GetToggleButton("Bold", "Bold Font",
        "bold.gif"));
      RadToolbar1.Items.Add(GetToggleButton("Italic", "Italic Font",
        "Italic.gif"));
      RadToolbar1.Items.Add(GetToggleButton("Underline", "Underline Font",
        "underline.gif"));
}

VB Example:
If Not IsPostBack Then
. . .

  RadToolbar1.Items.Add(New RadToolbarSeparator)
 RadToolbar1.Items.Add(GetToggleButton("Left", "Align Left", _
 "AlignLeft.gif", "Align"))
 RadToolbar1.Items.Add(GetToggleButton("Center", "Align Center", _
 "Center.gif", "Align"))
 RadToolbar1.Items.Add(GetToggleButton("Right", "Align Right", _
 "AlignRight.gif", "Align"))
 RadToolbar1.Items.Add(GetToggleButton("Justify", "Align Justified", _
 "Justify.gif", "Align"))
  RadToolbar1.Items.Add(New RadToolbarSeparator)
 RadToolbar1.Items.Add(GetToggleButton("Bold", "Bold Font", _
 "bold.gif"))
 RadToolbar1.Items.Add(GetToggleButton("Italic", "Italic Font", _
```

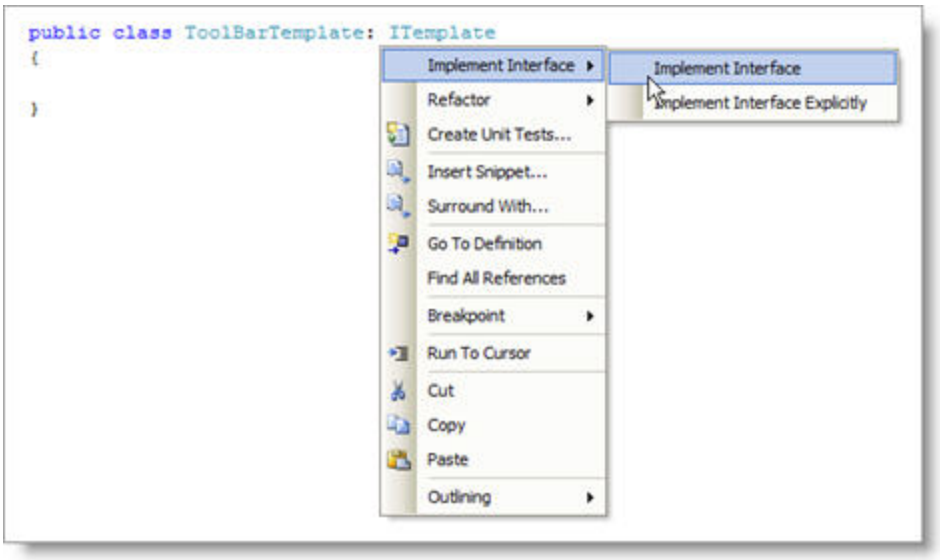
```
  "Italic.gif"))
  RadToolbar1.Items.Add(GetToggleButton("Underline",  "Underline  Font",  _
  "underline.gif"))
End If
```

13.Run the application and test the behavior of the toggle buttons.

So far three types of tool bar items have been created and responded to programmatically. The last part of the exercise shows how to create a template tool bar item, locate items within a template and to respond programmatically. The critical piece of this section is creating the template class. You must implement the ITemplate interface from the System.Web.UI namespace. ITemplate has a single method InstantiateIn() which takes a single parameter, a Control type to act as a container for other controls in your template.

14.First add a new class "ToolbarTemplate.cs" to the project.

15.Declare the class as implementing ITemplate. Right click ITemplate and choose the "Implement Interface" context menu item.



Implementing ITemplate

Now the code should look like this:

```
C# Example:
public class ToolBarTemplate: ITemplate
{

    #region ITemplate Members

    public void InstantiateIn(Control container)
    {
        throw new Exception("The method or operation is not implemented.");
    }

    #endregion
}

VB Example:
Public Class ToolBarTemplate
Implements ITemplate

 Public Sub InstantiateIn(ByVal container As Control)
```

```
    Throw New Exception("The method or operation is not implemented.")
 End Sub
End Class
```

16.Implement InstantiateIn() by:

- Removing the exception.
- Create a new LiteralControl with the text string "Search:" and add it to the container parameter Controls collection.
- Create a new TextBox control, set its ID property to "tbSearch", AutoPostBack property to "True" and add it to the container controls collection.

The code should now look similar to the example below:

```
C# Example:
public void InstantiateIn(Control container)
{
     container.Controls.Add(new LiteralControl("Search:"));
    TextBox tbSearch = new TextBox();
    tbSearch.ID = "tbSearch";
     tbSearch.AutoPostBack = true;
      container.Controls.Add(tbSearch);
}

VB Example:
Public Sub InstantiateIn(ByVal container As Control)
  container.Controls.Add(New LiteralControl("Search:"))
 Dim tbSearch As TextBox = New TextBox
 tbSearch.ID = "tbSearch"
 tbSearch.AutoPostBack = True
  container.Controls.Add(tbSearch)
End Sub
```

17.Code a Page_Init event handler. In the Page_Init:

- Create a RadToolbarTemplateButton to contain our ToolbarTemplate object.
- Create a new ToolbarTemplate instance.
- Set the RadToolbarTemplateButton "ButtonTemplate" property to the ToolbarTemplate instance.
- Add the RadToolbarTemplateButton to the collection of tool bar items.

See below for the complete code. This much alone provides a template tool bar item, but there isn't much you can do with it. The next two lines allow you to get at the contents of the template and respond to events.

- Call the FindControl() method of the RadToolbarTemplateButton, looking for "tbSearch" (defined in ToolbarTemplate.cs) and cast it to a TextBox.
- Use the reference to tbSearch and attach a TextChanged event handler. This event will fire when the user moves off the TextBox control.

Here's the completed Page_Init event handler:

```
C# Example:
protected void Page_Init(object sender, EventArgs e)
{
     RadToolbarTemplateButton radToolbarTemplateButton = new RadToolbarTemplateButton();
    ToolBarTemplate toolBarTemplate = new ToolBarTemplate();
     radToolbarTemplateButton.ButtonTemplate = toolBarTemplate;
```

```
      RadToolbar1.Items.Add(radToolbarTemplateButton);

     TextBox tbSearch = radToolbarTemplateButton.FindControl("tbSearch") as TextBox;
    tbSearch.TextChanged += new EventHandler(tbSearch_TextChanged);
}

VB Example:
Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs)
 Dim radToolbarTemplateButton As RadToolbarTemplateButton = _
  New RadToolbarTemplateButton
 Dim toolBarTemplate As ToolBarTemplate = New ToolBarTemplate
  radToolbarTemplateButton.ButtonTemplate = toolBarTemplate
    RadToolbar1.Items.Add(radToolbarTemplateButton)
 Dim tbSearch As TextBox = CType(ConversionHelpers.AsWorkaround(_
    radToolbarTemplateButton.FindControl("tbSearch"), GetType(TextBox)), TextBox)
  tbSearch.TextChanged += New EventHandler(tbSearch_TextChanged)
End Sub
```

18.Code code the TextChanged event handler:

```
C# Example:
void tbSearch_TextChanged(object sender, EventArgs e)
{
    lblStatus.Text = (sender as TextBox).Text;
}

VB Example:
Sub tbSearch_TextChanged(ByVal sender As Object, ByVal e As EventArgs)
  lblStatus.Text = (CType(ConversionHelpers.AsWorkaround(sender, GetType(TextBox)), _
   TextBox)).Text
End Sub
```

This handler simply echoes the text typed into the TextBox to the status label.

19.Run the application. Notice the new template item showing label and TextBox in the toolbar. Type new text into the "Search" text box and tab off the control. Notice the text is displayed on the screen.

Note: Because the Page_Init fires before the Page_Load the template will be added first and display on the left side of the toolbar. You can insert the remaining buttons to the left of the template using RadToolBar.Controls.Insert() instead of Add().

Lab: Binding to an ObjectDataSource

This lab will demonstrate binding the RadToolBar control to an ObjectDataSource.

RadToolBar binds to IListSource or IEnumerable interfaces. SqlDataSource, and ObjectDataSource descend from DataSourceControl, which in turns implements IListSource. Other controls like the SiteMapDataSource and XmlDataSource implement IListSource directly.

Note: This example uses ObjectDataSource. ObjectDataSource represents a middle business object that knows how to select and update data. ObjectDataSource isn't the business object itself, but rather an adapter that allows your business tier object to plug into standard data binding behavior in ASP.NET. How the business object manipulates the data is entirely specific to the business object and not visible from the ObjectDataSource. Our example will only use the ability to select data. See the MSDN for more on ObjectDataSource: http://msdn2.microsoft.com/en-us/library/9a4kyhcx.aspx (ObjectDataSource Control Overview).

The steps to this lab are different from the static XML binding example in the following respects:

- A business object will select data and return it via an ObjectDataSource control. The ObjectDataSource will be configured at design time.

- Some code is involved during page loading to bind the ObjectDataSource and to populate the tool tip for the button. The ItemDataBoundHandler handles retrieving the tool tip text from the ObjectDataSource and placing it in the ToolTip property of the button.
- The *ButtonCommandField* and *ButtonImageField* properties must be set. RadToolBar has three properties that govern which data elements are bound to, all ending with the suffix "Field": *ButtonCommandField, ButtonImageField* and *ButtonTextField*.

1. Create a new web application "BindObject".
2. Create a RadControls directory in the project and copy the ToolBar folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls

3. Drop a RadToolBar on the default page.
4. Create a new class "MyButton.cs" in the project. This class represents a single button in the toolbar and will be used later by the business object. MyButton defines the button command, tool tip and image path, but could also represent information for other parts of the button. Copy the following code into your class object:

C# Example:

```
public class MyButton
{
    public MyButton(string command, string tip, string image)
    {
        this._command = command;
        this._tip = tip;
        this._image = image;
    }
    private string _command;
    public string ButtonCommand
    {
        get
        {
            return _command;
        }
    }
    private string _tip;
    public string ToolTip
    {
        get
        {
            return _tip;
        }
    }
    private string _image;
    public string ButtonImage
    {
        get
        {
            return _image;
        }
    }
}
```

VB Example:

```
Public Class MyButton
```

```
Public Sub New(ByVal command As String, ByVal tip As String, ByVal image As String)
   Me._command = command
   Me._tip = tip
   Me._image = image
End Sub
Private _command As String

Public ReadOnly Property ButtonCommand() As String
  Get
    Return _command
  End Get
End Property
Private _tip As String

Public ReadOnly Property ToolTip() As String
  Get
    Return _tip
  End Get
End Property
Private _image As String

Public ReadOnly Property ButtonImage() As String
  Get
    Return _image
  End Get
End Property
End Class
```

5. Create another new class "MyBusinessObject.cs". Copy the code below to this class.

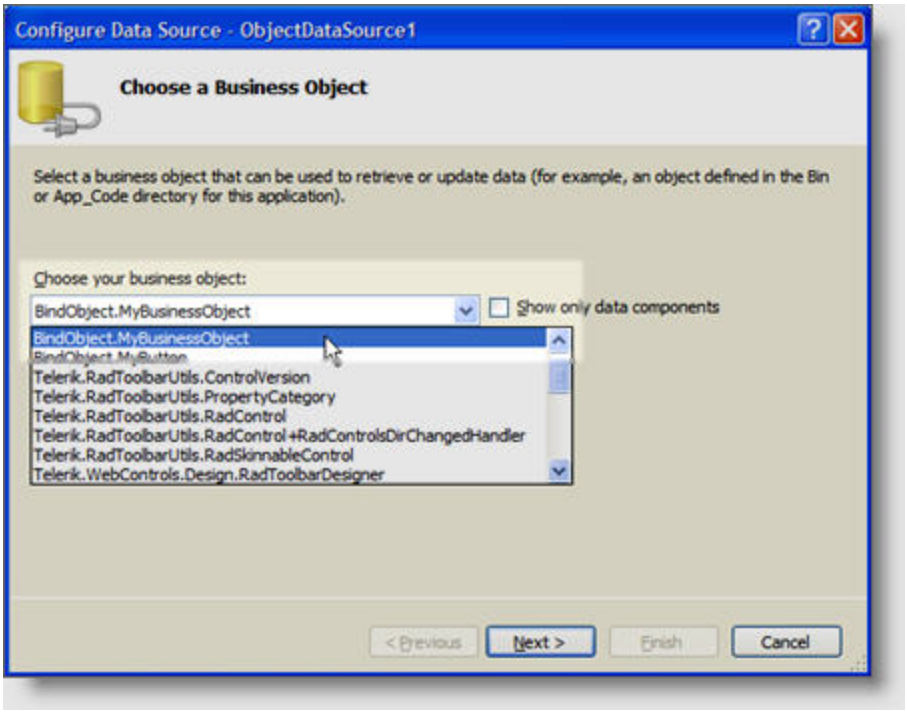
```
C# Example:
...
using System.Collections.Generic; // supports List<> object
using System.ComponentModel; // supports DataObjectMethod attribute

namespace BindObject
{

    public class MyBusinessObject
    {
         List<MyButton> _buttons = new List<MyButton>();

           [DataObjectMethod(DataObjectMethodType.Select)]
        public List<MyButton> MySelect()
        {
            _buttons.Add(new MyButton("new", "New Document", "new.gif"));
            _buttons.Add(new MyButton("open", "Open Document", "open.gif"));
            _buttons.Add(new MyButton("save", "Save Document", "save.gif"));
           return _buttons;
        }
    }
}

VB Example:
...
Imports System.Collections.Generic
Imports System.ComponentModel
...
Namespace BindObject
```

```
Public Class MyBusinessObject

  Dim _buttons = New List(Of MyButton)

    <DataObjectMethod(DataObjectMethodType.Select, True)>
  Public Function MySelect() As List(Of MyButton)
    _buttons.Add(New MyButton("new", "New Document", "new.gif"))
    _buttons.Add(New MyButton("open", "Open Document", "open.gif"))
    _buttons.Add(New MyButton("save", "Save Document", "save.gif"))
   Return _buttons
  End Function

End Class
End Namespace
```

This object gets used in the designer when configuring the ObjectDataSource. Notice that the MySelect () method has a DataObjectMethod attribute defined that lets ObjectDataSource know that this method performs a select and returns data. The data can be IEnumerable, DataTable, DataView, DataSet. Here we use a generic list of MyButton objects. The generic list type implements IEnumerable and so will work in this scenario. Alternatively you could build a DataTable and return that here.

6. Back in the designer, drop a ObjectDataSource from the toolbox "Data" section.

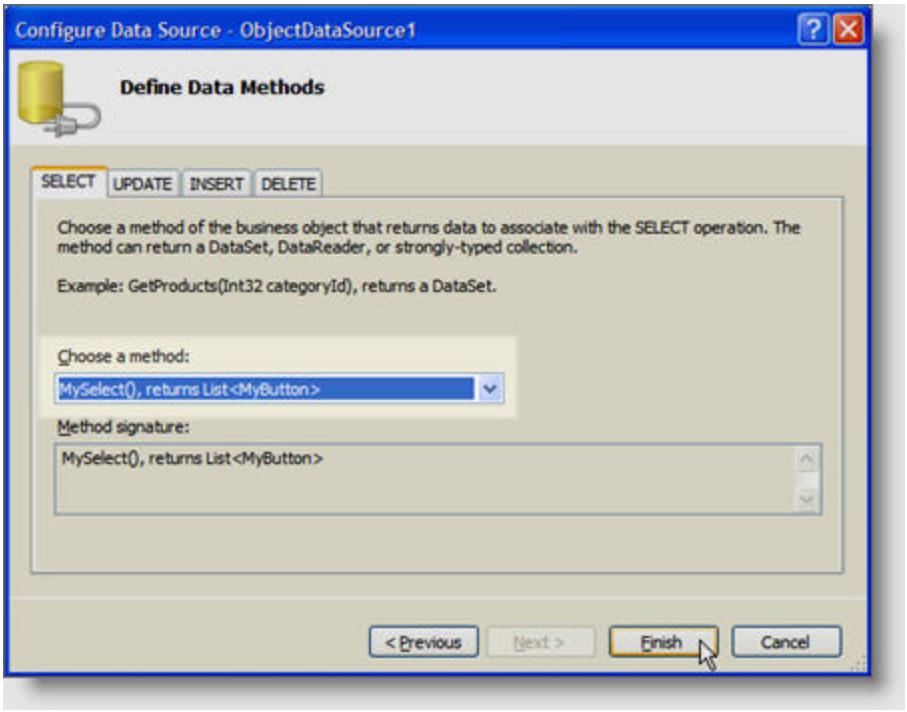
7. From the ObjectDataSource Smart Tag select "Configure Data Source".



Selecting MyBusinessObject

8. Select "MyBusinessObject" from the list, then click the Next button.

9. In the Define Data Methods part of the configuration notice that the designer detects that MySelect() method of the business object exists and is an appropriate select method. This is a result of the [DataObjectMethod] attribute placed above the MySelect() method.



MySelect() method showing in the list of data methods

10. In the code behind for the page load set the *ButtonCommandField* and *ButtonImageField* RadToolBar properties to the names of the properties in MyButton, and bind the RadToolBar *DataSource* property.

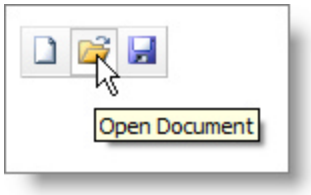
```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
     RadToolbar1.ButtonCommandField = "ButtonCommand";
     RadToolbar1.ButtonImageField = "ButtonImage";

    if (!IsPostBack)
    {
         RadToolbar1.DataSource = ObjectDataSource1;
        RadToolbar1.DataBind();
    }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  RadToolbar1.ButtonCommandField = "ButtonCommand"
  RadToolbar1.ButtonImageField = "ButtonImage"
 If Not IsPostBack Then
    RadToolbar1.DataSource = ObjectDataSource1
    RadToolbar1.DataBind
 End If
End Sub
```

11. Also in the Page_Load add a handler for the ItemDataBound event. Properties for the button command, text and image can be assigned directly by the properties ending in "Field". Any other aspects of items in the toolbar can be taken care of here in the ItemDataBound.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
```

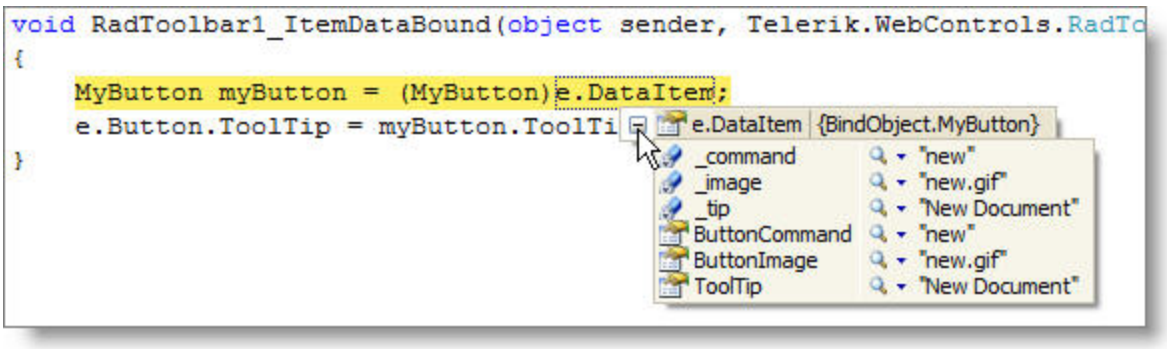
```
...
     RadToolbar1.ItemDataBound +=
   new    Telerik.WebControls.RadToolbar.OnItemDataBoundDelegate(
       RadToolbar1_ItemDataBound);
...
}

void  RadToolbar1_ItemDataBound(object sender,
    Telerik.WebControls.RadToolbarItemDataBoundEventArgs  e)
{
     MyButton myButton = (MyButton)e.DataItem;
     e.Button.ToolTip = myButton.ToolTip;
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
...
  AddHandler  RadToolbar1.ItemDataBound, AddressOf  RadToolbar1_ItemDataBound
...
End Sub

Sub  RadToolbar1_ItemDataBound(ByVal sender As Object,_
  ByVal e As    Telerik.WebControls.RadToolbarItemDataBoundEventArgs)
 Dim myButton As MyButton = CType(e.DataItem, MyButton)
  e.Button.ToolTip = myButton.ToolTip
End Sub
```

12.Run the application. Notice the tooltip is available for each button.



Running application with tooltip

Before closing the application put a break point at the first line of the ItemDataBound event handler and examine the "e" argument DataItem property. DataItem contains one "column" worth of data from the business object. If we had returned a DataTable DataItem might be a DataRow or DataRowView, but here it contains a MyButton object instance. Knowing this you can include any other data in MyButton that can be assigned during ItemDataBound.



DataItem with access to business object

2.3.4 Client Scripting with RadToolBar

This section will describe some of the important RadToolBar properties, methods and events available on the client, including:

- Getting references to RadToolBar and controls embedded in templates.
- Manipulating button properties such as visibility, enabled, toggled and hot keys.
- Iterating collections available on RadToolBar: commands, separators and IDs.
- Responding to client events that signal changes in the RadToolBar orientation, or actions of the mouse in relation to a given button.

Client API

Using the RadMenu client API you can:

- Iterate the buttons and commands in the tool bar.
- Enable and disable individual buttons.
- Show or hide individual buttons.
- Set the toggle for individual toggle buttons on and off.
- Change the orientation of the tool bar between vertical and horizontal.
- Change the access key for individual buttons.
- Change the fade effect for each button.

This section will also demonstrate how to work with server code to emit correct client IDs, particularly for controls embedded inside other controls.

To get an instance of RadToolBar in the client use the ClientID property to emit the name:

```
<%=  RadToolbar1.ClientID  %>
```

This way of getting an instance on the client will work even when controls are nested or in master/content page scenarios. From there you can call the client methods of RadToolBar:

```
<%=   RadToolbar1.ClientID   %>.DisableButton("Save");
```

Remember that the contents within <%= %> are being evaluated on the server and emitted. For simple situations like the one above the code simply evaluates as:

```
   RadToolbar1.DisableButton("Save");
```

If the object is nested in some other control, say within a template, then the use of ClientID to get the correct ID becomes more critical. Consider the case of a combo box sitting in a template button item on a RadToolBar. So the nesting is:

```
RadToolBar
   Template button
     Combo box
```

In this case the ClientID of the RadComboBox might be something like:

```
RadToolbar1_Radtoolbartemplatebutton1_ctl00_RadComboBox1
```

To flesh out the example a bit more lets say we have two tool bar buttons "Forward" and "Back" and the combo box inside the template. The Forward button should be disabled when the last item is selected and the Back button should be disabled when the first item is selected. To help visualize, here's what the page would look like:

First item selected in list

The first item in the list is selected and so the back button is disabled. To know when to disable the button we need to get a reference to the combo box in the template and check the selected index to see where its positioned in the list. Keep in mind that we have the full capabilities of the server inside the <%= %>, so here is how we get the reference the combo box in the template:

```
var combo = <%=((RadComboBox)((Control)RadToolbar1.Items.TemplatedButtons[0]).
FindControl("RadComboBox1")).ClientID%>;
```

Here's the sequence of events for the line of code above:

- Get the RadToolBar items collection
- Index into an array of TemplatedButtons (This is an array of objects).
- Cast the template button to a Control so that we can use the FindControl() method.
- FindControl() looking for the server side instance of "RadComboBox1" and return the Control.
- Cast the reference to RadComboBox1 to in fact be a RadComboBox.
- Emit the ClientID property to the JavaScript.

The <%= expression is shorthand for Response.Write() on the server, so we can expand this out and make it easier to understand by calling Response.Write() explicitly. This allows us to break the steps down to the retrieval of the template button followed by finding the RadComboBox control.

```
var rcbTables =
<%
    RadToolbarTemplateButton rttbTables =
       (RadToolbarTemplateButton)RadToolbar1.Items.TemplatedButtons[0];
     RadComboBox rcbTables = (RadComboBox)rttbTables.FindControl("rcbTables");
     Response.Write(rcbTables.ClientID);
%>
```

Here is a larger example showing how these pieces can work together:

```
function click_handler(sender, e)
{
   var combo = <%=((RadComboBox)((Control)RadToolbar1.Items.TemplatedButtons[0]).
FindControl("RadComboBox1")).ClientID%>;

  var text = combo.GetText();
  var item = combo.FindItemByText(text);

  if (item.Index == combo.Items.length - 1)
 {
      <%= RadToolbar1.ClientID %>.DisableButton("Forward");
```

```
  }
  else
  {
        <%= RadToolbar1.ClientID %>.EnableButton("Forward");
  }

  if (item.Index == 0)
  {
      <%= RadToolbar1.ClientID %>.DisableButton("Back");
  }
  else
  {
        <%= RadToolbar1.ClientID %>.EnableButton("Back");
  }
}
```

The command pairs for EnableButton/DisableButton, HideButton/ShowButton, and ToggleONButton/ToggleOFFButton all take a parameter for the name of the button to be acted on.

RadToolBar has arrays for commands, IDs and separators. For example, the following code iterates all commands in the RadToolBar button item collection and writes them to a text area.

```
var TextArea1 = document.getElementById("TextArea1");
var commands = <%= RadToolbar1.ClientID %>.ButtonCommandArray;
for(i=0;i<commands.length;i++){
   TextArea1.value += commands[i] + "\n";
}
```

You can change the orientation of RadToolBar simply by calling one of two methods:

```
<%= RadToolbar1.ClientID %>.MakeVertical();
<%= RadToolbar1.ClientID %>.MakeHorizontal();
```

Use the GetButtonByCommand() function to get a RadToolbarButton object. From there you can use the button object methods to

- Get and set the AccessKey.
- Get the command name for the button.
- Get Enabled or Visible properties for the button.

This example gets the button with the command "Forward" and sets the hot key to "F".

```
<%= RadToolbar1.ClientID %>.GetButtonByCommand("Forward").AccessKey = "F";
```

See the online help at http://www.telerik.com/help/aspnet/toolbar/ for a complete list of client API calls.

Client Events

You can receive notification of RadToolBar client side events:

- OnClientMouseOver: This event fires when the mouse is over a tool bar button.
- OnClientMouseOut: This event fires when the mouse leaves a button.
- OnClientClick: This event fires when the mouse clicks a button. Note: This event can be canceled.

- OnClientMouseDown: This event fires when the mouse is down when clicking a button.
- OnClientButtonToggled: This event fires when the a button is toggled.
- OnClientOrientationChanged: When the toolbar changes orientation to horizontal or vertical.

Subscribe to these events by using the attachEvent client method of the RadToolBar:

```
<%=    RadToolbar1.ClientID    %>.attachEvent("OnClientButtonToggled","ClientButtonToggledHandler");
```

Remove the event handler at any time by called the corresponding detachEvent() method:

```
<%=    RadToolbar1.ClientID    %>.detachEvent("OnClientButtonToggled","ClientButtonToggledHandler");
```

Lab: Confirming a client event

This lab shows how to attach a OnClientClick event handler, use techniques outlined earlier for getting template items and finally allowing or canceling the event based on user response from a confirmation dialog.

1. Open the "Database Explorer" example from the "Respond to RadToolBar Events" lab.

2. Inside the body element of the html for the default page add the following client code:

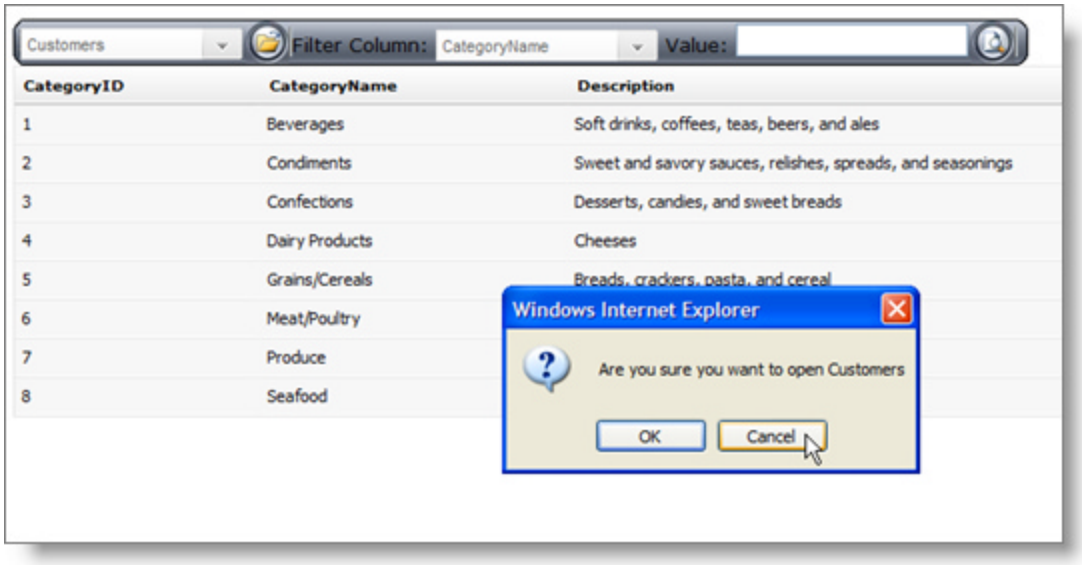
```
<script   type="text/javascript">
//<!--
      function  ClientClickHandler(sender,  e)
    {
          if (sender.CommandName == "Open")
        {
           var rcbTables =
           <%
                RadToolbarTemplateButton rttbTables =
                   (RadToolbarTemplateButton)RadToolbar1.Items.TemplatedButtons[0];
                 RadComboBox rcbTables = (RadComboBox)rttbTables.FindControl("rcbTables");
                Response.Write(rcbTables.ClientID);
           %>
           var tableName = rcbTables.GetText();
            return window.confirm("Are you sure you want to open " + tableName);
        }
    }
//-->
</script>
```

The code above only runs if the button with the "Open" command is clicked. Using the technique described in Client API section we retrieve a reference to the RadComboBox embedded in the button template. Once we have that reference we can call its client side GetText() function which contains the name of a table. Finally we call the JavaScript function window.confirm(). The user clicks OK or Cancel on the confirm dialog and we return this result from the ClientClickHandler function. If we return false then the client side click event itself is canceled and no server side action takes place.

3. After the ending Form tag add the following script to attach the ClientClickHandler:

```
<script   type="text/javascript">
//<!--
   <%= RadToolbar1.ClientID %>.attachEvent("OnClientClick","ClientClickHandler");
//-->
</script>
```

4. Run the application and click the Open button. Try both OK and cancel options in the confirm dialog.



Canceling the confirm dialog

2.3.5 Summary

This section on RadToolBar covered the basics of configuring the tool bar at design time, demonstrated building and responding to RadToolBar at runtime, and explored the capabilities of the client side API and event model.

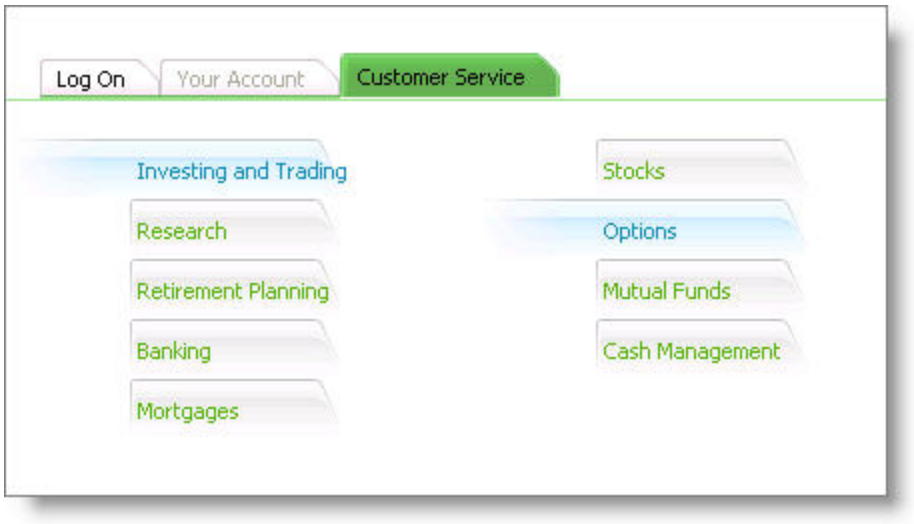
2.4 RadTabStrip

2.4.1 Getting Started

The Telerik RadTabStrip consists of the RadTabStrip itself and a RadMultiPage control that displays content for each tab selected. The MultiPage contains one or more PageView controls that store and display content. The flexible design of RadTabStrip allows you to use TabStrip and MultiPage controls separately or integrated together as your purpose requires. Some possible uses for RadTabStrip are:

- Organize UI complexity by hiding features not in use.
- Create vertical and horizontal menus with one or more levels
- Create Wizard style interfaces with or without validation for each page.
- Navigate local web sites or external Urls.
- Imitate the Office 2007 "Ribbon Bar" user interface.

Here's an example of RadTabStrip in vertical and horizontal orientations and with various skins running in the browser:



RadTabStrip instances running in the browser

The TabStrip and MultiPage can be populated manually in the designer, dynamically in code and bound to data:

Manually in the designer: The RadTabStrip Tabs collection can be edited from the properties window or using the Smart Tag "Edit Tabs" button. The Smart Tag Auto Format button applies any of a list of predefined skins to the control. The MultiPage contains one or more PageView controls that can be used to as containers for whatever content you care to put in them. Adding pages is performed by dragging PageView controls from the toolbox to the MultiPage.

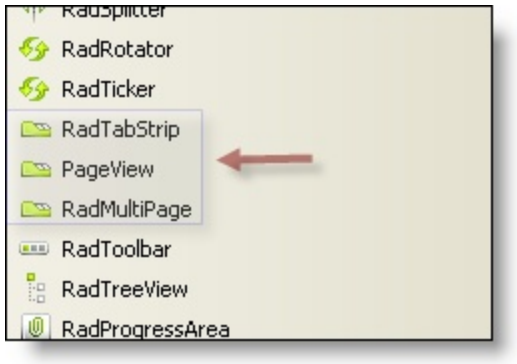
Dynamically in Code: To populate the RadTabStrip programmatically, create instances of the Tab object and add them to the Tabs collection. To populate the MultiPage, create PageView objects and add them to the PageViews collection.

Data Binding: You can bind the tabs to data sources including SqlDataSource, ObjectDataSource, AccessDataSource, SiteMapSource, XmlDataSource. The data can be a single level or hierarchical. You can bind to the datasource programmatically or in the designer. Be aware that the MultiPage does

not currently allow data binding. Instead you use the PageViewItemCreated event to re-populate each PageView.

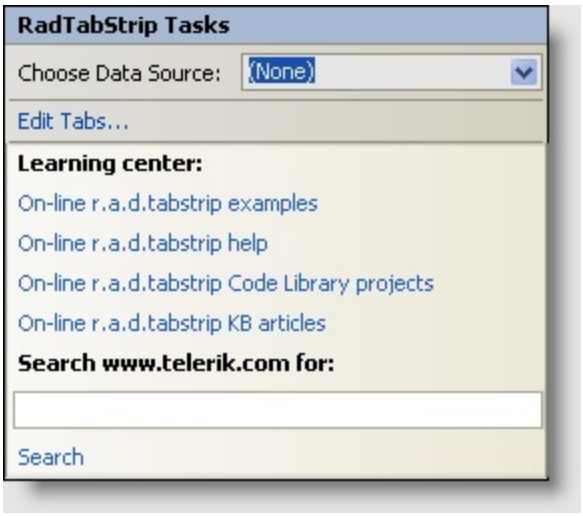
The RadMultiPage is a good choice when you need to hide and show groups of controls or when you require a "Wizard" style interface that steps the user sequentially through a set of steps.

The typical development scenario is to first drop a RadTabStrip on the page and a RadMultiPage just below. The figure below shows the RadTabStrip, PageView and RadMultiPage controls in the toolbar.



The TabStrip, MultiPage and PageView in the toolbox

The Smart Tag for the TabStrip allows you to bind data sources, edit tabs manually and contains additional links for Telerik site search, examples, help, sample projects and knowledge base articles.



More information available on the Smart Tag

2.4.2 Using RadTabStrip in the Designer

In this section you will learn tasks that can be performed on RadTabStrip and RadMultiPage at design time without code:

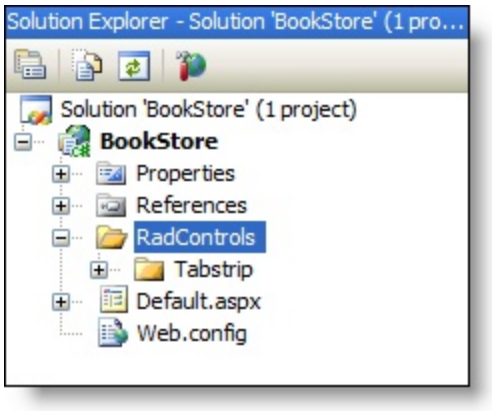
- Creating a single level menu
- Coordinating the RadTabStrip and RadMultiPage
- Manually editing Tabs

- Applying skins to RadTabStrip
- Creating multiple levels of tabs
- Binding to XML data
- Binding to hierarchical database data
- Using the RadTabStrip to navigate to external URLs and internal web pages
- Define hotkeys

Lab: Create a single level menu

In this lab you will learn to build a simple, single level set of tabs and matching page content using the designer only. You will also learn how to apply skins to the RadTabStrip.

1. Create a new ASP.NET web application project and name it "BookStore".
2. Create a RadControls directory to your project. Copy the \TabStrip directory to your new RadControls directory. You can find the RadControls directory containing skins, style sheets and javascript in the *\Program Files\telerik\r.a.d.controlsQ3 2006\NET2\RadControls* directory. Note: This step supplies the Smart Tag with the option to apply visual styles to the entire RadTabStrip using AutoFormat.

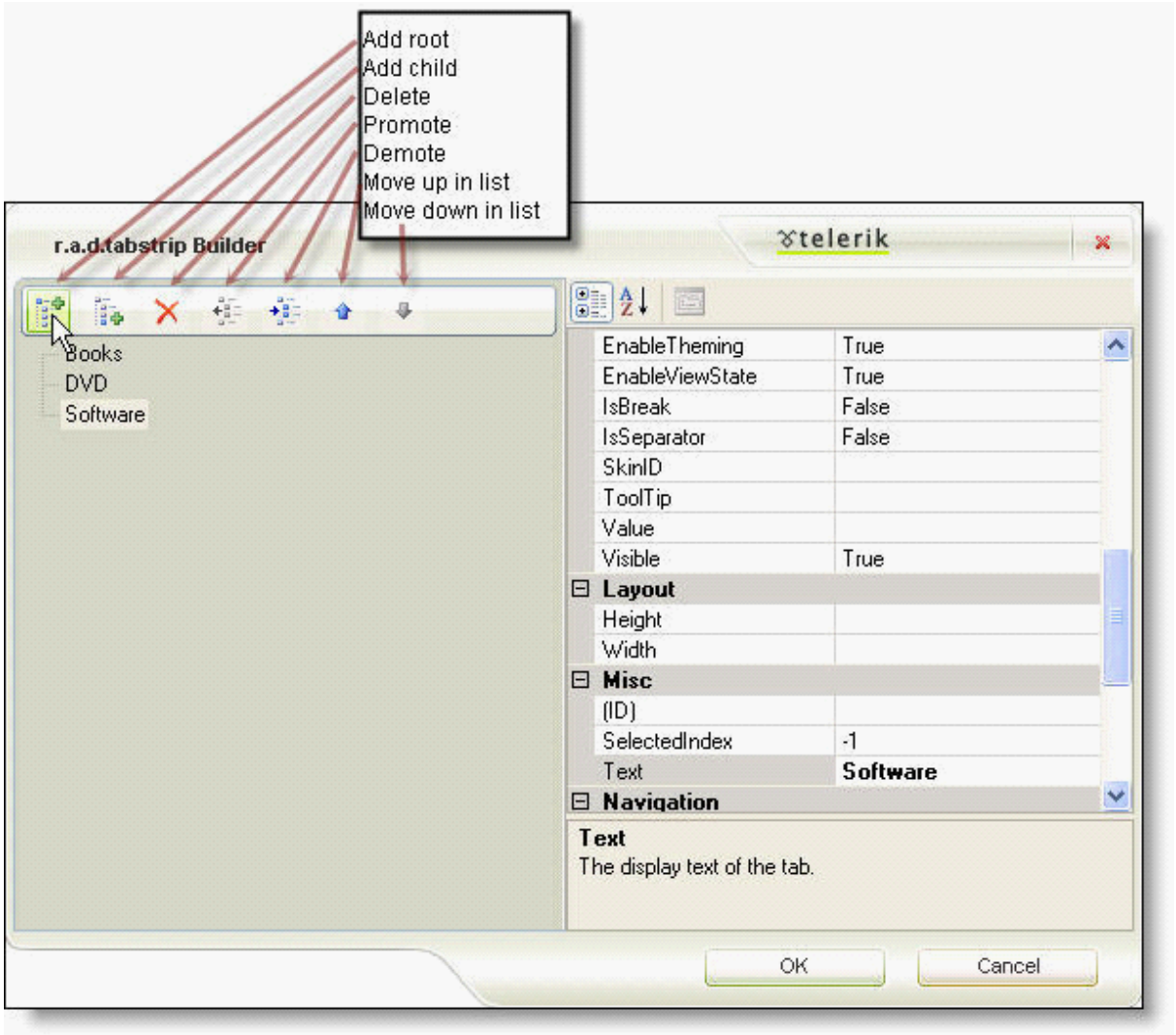


RadControls TabStrip directory added to project

3. Add a RadTabStrip to the form and set the ID to "rtsMain". Set the *SelectedIndex* property to "0".
4. Click the Smart Tag button and select Edit Tabs to display the RadTabStrip Builder dialog.

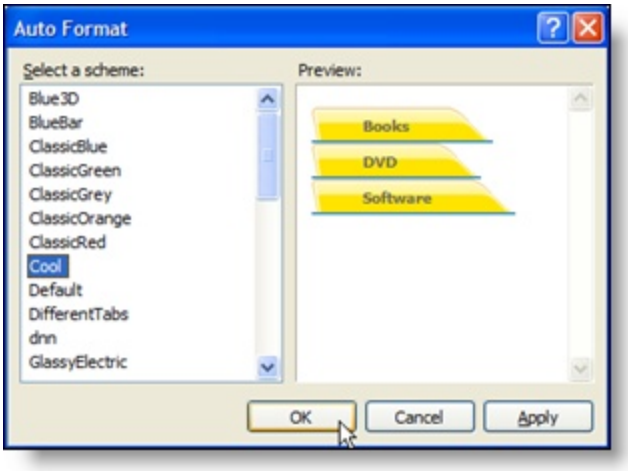
Using the RadTabStrip SmartTag

5. Create a tab by clicking the "Add root" button and set the *Text* property to "Books". Create two more tabs and set the text to "DVD" and "Software" respectively. Click OK to close the dialog.



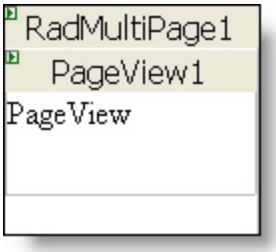
Adding tabs in the designer

6. Click the Smart Tag AutoFormat button. Select the "Cool" scheme and click OK to close the dialog.



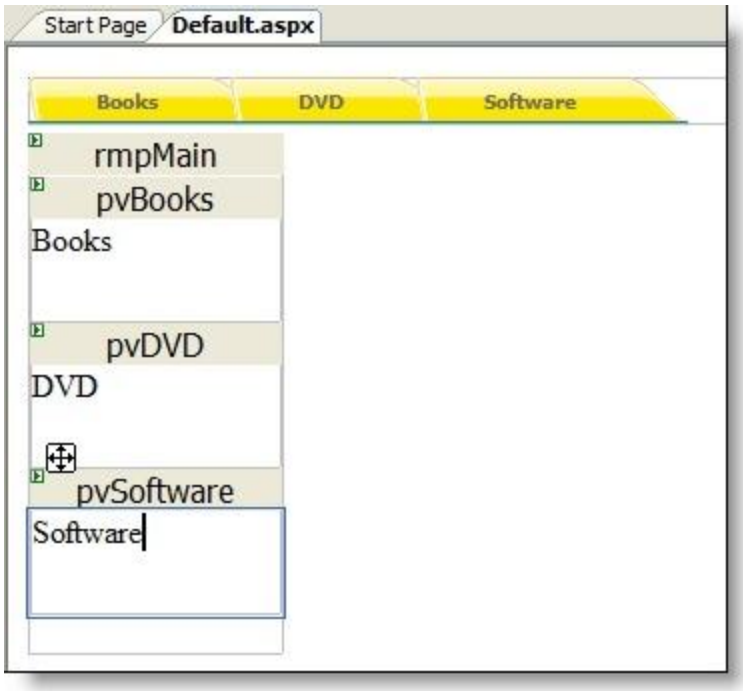
Use Auto Format to set the scheme

7. The RadMultiPage acts as a container, or set of containers really. It consists of the RadMultiPage itself and one or more PageView controls. By default it looks as the figure below and already contains one PageView. To add more PageViews, drag them from the ToolBox to the RadMultiPage.



RadMultiPage when first dropped on the page

8. Add a RadMultiPage control just below the RadTabStrip. Set the RadTabStrip ID to "rmpMain". It will contain a single MultiPage control by default. Set the MultiPage ID to "pvBooks".
9. Add two more PageView components to the RadMultiPage control. Set their ID's to "pvDVD" and "pvSoftware" respectively.
10. Edit the literal text inside of each PageView to read "Books", "DVD" and "Software" respectively. The page should now look like the example below.



RadTabStrip and RadMultiPage together on the form

11. Hook up the two controls so that they act in concert:
    - Set the RadTabStrip *MultiPageID* property to point at the MultiPage control.
    - Bring up the Tabs collection editor again and for each tab, set the PageViewID to the corresponding PageView.
12. Run the project. The content should match each selected tab.

Lab: Create a multiple level menu

In this lab you will learn how to create a menu with multiple levels.

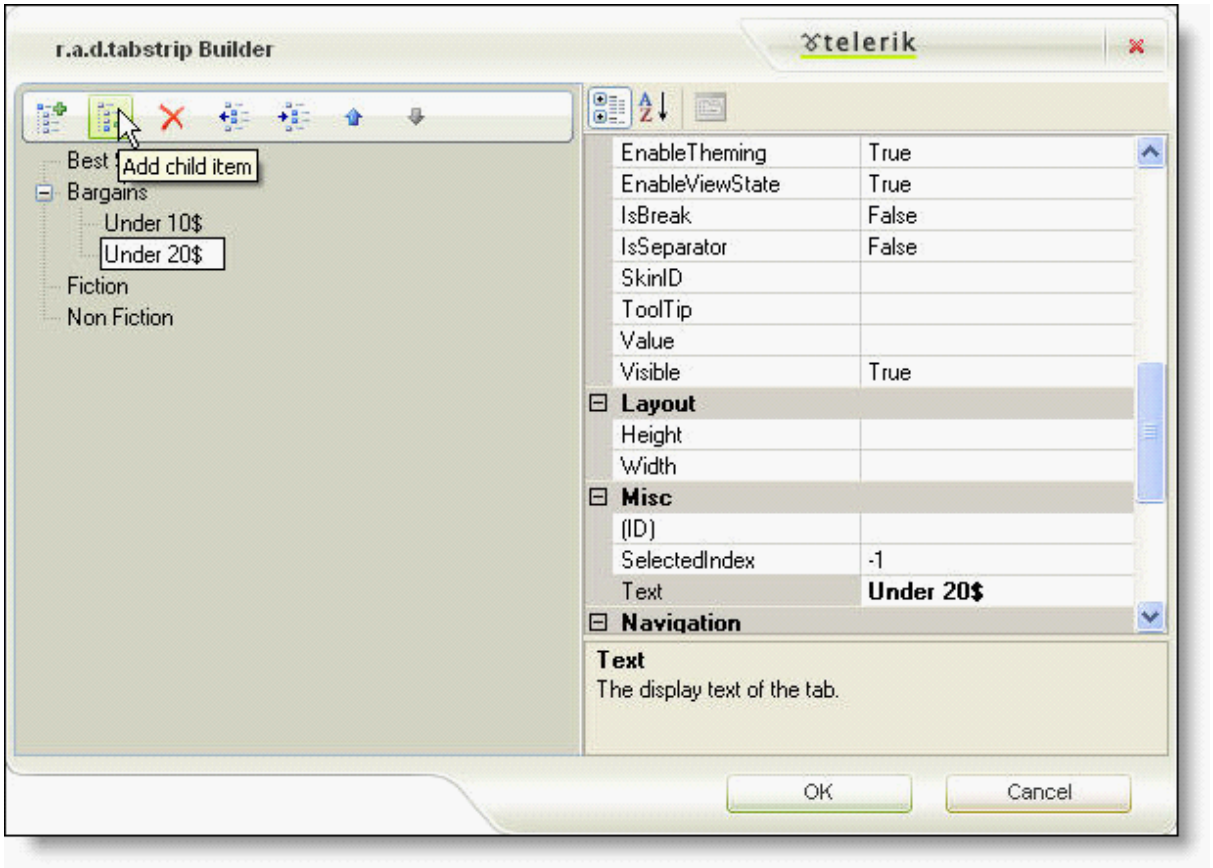
This project will build off the previous lab "Create a single level menu". Here we will use RadTabStrip to create a set of menu items running along down the left side. This menu will contain a second level of detail that will display when the parent item is clicked on.

The work will be done within the PageView "pvBooks".

1. Remove the literal text "Books". Create an HTML table within pvBooks. The table should contain a single row and only two columns, one to contain the tab strip on the left, the other to hold a MultiPage. Set the VAlign of the left hand cell to "Top".
2. In the left hand cell of the HTML table add a new RadTabStrip,
3. Set the ID to "rtsBooks"
4. Set the *Orientation* property to "VerticalLeftToRight"
5. Set the *Skin* property to ""VerticalVS2005".
6. Set the *SelectedIndex* property to "0".
7. Using the Edit Tabs Smart Tag option, edit the tabs to look like the list in the figure below. The items are hierarchical:

```
Best sellers
Bargains
  Under 10$
  Under 20$
Fiction
Non Fiction
```

8. To create the child items "Under 10$" and "Under 20$" in the tabstrip builder, first click on the Bargains item, then click the "Add child" button.



Adding child items

9. In the right hand cell of the HTML table add a new MultiPage, set the ID to "rmpBooks". Create a total of six PageViews in rmpBooks and set the ID's to the following:

```
pvBestSeller
pvBargain
pvFiction
pvNonFiction
pvUnder10
pvUnder20
```

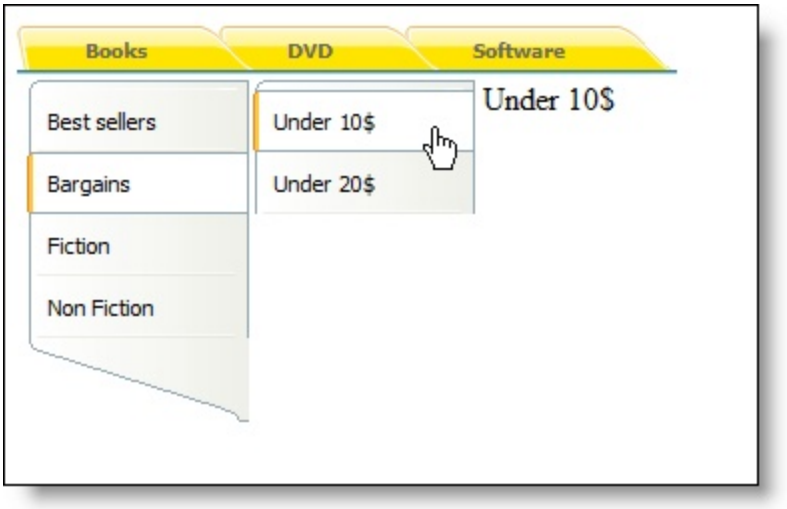
10. Set the literal text for each PageView to:

```
BestSeller
Bargain
Fiction
Non Fiction
Under 10$
Under 20$
```

11. Set the *MultiPageID* property for rtsBooks to "rmpBooks".

12. Edit the tabs for rtsBooks and set the *PageViewID* property for each tab to the corresponding PageView component.

13. Run the project to test the child tabs.



The running project

Lab: Binding to xml data

In this lab you will learn how to bind data from an XML file to the RadTabStrip.

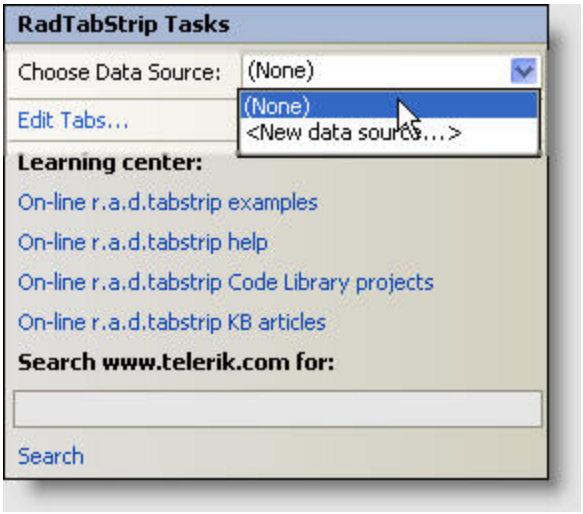
For relatively static tabs you can bind to XML data using an XMLDataSource. The xml file can be named anything as long as it has the xml extension and is part of the project. Likewise, the XML doesn't need a particular naming convention. The "tabs" and "tab" element names below are completely arbitrary, as are the "id" and "title" attributes. The structure is what the tab strip is looking for. Later, in the designer we can assign the meaning of particular attributes to be text or value fields in the tab strip.

The XML structure starts with some outer element, "<tabs>" in this case. Within tabs are single xml elements representing each tab. Notice in the xml sample below that the elements with id's "books", "bargains" and "under 10$" items are nested.

Tabs.xml:

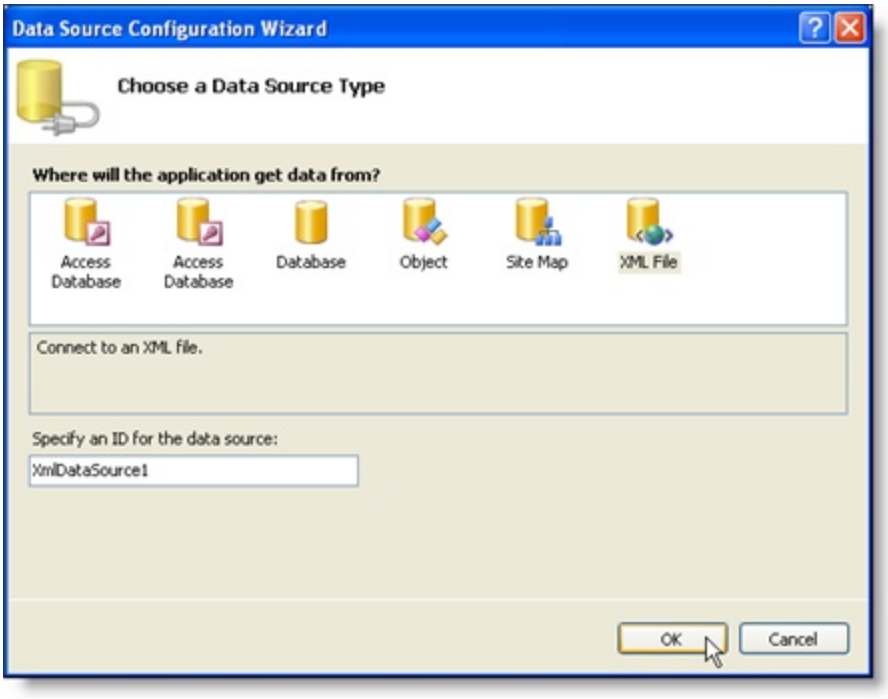
```
<?xml  version="1.0"  encoding="utf-8"?>
<tabs>
   <tab  id="books"  title="Books"  >
      <tab  id="bestseller"  title="Best  Sellers"  />
      <tab  id="bargains"  title="Bargains"   >
        <tab  id="under10"  title="Under  10$"   />
        <tab  id="under20"  title="Under  20$"   />
     </tab>
   </tab>
   <tab  id="music"  title="Music"  />
   <tab  id="software"  title="Software"   />
</tabs>
```

1. Create a new web project "XMLData".
2. Add an XML file to the project and paste the code for "Tabs.xml" show above.
3. Using the Smart Tag "Choose Data Source", select New data source.



Creating a new data source for the tab strip

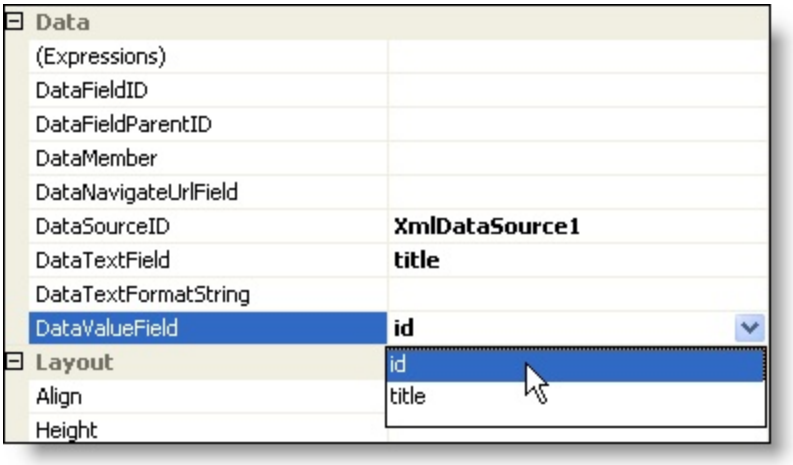
4. In the Data Source Configuration Wizard choose XML File as the data source type, leave the default ID "XmlDataSource1" and click OK.



Selecting the data source type

5. The Configure Data Source dialog will display next. Click the browse button, select tabs.xml from the list. Click OK on all each open dialog to finish configuring the data source.

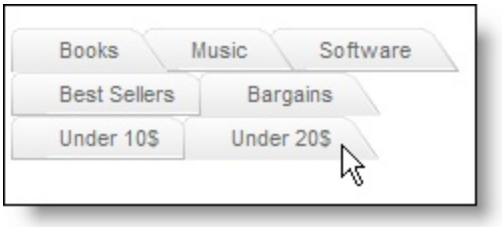
6. Now that the data source is hooked up we need to set properties to let RadTabStrip know which part of the xml file to display. The two properties we're concerned with now are the *DataTextField* that will be displayed in the visible tab itself and the *DataValueField* typically used to retain database id column information. Set *DataTextField* to "title" and the *DataValueField* to "id".



Setting the DataValueField property

7. Set the *SelectedIndex* of the RadTabStrip to "0". This will cause the "Books" tab to show selected when the application first runs.

8. At this point you should see the first level of menu items in the designer. Run the application to see test the functionality of the nested items.



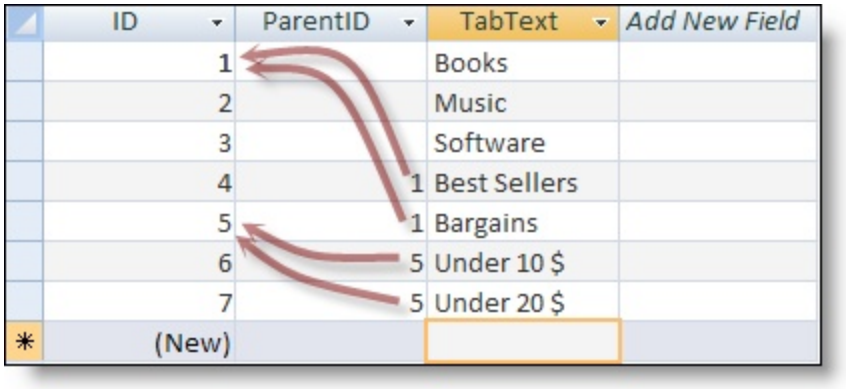
Completed project output

Lab: Binding to hierarchical database data

In this lab you will learn how to bind the RadTabStrip to hierarchical data.

We can build the same set of tabs as the previous xml example by binding to hierarchical data in a database. Again, the structure of the data is the key element. We will use an Access table "tabs.mdb" to bind to. The column names aren't significant except to make it easier to see what the data is intended for. You need at least three fields:

- A unique numeric id for each tab in the TabStrip. The TabStrip *DataFieldID* property will point to this column name.
- A non-unique numeric parent id column. The TabStrip *DataFieldParentID* will point to this column name. The value for the column will be null for all root level tabs, and will indicate a parent tab for all child tabs. In the example below "Best Sellers" and "Bargains" are child nodes of "Books".
- A text column to hold the string that will display in the tab. The TabStrip *DataTextField* property will point to this column.



	ID	ParentID	TabText	Add New Field
	1		Books	
	2		Music	
	3		Software	
	4	1	Best Sellers	
	5	1	Bargains	
	6	5	Under 10 $	
	7	5	Under 20 $	
*	(New)			

Hierarchical data

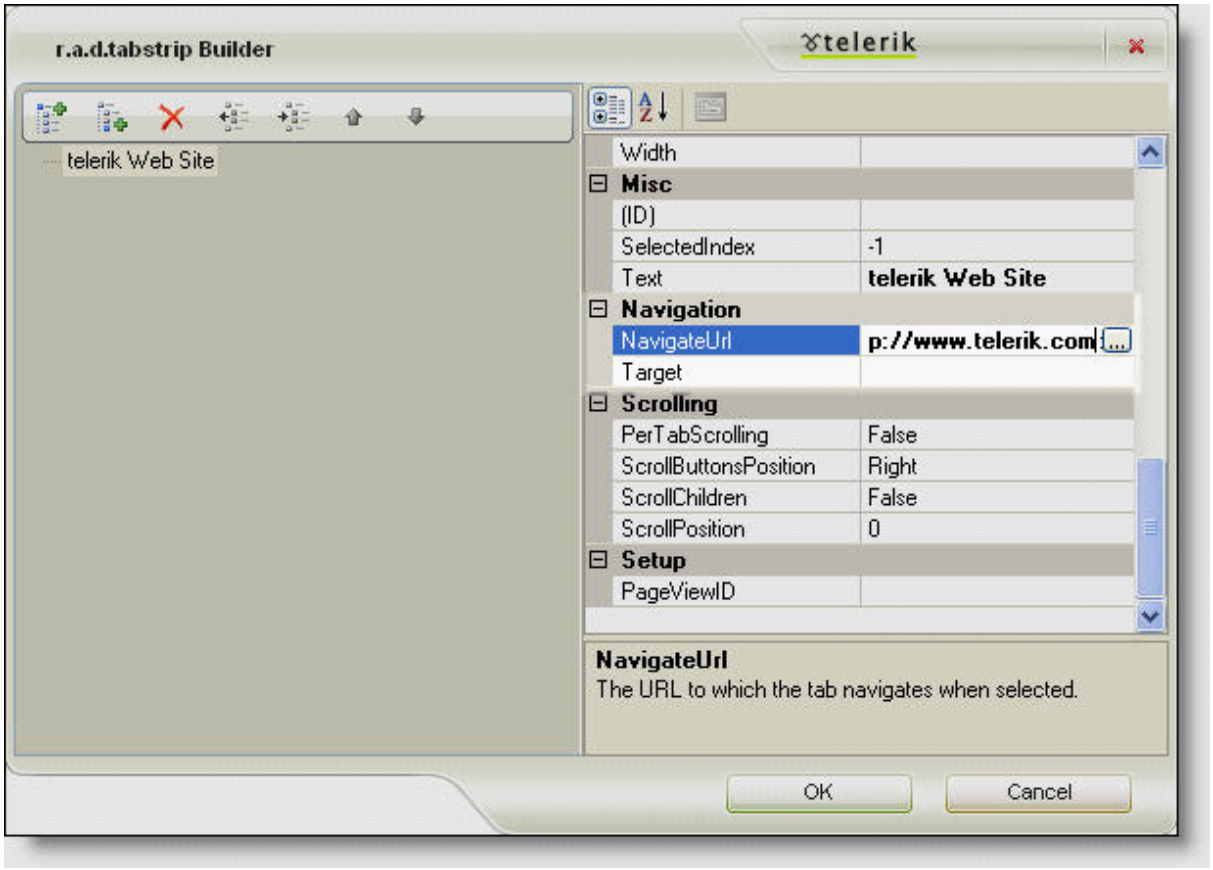
1. Create a new web project "Hierarchy".
2. Add a RadTabStrip control.
3. Add the file "Menu.mdb" to the project. Menu.mdb can be found in the \Rad TabStrip\Projects\Data directory. If the "Data Source Configuration Wizard" appears, cancel the dialog.
4. From the ToolBox drag a AccessDataSource to the page.
5. Choose the "Configure Data Source" Smart Tag. In the Configure Data Source dialog, browse to menu.mdb and select it. The Microsoft Access data file entry should now read "~/Menu.mdb". Click the Next button. Select all fields of the table "Menu". Click the Next button. Click the Finish button.
6. In the properties window for the RadTabStrip set the following properties:

Property	Value
DataSourceID	AccessDataSource1
DataFieldID	ID
DataFieldParentID	ParentID
DataTextField	TabText
DataValueField	ID

7. Run the project to test the functionality. It should look and behave as the earlier, static XML project did.

Navigating with the TabStrip

Before we move on to using the MultiPage in concert with the TabStrip you should know that the TabStrip automatically handles navigating to other urls. The locations can be either within your application or anywhere on the web (connection willing). Here's an example of creating tabs manually using the *NavigateUrl* property. Also, the *Target* property can be used to control where the NavigateUrl web page will display. By default the page will display in the same browser. If you set Target to "_blank", the url will display in a new browser window.

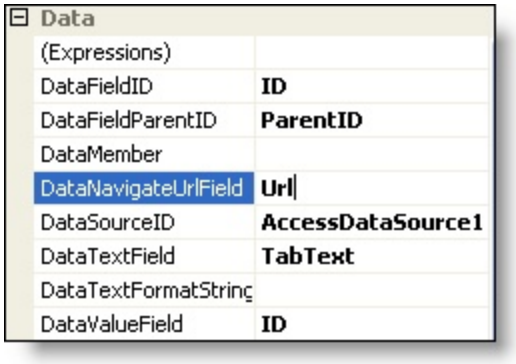


Defining the NavigateUrl property

If you're data binding, add another column to your database that will contain the url. The DataNavigateUrlField field takes the name of that column and causes a redirection to the specified url corresponding to the tab you clicked on. Taking the previous Hierarchy example we add a column "Url".

Defining a column to contain navigation targets

Set the *DataNavigateUrl* property at your new "Url" column in the Menu table and that's all it takes. When you run the project and click the "Software" tab you will be redirected to the Telerik web site.



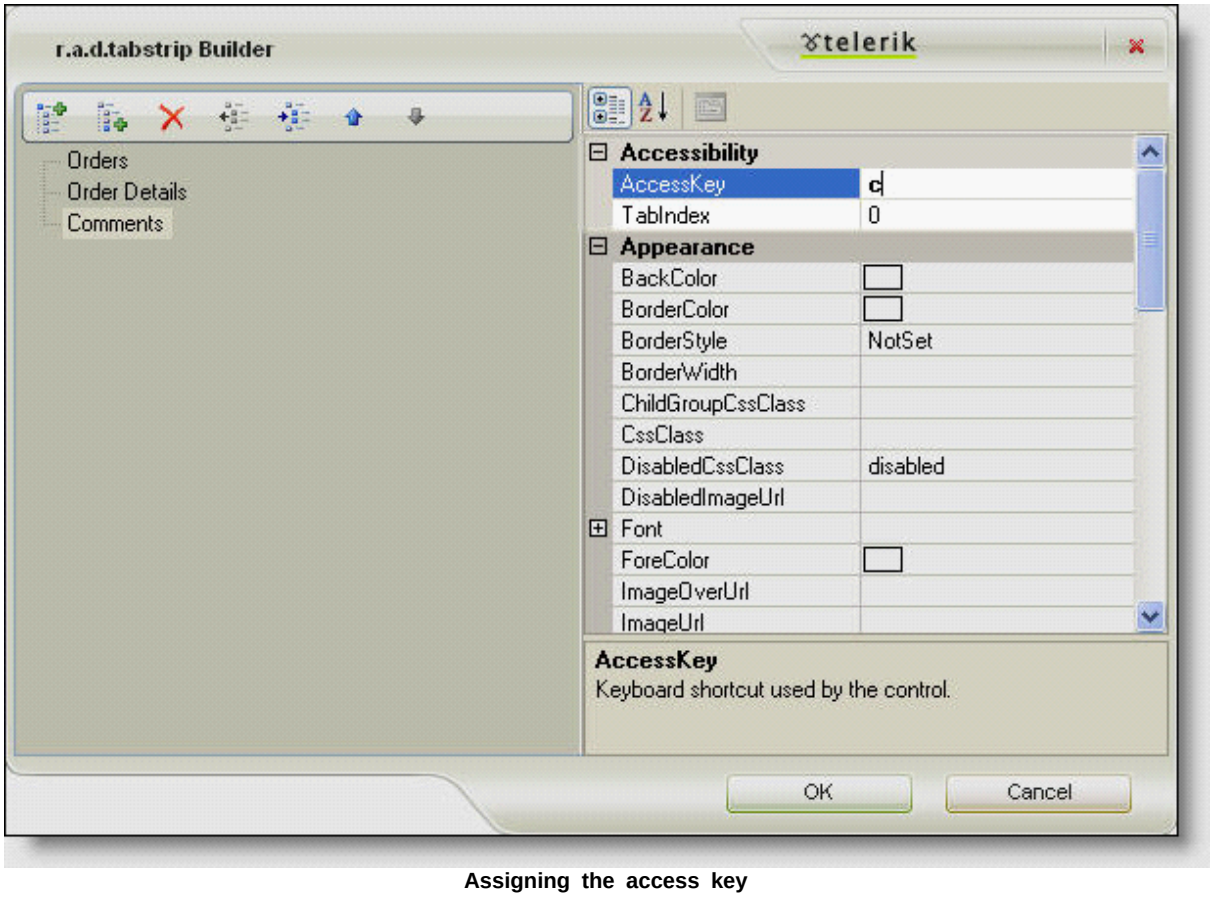
Setting the DataNavigateUrlField property

Defining Hotkeys

Production ready applications require speedy, easy-to-use data entry functionality. Data entry tasks should have the option of being entered from the keyboard only. The use of hotkeys frees the user from having to use the mouse or switch between mouse and keyboard.

To implement hotkeys in your application:

- Set the *AutoPostBack* property of the TabStrip to true.
- Assign the *AccessKey* property of each tab. In the example below, when the user keys "alt-c", the "Comments" tab will show and the TabClick event will fire.



Assigning the access key

2.4.3 Using RadTabStrip at Runtime

At some point in the development process you will want more fine grain control to RadTabStrip and RadMultiPage than design time access can provide. Telerik provides rich server and client API's for creating, iterating, locating and manipulating tabs and pages programmatically. In this section you will learn how to manipulate RadTabStrip and RadMultiPage in code at runtime:

- Create new tabs and page views dynamically and add them to the web page controls.
- Iterate the RadTabStrip Tabs collection and RadMultiPage PageViews collection.
- Navigate dynamically created PageViews at runtime.
- Validate controls on a PageView.
- Dynamcailly add user controls to PageViews.
- Programatically set focus to a control based on the current selected tab.

RadTabStrip at runtime

To create tabs programmatically create instances of the Tab object and add them to the Tabs collection.

```
C# Example:
Tab tab = new Tab("a tab name");
RadTabStrip1.Tabs.Add(tab);

VB Example:
Dim Tab As Tab = New Tab("a tab name")
RadTabStrip1.Tabs.Add(Tab)
```

The Tab constructor has three overloads:

```
Tab constructors:
public Tab();
public Tab(string text);
public Tab(string text, string value);
```

Using the last of these we can replicate the menu we created manually in "Create a single level menu".

```
C# Example:
RadTabStrip1.Tabs.Add(new Tab("Books", "1"));
RadTabStrip1.Tabs.Add(new Tab("Music", "2"));
RadTabStrip1.Tabs.Add(new Tab("Software", "3"));

VB Example:
RadTabStrip1.Tabs.Add(New Tab("Books", "1"))
RadTabStrip1.Tabs.Add(New Tab("Music", "2"))
RadTabStrip1.Tabs.Add(New Tab("Software", "3"))
```

Each Tab object contains a Tabs collection. To make the tabs form a hierarchy, add child Tab objects to the parent Tabs collection.

```
C# Example:
```

```
// create all the tabs
Tab books = new Tab("Books");
Tab bestSellers = new Tab("Best Sellers");
Tab bargains = new Tab("Bargains");
Tab under10 = new Tab("Under 10 $");
Tab under20 = new Tab("Under 20 $");
// add each child tab to parent Tabs collection
bargains.Tabs.Add(under10);
bargains.Tabs.Add(under20);
books.Tabs.Add(bestSellers);
books.Tabs.Add(bargains);
RadTabStrip1.Tabs.Add(books);
RadTabStrip1.Tabs.Add(new Tab("Music"));
RadTabStrip1.Tabs.Add(new Tab("Software"));

VB Example:
'create all the tabs
Dim books As Tab = New Tab("Books")
Dim bestSellers As Tab = New Tab("Best Sellers")
Dim bargains As Tab = New Tab("Bargains")
Dim under10 As Tab = New Tab("Under 10 $")
Dim under20 As Tab = New Tab("Under 20 $")
'add each child tab to parent Tabs collection
bargains.Tabs.Add(under10)
bargains.Tabs.Add(under20)
books.Tabs.Add(bestSellers)
books.Tabs.Add(bargains)
RadTabStrip1.Tabs.Add(books)
RadTabStrip1.Tabs.Add(New Tab("Music"))
RadTabStrip1.Tabs.Add(New Tab("Software"))
```

You can access any of the tabs by index:

```
C# Example:
this.Controls.Add(new LiteralControl(RadTabStrip1.Tabs[0].Text));

VB Example:
Me.Controls.Add(New LiteralControl(RadTabStrip1.Tabs(0).Text))
```

...or by iterating the Tabs collection:

```
C# Example:
foreach (Tab tab in RadTabStrip1.Tabs)
{
    this.Controls.Add(new LiteralControl(tab.Text));
}

VB Example:
For Each Tab As Tab In RadTabStrip1.Tabs
  Me.Controls.Add(New LiteralControl(tab.Text))
Next
```

RadMultiPage at runtime

To add to the MultiPage in code, create PageView instances, add content to the PageView controls array, then add the PageView instance to the MultiPage PageViews collection.

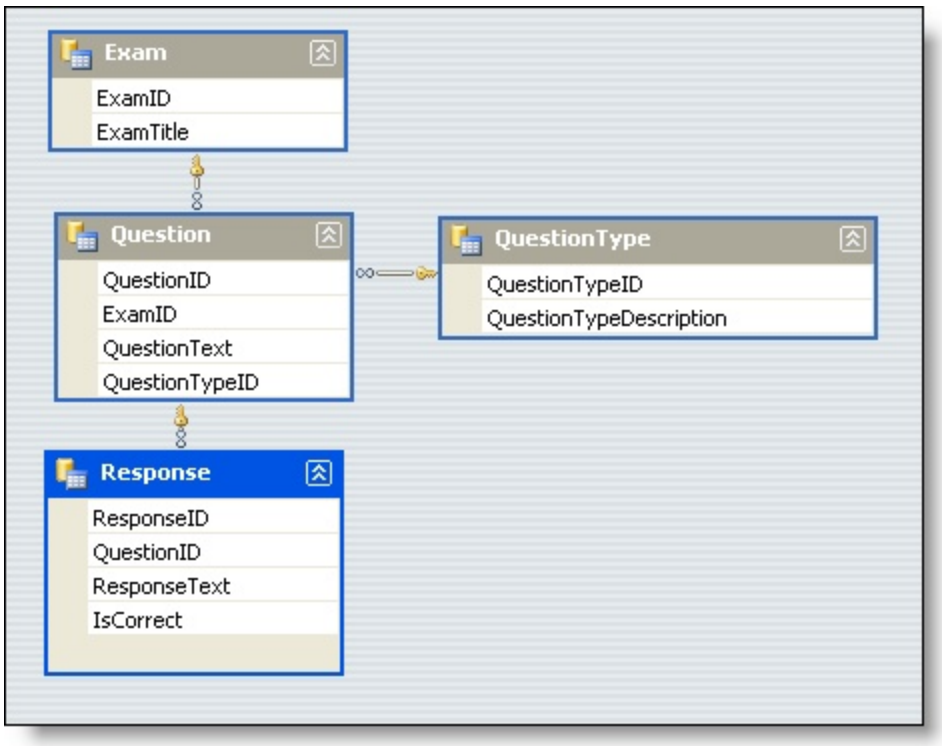
```
C# Example:
PageView myPageView = new PageView();
myPageView.Controls.Add(new LiteralControl("Some content here"));
RadMultiPage1.PageViews.Add(myPageView);

VB Example:
Dim myPageView As PageView = New PageView
myPageView.Controls.Add(New LiteralControl("Some content here"))
RadMultiPage1.PageViews.Add(myPageView)
```

Lab: Quiz Wizard

In this lab you will learn how to build a "wizard" style interface and along the way create and add PageViews dynamically, navigate the RadMultiPage, and refresh the PageView.

The user sees an introductory screen about an online quiz they can take, steps through an arbitrary set of questions and displays a summary screen. The application is fairly light weight, both in terms of the data structure and functionality to keep code size workable. The data structure and content are contained in files ExamDataset.xsd and Exam.xml. Both files are available in the "\rad tabstrip\projects \data" directory.

The data is basically a three tiered structure. We have an exam (there's only the one exam in the database for our purposes), the exam has questions, and each question has a possible response. There are only two types of questions, those with a single response displayed as a series of radio buttons and those with multiple possible correct responses shown as a series of checkboxes.



The Exam database

Create a new web application project "WizardInterface".

1. Add Exam.xml and Exam.xsd to the project.
2. Add an HTML table with a single column and two rows. The top cell will contain the MultiPage and the second will contain a "Next" button for navigating through the pages.
3. Add a RadMultiPage to the upper table cell.
4. Add standard ASP button to the lower table cell. Set the button ID to "btnNext" and the Text property to "Start".
5. The aspx code should look similar to the example below:

```
From Default.aspx:
<table>
    <tr>
        <td>
              <radTS:RadMultiPage ID="RadMultiPage1" runat="server" />
        </td>
    </tr>
    <tr>x
         <td align="right">
              <asp:Button ID="btnNext" runat="server" Text="Start" />
        </td>
    </tr>
</table
```

..and the user interface should look like the figure below. Except for hooking up a couple of events, this will be essentially all the design time work we do. The rest will take place in code.

The configured UI

6. Reference the Telerik.WebControls namespace:

```
using Telerik.WebControls;
```

7. In the designer, double click the button to create an Click event handler. Enter the following code to advance the RadMultiPage SelectedIndex each time the button is clicked.

```
C# Example:
protected void StartButton_Click(object sender, EventArgs e)
{
     RadMultiPage1.SelectedIndex += 1;
}

VB Example:
protected Sub StartButton_Click(ByVal sender As Object, ByVal e As EventArgs)
  RadMultiPage1.SelectedIndex += 1
End Sub
```

Next we need to create a few properties. The first, "Exam", will read in and cache the data needed to populate the test. Two other properties "IsFirstPage" and "IsLastPage" help control the button visibility (we don't want to see the button on the very last page), and the text (we should see "Start" on the first page and "Next" on the following pages.

8. Enter the code that defines the Exam property. The property returns a populated instance of the ExamDataset you added to the project earlier. This is the typed dataset contains structure for exams, questions and responses. The accessor for this property checks to see if the dataset exists in the cache and creates it if not found. When the dataset is first created call the dataset's ReadXML() method to populate it with data from "exam.xml".

```
C# Example:
public ExamDataset Exam
{
    get
    {
          ExamDataset examDataSet = (ExamDataset)Cache["Exam"];
        if (examDataSet == null)
        {
            examDataSet = new ExamDataset();
              examDataSet.ReadXml(Server.MapPath("exam.xml"));
            Cache["Exam"] = examDataSet;
        }
        return examDataSet;
    }
}

VB Example:
Public ReadOnly Property Exam() As ExamDataset
 Get
   Dim examDataSet As ExamDataset = CType(Cache("Exam"), ExamDataset)
   If examDataSet Is Nothing Then
      examDataSet = New ExamDataset
        examDataSet.ReadXml(Server.MapPath("exam.xml"))
      Cache("Exam") = examDataSet
   End If
   Return examDataSet
 End Get
End Property
```

9. Enter code for IsLastPage and IsFirstPage properties. These properties indicate the relative position of the selected property to the total number of pages in the MultiPage. These are included to help readability in later code.

```
C# Example:
public bool IsLastPage
{
    get
    {
        return RadMultiPage1.SelectedIndex == RadMultiPage1.PageViews.Count - 1;
    }
}

public bool IsFirstPage
{
    get
    {
        return RadMultiPage1.SelectedIndex == 0;
    }
```

```
}

VB Example:
Public ReadOnly Property IsLastPage() As Boolean
 Get
   Return RadMultiPage1.SelectedIndex = RadMultiPage1.PageViews.Count - 1
 End Get
End Property

Public ReadOnly Property IsFirstPage() As Boolean
 Get
   Return RadMultiPage1.SelectedIndex = 0
 End Get
End Property
```

10. To control the buttons text and visibility, override OnPreRender:

```
C# Example:
protected override void OnPreRender(EventArgs e)
{
    base.OnPreRender(e);
     btnNext.Text = IsFirstPage ? "Start" : "Next";
     btnNext.Visible = !IsLastPage;
}

VB Example:
Protected Overloads Overrides Sub OnPreRender(ByVal e As EventArgs)
  MyBase.OnPreRender(e)
  btnNext.Text = Microsoft.VisualBasic.IIf(IsFirstPage,"Start","Next")
 btnNext.Visible = Not IsLastPage
End Sub
```

11. The next few steps involve creating and populating all the pages for the exam. Pseudo code for the general calling order will start on the page load:

```
BindMultiPage()
   BuildStartPage(PageView)
  for each question in the exam:
     BuildPageViewContents(PageView, Question)
   BuildEndPage(PageView)
```

12. Notice that these methods take a PageView as a parameter where you might expect them to create a PageView and *return* the PageView instead. The reason for this is that because the MultiPage is not databound, an event called PageViewItemCreated(PageView, viewIndex) is provided to allow you to recreate the contents of each page. We will want to reuse our page building code and so the signatures must match.

13. Start by adding beginning and ending page building methods BuildStartPage and BuildEndPage. These methods just create literal controls with the test information and add them to the PageView controls array.

```
C# Example:
private void BuildStartPage(PageView pageView)
{
    const string format = "Welcome to the {0} test." +
   "<br>There are {1} questions. Press start to begin the test.";
    string message = string.Format(format, this.Exam.
   _Exam[0].ExamTitle, this.Exam.Question.Count);
     pageView.Controls.Add(new LiteralControl(message));
```

```
}

private void  BuildEndPage(PageView  pageView)
{
    const string format = "This concludes the {0} test.";
    string message = string.Format(format, this.Exam._Exam[0].ExamTitle);
     pageView.Controls.Add(new  LiteralControl(message));
}

VB Example:
Private Sub  BuildStartPage(ByVal  pageView As  PageView)
 Const format As String = "Welcome to the {0} test.<br>" + _
    There are {1} questions. Press start To begin the test."
 Dim message As String  =  String.Format(format,  Me.Exam._Exam(0).ExamTitle,  _
    Me.Exam.Question.Count)
  pageView.Controls.Add(New  LiteralControl(message))
End Sub

Private Sub  BuildEndPage(ByVal  pageView As  PageView)
 Const format As String = "This concludes the {0} test."
 Dim message As String  =  String.Format(format,  Me.Exam._Exam(0).ExamTitle)
  pageView.Controls.Add(New  LiteralControl(message))
End Sub
```

14. Next create a method that will state the question and list the responses for that question on a page. This method will be sent a populated QuestionRow from the Exam dataset. The question will be added to the page as a literal control. Then a method called LoadResponses() will be called.

```
C# Example:
private void  BuildPageViewContents(PageView  pageView,  ExamDataset.QuestionRow  question)
{
    // Load the question
     pageView.Controls.Add(new  LiteralControl(question.QuestionText));
     pageView.Controls.Add(new  LiteralControl("  "));
    // Load responses to the question
     LoadResponses(pageView, question);
}

VB Example:
Private Sub  BuildPageViewContents(ByVal  pageView As  PageView, _
   ByVal  question As  ExamDataset.QuestionRow)
  pageView.Controls.Add(New  LiteralControl(question.QuestionText))
  pageView.Controls.Add(New  LiteralControl("  "))
  LoadResponses(pageView,  question)
End Sub
```

15. Create the LoadResponses() method which passes PageView and QuestionRow parameters. The method filters the dataset to only responses for the question passed in, cycle through the responses for that question, creating and adding controls to the PageView as it goes. Based on the type of question (single or multiple response) either radio buttons or check boxes are used to populate the list of responses.

```
C# Example:
private void  LoadResponses(PageView  pageView,  ExamDataset.QuestionRow  question)
{
    string filter = "QuestionID = "  +  question.QuestionID.ToString();
     DataRow[] responses = this.Exam.Response.Select(filter);
```

```
    if (question.QuestionTypeID == 1) // single response
    {
         RadioButtonList radioButtonList = new RadioButtonList();
        foreach (ExamDataset.ResponseRow row in responses)
             radioButtonList.Items.Add(
          new ListItem(row.ResponseText, row.ResponseID.ToString()));
           pageView.Controls.Add(radioButtonList);
    }
    else // multiple response
    {
         CheckBoxList checkBoxList = new CheckBoxList();
        foreach (ExamDataset.ResponseRow row in responses)
             checkBoxList.Items.Add(
          new ListItem(row.ResponseText, row.ResponseID.ToString()));
           pageView.Controls.Add(checkBoxList);
    }
}

VB Example:
Private Sub LoadResponses(ByVal pageView As PageView, _
   ByVal question As ExamDataset.QuestionRow)
 Dim filter As String = "QuestionID = " + question.QuestionID.ToString
 Dim responses As DataRow() = Me.Exam.Response.Select(filter)
 If question.QuestionTypeID = 1 Then
   Dim radioButtonList As RadioButtonList = New RadioButtonList
   For Each row As ExamDataset.ResponseRow In responses
      radioButtonList.Items.Add(New ListItem(row.ResponseText, row.ResponseID.ToString))
   Next
     pageView.Controls.Add(radioButtonList)
 Else
   Dim checkBoxList As CheckBoxList = New CheckBoxList
   For Each row As ExamDataset.ResponseRow In responses
      checkBoxList.Items.Add(New ListItem(row.ResponseText, row.ResponseID.ToString))
   Next
     pageView.Controls.Add(checkBoxList)
 End If
End Sub
```

16. Now pull the pieces together with the BindMultiPage() method. The basic flow for each part of this method is, create a PageView, call a method to populate it, then add the PageView to the MultiPageView PageViews collection. The only exception is that BuildPageViewContents requires a QuestionRow to be passed. Finally the SelectedIndex is set to zero so the start page will be visible when the application first runs. Notice that we're using the PageView ID property to store the Question ID for later retrieval. We assign the start and end page -1 and -2 respectively.

```
C# Example:
private void BindMultiPage()
{
     PageView startPageView = new PageView();
     startPageView.ID = "-1";
      BuildStartPage(startPageView);
       RadMultiPage1.PageViews.Add(startPageView);

    foreach (ExamDataset.QuestionRow question in this.Exam.Question)
    {
         PageView pageView = new PageView();
          pageView.ID = question.QuestionID.ToString();
          BuildPageViewContents(pageView, question);
          RadMultiPage1.PageViews.Add(pageView);
```

```
    }

     PageView endPageView = new  PageView();
     endPageView.ID = "-2";
      BuildEndPage(endPageView);
       RadMultiPage1.PageViews.Add(endPageView);

      RadMultiPage1.SelectedIndex = 0;
}
```

```
VB Example:
Private Sub  BindMultiPage()
 Dim  startPageView As PageView = New  PageView
 startPageView.ID = "-1"
  BuildStartPage(startPageView)
   RadMultiPage1.PageViews.Add(startPageView)
 For Each  question As  ExamDataset.QuestionRow In  Me.Exam.Question
   Dim  pageView As PageView = New  PageView
    pageView.ID = question.QuestionID.ToString
     BuildPageViewContents(pageView,  question)
      RadMultiPage1.PageViews.Add(pageView)
 Next
 Dim  endPageView As PageView = New  PageView
 endPageView.ID = "-2"
  BuildEndPage(endPageView)
   RadMultiPage1.PageViews.Add(endPageView)
 RadMultiPage1.SelectedIndex = 0
End Sub
```

17.Call BindMultiPage() on the PageLoad:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
         BindMultiPage();
    }
}
```

```
VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
    BindMultiPage
 End If
End Sub
```

18.The last issue to be dealt with is that page contents must be recreated. The PageViewItemCreated event passes a PageView and index of the page. We are less concerned with what index the page is and more with what question was on that page. So having populated the PageView ID property with the questions id helps us reconnect to the underlying data. First create a helper function that retrieves the QuestionRow using the PageView ID. Here the PageView ID, which is actually the question ID, is used to build a filter statement.

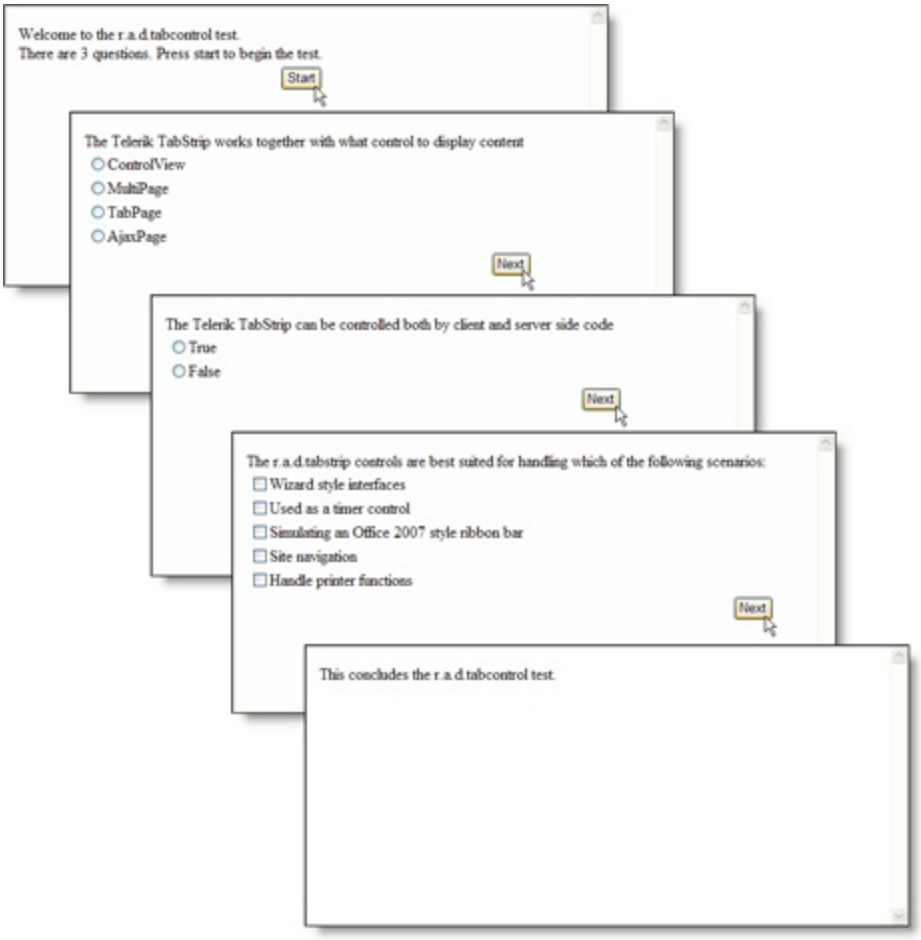
```
C# Example:
private  ExamDataset.QuestionRow  GetQuestionRow(PageView  view)
{
     DataRow[] row = Exam.Question.Select("QuestionID = " + view.ID);
```

```
    return   (ExamDataset.QuestionRow)row[0];
}

VB  Example:
Private Function  GetQuestionRow(ByVal view As  PageView) As   ExamDataset.QuestionRow
 Dim row As  DataRow() = Exam.Question.Select("QuestionID = " + view.ID)
 Return  CType(row(0),  ExamDataset.QuestionRow)
End Function
```

19.Use TryParse() to convert the id into a number that can be used in the switch statement. The first and last pages are indicated by ID's of -1 and -2. For all others we use GetQuestionRow to retrieve the corresponding question used to build the page content.

```
C#  Example:
protected void  RadMultiPage1_PageViewItemCreated(PageView  view, int  viewIndex)
{
    int id = 0;
    if (int.TryParse(view.ID, out id))
    {
        switch (id)
        {
            case -1:  BuildStartPage(view); break;
            case -2:  BuildEndPage(view); break;
            default:   BuildPageViewContents(view,   GetQuestionRow(view)); break;
        }
    }
}

VB  Example:
protected Sub   RadMultiPage1_PageViewItemCreated(ByVal view As PageView, _
   ByVal  viewIndex As Integer)
 Dim id As Integer = 0
 If  Integer.TryParse(view.ID,  id) Then
   Select id
   Case -1
      BuildStartPage(view)
     ' break
   Case -2
      BuildEndPage(view)
     ' break
   Case Else
       BuildPageViewContents(view,  GetQuestionRow(view))
     ' break
   End Select
 End If
End Sub
```

20.Run the application.

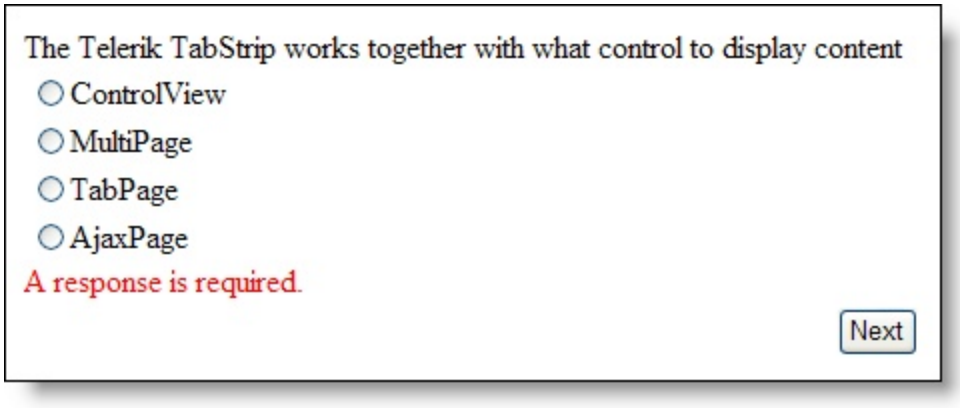


The running application

Lab: Validation

In this lab you will learn how to validate the contents of a PageView and prevent navigation for invalid input.

You can perform validation on the content of each PageView in a RadMultiPage. In this example we extend the Quiz Wizard to validate that the user has entered at least one response before allowing navigation to the next page.



Validator displaying message and preventing navigation

1. Set the RadMultiPage *RenderSelectedPageOnly* property to "True". This is a critical piece to remember when validating against the RadMultiPage to prevent all the validators from firing at once.

2. Create a helper property to get the current page view. This will enhance the readability of the code.

```
C# Example:
public PageView CurrentPageView
{
    get
    {
        return    RadMultiPage1.PageViews[RadMultiPage1.SelectedIndex];

    }
}

VB Example:
Public readonly Property CurrentPageView() As PageView
 Get
   Return    RadMultiPage1.PageViews(RadMultiPage1.SelectedIndex)
 End Get
End Property
```

3. In the StartButton_Click event handler, verify that the page is validated on the server and don't navigate to the next page if the page is not valid.

```
C# Example:
protected void StartButton_Click(object sender, EventArgs e)
{
     Page.Validate();
    if (Page.IsValid)
    {
          RadMultiPage1.SelectedIndex += 1;
    }
}

VB Example:
protected Sub StartButton_Click(ByVal sender As Object, ByVal e As EventArgs)
Page.Validate
If Page.IsValid Then
RadMultiPage1.SelectedIndex += 1
End If
End Sub
```

4. At the end of the LoadResponses() method add a CustomValidator to the PageView. We're using a CustomValidator because we want to validate CheckBoxList and RadioButtonList controls which is not supported by the RequiredFieldValidator. Add a ServerValidate event handler to be coded later.

```
C# Example:
CustomValidator valResponses = new CustomValidator();
valResponses.ErrorMessage = "A response is required.";
valResponses.ServerValidate += new ServerValidateEventHandler(valResponses_ServerValidate);
pageView.Controls.Add(valResponses);

VB Example:
Dim valResponses As CustomValidator = New CustomValidator
valResponses.ErrorMessage = "A response is required."
valResponses.ServerValidate += New ServerValidateEventHandler(valResponses_ServerValidate)
pageView.Controls.Add(valResponses)
```

5. In the ServerValidate event handler retrieve the QuestionRow for the current page. The QuestionRow tells us what kind of control to look for, RadioButtonList or CheckBoxList. Find the control on the page, cast it to the correct type and see if anything has been selected from the list.

```
C# Example:
void valResponses_ServerValidate(object source, ServerValidateEventArgs args)
{
     ExamDataset.QuestionRow question = GetQuestionRow(this.CurrentPageView);
    Control list = this.CurrentPageView.FindControl("List" + question.QuestionID);
    bool isSingleResponse = (question.QuestionTypeID == 1);
     args.IsValid = isSingleResponse ?
   (list as RadioButtonList).SelectedIndex != -1 :
       (list as CheckBoxList).SelectedIndex != -1;
}

VB Example:
Sub valResponses_ServerValidate(ByVal source As Object, ByVal args As ServerValidateEventArgs)
 Dim question As ExamDataset.QuestionRow = GetQuestionRow(Me.CurrentPageView)
 Dim list As Control = Me.CurrentPageView.FindControl("List" + question.QuestionID)
 Dim isSingleResponse As Boolean = (question.QuestionTypeID = 1)
 args.IsValid =
      Microsoft.VisualBasic.IIf(isSingleResponse,Not
      ((CType(ConversionHelpers.AsWorkaround(list, _
    GetType(RadioButtonList)), RadioButtonList)).SelectedIndex = -1),Not
      ((CType(ConversionHelpers.AsWorkaround(list, _
    GetType(CheckBoxList)), CheckBoxList)).SelectedIndex = -1))
End Sub
```

Loading User Controls

User controls are used extensively in many production applications. To integrate with existing applications you may need to add user controls dynamically to a PageView. This can be done by using the Page's LoadControl() method, as you would in any other ASP.NET application, and then add the control to the appropriate PageView.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
         PageView1.Controls.Add(this.LoadControl("WebUserControl1.ascx"));
         PageView2.Controls.Add(this.LoadControl("WebUserControl2.ascx"));
```

```
        PageView3.Controls.Add(this.LoadControl("WebUserControl3.ascx"));
    }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
      PageView1.Controls.Add(Me.LoadControl("WebUserControl1.ascx"))
      PageView2.Controls.Add(Me.LoadControl("WebUserControl2.ascx"))
      PageView3.Controls.Add(Me.LoadControl("WebUserControl3.ascx"))
 End If
End Sub
```

Focus To Control

Most data entry applications control the initial focus on the form to eliminate unnecessary mouse clicks. You can use the Focus() method of a control if there is a post back. In this example the RadTabStrip *AutoPostBack* property is set to true, then the Focus() method acts in the same manner as any standard ASP.NET control.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
     RadDateInput1.Focus();
}

protected void RadTabStrip1_TabClick(object sender, Telerik.WebControls.TabStripEventArgs e)
{
    switch (e.Tab.Index)
    {
        case 0:  RadDateInput1.Focus(); break;
        case 1:  RadMaskedTextBox2.Focus(); break;
        case 2:  RadDateInput3.Focus(); break;
    }
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  RadDateInput1.Focus
End Sub

protected Sub RadTabStrip1_TabClick(ByVal sender As Object, _
 ByVal e As   Telerik.WebControls.TabStripEventArgs)
 Select e.Tab.Index
 Case 0
    RadDateInput1.Focus
   ' break
 Case 1
    RadMaskedTextBox2.Focus
   ' break
 Case 2
    RadDateInput3.Focus
   ' break
 End Select
End Sub
```

2.4.4 Client Scripting with RadTabStrip

In this section you will learn how to manipulate properties and events in JavaScript on the client.

- Obtain references to RadTabStrip and RadMultiPage in JavaScript.
- Iterate arrays of tabs.
- Find a specific tab.
- Change the properties of a tab on the client.
- Learn multiple ways to add and remove client side events.
- Navigate the RadMultiPage on the client.

Client side access to RadTabStrip and RadMultiPage

Much of the server side functionality of RadTabStrip is replicated on the client. In client script you can reference a RadTabStrip and its collection of tabs. From there you have access to client side events, methods and properties. The first step is to get a reference to the RadTabStrip or RadMultiPage control. The recommend way of doing this (which will work when your control is nested inside a user control or master page) is to emit the ClientID into the JavaScript:

```
<%= RadTabStrip1.ClientID %>
```

For example, get a reference to the RadTabStrip and iterate the Tabs collection:

```
var tabStrip = <%= RadTabStrip1.ClientID %>;

var allTabs = tabStrip.GetAllTabs();
```

Likewise the RadMultiPage can be accessed through its ClientID:

```
var multiPage = <%= RadMultiPage1.ClientID %>;

multiPage.SelectPageByIndex(1);
```

Iterating and finding Tabs

Telerik provides two tab collections:

- Tabs: Just the root level tabs only.
- AllTabs: Root and child level tabs.

You can iterate these on the client:

```
var tabStrip = <%= RadTabStrip1.ClientID %>;

for (var i=0; i<tabStrip.Tabs.length; i++)
{
   // do something with tabStrip.Tabs[i]
}
```

There are four "FindTabByX" client methods to locate a given tab instance on the client:

- FindTabById
- FindTabByText
- FindTabByValue
- FindTabByUrl

Each method takes a string value.

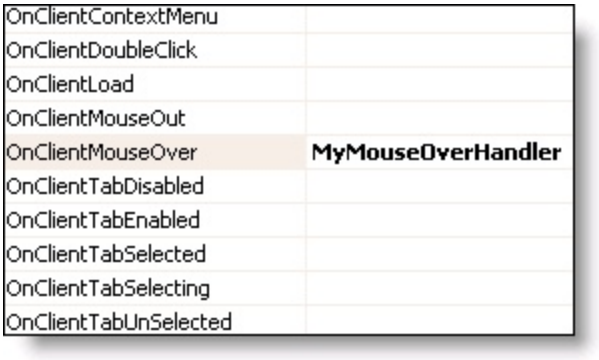
```
var decafTab = tabStrip.FindTabByText("Decaf");
decafTab.Disable();
```

Once you have access to a tab instance you can use its methods and properties. For example:

```
myTab.Enable();
myTab.Select();
myTab.SetText("some text that I want to persist between post backs");
alert("location within the AllTabs collection is " + myTab.GlobalIndex);
alert("this tab has " + myTab.Tabs.Length + " child tabs");
```

RadTabStrip events

For straightforward cases, client side event handlers can be assigned directly in the designer:



RadTabStrip client side event handlers

Or you can assign an event handler in JavaScript code:

```
function MyMouseOverHandler(sender, eventArgs)
{
   alert(sender.ID + " " + eventArgs.Tab.Text);
}

tabStrip.OnClientMouseOver = MyMouseOverHandler;
```

The handlers can be assigned by reference:

```
tabStrip.OnClientMouseOver = MyMouseOverHandler;
```

...or by name:

```
tabStrip.OnClientMouseOver = "MyMouseOverHandler";
```

...or by directly assigning JavaScript:

```
tabStrip.OnClientMouseOver = "alert('OnClientMouseOver event fired')";
```

These are all examples of "singlecast" events. The event fires and a single event handler responds. If you require more flexibility, the RadTabStrip "multicast" event handling allows multiple listeners for a single event. The following example shows two event handlers responding to the same OnClientMouseOver event. The events are handled in the order they are attached.

```
tabStrip.AttachEvent('OnClientMouseOver', 'MouseOverHandler1');
tabStrip.AttachEvent('OnClientMouseOver', 'MouseOverHandler2');
```

If you want to stop or start events from firing you can use the RadTabStrip EnableEvents() and DisableEvents() methods. Typically, for events ending in "ing" you can cancel the event by returning false. But what if you want to selectively shut off some events altogether but leave other events firing? Then the client events can be selectively detached:

```
tabStrip.DetachEvent('OnClientTabSelecting', 'SelectingHandler');
```

Note: If you want to detach events in your client code, be sure to use the AttachEvent() method to assign them in the first place. If you assign them in the designer for example, the DetachEvent() method will have no effect.

Lab: Exploring Client events, functions and properties

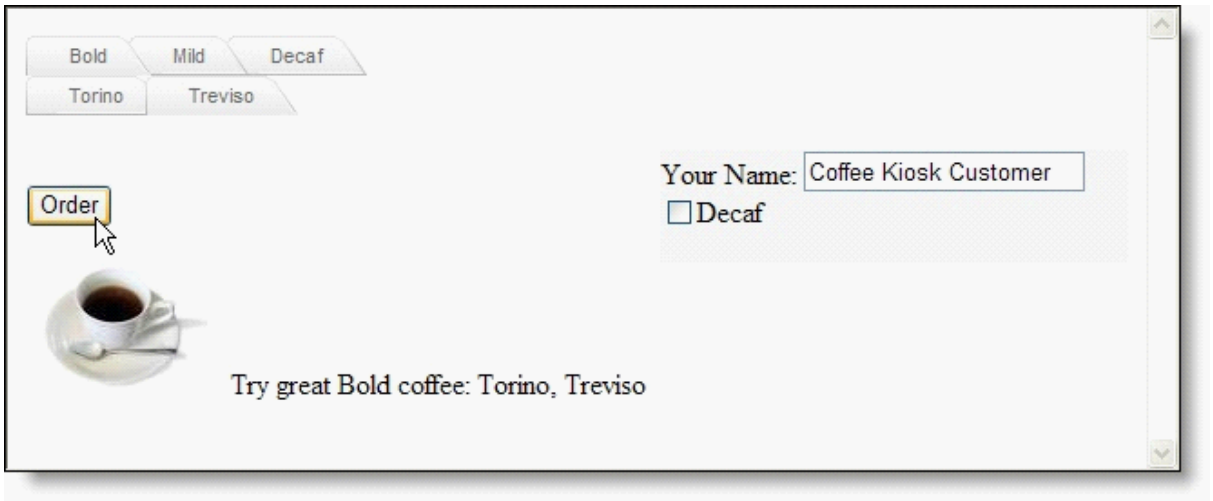
The following project combines a number of client side techniques that show the RadTabStrip client API in action. In this lab you will learn client side techniques to:

- Reference the RadTabStrip and RadMultiPage.
- Enumerate root level and child tabs.
- Enable/Disable tabs.
- Hook up multiple event handlers to a single client event.
- Respond to events for tab selection and mouse over.
- Prevent navigation to a tab.
- Attach and detach events at runtime.
- Find tabs by text.
- Navigate to a specific page within a RadMultiPage.

The Coffee Kiosk project performs its work on the client side only and has the following behavior:

- Allows the user to move the mouse over types of coffee (Bold, Mild, Decaf) to see a message below with the coffees available for that type. The coffee cup image also changes as the mouse moves over the tabs to provide additional visual cues.
- When the user clicks a type-of-coffee (one of the root level tabs) a confirmation dialog either allows or prevents navigation to the tab.
- When a coffee is selected (one of the child items), the order button becomes enabled.

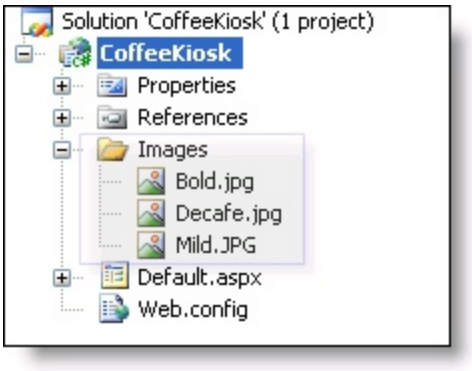
- If the Decaf checkbox is checked, Bold and Mild tabs are disabled.
- When the order button is clicked, a status message shows the selected coffee.



Placing an order at the Coffee Kiosk

Setting up the project

1. Create a new project named "CoffeeKiosk".
2. Add an "Images" folder and copy the images "bold.jpg", "medium.jpg" and "decaf.jpg" from the rad tabstrip\projects\images folder. Your project should now look like the figure below.

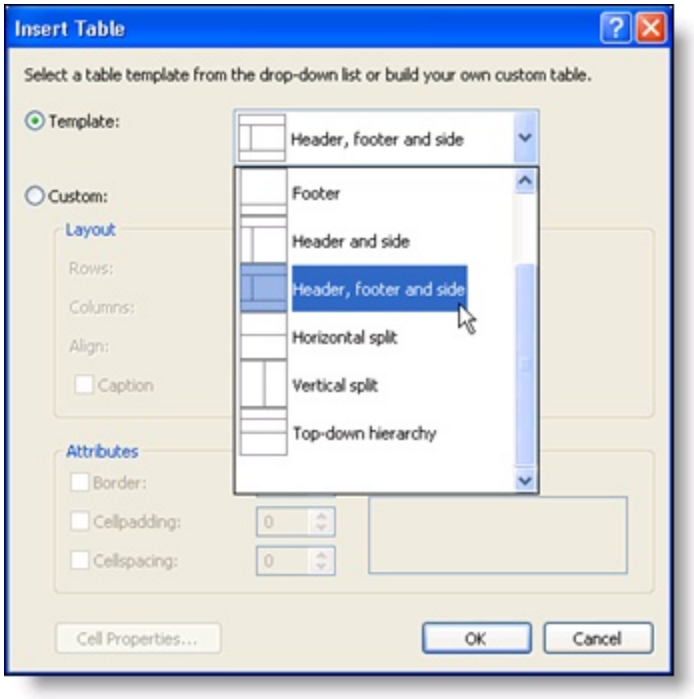


Project with images added

Setting up the page design

3. Setup an HTML table to organize the visual elements of the project.

- In the Visual Studio design view of the default web form select Layout | Insert Table.
- Select "Header, footer and side" from the list.
- Check each of the four cells of the table in the Properties window and remove any Style settings (there may be hard coded width settings).
- In the right side cell, set the Style property to: "BACKGROUND-COLOR: whitesmoke".



Inserting an html table from the Template option

4. Populate the html table with controls:

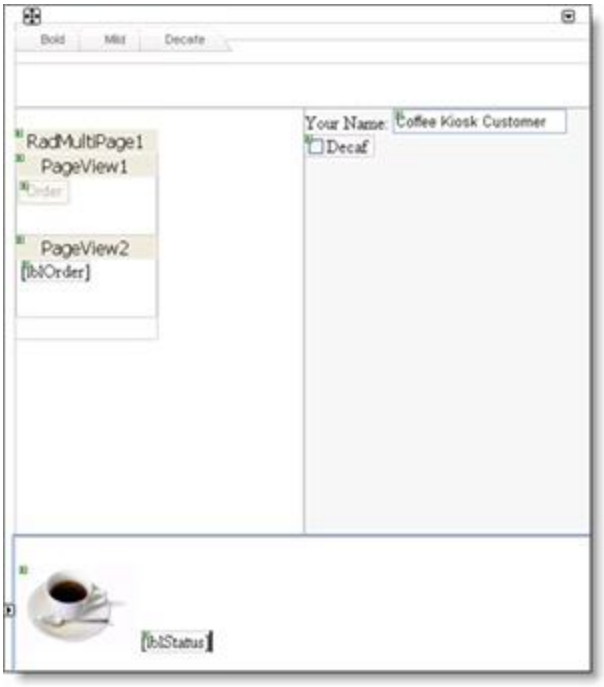
- Drag a new RadTabStrip component to the upper table cell. Add new root level and child tabs to match the figure below:

Tabs setup

- Drag a new RadMultiPage component to the left hand cell.
- By default the RadMultiPage will contain PageView1. Drag a second PageView to the RadMultiPage for a total of two pages.
- In PageView1 place a Standard ASP.NET Button control and set the *ID* property to "btnOrder" the *UseSubmitBehavior* property to false, and *Enabled* to false. Setting the UseSubmitBehavior property to false prevents a postbox.

- In PageView2 place a Standard ASP.NET Label control and set the *ID* property to "lblOrder".
- In the right hand table cell write the literal text "Your Name:".
- To the right of "Your Name:" drag a new Standard ASP.NET TextBox and set its *ID* property to "tbName" and *Text* property to "Coffee Kiosk Customer".
- Below the TextBox control add a Standard ASP.NET Checkbox, set its *ID* property to "cbDecaf" and *Text* property to "Decaf".
- In the lower cell of the HTML table place a Standard ASP.NET Image control, set the *ID* property to "imgCup" and *ImageUrl* property to "~/Images/Bold.jpg".
- To the right of the image place another Standard ASP.NET Label control and set the *ID* property to "lblStatus".

The page design should now look like the figure below:



The completed page design

Setting up the client code

5. Edit the HTML In the Visual Studio Source view so that the <head> tag includes <title> and <script> tags. The rest of the lab focuses on creating the JavaScript code that fills the script tag.

```
<head>

     <title>Coffee Kiosk</title>

     <script type="text/javascript">
        //<!--

              // your JavaScript code here

        //-->
```

```
    </script>

</head>
```

Setting up the code behind

6. The only code required by this project is to hook up client events for standard ASP.NET controls. The JavaScript functions themselves we will write later. Enter the following code to the form page load.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
     btnOrder.Attributes["onclick"] = "PlaceOrderHandler()";
     cbDecaf.Attributes["onclick"] = "DecafClickHandler()";
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
      btnOrder.Attributes("onclick") = "PlaceOrderHandler()"
      cbDecaf.Attributes("onclick") = "DecafClickHandler()"
End Sub
```

Attaching events

7. If we attach the client side event handlers in code, we can remove them later at will. Here we add handlers for OnClientTabSelected, OnClientTabSelecting, and OnClientMouseOver. We will detach the Selected and Selecting handlers later when the order is placed but leave the mouse over handlers. Also notice that there are two handlers for the MouseOver event. Add the following code:

```
...
    </form>
</body>
     <script type="text/javascript">
        //<!--
             var tabStrip = <%= RadTabStrip1.ClientID %>;

               tabStrip.AttachEvent('OnClientTabSelected', 'SelectedHandler');
               tabStrip.AttachEvent('OnClientTabSelecting', 'SelectingHandler');
             tabStrip.AttachEvent('OnClientMouseOver', 'MouseOverHandler1');
             tabStrip.AttachEvent('OnClientMouseOver', 'MouseOverHandler2');
        //-->
    </script>
</html>
```

Enumerating tabs

8. Before we implement the event handlers we need a pair of helper functions. First create a function to retrieve a reference to the RadTabStrip object, iterate its Tabs collection and compare against a Tab object parameter. Because the Tabs collection only contains root level items, if we find a match, then the parameter "tab" is a root level tab.

Add the following JavaScript code:

```
function  isTopLevel(tab)
{
     // retrieve a reference to the RadTabStrip
     var tabStrip = <%= RadTabStrip1.ClientID %>;
```

```
    // iterate the root level tabs
    for (var i=0;  i<tabStrip.Tabs.length;  i++)
    {
       // if the text matches, "tab" is a root level tab.
        if (tab.Text  ==  tabStrip.Tabs[i].Text)
            return true;
    }
    return false;
}
```

Add a similar function to collect and concatenate the text from a collection of child tabs.

```
function   GetChildTabsText(tab)
{
    var childTabText = "";

    // iterate the child tabs of the passed in tab
    for (var i=0; i<tab.Tabs.length; i++)
    {
        // separate with commas
        if (childTabText != '')
        {
            childTabText += ", ";
        }
        // add the child tab text
         childTabText += tab.Tabs[i].Text;
    }
    return childTabText;
}
```

Implementing event handlers

Selecting

9. The OnClientTabSelecting eventArgs parameter provides a reference to the tab that's being selected. If you return false in this event handler the event is canceled, thus preventing navigation to the tab. If the tab is a child tab (one of the coffee choices), then we always display the tab. If the tab is a root level parent tab then we use a confirm dialog to ask the user if they want to go to that tab and cancel the event if they don't. Add this code to the JavaScript section.

```
function  SelectingHandler(sender,  eventArgs)
{
    // if this is a top level tab, ask the user if
    // they want to navigate to that tab; cancel
    // the event if they don't.
     if (isTopLevel(eventArgs.Tab))
    {
        return confirm("Do you want to see "+ eventArgs.Tab.Text + " coffee?");
    }
    else
    {
        return true;
    }
}
```

Selected

10. After the OnClientTabSelecting fires the OnClientTabSelected event takes over. At the top of the JavaScript tag section add a global variable that will contain a reference to a selected tab.

```
<script   type="text/javascript">
//<!--

    var selectedTab;
...
```

11. In the handler for OnClientTabSelected check if this is a root level tab and enable the order button only if the user has selected a coffee. Save a reference to this last selected tab so if the user clicks the order button we can tell what was ordered. Add the following to the JavaScript code.

```
function  SelectedHandler(sender,  eventArgs)
{
    // get a reference to standard asp.net control
     var  btnOrder  =  document.getElementById('btnOrder');

    // only place an order when a coffee tab is selected
     btnOrder.disabled  =  isTopLevel(eventArgs.Tab);

    // save the selected tab to use when placing the order
    selectedTab  =  eventArgs.Tab;
}
```

Mouse Over

12. Here we have two event handlers for the same OnClientMouseOver event. The event handlers fire in the order they were attached. The first mouse over event handler changes some status text to display the current type of coffee (the root level tab text) and the results of GetChildTabsText() (the concatenated text of the child tabs). Add this code to the JavaScript tag section.

```
function  MouseOverHandler1(sender,  eventArgs)
{
    // get a reference to a standard asp.net control
     var  lblOrder  =  document.getElementById('lblStatus');

    // if this is a top level control, display a status message
    // that contains the coffee type and the coffees of that
    // type.
     if  (isTopLevel(eventArgs.Tab))
    {
         lblOrder.innerHTML = "Try great " +
             eventArgs.Tab.Text +
            " coffee: " +
              GetChildTabsText(eventArgs.Tab);
    }
    return true;
}
```

13. This second handler is similar but uses the current root level tab text to build a corresponding image file name. Add this code to the JavaScript tag section.

```
function  MouseOverHandler2(sender,  eventArgs)
{
     // get a reference to a standard asp.net image control
      var imgCup = document.getElementById('imgCup');

     // if this is a top level tab, use the tab text to
     // get the corresponding image file name.
      if (isTopLevel(eventArgs.Tab))
    {
          imgCup.src = "images/" + eventArgs.Tab.Text + ".jpg";
    }
     return true;
}
```

Finding tabs, disabling, enabling

14. This function fires when the "Decaf" checkbox is clicked. It locates the top level tabs and enables or disables each tab depending on if they are decaf types of coffee. Here we use the FindTabByText() method to locate the tab because that's the only identifying info available. Once you have a reference to each of the top level tabs run Enable()/Disable() methods as appropriate. Add the following code to the javascript tag:

```
function  DecafClickHandler()
{
     // get a reference to the RadTabStrip control
     var tabStrip = <%= RadTabStrip1.ClientID %>;

     // get a reference to the checkbox control
      var cbDecaf = document.getElementById('cbDecaf');

     // get references to each top level tab
      var boldTab = tabStrip.FindTabByText("Bold");
      var mildTab = tabStrip.FindTabByText("Mild");
      var decafTab = tabStrip.FindTabByText("Decaf");

     // enable/disable if the tab represents a type of decaf coffee
      //if (document.form1.cbDecaf.checked)
     if (cbDecaf.checked)
    {
          boldTab.Disable();
          mildTab.Disable();
          decafTab.Enable();
    }
    else
    {
          boldTab.Enable();
          mildTab.Enable();
         decafTab.Disable();
    }
}
```

Detaching events, MultiPage navigation

15. This function is called when the order button is clicked by the user. It navigates the RadMultiPage to PageView2 that contains "lblOrder" with a "here's what you ordered" message. Notice that the tab selecting and selected client event handlers are detached at this point. From here on only the mouse over events that were not detached will continue to fire.

```
function  PlaceOrderHandler()
```

```
{
     // get a reference to the tab strip control
     var tabStrip = <%= RadTabStrip1.ClientID %>;

     // get a reference to the multi page control
     var multiPage = <%= RadMultiPage1.ClientID %>;

     // get references to standard asp.net controls
      var lblOrder = document.getElementById('lblOrder');
      var tbName = document.getElementById('tbName');

    //
     lblOrder.innerHTML = tbName.value + ", your " + selectedTab.Text + " is ready.";
      multiPage.SelectPageByIndex(1);
       tabStrip.DetachEvent('OnClientTabSelected', 'SelectedHandler');
       tabStrip.DetachEvent('OnClientTabSelecting', 'SelectingHandler');
}
```

2.4.5 Summary

This chapter covered how to use RadTabStrip controls in the designer to control visual appearance and behavior, set basic tab structures, design-time binding to data sources for xml and hierarchical data, and using tabstrip as a vehicle for navigating websites.

2.5 RadPanelBar

2.5.1 Getting Started

The Telerik RadPanelBar is for creating collapsible side-panelbar systems similar to Outlook®-bars to fully customizable graphical side-panelbars. The panelbar provides left/right side web-site navigation not suitable for dropdown navigation.

Like most Telerik controls, one uses the panelbar by dropping it from the ToolBox onto the WebForm designer and populating it manually in the designer, or programmatically in code-behind. Additionally, you can populate the panelbar by binding it to a datasource.

RadPanelBar in the toolbox

2.5.2 Using RadPanelBar in the Designer

This section will teach you how to create and configure a simple web page using the RadPanelBar. It will also illustrate how to modify various properties within the designer.

Lab: Create a Simple panelbar

This lab will teach you to build a simple radpanel using the designer. You will also learn how to apply skins to the radpanel.

1. Create a new ASP.NET Project named "Simple".
2. Create a folder in this project named \RadControls as shown in the figure below.

RadControls folder in Project

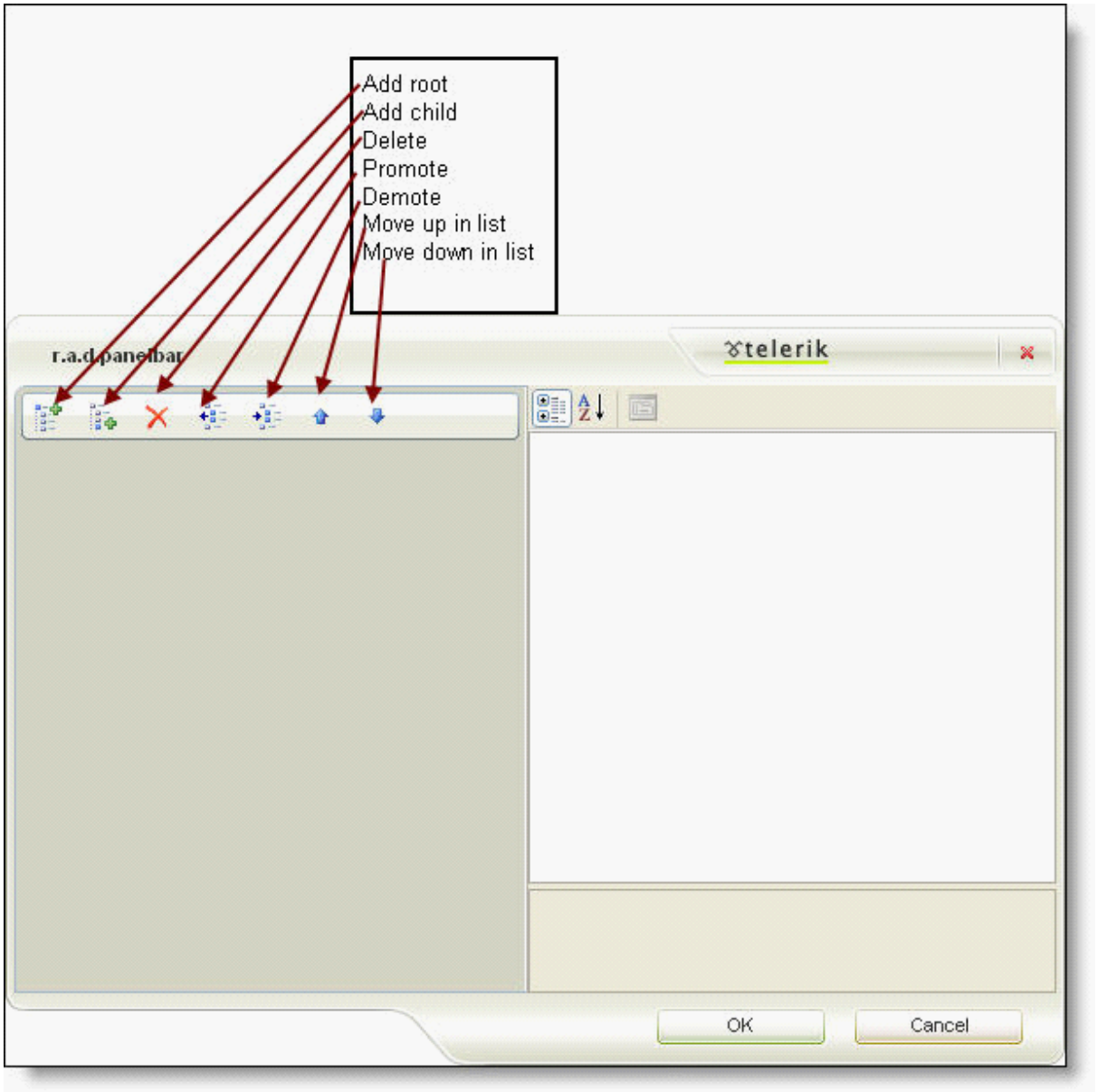
3. Copy the contents of the \PanelBar folder form the Telerik installation directory. This step supplies the Smart Tag with the option to apply visual styles to the entire RadPanelbar using AutoFormat.

   Note: Assuming you selected the default installation path, the directory location would be:
   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls\RadPanel

4. Add a RadPanelbar control to the default page and set it's ID property to rpbMain.
5. Click the Smart Tag button to invoke RadPanelBar Tasks as shown in the figure below.

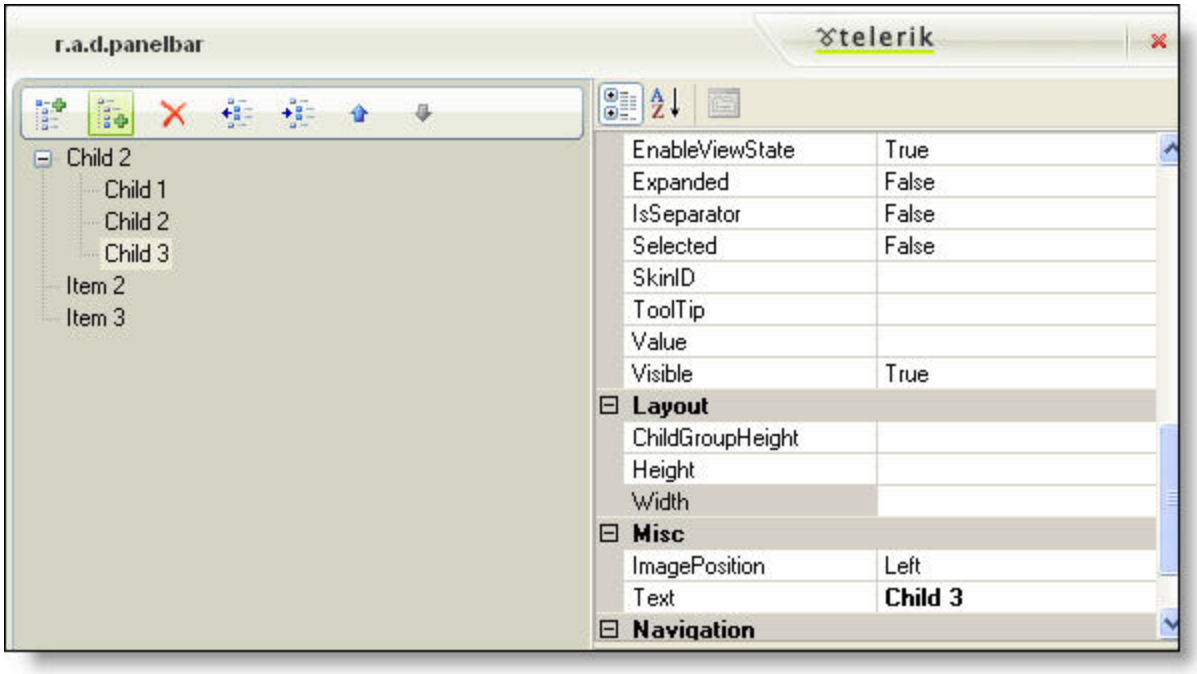
RadPanelBar Tasks Window

6. Start Building the panelbar by selecting "Build RadPanelbar..." link which will invoke the r.a.d RadPanelbar Builder dialog.



RadPanelbar Builder dialog

7. Add panel items as depicted in the figure below. To change the title of the panel items you must change the Text property accordingly. Press Ok to close the RadPanelbar Builder dialog.



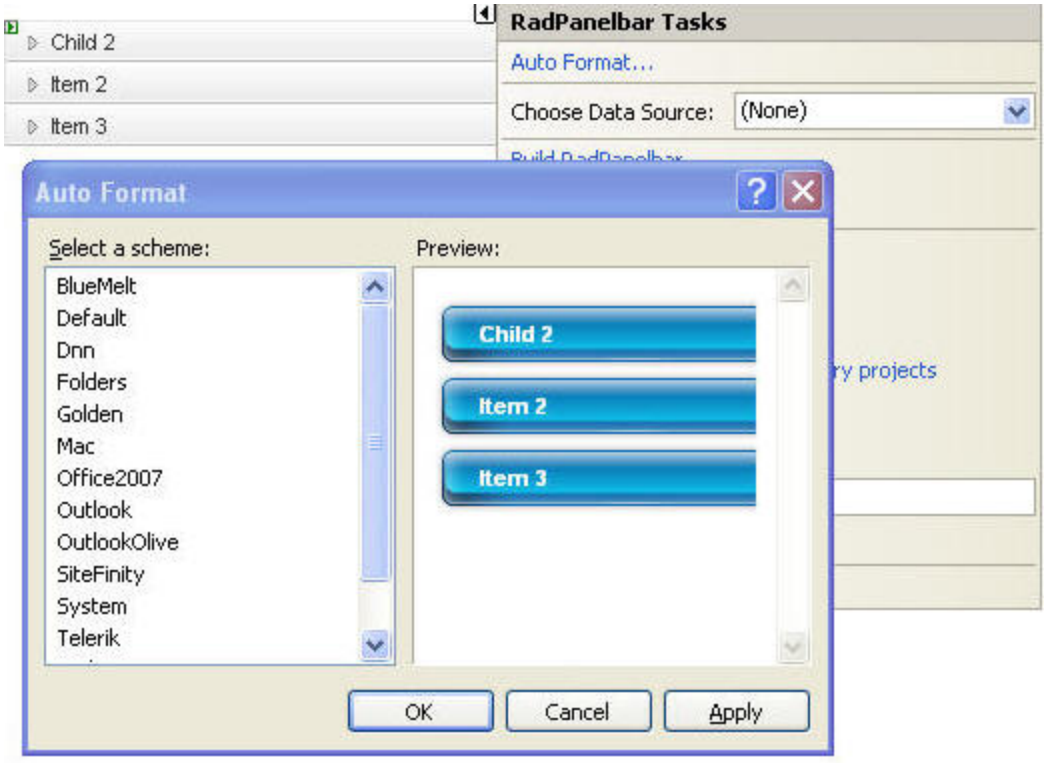
Adding panel items

8. Run the application to examine the radpanel.

Lab: Modify the RadPanelBar Skin

This Lab will illustrate how to change the RadPanelbar skin using the Auto Format dialog in the designer.

1. Using the project from the previous Lab, you can change the Skin of the RadPanelbar by clicking on the Smart Tag and then selecting "Auto Format..." link. This will invoke the Auto Format dialog shown in Figure 1.



Auto Format Dialog

2. Select the "Golden" skin and press OK.

3. This completes this lab. Execute the project to examine the modified skin.

2.5.3 Databinding with RadPanelBar

The r.a.d panelbar supports databinding to IListSource implementations such as DataSet, DataTable and DataView. It can also bind to IEnumerable and ICollection implementations such as ArrayLists.

The RadPanelBar can bind to a self-referencing table from which it obtains a hierarchical structure. The r.a.d panelbar automatically realizes the inherent hierarchy of hierarchical declarative data sources such as XmlDataSource and SiteMapDataSource. For flat declarative data sources such as SqlDataSource, you specify the hierarchy using the ID -> ParentID relationship model. These are specified in the RadPanelbar.DataFieldID and RadPanelBar.DataFieldParentID properties.

Lab: Binding to xml data

This lab will teach you how to bind the RadPanelBar to XML data sources.

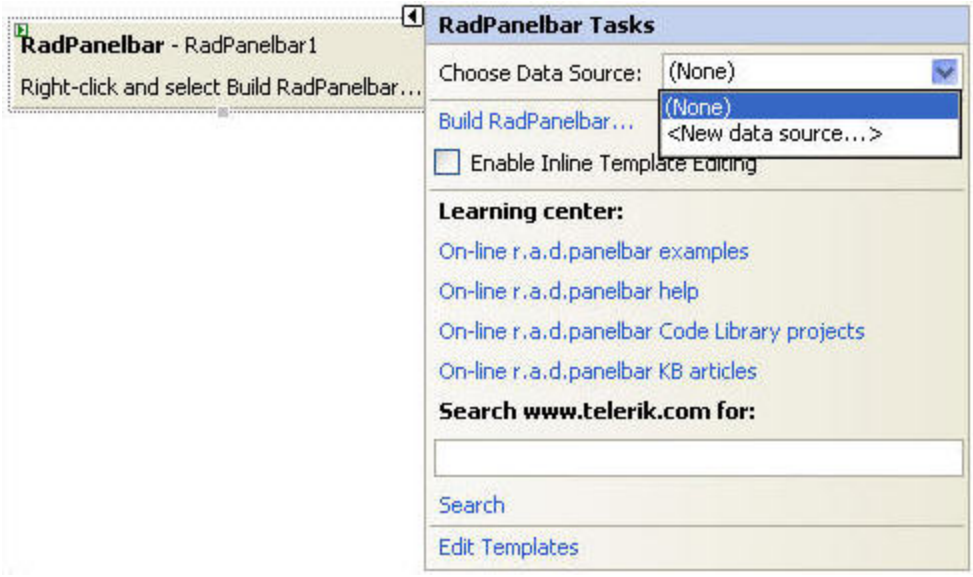
The RadPanelBar can easily bind to xml formatted data. This is particularly useful when the structure of the panelbar is relatively static, yet when change is required you need only modify the xml file.

This lab will use the following xml file (panelbar.xml).

```
<?xml  version="1.0"  encoding="utf-8"  ?>
<PanelItems>
     <PanelItem  Text="Products"  Expanded="True"  >
         <PanelItem Text="r.a.d.editor" />
         <PanelItem Text="RadPanelBar" />
         <PanelItem Text="r.a.d.menu" />
```

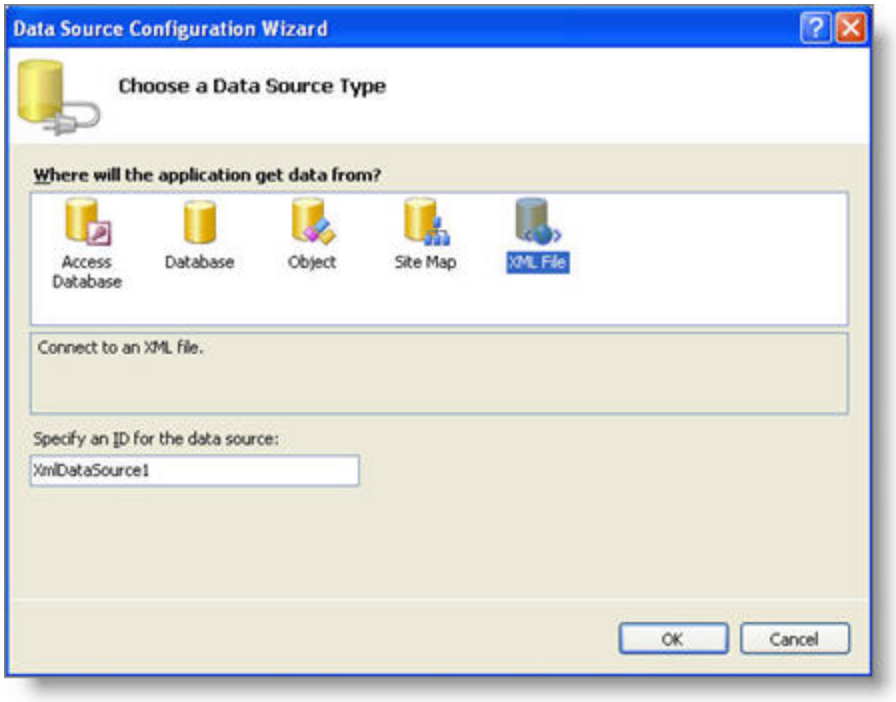
```
        <PanelItem Text="r.a.d.tabstrip" />
    </PanelItem>
    <PanelItem Text="Support" >
        <PanelItem Text="Knowledge Base" />
        <PanelItem Text="Forums" />
        <PanelItem Text="Articles" />
        <PanelItem Text="FAQ" />
    </PanelItem>
</PanelItems>
```

1. Create a new project named "XMLFile".
2. Add an XML file to the project and paste the panelbar.xml code from Listing 1 into this file.
3. Add a RadPanelbar from the ToolBox to the Web Form.
4. Using the Smart Tag "Choose Data Source", select New data source as shown in the figure below.



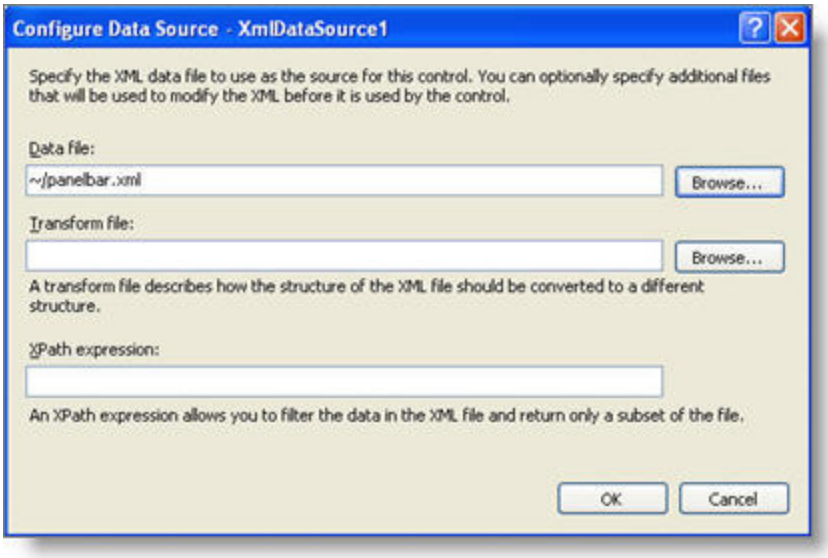
Selecting a New Data Source

5. In the Data Source Configuration Wizard choose XML File as the data source type, leave the default ID "XmlDataSource1" and click OK.



Data Source Configuration Wizard

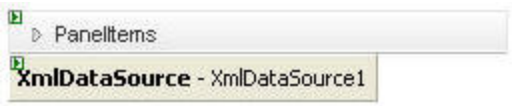
6. When the Configure Data Source Dialog is invoked, click the browse button to select panelbar.xml from the list. Click OK on each open dialog to finish configuring the data source.



Configure Data Source Dialog

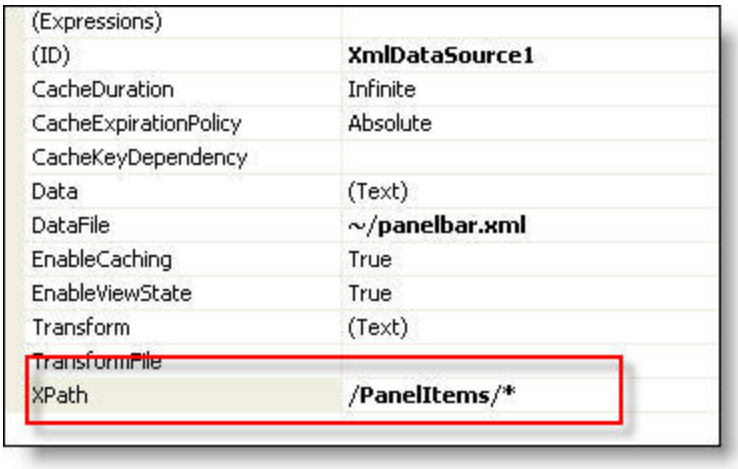
7. Set the RadPanelbar.DataTextField property to Text. This tells the panelbar which attribute of the XML file to display.

8. Notice that the first item within the RadPanelbar is the outer element <PanelItems>. To change this. select the XmlDataSource component.



Outer element set as first item

9. Set the XmlDataSource.XPath property to "/PanelItems/*" to specify the children of PanelItems as the root level for the panelbar.



Setting the XmlDataSource.XPath property

10. Execute this project to examine the XML data binding functionality.

Lab: Binding to hierarchical database data

This lab demonstrates how to bind the RadPanelBar to a database data source programmatically.

RadPanelbar has a set of properties that control binding characteristics. As you might expect there are properties that specify the data source, the visible text and the underlying value. Also look for properties in the table below that support hierarchical data and allow automatic navigation to specified URLs.

Property Name	Description
DataTextField	Used to bind the Text property to a DataColumn in the data source.
DataValueField	Used to bind the Value property to a DataColumn in the data source.
DataFieldID	Set this optional property to that of a DataColumn containing an ID.
DataFieldParentID	Set this optional property to that of a DataColumn containing the ParentID. When used, the DataFieldID and DataFieldParentID enforce the self-referencing model for hierarchical panel items.
DataSource	Set this mandatory field to the data source to which to bind.
DataNavigateUrlField	When used, binds the NavigateUrl property to a DataColumn in the data source.

You must first have a data source. This lab uses a database within SQL Server or SQL Server Express (MSDE). The following script creates the required table for this lab:

Table Creation script

```
SET ANSI_NULLS ON
```

```
GO
SET  QUOTED_IDENTIFIER  ON
GO
SET  ANSI_PADDING  ON
GO
CREATE  TABLE  [dbo].[radpanel](
       [Text]  [varchar](50)  COLLATE  SQL_Latin1_General_CP1_CI_AS  NOT  NULL,
       [ID] [int] NOT NULL,
       [ParentID]  [int]  NULL
) ON [PRIMARY]

GO
SET  ANSI_PADDING  OFF
```

Here is the script needed to populate the table just created:

Populate Script

```
insert into radpanel (Text, ID) values ('Politics', 1)
insert into radpanel (Text, ID, ParentID) values ('CNN', 2, 1)
insert into radpanel (Text, ID, ParentID) values ('NBC', 3, 1)
insert into radpanel (Text, ID, ParentID) values ('ABC', 4, 1)
insert into radpanel (Text, ID) values ('Sports', 5)
insert into radpanel (Text, ID, ParentID) values ('US Sports', 6, 5 )
insert into radpanel (Text, ID, ParentID) values ('European Sports', 7, 5)
insert into radpanel (Text, ID) values ('Events', 8)
insert into radpanel (Text, ID, ParentID) values ('Oscar Awards', 9, 8)
insert into radpanel (Text, ID, ParentID) values ('MTV Movie Awards', 10, 8)
```

To programmatically populate a r.a.d panelbar from a database

1) Create a new Web Application project

2) Drop a RadPanelbar onto the Web Page Designer

3) Go to the Code View for the Page and add the following code to the Page_Load event:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   if(!Page.IsPostBack)
  {
     SqlConnection sqlCon =
         new SqlConnection(@"Data Source=YOURDATABASE;Initial Catalog=telerik;" +
          "User ID=sa;Password=mypw");
     SqlDataAdapter sqlDa = new SqlDataAdapter( "select * from radpanel", sqlCon );
     DataSet dsradPanel = new DataSet();
     sqlDa.Fill( dsradPanel );
     RadPanelbar1.DataTextField = "Text";
     RadPanelbar1.DataFieldID = "ID";
    RadPanelbar1.DataFieldParentID  =  "ParentID";
     RadPanelbar1.DataSource = dsradPanel;
     RadPanelbar1.DataBind();
     RadPanelbar1.Items[1].Expanded = true;
  }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not Page.IsPostBack Then
```

```
    Dim sqlCon As SqlConnection = _
       New SqlConnection("Data Source=YOURDATABASE;_
         Initial Catalog=telerik;User ID=sa;Password=mypw")
    Dim sqlDa As SqlDataAdapter = New SqlDataAdapter("select * from radpanel", sqlCon)
    Dim dsradPanel As DataSet = New DataSet
     sqlDa.Fill(dsradPanel)
     RadPanelbar1.DataTextField = "Text"
     RadPanelbar1.DataFieldID = "ID"
     RadPanelbar1.DataFieldParentID = "ParentID"
     RadPanelbar1.DataSource = dsradPanel
     RadPanelbar1.DataBind
     RadPanelbar1.Items(1).Expanded = True
 End If
End Sub
```

4) When you run the application, you will see the RadPanelbar as shown in Figure 1.

RadPanelbar populated from database

2.5.4 Using RadPanelBar at Runtime

This section demonstrates how to work with the r.a.d panelbar at runtime. You will learn how to populate the panelbar using some of the classes and api functions specific to the r.a.d panelbar. You will also work with server side events.

Lab: Populating the panelbar

This lab demonstrates how to populate the r.a.d panelbar on the fly for those instances where RadPanelbar contents are not static and are determined at runtime. You will dynamically populate a RadPanelbar in code-behind using the constructor and properties of the RadPanelItem class.

The class, RadPanelItem, represents a single item in the RadPanelbar. The entire RadPanelbar is composed of a hierarchical list of RadPanelItem's. Root panel items are those that appear at level zero (see figure below). A panel item having a parent is called a sub-panel item.

Root PanelItems

1) Create a new project and name is "RuntimePop".

2) Add a RadPanelbar from the ToolBox to the Web Form.

3) In the code-behind add the following using statement which will make the RadPanelbar and RadPanelItem classes accessible in your code:

```
C# Example:
using Telerik.WebControls;

VB Example:
Imports Telerik.WebControls
```

4) Also in the code behind, add the following code to the Page_Load event handler (Listing 1). The Page_Load populates a RadPanelbar at runtime. Note that by using this approach, you could deterministically set properties to values based on some external criteria. For example, you could determine whether or not to add a particular item to the RadPanelbar based on a user's access rights.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  // Create first root item
   RadPanelItem rpItem = new RadPanelItem();
   rpItem.Text = "Politics";
   rpItem.CssClass = "MainItem";
   rpItem.Expanded = true;

   RadPanelbar1.Items.Add( rpItem ); // add root items directly to the RadPanelbar

  // Add sub-items
   RadPanelItem rpiSubItem = new RadPanelItem();
   rpiSubItem.Text = "CNN";
   rpItem.Items.Add( rpiSubItem ); // add sub-items to a root item.

   rpiSubItem = new RadPanelItem();
   rpiSubItem.Text = "NBC";
   rpItem.Items.Add( rpiSubItem );
```

```
   rpiSubItem = new  RadPanelItem();
   rpiSubItem.Text = "ABC";
   rpItem.Items.Add( rpiSubItem );


  // Create second root item
  rpItem = new  RadPanelItem();
  rpItem.Text = "Sports";
  rpItem.CssClass = "MainItem";
  rpItem.Expanded = false;

   RadPanelbar1.Items.Add( rpItem );

  // Add sub-items
  rpiSubItem = new  RadPanelItem();
  rpiSubItem.Text = "US Sports";
  rpItem.Items.Add( rpiSubItem );

  rpiSubItem = new  RadPanelItem();
  rpiSubItem.Text = "European Sports";
  rpItem.Items.Add( rpiSubItem );

  // Create third root item
  rpItem = new  RadPanelItem();
  rpItem.Text = "Events";
  rpItem.CssClass = "MainItem";
  rpItem.Expanded = true;

   RadPanelbar1.Items.Add( rpItem );

  // Add sub-items
  rpiSubItem = new  RadPanelItem();
  rpiSubItem.Text = "Oscar Awards";
  rpItem.Items.Add( rpiSubItem );

  rpiSubItem = new  RadPanelItem();
  rpiSubItem.Text = "MTV Movie Awards";
  rpItem.Items.Add( rpiSubItem );
}
```

VB Example:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim rpItem As RadPanelItem = New RadPanelItem
     rpItem.Text = "Politics"
     rpItem.CssClass = "MainItem"
     rpItem.Expanded = True
      RadPanelbar1.Items.Add(rpItem)
    Dim rpiSubItem As RadPanelItem = New RadPanelItem
     rpiSubItem.Text = "CNN"
      rpItem.Items.Add(rpiSubItem)
     rpiSubItem = New RadPanelItem
     rpiSubItem.Text = "NBC"
      rpItem.Items.Add(rpiSubItem)
     rpiSubItem = New RadPanelItem
     rpiSubItem.Text = "ABC"
      rpItem.Items.Add(rpiSubItem)
     rpItem = New RadPanelItem
     rpItem.Text = "Sports"
     rpItem.CssClass = "MainItem"
     rpItem.Expanded = False
```

```
     RadPanelbar1.Items.Add(rpItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "US Sports"
     rpItem.Items.Add(rpiSubItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "European Sports"
     rpItem.Items.Add(rpiSubItem)
    rpItem = New RadPanelItem
    rpItem.Text = "Events"
    rpItem.CssClass = "MainItem"
    rpItem.Expanded = True
     RadPanelbar1.Items.Add(rpItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "Oscar Awards"
     rpItem.Items.Add(rpiSubItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "MTV Movie Awards"
     rpItem.Items.Add(rpiSubItem)
End Sub
```

5) Run the application.

Lab: Handling Server Events

This lab will teach you how to use the three server side events available to the RadPanelBar control.

The RadPanelbar provides three server side events:

Event	Description
ItemClick	This event is fired when the user clicks on a panelbar item (a panelbar postback).
ItemCreated	This event is fired whenever a new panel item is added to the RadPanelbar.
ItemDataBound	This event is fired whenever the RadPanelbar is being bound to a datasource.

In this lab, you will create a single project in which you will handle all three events.

Handling the ItemClick Event

1. Create a new project and name it ServerSide.
2. Drop a RadPanelbar control on to the default page for this project.
3. Drop a standard Label control on to the default page for this project.
4. Add the CreateTable() method shown below to the code-behind Page class for this page. This method creates a DataTable and populates it.

```
C# Example:
private DataTable CreateTable()
{
   DataTable tbl = new DataTable();
    tbl.Columns.Add( "ID" );
    tbl.Columns.Add( "Text" );
    tbl.Columns.Add( "Value" );
    tbl.Columns.Add( "ParentID" );
```

```
   tbl.Rows.Add( "1", "Products", "You clicked Products", null);
   tbl.Rows.Add( "2", "Support", "You clicked Support", null );
   tbl.Rows.Add( "3", "r.a.d.editor", "You clicked r.a.d.editor", 1 );
   tbl.Rows.Add( "4", "RadPanelBar", "You clicked RadPanelBar", 1 );
   tbl.Rows.Add( "5", "r.a.d.menu", "You clicked r.a.d.menu", 1 );
   tbl.Rows.Add( "6", "r.a.d.tabstrip", "You clicked r.a.d.tabstrip", 1 );
   tbl.Rows.Add( "7", "Knowledge Base", "You clicked Knowledge Base", 2 );
   tbl.Rows.Add( "8", "Forums", "You clicked Forums", 2 );
   tbl.Rows.Add( "9", "Articles", "You clicked Articles", 2 );
   tbl.Rows.Add( "10", "FAQ", "You clicked FAQ", 2 );
  return tbl;
}

VB Example:
Private Function CreateTable() As DataTable
  Dim tbl As DataTable = New DataTable
   tbl.Columns.Add("ID")
   tbl.Columns.Add("Text")
   tbl.Columns.Add("Value")
   tbl.Columns.Add("ParentID")
   tbl.Rows.Add("1", "Products", "You clicked Products", Nothing)
   tbl.Rows.Add("2", "Support", "You clicked Support", Nothing)
   tbl.Rows.Add("3", "r.a.d.editor", "You clicked r.a.d.editor", 1)
   tbl.Rows.Add("4", "RadPanelBar", "You clicked RadPanelBar", 1)
   tbl.Rows.Add("5", "r.a.d.menu", "You clicked r.a.d.menu", 1)
   tbl.Rows.Add("6", "r.a.d.tabstrip", "You clicked r.a.d.tabstrip", 1)
   tbl.Rows.Add("7", "Knowledge Base", "You clicked Knowledge Base", 2)
   tbl.Rows.Add("8", "Forums", "You clicked Forums", 2)
   tbl.Rows.Add("9", "Articles", "You clicked Articles", 2)
   tbl.Rows.Add("10", "FAQ", "You clicked FAQ", 2)
  Return tbl
End Function
```

5. Add the code to the Page_Load() event. This method assigns the data source and sets up the data binding related properties.

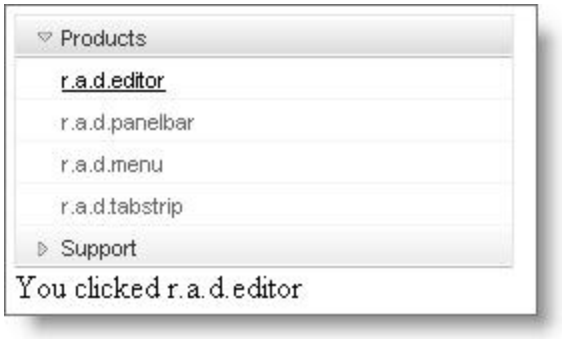
```
C# Example:
protected void Page_Load( object sender, EventArgs e )
{
   RadPanelbar1.DataSource = CreateTable();
   RadPanelbar1.DataFieldID = "ID";
   RadPanelbar1.DataTextField = "Text";
   RadPanelbar1.DataValueField = "Value";
   RadPanelbar1.DataFieldParentID = "ParentID";
   RadPanelbar1.DataBind();
  // this can also be set in the Properties Window
   RadPanelbar1.PersistStateInCookie = true;
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
   RadPanelbar1.DataSource = CreateTable
   RadPanelbar1.DataFieldID = "ID"
   RadPanelbar1.DataTextField = "Text"
   RadPanelbar1.DataValueField = "Value"
   RadPanelbar1.DataFieldParentID = "ParentID"
   RadPanelbar1.DataBind
   RadPanelbar1.PersistStateInCookie = True
End Sub
```

6. Add an ItemClick event to the RadPanelbar. You can do this through the Properties Window in Visual Studio.

7. In the ItemClick event handler assign the label text from the value of the clicked on item:

```
C# Example:
protected void RadPanelbar1_ItemClick( object sender,
Telerik.WebControls.RadPanelbarEventArgs e )
{
   Label1.Text = e.Item.Value;
}

VB Example:
Protected Sub RadPanelbar1_ItemClick(ByVal sender As Object, _
  ByVal e As Telerik.WebControls.RadPanelbarEventArgs)
   Label1.Text = e.Item.Value
End Sub
```

8. Now when you run the application and click on an item, the Label will display the value you provided as the Value column within the DataTable as shown in the figure below.



Selected item reflected in the Label

Note how this was done by accessing the Value field of the RadPanelbarEventArgs parameter to the event. This class contains an Item property which refers to a single item in the RadPanelbar control.

Handling the ItemCreated Event

1. In the same project, add another Label control to the page and clear its Text property. Place this label somewhere underneath the previous Label.

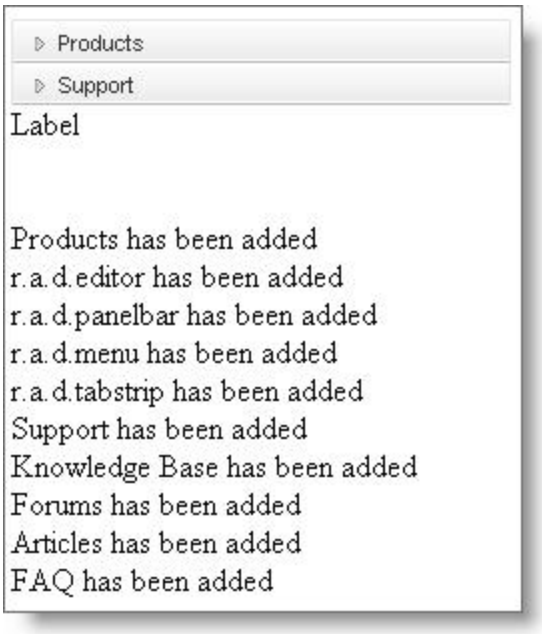
2. Add the ItemCreated event to the RadPanelbar. You can do this through the Properties Window in Visual Studio.

3. In the ItemCreated event handler add the following code to assign the text for a second label.

```
C# Example:
protected void RadPanelbar1_ItemCreated( object sender,
   Telerik.WebControls.RadPanelbarEventArgs e )
{
   Label2.Text += e.Item.Text + " has been added" + "<br/>";
}

VB Example:
```

```
protected Sub  RadPanelbar1_ItemCreated(ByVal sender As Object, _
  ByVal e As   Telerik.WebControls.RadPanelbarEventArgs)
   Label2.Text += e.Item.Text + " has been added" + "<br/>"
End Sub
```

4. When you run the application you should see something like that shown in Figure 2.



Added Items Displayed

Handling the ItemDataBound Event

The ItemDataBound event is fired for each item in a data source being bound to the RadPanelbar. This event gives you the opportunity to handle any processing that requires access to the row of data being displayed. The item being bound is accessible through the DataItem property of the RadPanelbarEventArgs argument. You must cast DataItem to the appropriate type, in this case a DataRowView since this example illustrates binding to a DataTable.

1. Add the ItemDataBound event to the RadPanelbar through the Properties Window in Visual Studio.
2. Add the code shown in Listing 4 to the RadPanelbar1_ItemDataBound method that was created for you.

```
C# Example:
protected void  RadPanelbar1_ItemDataBound( object sender,
   Telerik.WebControls.RadPanelbarEventArgs e )
{
   DataRowView row = (DataRowView)e.Item.DataItem;
   e.Item.Text = row["Text"].ToString() + " from IDB";
}

VB Example:
Protected Sub  RadPanelbar1_ItemDataBound(ByVal sender As Object, _
```

```
 ByVal e As    Telerik.WebControls.RadPanelbarEventArgs)
Dim row As  DataRowView  =  CType(e.Item.DataItem,  DataRowView)
 e.Item.Text  =  row("Text").ToString  +  "  from  IDB"
End Sub
```

3. When you run the application, you will see that the text for the menu items is change that specified in the event.

Items modified in the ItemDataBound event

2.5.5 Client Scripting with RadPanelBar

With the r.a.d panelbar client API you can perform various client side functions and avoid round trips to the server. This section will instruct you on some of the client side features of the RadPanelBar control.

Lab: Responding Client Events

This lab will teach you how to setup and to respond to events on the client. The r.a.d panelbar contains nine events to which you may respond:

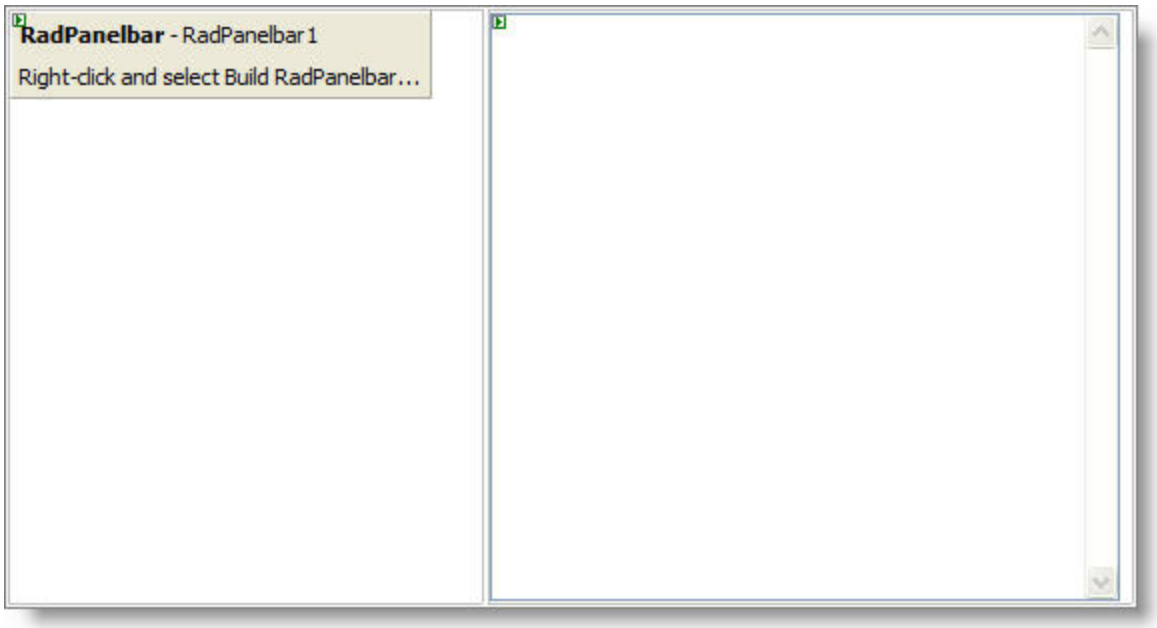
Event	Description
OnClientLoad	This event is fired after the panelbar is fully loaded on the client.
OnClientMouseOver	This event is fired when the mouse moves over the panelbar item.
OnClientMouseOut	This event is fired when the mouse moves out of the panelbar item.
OnClientItemExpand	This event is fired when a group of child items is expanded.
OnClientItemFocus	This event is fired when a panelbar item is selected using either the keyboard (the [TAB] or arrow keys) or by clicking it
OnClientItemBlur	This event is fired when a panelbar item loses focus as a result of the user pressing a key or clicking elsewhere on the page
OnClientItemClicking	This event is fired just before a panelbar item is clicked upon.

Event	Description
OnClientItemClicked	This event is fired just after a panelbar item is clicked upon.
OnClientItemCollapse	This event is fired just before a panelbar item is closed.

The example in this lab will illustrate responding to all of the above listed events.

Setting up the WebForm

1. Create a new project and name it ClientSide.
2. Add a single-row, two-column html table to the WebForm. This is for the purpose of layout.
3. Set the align and valign properties of each table column to Left and Top respectively.
4. Add a RadPanelbar control to the first column in the html table.
5. Add a TextBox control to the second column in the html table.
6. Set the TextBox's TextMode property to MultiLine.
7. Adjust the TextBox's size to so that it looks similar to that shown in the figure below.



WebForm for this example

Populating the RadPanelbar and Initializing client events

1. Add the method shown in Listing 1 to the page class in the code-behind. This method initializes the client events with client-side functions which you have not yet written.

```
C# Example:
private void  InitializeClientEvents()
{
```

```
    RadPanelbar1.OnClientMouseOut  = "OnMouseOut";
    RadPanelbar1.OnClientMouseOver  = "OnMouseOver";
    RadPanelbar1.OnClientItemClicking  = "OnClicking";
    RadPanelbar1.OnClientItemClicked  = "OnClicked";
    RadPanelbar1.OnClientItemCollapse  = "OnCollapse";
    RadPanelbar1.OnClientItemFocus  = "OnFocus";
    RadPanelbar1.OnClientItemBlur  = "OnBlur";
    RadPanelbar1.OnClientItemExpand  = "OnExpand";
}

VB  Example:
Private Sub  InitializeClientEvents()
    RadPanelbar1.OnClientMouseOut  =  "OnMouseOut"
    RadPanelbar1.OnClientMouseOver  =  "OnMouseOver"
    RadPanelbar1.OnClientItemClicking  =  "OnClicking"
    RadPanelbar1.OnClientItemClicked  =  "OnClicked"
    RadPanelbar1.OnClientItemCollapse  =  "OnCollapse"
    RadPanelbar1.OnClientItemFocus  =  "OnFocus"
    RadPanelbar1.OnClientItemBlur  =  "OnBlur"
    RadPanelbar1.OnClientItemExpand  =  "OnExpand"
End Sub Sub
```

2. In the code-behind add the following using statement which will make the RadPanelbar and RadPanelItem classes accessible in your code :

```
C#  Example:
using  Telerik.WebControls;

VB  Example:
Imports  Telerik.WebControls;
```

3. Also in the code behind, add the following code to the Page_Load event handler. Note the highlighted comments.

```
C#  Example:
protected void  Page_Load(object  sender,  EventArgs  e)
{
  // Create  first  root  item
   RadPanelItem  rpItem  =  new  RadPanelItem();
   rpItem.Text  =  "Politics";
   rpItem.CssClass  =  "MainItem";
   rpItem.Expanded  =  true;

  // add  root  items  directly  to  the  RadPanelbar
   RadPanelbar1.Items.Add(  rpItem  );

  // Add  sub-items
   RadPanelItem  rpiSubItem  =  new  RadPanelItem();
   rpiSubItem.Text  =  "CNN";
  // add  sub-items  to  a  root  item.
   rpItem.Items.Add(  rpiSubItem  );

   rpiSubItem  =  new  RadPanelItem();
   rpiSubItem.Text  =  "NBC";
   rpItem.Items.Add(  rpiSubItem  );

   rpiSubItem  =  new  RadPanelItem();
   rpiSubItem.Text  =  "ABC";
   rpItem.Items.Add(  rpiSubItem  );
```

```
  // Create second root item
   rpItem = new  RadPanelItem();
   rpItem.Text = "Sports";
   rpItem.CssClass = "MainItem";
   rpItem.Expanded = false;

   RadPanelbar1.Items.Add( rpItem );

  // Add sub-items
   rpiSubItem = new  RadPanelItem();
   rpiSubItem.Text = "US Sports";
   rpItem.Items.Add( rpiSubItem );

   rpiSubItem = new  RadPanelItem();
   rpiSubItem.Text = "European Sports";
   rpItem.Items.Add( rpiSubItem );

  // Create third root item
   rpItem = new  RadPanelItem();
   rpItem.Text = "Events";
   rpItem.CssClass = "MainItem";
   rpItem.Expanded = true;

   RadPanelbar1.Items.Add( rpItem );

  // Add sub-items
   rpiSubItem = new  RadPanelItem();
   rpiSubItem.Text = "Oscar Awards";
   rpItem.Items.Add( rpiSubItem );

   rpiSubItem = new  RadPanelItem();
   rpiSubItem.Text = "MTV Movie Awards";
   rpItem.Items.Add( rpiSubItem );

    InitializeClientEvents();
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  Dim rpItem As RadPanelItem = New RadPanelItem
   rpItem.Text = "Politics"
   rpItem.CssClass = "MainItem"
   rpItem.Expanded = True
    RadPanelbar1.Items.Add(rpItem)
  Dim rpiSubItem As RadPanelItem = New RadPanelItem
   rpiSubItem.Text = "CNN"
    rpItem.Items.Add(rpiSubItem)
   rpiSubItem = New RadPanelItem
   rpiSubItem.Text = "NBC"
    rpItem.Items.Add(rpiSubItem)
   rpiSubItem = New RadPanelItem
   rpiSubItem.Text = "ABC"
    rpItem.Items.Add(rpiSubItem)
  rpItem = New RadPanelItem
   rpItem.Text = "Sports"
   rpItem.CssClass = "MainItem"
   rpItem.Expanded = False
    RadPanelbar1.Items.Add(rpItem)
   rpiSubItem = New RadPanelItem
```

```
   rpiSubItem.Text = "US Sports"
    rpItem.Items.Add(rpiSubItem)
   rpiSubItem = New RadPanelItem
   rpiSubItem.Text = "European Sports"
    rpItem.Items.Add(rpiSubItem)
  rpItem = New RadPanelItem
   rpItem.Text = "Events"
   rpItem.CssClass = "MainItem"
   rpItem.Expanded = True
    RadPanelbar1.Items.Add(rpItem)
   rpiSubItem = New RadPanelItem
   rpiSubItem.Text = "Oscar Awards"
    rpItem.Items.Add(rpiSubItem)
   rpiSubItem = New RadPanelItem
   rpiSubItem.Text = "MTV Movie Awards"
    rpItem.Items.Add(rpiSubItem)
    InitializeClientEvents
End Sub
```

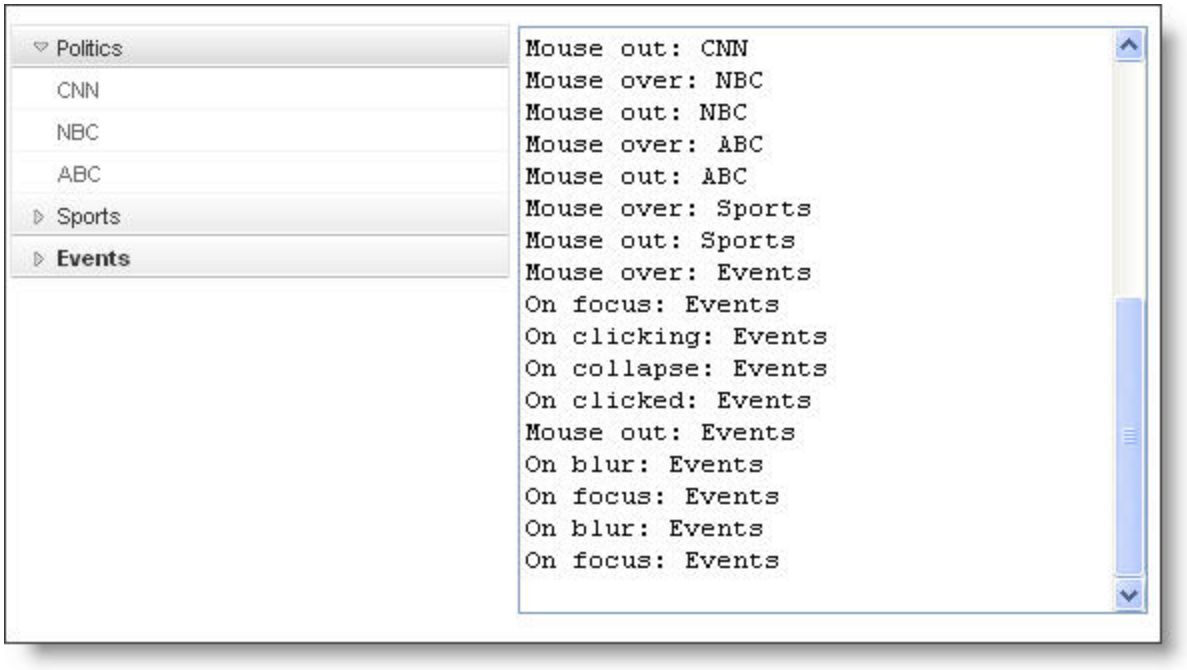
Providing the Client-side Functions

1. Add the <script> block shown in Listing 3 to the <head> section of the WebForm as shown. This listing contains a function for each client-side event. When the event is fired, it invokes a LogEvent() function which writes information about that event to the TextBox.

```
<head  runat="server">
     <title>Untitled Page</title>

     <script type="text/javascript">
          function OnMouseOver(sender, eventArgs)
       {
            LogEvent("Mouse over: " + eventArgs.Item.Text);
       }

          function OnMouseOut(sender, eventArgs)
       {
            LogEvent("Mouse out: " + eventArgs.Item.Text);
       }

          function OnClicking(sender, eventArgs)
       {
            LogEvent("On clicking: " + eventArgs.Item.Text);

       }

          function OnClicked(sender, eventArgs)
       {
            LogEvent("On clicked: " + eventArgs.Item.Text);
       }

          function OnExpand(sender, eventArgs)
       {
            LogEvent("On expand: " + eventArgs.Item.Text);
       }

          function OnCollapse(sender, eventArgs)
       {
            LogEvent("On collapse: " + eventArgs.Item.Text);
       }
```

```
        function OnFocus(sender, eventArgs)
        {
            LogEvent("On focus: " + eventArgs.Item.Text);
        }

        function OnBlur(sender, eventArgs)
        {
            LogEvent("On blur: " + eventArgs.Item.Text);
        }

        function ClearLog()
        {
            var log = document.getElementById("LogEvent");

            log.value = "";
        }

        function LogEvent(text)
        {
            var tb = document.getElementById("Textbox1");

            if (tb)
            {
                tb.value += text + "\n";
                tb.scrollTop = tb.scrollHeight;
            }
        }

    </script>

</head>
```

2. Run the application to examine the client-side functionality. As you perform various tasks on the panelbar, you will see the log entries in the TextBox as shown in the figure below.



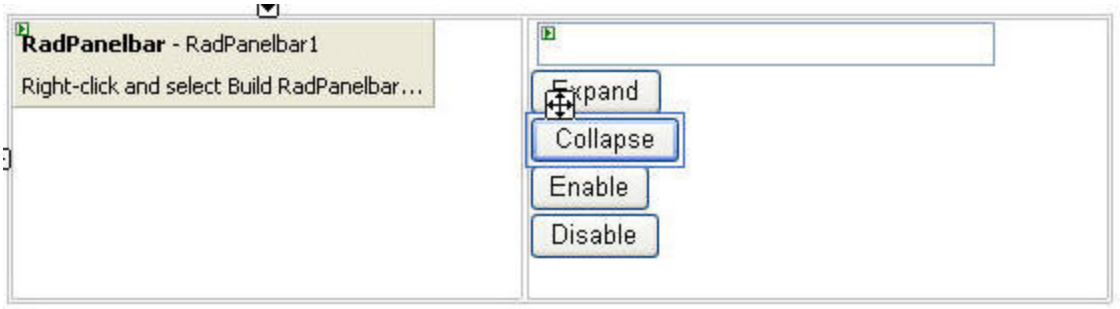
Executing the Example

Lab: Using the Client API

This lab will teach you how to interact with the client API for the r.a.d panelbar. In the first part of this example, you will learn how to find a specific panelbar item on the form. Then, you will write code to interact with that item on the client.

Setting up the WebForm

1. Create a new project and name it "ClientAPI".
2. Add a single-row, two-column html table to the WebForm. This is for the purpose of layout.
3. Set the align and valign properties of each table column to Left and Top respectively.
4. Add a RadPanelbar control to the first column in the html table.
5. Add a TextBox control to the second column in the html table.
6. Set the TextBox's TextMode property to MultiLine.
7. Add four HTML Buttons to the second column and change their value properties to: "Expand" "Collapse", "Enable" and "Disable"
8. Your form should appear as that shown in the figure below.



WebForm for this example

Populating the RadPanelbar

1. In the code-behind add the following using statement which will make the RadPanelbar and RadPanelItem classes accessible in your code :

```
C# Example:
using Telerik.WebControls;

VB Example:
Imports Telerik.WebControls
```

2. Also in the code behind, add the following code to the Page_Load event handler. The code here simply populates the RadPanelbar with data. Note the highlighted comments.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  // Create first root item
   RadPanelItem rpItem = new RadPanelItem();
   rpItem.Text = "Politics";
   rpItem.CssClass = "MainItem";
   rpItem.Expanded = true;

  // add root items directly to the RadPanelbar
   RadPanelbar1.Items.Add( rpItem );

  // Add sub-items
   RadPanelItem rpiSubItem = new RadPanelItem();
   rpiSubItem.Text = "CNN";
  // add sub-items to a root item.
   rpItem.Items.Add( rpiSubItem );

   rpiSubItem = new RadPanelItem();
   rpiSubItem.Text = "NBC";
   rpItem.Items.Add( rpiSubItem );

   rpiSubItem = new RadPanelItem();
   rpiSubItem.Text = "ABC";
   rpItem.Items.Add( rpiSubItem );


  // Create second root item
  rpItem = new RadPanelItem();
   rpItem.Text = "Sports";
   rpItem.CssClass = "MainItem";
   rpItem.Expanded = false;
```

```
    RadPanelbar1.Items.Add( rpItem );

   // Add sub-items
    rpiSubItem = new RadPanelItem();
    rpiSubItem.Text = "US Sports";
    rpItem.Items.Add( rpiSubItem );

    rpiSubItem = new RadPanelItem();
    rpiSubItem.Text = "European Sports";
    rpItem.Items.Add( rpiSubItem );

   // Create third root item
    rpItem = new RadPanelItem();
    rpItem.Text = "Events";
    rpItem.CssClass = "MainItem";
    rpItem.Expanded = true;

    RadPanelbar1.Items.Add( rpItem );

   // Add sub-items
    rpiSubItem = new RadPanelItem();
    rpiSubItem.Text = "Oscar Awards";
    rpItem.Items.Add( rpiSubItem );

    rpiSubItem = new RadPanelItem();
    rpiSubItem.Text = "MTV Movie Awards";
    rpItem.Items.Add( rpiSubItem );
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
   Dim rpItem As RadPanelItem = New RadPanelItem
    rpItem.Text = "Politics"
    rpItem.CssClass = "MainItem"
    rpItem.Expanded = True
     RadPanelbar1.Items.Add(rpItem)
   Dim rpiSubItem As RadPanelItem = New RadPanelItem
    rpiSubItem.Text = "CNN"
     rpItem.Items.Add(rpiSubItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "NBC"
     rpItem.Items.Add(rpiSubItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "ABC"
     rpItem.Items.Add(rpiSubItem)
   rpItem = New RadPanelItem
    rpItem.Text = "Sports"
    rpItem.CssClass = "MainItem"
    rpItem.Expanded = False
     RadPanelbar1.Items.Add(rpItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "US Sports"
     rpItem.Items.Add(rpiSubItem)
    rpiSubItem = New RadPanelItem
    rpiSubItem.Text = "European Sports"
     rpItem.Items.Add(rpiSubItem)
   rpItem = New RadPanelItem
    rpItem.Text = "Events"
    rpItem.CssClass = "MainItem"
    rpItem.Expanded = True
     RadPanelbar1.Items.Add(rpItem)
```

```
   rpiSubItem = New  RadPanelItem
   rpiSubItem.Text  =  "Oscar  Awards"
    rpItem.Items.Add(rpiSubItem)
   rpiSubItem = New  RadPanelItem
   rpiSubItem.Text  =  "MTV  Movie  Awards"
    rpItem.Items.Add(rpiSubItem)
End Sub
```

Adding the Client Script Functionality

1. Add the javascript functions in the listing below to the WebForm. You can place these within the <Form> block. The listing contains three client-side functions:

   - ExpandItem() expands or collapses the item specified in the Textbox by using the client-side Expand() and Collapse() functions. Note how a reference to the RadPanelbar instance is retrieved with the line:

     ```
     var  panelbar  =  <%=  RadPanelbar1.ClientID  %>;
     ```

   - EnableItem() enables or disables the item specified in the Textbox. This function illustrates how to use the Enable() and Disable() client-side functions.

   - Finally, EnableAll() illustrates how one can enumerate through the list of items.

```
<script   type="text/javascript">
   function  ExpandItem(  ExpandIt  )
  {
        var  panelbar  =  <%=  RadPanelbar1.ClientID  %>;
         var  text  =  document.getElementById("TextBox1").value;

         var  item  =  panelbar.FindItemByText(text);

       if (item)
       {
           if ( ExpandIt )
               item.Expand();
           else
               item.Collapse();
       }
       else
       {
            alert("Item with text '" + text + "' not found.");
       }
  }

   function  EnableItem(  EnableIt  )
  {
        var  panelbar  =  <%=  RadPanelbar1.ClientID  %>;
         var  text  =  document.getElementById("TextBox1").value;

         var  item  =  panelbar.FindItemByText(text);

       if (item)
       {
           if ( EnableIt )
             item.Enable();
           else
             item.Disable();
       }
       else
       {
```

```
            alert("Item with text '" + text + "' not found.");
        }
    }

    function EnableAll( EnableThem )
    {
        var panelbar = <%= RadPanelbar1.ClientID %>;

        for (var i = 0; i < panelbar.AllItems.length; i++)
        {
            if ( EnableThem )
                panelbar.AllItems[i].Enable();
            else
                panelbar.AllItems[i].Disable();
        }
    }
</script>
```

2. Now modify the Button tag to contain an onclick handler that will reference one of the client-side scripts. Listing 3 depicts what this should look like with emphasis on the necessary setting.

```
<button class="button" onclick="ExpandItem( true );return false;">Expand</button><br />
<button class="button" onclick="ExpandItem( false );return false;">Collapse</button><br />
<button class="button" onclick="EnableItem( true );return false;">Enable</button><br />
<button class="button" onclick="EnableItem( false );return false;">Disable</button><br />
```

Notice that in each client side function, you obtain a reference to the RadPanelbar instance using the following line of code:

```
var panelbar = <%= RadPanelbar1.ClientID %>;
```

One you have that reference, you can start executing client-side function such as those illustrated in Listing 2 (Expand, Collapse, Enable, Disable).

3. Run the application to test the client side functionality.

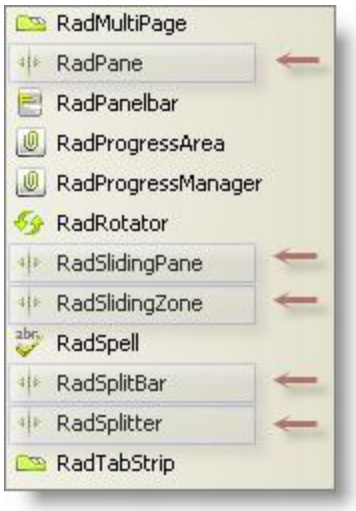
2.5.6 Summary

This section on RadPanelbar covered the basics of configuring at design time, binding to flat and hierarchical data, demonstrated building and responding to RadPanelbar at runtime, and explored the capabilities of the client side API and event model.

2.6 RadSplitter

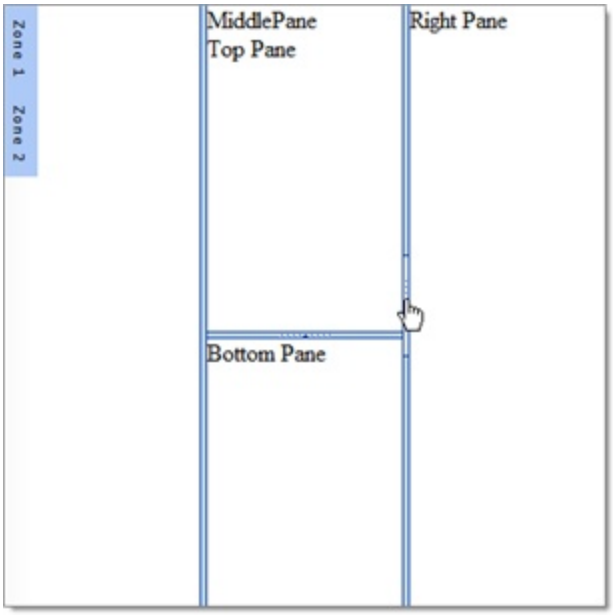
2.6.1 Getting Started

RadSplitter supplies you with powerful screen real estate management tools. You can slice up the screen in configurations previously only possible in Window's applications where you can resize any portion of the screen or collapse and expand certain areas that don't need to be continuously visible.



RadSplitter controls in the toolbox

For example, if you want an "explorer" style interface you could have the left most pane defining the top level actions, perhaps containing a tree view or list, then selections from that showing more detailed information in the right/top pane and finally the most detailed information appearing on the right bottom. In addition you can have expandable zones that work like Visual Studio windows that can be expanded and pinned into place. More complex arrangements can be handled by splitting areas up into smaller sections. The figure below shows three main panes (left, middle and right) where the middle pane is further subdivided to top and bottom panes. The left most pane also has two expanding zones.



RadSplitter in action allows resizing all portions of the screen and collapsible panels

The RadSplitter control is the overall manager for resizable and sliding panes. It defines properties that apply to all panes such as orientation and Skin. Within the RadSplitter you place one or more RadPane controls. RadPane acts as a container for other controls placed inside the scope of its tag and also controls scroll behavior. Instead of populating the RadPane directly with controls you can also assign its *ContentUrl* property to automatically load external web sites or web forms from your project.

RadSplitBars are placed between RadPanes to allow resizing. RadSplitBar also controls the presence and behavior of collapse/expand buttons. The basic pattern in markup looks something like this simplified example:

```
<RadSplitter   Orientation="Vertical">
   <RadPane>Left  pane</RadPane>
   <RadSplitBar />
   <RadPane>Right  pane</RadPane>
</RadSplitter>
```

You can take this model a step farther by adding sliding panels. RadSlidingZones are placed within panes with one or more RadSlidingPanes within the sliding zone. The simplified model now looks like:

```
<RadSplitter   Orientation="Vertical">
   <RadPane>
      <RadSlidingZone>
         <RadSlidingPane>Content  goes  here...</RadSlidingPane>
      </RadSlidingZone>
   </RadPane>
   <RadSplitBar />
   <RadPane>Right  pane</RadPane>
</RadSplitter>
```

Sliding zones define which direction the pane slides out, the amount of time the sliding animation takes to run, and the granularity in pixels that the slider will move in. Sliding zones also define the initial docked and expanded sliding panes.

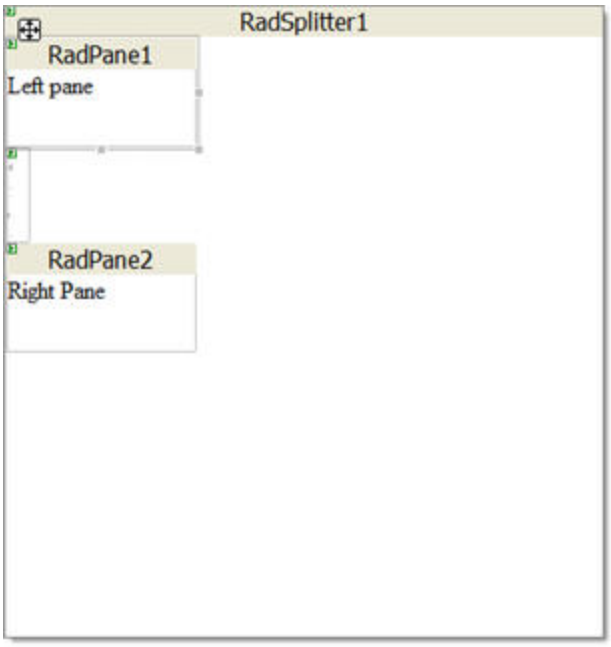
Each sliding pane has a *Title* property that shows in both the collapsed menu like area and in the heading when expanded.

Lab: RadSplitter Basics

This lab will demonstrate building basic screen layouts using the splitter.

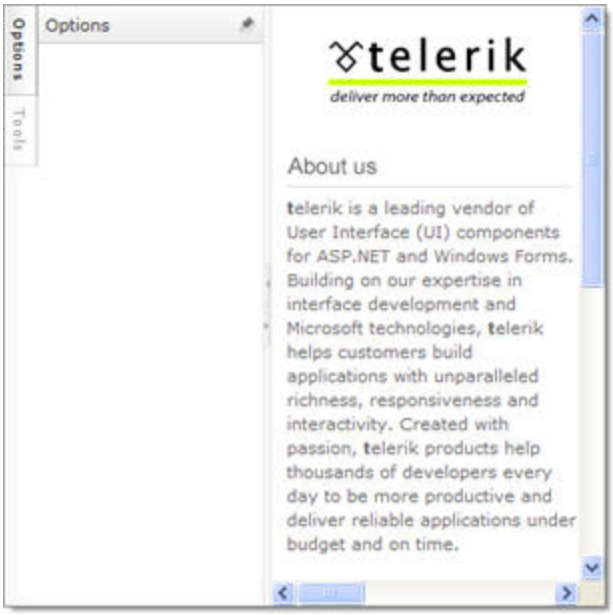
The first step in getting control of your screen real estate is to design the general layout. The control set in design mode is not WYSIWYG and so layout of any complexity tends to be more of a logical design. This is much easier to do up front than when the form is heavily populated with other controls.

1. Create a new web application "Getting Started".
2. Drop a RadSplitter control on the default page.
3. Drop a RadPane control on the RadSplitter. Enter the literal text "Left Pane" to the RadPane control.
4. Drop a RadSplitBar just after the RadPane. Set the *CollapseMode* property to "Both".
5. Drop a second RadPane control on the RadSplitter. Enter the literal text "Right Pane" to the RadPane control. The layout should now look like the example in the figure below:



Initial splitter layout with two panels only

6. Run the application and test the splitter bar and the collapse/expand buttons.
7. Stop the application.
8. In the left pane:
   - Remove the literal text "Left pane".
   - Drop a RadSlidingZone. Set the *InitiallyDockedPaneId* to "RadSlidingPane1".
   - Within the sliding zone drop a RadSlidingPane and set the *Title* property to "Options".
   - Drop a second RadSlidingPane just after the first and set the *Title* property to "Tools".
9. In the right pane set the *ContentUrl* property to "http://www.telerik.com".
10. Run the application. Notice that RadSlidingPane1 is displayed initially expanded and pinned into place.



The running application.

2.6.2 Using RadSplitter in the Designer

Key Properties

The following reviews a few key properties for:

- RadSplitter
- RadPane
- RadSplitBar
- RadSlidingZone
- RadSlidingPane

RadSplitter

The RadSplitter has a few properties of interest that affect overall behavior and appearance:

VisibleDuringInit: The helpful property when set true reduces the amount of redrawing the user sees when starting up the application. Particularly good for complex arrangements of panes that recalculate width and height.

FullScreenMode: If true, this property automatically sets the width and height to 100%.

HeightOffset: The number of pixels subtracted from the splitter height when calculating the splitter height as a percentage.

LiveResize: If true the splitter panes are previewed during resizing.

Orientation: A very important property that controls how panes are arranged. The default Vertical orientation places panes side by side from left to right. Horizontal orientation places each pane below the next.

SplitBarSize: The size in pixels for the split bars. Use this property to trade off between ease of finding and dragging on the one hand and space constraints on the other.

Resizing: *ResizeMode* determines what happens to the other panes when you resize a given pane. "AdjacentPane" resizes the panes next to the resized pane, "Proportional" resizes the other panes proportionally and "EndPane" causes the end pane to be resized. *ResizeWithBrowserWindow* and *ResizeWithParentPane* determine if the splitter is automatically resized when the browser and parent pane are resized.

RadPane

You can control the behavior of RadPane using properties:

Locked: Prevent resizing this panel by setting Locked to true.

InitiallyCollapsed: Leave a pane in its collapsed state when the app is first started.

Scrolling: Set the *Scrolling* property to Both, X, Y or None. If the content exceeds the pane the Scrolling property takes effect. This can be a handy property when you're trying to suppress scrollbars altogether.

PersistScrollPosition: Set this property true to persist scrollbar position across postbacks.

ContentUrl: Instead of loading a pane with controls you can point it at a page within your application or an external url.

RadSplitBar

CollapseMode: Can be Forward, Backward, Both or None and controls whether buttons appear on the split bar to allow collapse the pane against either side of the screen.

ResizeStep: The number of pixels the resize bar will jump to when dragged.

RadSlidingZone

ClickToOpen: If true the user must click the pane to open it. By default this property is false and the user merely has to pass the mouse over the pane to open.

InitiallyDockedPaneId, InitiallyExpandedPaneId: If you want a sliding pane expanded an pinned at startup use the InitiallyDockedPaneId. InitiallyExpandedPaneId is similar but at startup the pane slides out but does not remain unless the user pins it.

ResizeStep: The number of pixels the resize bar will jump to when dragged.

SlideDirection: The panel can slide in from the Left, Right, Top or Bottom.

SlideDuration: Controls the number of milliseconds it takes for the slide animation.

RadSlidingPane

- Set whether the sliding pane can be docked or resized is governed by *EnableDock* and *EnableResize* properties.

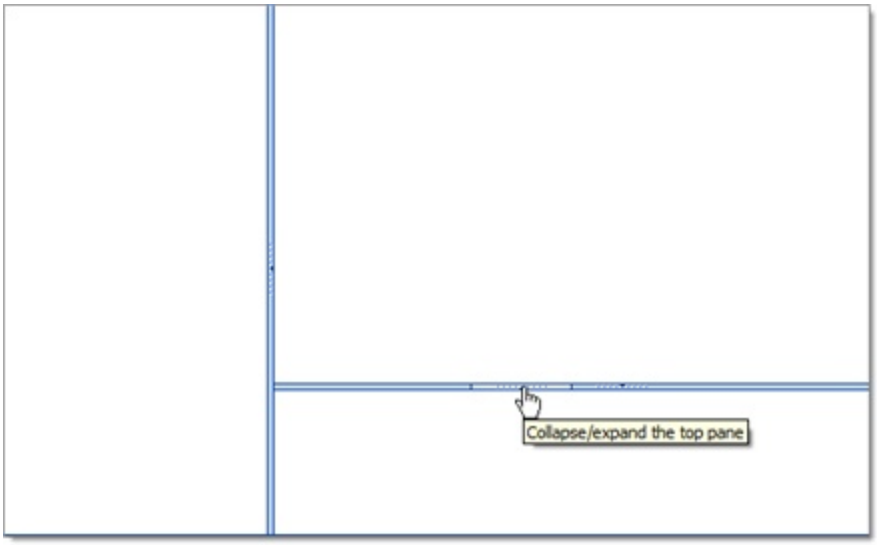
- Display an icon for the pane tab using the *IconUrl* property.
- *TabView* determines if the tab for the pane is rendered as TextOnly, ImageOnly or TextAndImage.
- The *Title* displays in the text for the pane tab.

Lab: Designing The Explorer Interface

This lab demonstrates the techniques used to create an explorer style interface with multiple panels oriented both vertically and horizontally.

1. Create a new web application "ExplorerUserInterface".
2. Create a new RadControls directory in your project. Copy the "Splitter" directory from the Telerik RadControls directory to the RadControls directory in your project.
3. In the designer drop a RadSplitter on the default web page. Set *FullScreenMode* to "True". Set the *Skin* property to "Outlook".
4. Drop a RadPane control onto the splitter. Set the *Scrolling* property to "None". Set the *Width* property to "30%".
5. Drop a RadSplitBar just below the RadPane. Set the *CollapseMode* property to "Forward".
6. Drop a second RadPane after the split bar. Set the *Scrolling* property to "None". Set the *Width* property to "70%".
7. Within the second RadPane drop another RadSplitter. Set the *Orientation* property to "Horizontal". Set the *Skin* property to "Outlook".
8. Within the second splitter drop a RadPane, a RadSplitBar and another RadPane. In both RadPanes set the *Scrolling* property to "None". Set the split bar *CollapseMode* property to "Both".
9. In the markup set the body tag *scroll* to "no". **Note**: This is important to remember when you want to suppress scrollbars for a full screen splitter.

```
<body  scroll="no">
```



The running application with horizontal and veritical splitters.

2.6.3 Using RadSplitter at Runtime

The Splitter *Items* collection contains all the panes and split bars. You can add to this collection on the server side but be aware that the collection is *not* persisted across postbacks and needs to be recreated every time. See the Telerik online demo "Save/Load State on server" that shows how to store and load pane properties from a session variable.

The example below creates the RadSplitter, panels and split bars at runtime and adds them to the page controls collection. Remember that you will still need to register the splitter assembly in the HTML source for the page. After creating the splitter the RadPane is created, pane properties are set and the pane is added to the splitters Items collection. This pattern continues with a RadSplitBar and second panel until the last chunk of code where "paneMiddle" is created and inserted between the other two panels. Notice that the scrolling for the middle panel is "Both" and the width is only 20%.

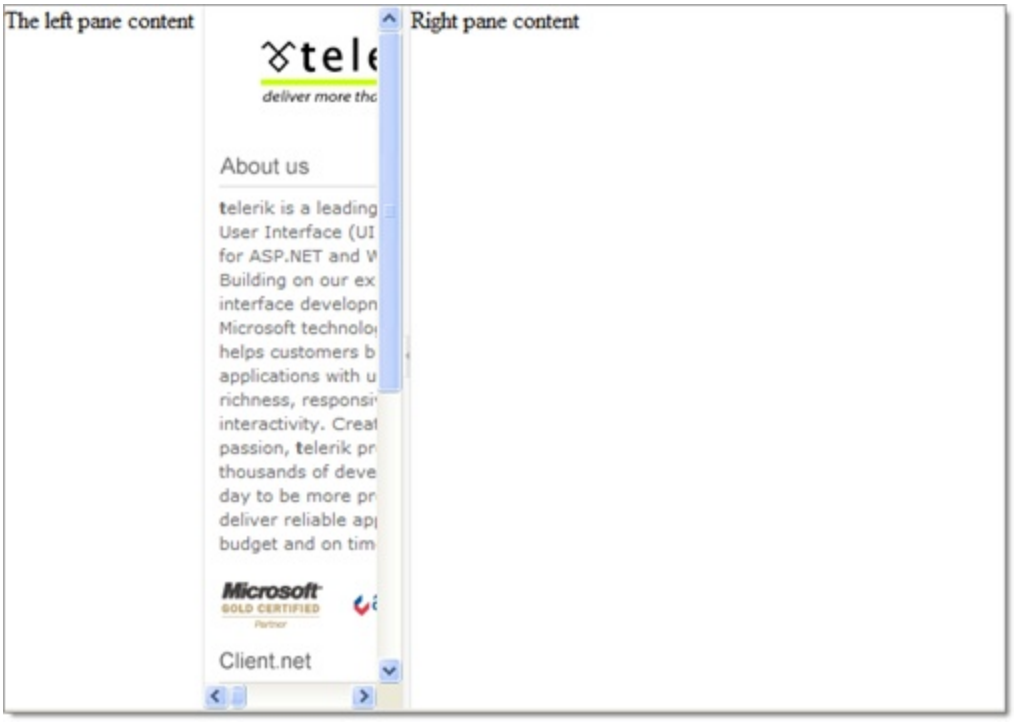
```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   RadSplitter splitter = new RadSplitter();
    splitter.FullScreenMode = true;

   RadPane paneLeft = new RadPane();
    paneLeft.Width = Unit.Percentage(20);
     paneLeft.Scrolling = RadSplitterPaneScrolling.None;
   Label labelLeft = new Label();
   labelLeft.Text = "The left pane content";
     paneLeft.Controls.Add(labelLeft);
     splitter.Items.Add(paneLeft);

   RadSplitBar splitBar = new RadSplitBar();
     splitBar.CollapseMode = RadSplitBarCollapseMode.Forward;
     splitter.Items.Add(splitBar);

   RadPane paneRight = new RadPane();
    paneRight.Width = Unit.Percentage(60);
     paneRight.Scrolling = RadSplitterPaneScrolling.None;
   Label labelRight = new Label();
   labelRight.Text = "Right pane content";
     paneRight.Controls.Add(labelRight);
```

```
    splitter.Items.Add(paneRight);

   RadPane paneMiddle = new RadPane();
    paneMiddle.Width = Unit.Percentage(20);
    paneMiddle.Scrolling = RadSplitterPaneScrolling.Both;
   paneMiddle.ContentUrl = "http://www.telerik.com";
    splitter.Items.Insert(1, paneMiddle);

  this.Controls.Add(splitter);
}
```

VB Example:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)

 Dim splitter As RadSplitter = New RadSplitter
  splitter.FullScreenMode = True

 Dim paneLeft As RadPane = New RadPane
  paneLeft.Width = Unit.Percentage(20)
  paneLeft.Scrolling = RadSplitterPaneScrolling.None
 Dim labelLeft As Label = New Label
 labelLeft.Text = "The left pane content"
   paneLeft.Controls.Add(labelLeft)
   splitter.Items.Add(paneLeft)

 Dim splitBar As RadSplitBar = New RadSplitBar
  splitBar.CollapseMode = RadSplitBarCollapseMode.Forward
   splitter.Items.Add(splitBar)

 Dim paneRight As RadPane = New RadPane
  paneRight.Width = Unit.Percentage(60)
  paneRight.Scrolling = RadSplitterPaneScrolling.None
 Dim labelRight As Label = New Label
 labelRight.Text = "Right pane content"
   paneRight.Controls.Add(labelRight)
   splitter.Items.Add(paneRight)

 Dim paneMiddle As RadPane = New RadPane
  paneMiddle.Width = Unit.Percentage(20)
  paneMiddle.Scrolling = RadSplitterPaneScrolling.Both
  paneMiddle.ContentUrl = "http://www.telerik.com"
  splitter.Items.Insert(1, paneMiddle)

   Me.Controls.Add(splitter)

End Sub
```

The running example looks like the figure below:



Dynamically created splitter panels and split bar.

2.6.4 Client Scripting with RadSplitter

This section touches on a few basics for calling client API functions and handling events for the splitter, panes and zones. See the online help for a full reference of published API and events for RadSplitter.

Client API

The splitter client API provides access to all the contained panes and splitbars. To use the splitter client side methods reference the ClientID. From there you can call methods an properties of the client RadSplitter object. You can retrieve panes and split bars by iterating an array of these objects or by calling one of the get methods that return objects by ID or index.

This example cycles through the panes and split bars, calling methods of each element. The Iteration through the panes collapses all the panes. The iteration through the split bars retrieves the HTML that defines the split bar itself.

```
var splitter = <%= splitterTopBottom.ClientID %>;
var panes = splitter.GetPanes();
for(i = 0; i < panes.length; i++)
{
   panes[i].Collapse(
     RadSplitterNamespace.RAD_SPLITTER_DIRECTION.Forward);
}

var splitBars = splitter.GetSplitBars();
for(i = 0; i < splitBars.length; i++)
{
     alert(splitBars[i].GetContainerElement().innerHTML);
}
```

You can also address one of the objects within the splitter directly and call its methods. This example gets a reference to sliding zone, undocks one of its panes and docks another.

```
var slidingZone = <%= zoneLeft.ClientID %>;
slidingZone.UnDockPane("slidingPaneOptions");
slidingZone.DockPane("slidingPaneFiles");
```

There are several methods for splitter, split bar and panes that return portions of themselves as raw HTML elements. This example sets the content url, then retrieves the HTML for the page.

```
var pane = <%= paneMain.ClientID %>;
pane.SetContentUrl("http://www.telerik.com");
alert(pane.GetExtContentContainerElement().outerHTML);
```

HTML for pane container element

Panes, sliding zones and sliding panes all know how to retrieve and restore their state as JSON strings:

```
var pane = <%= paneBottom.ClientID %>;
alert(pane.GetState());
```

Panel state returned as a JSON string

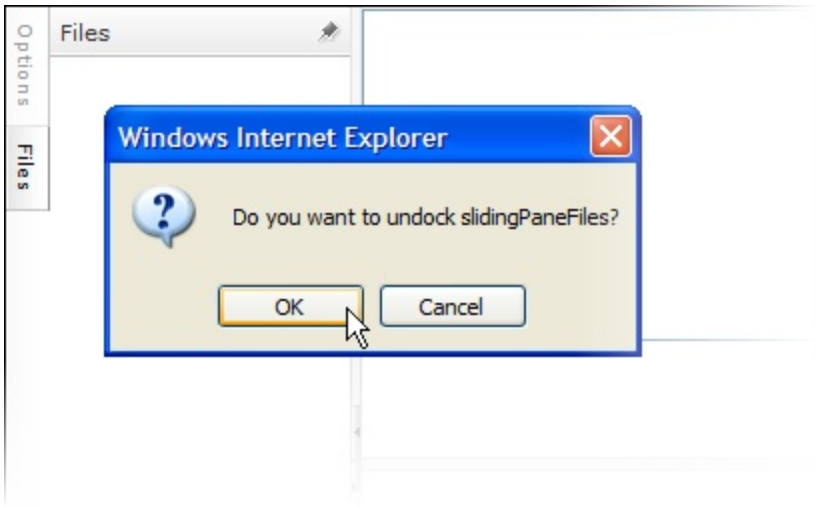
Client Events

The client API surfaces events for splitter, panes and zones to notify your application of changes in object state. Typically these state conditions are:

- The object is loaded and ready for use. OnClientLoaded events are published for the splitter object and sliding zones.
- A change is about to occur or has occurred to the visual representation of the object. For example, panel resize, collapse or docking changes fire client events. The event args will include an identification of the object the event is firing for and if a resize event will typically pass the new dimensions. For events that are OnClientBefore<some action> can be canceled by returning false. Here is an example that prevents undocking based on user feedback in a confirm dialog.

```
function BeforePaneUnDock(sender, eventArgs)
{
```

```
  return confirm("Do you want to undock " +
     eventArgs.paneObj.ID + "?");
}
```



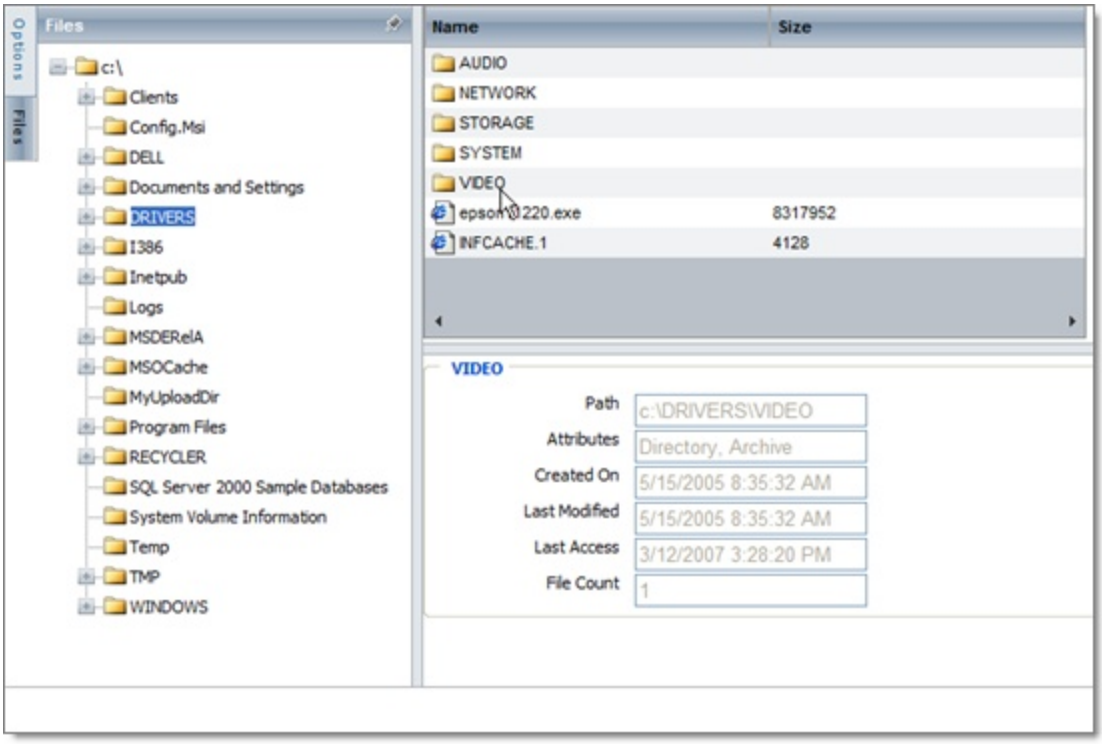
Confirming a panel undock event

2.6.5 Lab: File Explorer

This lab will integrate a number of Telerik controls and techniques. The splitter setup will be in the pattern of the explorer interface and will show a list of directories on the left side, a list of sub directories and files on the top right side and details about a selected file or directory in the bottom right pane.

Due to the number of controls involved the lab is somewhat extensive. The general steps in the lab are:

- Setup the application files and directories, i.e. RadControls directory and Styles directory.
- Create the overall splitter layout.
- Populate the splitter layout with controls.
- Add server code.
- Add client code.
- Hook up events in HTML.



The running file explorer application

Application setup

1. Create a new web application "FileExplorer".
2. Create a new RadControls directory in your project. Copy "Splitter", "Grid" and "TreeView" directories from the Telerik RadControls directory to the RadControls directory in your project.
3. Create a new "Img" directory in the project. Copy the images from \rad splitter\projects\images directory. This directory contains the icon images for several different types of files we may encounter.
4. Create a new "Styles" directory in the project. Copy "Style.css" from the \rad splitter\projects\styles directory. The style sheet content is shown below:

```
form
{
     font-family: Tahoma;
     font-size: 8pt;
}

fieldset
{
     width: 450px;
}

legend
{
     font-weight: bold;
     font-variant: small-caps;
     padding: 2px 6px;
     margin-bottom: 8px;
}
```

```
label
{
     line-height: normal;
     text-align: right;
     margin-right: 10px;
     position: relative;
     display: block;
     float: left;
     width: 125px;
}

.UpdateImage
{
     padding-top:6px;
     padding-left:2px;
}

.OptionInput
{
     width: 100px;
     border-style:none;
}
```

5. In the <head> tag for the default page place a link to the style sheet:

```
<link  rel="stylesheet"  type="text/css"  href="Styles/Style.css"  />
```

6. Also in the html set the body tag *scroll* to "no".

```
<body  scroll="no">
```

In the HTML source for the default page register the treeview, grid, splitter and rad AJAX assemblies. **Note**: You could also perform this step by dragging the appropriate controls on to the default page.

```
<%@  Register  Assembly="RadAjax.Net2"  Namespace="Telerik.WebControls"
   TagPrefix="radA" %>
<%@  Register  Assembly="RadGrid.Net2"  Namespace="Telerik.WebControls"
   TagPrefix="radG" %>
<%@  Register  Assembly="RadSplitter.Net2"  Namespace="Telerik.WebControls"
   TagPrefix="radspl" %>
<%@  Register  Assembly="RadTreeView.Net2"  Namespace="Telerik.WebControls"
   TagPrefix="radT" %>
```

Splitter Layout

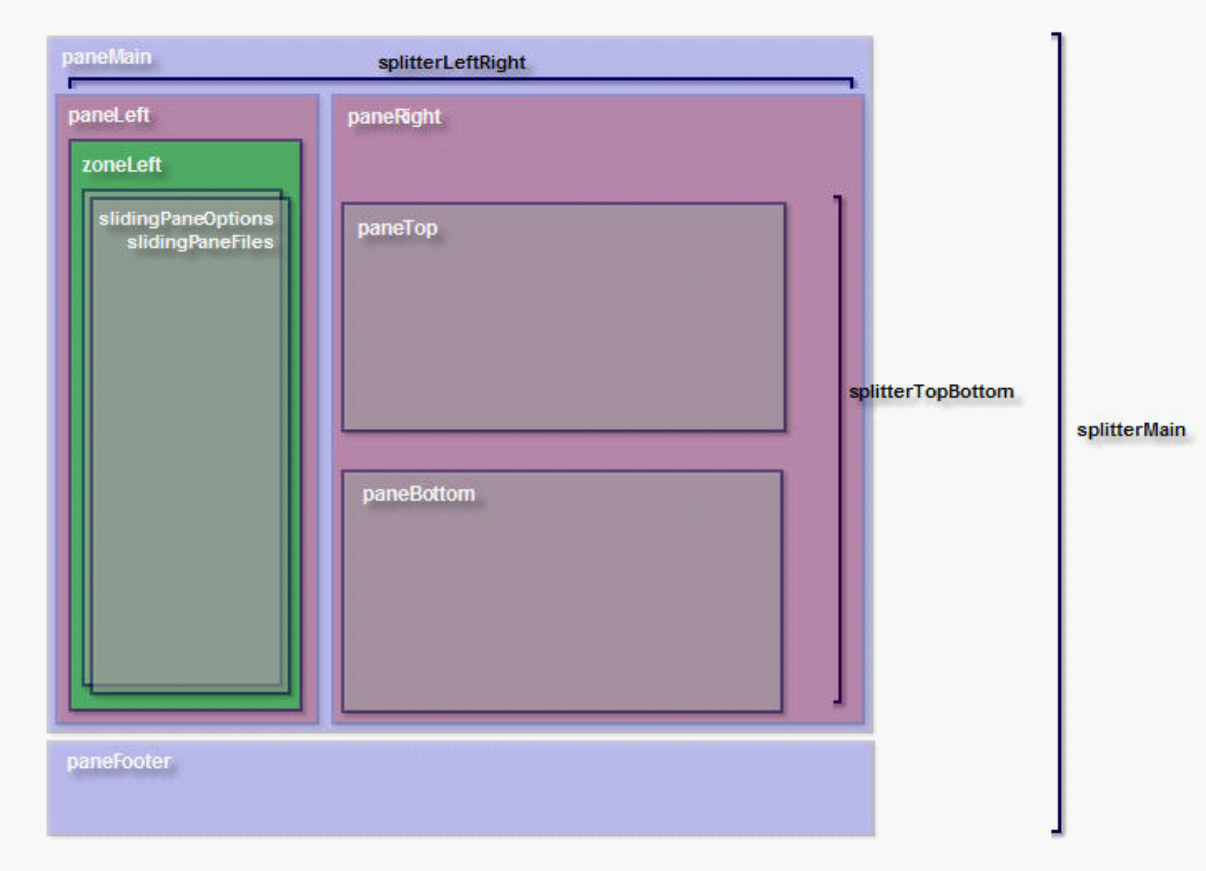
In the HTML source for the default page paste the markup below inside the form tag.

```
<radspl:RadSplitter ID="splitterMain" runat="server" Height="90%" Width="100%"
    EnableClientDebug="False" HeightOffset="8" BorderSize="0" BorderStyle="None"
    Skin="WebBlue" Orientation="Horizontal" VisibleDuringInit="False"
    FullScreenMode="true" LiveResize="True">

   <radspl:RadPane ID="paneMain" runat="server" Height="100%" Width="100%"
     Scrolling="None">
      <radspl:RadSplitter ID="splitterLeftRight" runat="server" Height="100%"
         Width="100%" EnableClientDebug="False" BorderSize="0" BorderStyle="None"
         Skin="WebBlue" Orientation="Vertical" VisibleDuringInit="False">

        <radspl:RadPane ID="paneLeft" runat="server" Height="80%"
         Scrolling="None">

          <radspl:RadSlidingZone ID="zoneLeft" runat="server"
            InitiallyDockedPaneId="slidingPaneFiles" >
             <radspl:RadSlidingPane ID="slidingPaneOptions" runat="server" Scrolling="none"
            Title="Options">
             <%-- Options controls go here --%>
           </radspl:RadSlidingPane>
             <radspl:RadSlidingPane ID="slidingPaneFiles" runat="server" Scrolling="none"
            Title="Files">
             <%-- File list goes here --%>
           </radspl:RadSlidingPane>
         </radspl:RadSlidingZone>

      </radspl:RadPane> <%-- end pane left--%>

       <radspl:RadSplitBar ID="RadSplitBar1" runat="server" />

       <radspl:RadPane ID="paneRight" runat="server" Height="40%" Width="60%"
        Scrolling="None">
          <radspl:RadSplitter ID="splitterTopBottom" runat="server" Height="100%"
              Width="100%" EnableClientDebug="False" BorderSize="0"
              BorderStyle="None" Skin="WebBlue" Orientation="Horizontal"
               VisibleDuringInit="False">

              <radspl:RadPane ID="paneTop" runat="server" Height="50%" Width="100%"
                  Scrolling="None">
               <%-- rad grid control goes here --%>
            </radspl:RadPane>

              <radspl:RadSplitBar ID="RadSplitBar2" runat="server" />

               <radspl:RadPane ID="paneBottom" runat="server" Height="50%" Width="100%"
                  Scrolling="None">
               <%-- field set for file detail goes here --%>
            </radspl:RadPane>

          </radspl:RadSplitter> <%-- end splitter right--%>
     </radspl:RadPane> <%-- end pane right--%>

    </radspl:RadSplitter> <%-- end splitter left--%>
  </radspl:RadPane> <%-- end main pane--%>

   <radspl:RadPane ID="paneFooter" runat="server" Height="30px" Width="100%"
     CssClass="footer" Scrolling="none">
```

```
    <%-- rad ajax loading panel goes here --%>
   </radspl:RadPane>

</radspl:RadSplitter>
```

Review the markup to see how the three different RadSplitter controls are nested. Be sure to notice the orientation property of each. Also check the *InitialDockedPanelId* setting for the sliding zone; this setting will cause the "Files" pane to be displayed at startup. Comments in the markup denote where controls will be placed later after the general layout is complete.

The result of the HTML so far just sets up the splitter bars without any content into the arrangement shown in the figure below.



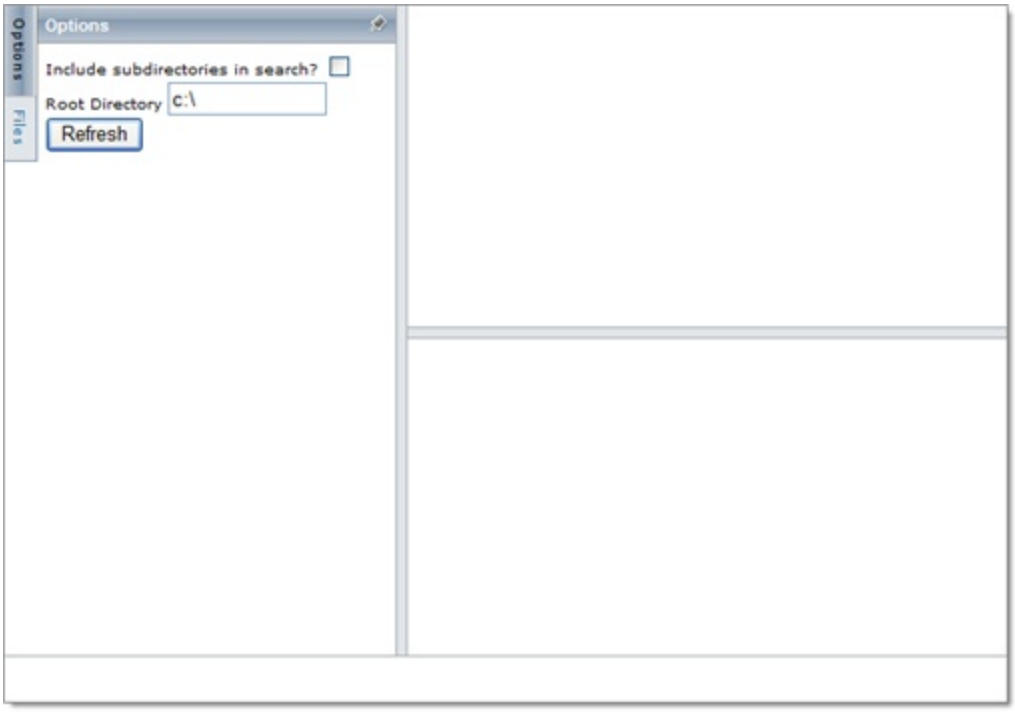
Splitter layout

Add Controls

1. In the html tag for slidingPaneOptions add this markup:

```
Include subdirectories in search?
<asp:CheckBox ID="cbSubDirectories" runat="server" Text="" Checked="false" />
<br />
Root Directory
<asp:TextBox ID="tbPath" CssClass="OptionInput" runat="server" Text="c:\"></asp:TextBox>
<input type="Button" id="btnRefresh" value="Refresh"/>
```

This step adds a checkbox "include subdirectories" option, a textbox for the root directory to display and a refresh button to update changes based on the options selected. The figure below shows how the screen would look running at this point.



The application running

2. The left pane titled "Files" will contain a treeview that displays a directory list. To the html tag for slidingPaneFiles add this markup:

```
<radT:RadTreeView  ID="RadTreeView1"  runat="server"  AutoPostBack="True"
   Skin="WebBlue"  Width="100%"  Height="100%"  AllowNodeEditing="False"
   DragAndDrop="False">
</radT:RadTreeView>
```

3. In the RadPane tag titled "paneTop" add a RadGrid:

```
<radG:RadGrid  ID="RadGrid1"  runat="server"  AutoGenerateColumns="False"  GridLines="None"
   EnableAJAX="true"  Height="98%"  Width="98%"  Skin="WebBlue">
   <MasterTableView  Width="100%">
    <Columns>
        <radG:GridTemplateColumn  HeaderText="Name"  SortExpression="Name"
        UniqueName="Name">
        <ItemTemplate>
           <asp:Image ID="icon" runat="server" />
           <asp:Label ID="itemLabel" runat="server"
              Text='<%# DataBinder.Eval(Container.DataItem, "Name") %>'></asp:Label>
        </ItemTemplate>
         <ItemStyle BorderStyle="None" />
       </radG:GridTemplateColumn>
        <radG:GridTemplateColumn  HeaderText="Size"  SortExpression="Size"
        UniqueName="Size">
         <ItemStyle BorderStyle="None" />
       </radG:GridTemplateColumn>
        <radG:GridTemplateColumn  Display="False"  HeaderText="Path"  UniqueName="Path">
        <ItemTemplate>
```

```
            <asp:Label ID="pathLabel" runat="server"
                Text='<%# DataBinder.Eval(Container.DataItem, "Path") %>'></asp:Label>
          </ItemTemplate>
        </radG:GridTemplateColumn>
      </Columns>
       <ExpandCollapseColumn Visible="False">
        <HeaderStyle Width="19px" />
       </ExpandCollapseColumn>
       <RowIndicatorColumn Visible="False">
        <HeaderStyle Width="20px" />
      </RowIndicatorColumn>
    </MasterTableView>
    <ItemStyle BorderStyle="None" />
     <ClientSettings ApplyStylesOnClient="True" ReorderColumnsOnClient="True">
      <Selecting AllowRowSelect="True" />
      <Resizing AllowColumnResize="True" />
       <Scrolling AllowScroll="True" UseStaticHeaders="True" ScrollHeight="99.9%" />
    </ClientSettings>
     <HeaderStyle BorderColor="DarkGray" BorderStyle="Solid" BorderWidth="1px" />
</radG:RadGrid>
```

Pay particular attention to the Columns tag. The grid is pre-configured with three templated columns for Name, Size and Path. DataBinding expressions are included for the text properties of Name and Path. The Name template includes an image control to display the file icon. We will add event handlers to respond to the grid later in this lab.

4. In the tag "paneBottom" add a field set to contain file details. The fieldset won't have any content yet. Content will be added in code later in this lab.

```
<fieldset id="details" runat="server"></fieldset>
```

5. Add a AjaxLoadingPanel to "paneFooter" tag. The footer is essentially a status bar. Updates performed by the RadAjaxManager will trigger the loading panel to display in the footer.

```
<radA:AjaxLoadingPanel ID="AjaxLoadingPanel1" runat="server"
    Height="100%" Width="100%" IsSticky="True">
     <asp:Image ID="Image1" runat="server" AlternateText="Updating..."
      ImageUrl="~/Img/LoadingProgressBar.gif"
     CssClass="UpdateImage" />  Updating...
</radA:AjaxLoadingPanel>
```

6. Just after the last splitter tag closes and before the end form tag add a RadAjaxManager to handle no-postback interactions between the treeview, grid and field set. This control will have both server and client side methods added later in this lab, but for now the RadAjaxManager settings just map out the update relationships between treeview, grid and fieldset. Notice the RadAjaxManager1 setting. This will be used later when we make an AjaxRequest() call to update grid row clicks and when the refresh button is clicked.

```
<radA:RadAjaxManager ID="RadAjaxManager1" runat="server"
     OnAjaxRequest="AjaxRequest" ClientEvents-OnResponseReceived="ResponseReceived"
     ClientEvents-OnRequestSent="RequestSent">
   <AjaxSettings>
      <radA:AjaxSetting AjaxControlID="RadTreeView1">
       <UpdatedControls>
          <radA:AjaxUpdatedControl ControlID="RadTreeView1"
```

```
            LoadingPanelID="AjaxLoadingPanel1" />
          <radA:AjaxUpdatedControl ControlID="RadGrid1"
            LoadingPanelID="AjaxLoadingPanel1" />
          <radA:AjaxUpdatedControl ControlID="details"
            LoadingPanelID="AjaxLoadingPanel1" />
        </UpdatedControls>
      </radA:AjaxSetting>
       <radA:AjaxSetting AjaxControlID="RadAjaxManager1">
        <UpdatedControls>
          <radA:AjaxUpdatedControl ControlID="RadTreeView1"
            LoadingPanelID="AjaxLoadingPanel1" />
          <radA:AjaxUpdatedControl ControlID="RadGrid1"
            LoadingPanelID="AjaxLoadingPanel1" />
          <radA:AjaxUpdatedControl ControlID="details"
            LoadingPanelID="AjaxLoadingPanel1" />
        </UpdatedControls>
      </radA:AjaxSetting>
    </AjaxSettings>
</radA:RadAjaxManager>
```

Server Code

1. In the code behind for the default page add using statements to reference Telerik and IO namespaces:

```
C# Example:
using System.IO;
using Telerik.WebControls;

VB Example:
Imports System.IO
Imports Telerik.WebControls
```

2. In the code behind for the default page add properties to get the path and current search option. Both of these values are from controls on "slidingPaneOptions". The current path has either been selected from the treeview by the user or its the root path from the options pane. The current search option by default is to search only top level directories (this just affects the file count) and is also set on the options pane.

```
C# Example:
private string CurrentPath
{
  get
  {
    return RadTreeView1.SelectedNode == null ?
        tbPath.Text : RadTreeView1.SelectedNode.Value;
  }
}

private SearchOption CurrentSearchOption
{
  get
  {
    return cbSubDirectories.Checked ?
          SearchOption.AllDirectories : SearchOption.TopDirectoryOnly;
  }
}

VB Example:
Private ReadOnly Property CurrentPath() As String
 Get
```

```
   Return   Microsoft.VisualBasic.IIf(_
      RadTreeView1.SelectedNode Is Nothing,tbPath.Text,RadTreeView1.SelectedNode.Value)
 End Get
End Property

Private ReadOnly Property  CurrentSearchOption() As  SearchOption
 Get
   Return   Microsoft.VisualBasic.IIf(_
         cbSubDirectories.Checked,SearchOption.AllDirectories,SearchOption.TopDirectoryOnly)
 End Get
End Property
```

3. We will be coding server side event handlers to populate and respond to treeview, grid and fieldset, but first we need a series of utilities routines to assist. Copy each of these methods to the code behind of the default page.

   - GetTreeNode() constructs a RadTreeNode using parameter supplied name, value and if the node should be expanded. This method is called with isExpanded = true to create the root node.

```
C# Example:
private RadTreeNode GetTreeNode(string name, string value, bool isExpanded)
{
   RadTreeNode node = new  RadTreeNode(name);
   node.ImageUrl = "Img/mailfolder.gif";
   node.Expanded = isExpanded;
   node.Value = value;
  return node;
}

VB Example:
Private Function  GetTreeNode(ByVal name As String, _
  ByVal value As String, ByVal isExpanded As Boolean) As  RadTreeNode
 Dim node As RadTreeNode = New  RadTreeNode(name)
  node.ImageUrl  =  "Img/mailfolder.gif"
  node.Expanded  =  isExpanded
  node.Value = value
  Return node
End Function
```

   - This method maps the file extension to an image for use when representing each file with an icon.

```
C# Example:
private string  GetImageForExtension(string  fileName)
{
  string image = "File.gif";

  switch   (Path.GetExtension(fileName))
  {
    case ".cs": image = "cs.gif"; break;
    case ".css": image = "css.gif"; break;
    case ".html": image = "html.gif"; break;
    case ".resx": image = "resx.gif"; break;
    case ".vb": image = "vb.gif"; break;
    case ".xml": image = "xml.gif"; break;
    case ".ascx":
    case ".aspx": image = "ascx.gif"; break;
    case ".gif":
    case ".jpg": image = "gif.gif"; break;
    case "": image = "mailfolder.gif"; break;
```

```
  }
  return image;
}

VB Example:
Private Function  GetImageForExtension(ByVal fileName As String) As String
 Dim image As String = "File.gif"
 Select  Path.GetExtension(fileName)
 Case ".cs"
    image = "cs.gif"
   ' break
 Case ".css"
    image = "css.gif"
   ' break
 Case ".html"
    image = "html.gif"
   ' break
 Case ".resx"
    image = "resx.gif"
   ' break
 Case ".vb"
    image = "vb.gif"
   ' break
 Case ".xml"
    image = "xml.gif"
   ' break
 Case ".ascx", ".aspx"
    image = "ascx.gif"
   ' break
 Case ".gif", ".jpg"
    image = "gif.gif"
   ' break
 Case ""
    image = "mailfolder.gif"
   ' break
 End Select
 Return image
End Function
```

- GetDirectories() wraps the System.IO Directory.GetDirectories() method. The UnauthorizedAccessException is caught and ignored for this example. This would also be an appropriate place to put additional exception handling to trap for things like "file not found".

```
C# Example:
private string[] GetDirectories(string path)
{
  string[] directories = new string[0];
  try
  {
     directories = Directory.GetDirectories(path, "*.*", this.CurrentSearchOption);
  }
  catch  (UnauthorizedAccessException)
  {
    // ignore
  }
  return directories;
}

VB Example:
Private Function GetDirectories(ByVal path As String) As String()
```

```
 Dim  directories(0) As String
 Try
    directories  =  Directory.GetDirectories(path,  "*.*",  Me.CurrentSearchOption)
  Catch  generatedExceptionVariable0 As  UnauthorizedAccessException
 End Try
 Return  directories
End Function
```

- GetFileList() creates a data table with columns for file and directory Name, Size and Path.

```
C#  Example:
private  DataTable  GetFileList(string  path)
{
   DataTable  fileList  =  new  DataTable();
    fileList.Columns.Add("Name");
    fileList.Columns.Add("Size");
    fileList.Columns.Add("Path");

   DirectoryInfo  dir  =  new  DirectoryInfo(path);

  try
  {
    foreach  (DirectoryInfo  subDir  in  dir.GetDirectories())
    {
       fileList.Rows.Add(
        new  string[]  {  subDir.Name,  "-1",  subDir.FullName  });
    }

    foreach  (FileInfo  file  in  dir.GetFiles())
    {
       fileList.Rows.Add(
        new  string[]  {  file.Name,  file.Length.ToString(),  file.FullName  });
    }
  }
  catch   (UnauthorizedAccessException)
  {
    //  ignore
  }

  return  fileList;
}

VB  Example:
Private Function  GetFileList(ByVal  path  As  String)  As  DataTable
 Dim  fileList  As  DataTable  =  New  DataTable
   fileList.Columns.Add("Name")
   fileList.Columns.Add("Size")
   fileList.Columns.Add("Path")
 Dim  dir  As  DirectoryInfo  =  New  DirectoryInfo(path)
 Try
   For Each  subDir  As  DirectoryInfo  In  dir.GetDirectories
      fileList.Rows.Add(New  String()  {subDir.Name,  "-1",  subDir.FullName})
   Next
   For Each  file  As  FileInfo  In  dir.GetFiles
      fileList.Rows.Add(New  String()  {file.Name,  file.Length.ToString,  file.FullName})
   Next
  Catch  generatedExceptionVariable0 As  UnauthorizedAccessException
 End Try
 Return  fileList
End Function
```

- The detail frameset in the lower right panel contains a legend element that displays the title and a number of Label and Input pairs for each property of a file, e.g. creation date.

```
C# Example:
private LiteralControl GetFramesetElement(string name, string value)
{
  const string elementFormat = "<label>{0}</label>" +
    "<input class=\"FileInput\" disabled=\"true\" type=\"text\": value=\"{1}\">" +
    "<br>";
  return new LiteralControl(string.Format(elementFormat, name, value));
}

private LiteralControl GetLegend(string legend)
{
  return new LiteralControl(
    string.Format("<legend>{0}</legend>", legend));
}

VB Example:
Private Function GetFramesetElement(ByVal name As String, _
  ByVal value As String) As LiteralControl
 Const elementFormat As String = "<label>{0}</label>" + _
   "<input class=""FileInput"" disabled=""True"" Type=""Text"": value=""{1}"">" + _
  "<br>"
 Return New LiteralControl(String.Format(elementFormat, name, value))
End Function

Private Function GetLegend(ByVal legend As String) As LiteralControl
 Return New LiteralControl(String.Format("<legend>{0}</legend>", legend))
End Function
```

- UpdateFrameset() takes a reference to the frameset tag control and populates the frameset using information retrieved from static methods of the System.IO.File class. The frameset is first cleared of all previous controls, the legend control is added with the file name, then a series of Label/Input pairs display the properties of the file or directory.

```
C# Example:
private void UpdateFrameset(HtmlGenericControl frameset, string path)
{
   FileAttributes attributes = File.GetAttributes(path);

   frameset.Controls.Clear();
   frameset.Controls.Add(
      GetLegend(Path.GetFileNameWithoutExtension(path)));
   frameset.Controls.Add(
     GetFramesetElement("Path", path));
   frameset.Controls.Add(
     GetFramesetElement("Attributes", attributes.ToString()));
   frameset.Controls.Add(
     GetFramesetElement("Created On", File.GetCreationTime(path).ToString()));
   frameset.Controls.Add(
     GetFramesetElement("Last Modified", File.GetLastWriteTime(path).ToString()));
   frameset.Controls.Add(
     GetFramesetElement("Last Access", File.GetLastAccessTime(path).ToString()));
  if ((attributes & FileAttributes.Directory) == FileAttributes.Directory)
  {
    try
    {
```

```
      string[] files =
         Directory.GetFiles(path, "*.*", this.CurrentSearchOption);
       frameset.Controls.Add(
         GetFramesetElement("File  Count",   files.Length.ToString()));
    }
    catch
    {
      // ignore access exception
    }
  }
}

VB Example:
Private Sub  UpdateFrameset(ByVal  frameset As  HtmlGenericControl,_
  ByVal path As String)
 Dim  attributes As  FileAttributes  =  File.GetAttributes(path)
  frameset.Controls.Clear
     frameset.Controls.Add(GetLegend(Path.GetFileNameWithoutExtension(path)))
  frameset.Controls.Add(_
    GetFramesetElement("Path",  path))
  frameset.Controls.Add(_
     GetFramesetElement("Attributes",  attributes.ToString))
  frameset.Controls.Add(_
     GetFramesetElement("Created  On",  File.GetCreationTime(path).ToString))
  frameset.Controls.Add(_
     GetFramesetElement("Last  Modified",  File.GetLastWriteTime(path).ToString))
  frameset.Controls.Add(_
     GetFramesetElement("Last  Access",  File.GetLastAccessTime(path).ToString))
 If  (attributes And  FileAttributes.Directory)  =  FileAttributes.Directory Then
   Try
     Dim files As String()  =  Directory.GetFiles(path,  "*.*",  Me.CurrentSearchOption)
        frameset.Controls.Add(GetFramesetElement("File  Count",  files.Length.ToString))
   Catch
   End Try
 End If
End Sub
```

- The LoadTreeView() method clears and rebuilds the treeview nodes. Most of the work populating the nodes occurs in the BindDirectory() method.

```
C# Example:
private void  LoadTreeView(RadTreeView  treeview)
{
   treeview.Nodes.Clear();
   RadTreeNode  rootNode  =  GetTreeNode(
    this.CurrentPath, this.CurrentPath, true);
    treeview.Nodes.Add(rootNode);

   BindDirectory(this.CurrentPath,   rootNode.Nodes);
}

VB Example:
Private Sub  LoadTreeView(ByVal  treeview As  RadTreeView)
  treeview.Nodes.Clear
 Dim  rootNode As  RadTreeNode  =  GetTreeNode(Me.CurrentPath,  Me.CurrentPath, True)
  treeview.Nodes.Add(rootNode)
  BindDirectory(Me.CurrentPath,  rootNode.Nodes)
End Sub
```

- BindDirectory() populates treeview nodes using the dirPath parameter as a starting point and adding to the node collection passed in. At startup "nodes" would be the nodes collection for the root node. Later when the user expands a node we use the nodes collection of the clicked on node.

  Notice the call to path.Split()? The last element of the directory path is extracted and placed in the name variable. For example if a path is "c:\\Documents and Settings\\All Users\\Application Data" then the name displayed in the treeview node is "Application Data". The assignment of node.ExpandMode to "ServerSide" forces the NodeExpand event handler to fire, giving us a chance to populate the newly opened set of nodes.

```
C# Example:
private void BindDirectory(string dirPath, RadTreeNodeCollection nodes)
{
  if (dirPath == string.Empty)
    return;

  string[] dirs = this.GetDirectories(dirPath);
  foreach (string path in dirs)
  {
    string[] parts = path.Split('\\');
    string name = parts[parts.Length - 1];
     RadTreeNode node = GetTreeNode(name, path, false);

    string[] directories = this.GetDirectories(path);
    if (directories.Length > 0)
    {
        node.ExpandMode = ExpandMode.ServerSide;
    }
     nodes.Add(node);
  }
}

VB Example:
Private Sub BindDirectory(ByVal dirPath As String,_
  ByVal nodes As  RadTreeNodeCollection)
 If dirPath = String.Empty Then
   Return
 End If
 Dim dirs As String() = Me.GetDirectories(dirPath)
 For Each path As String In dirs
   Dim parts As String() = path.Split("\"C)
   Dim name As String = parts(parts.Length - 1)
   Dim node As RadTreeNode = GetTreeNode(name, path, False)
   Dim directories As String() = Me.GetDirectories(path)
   If directories.Length > 0 Then
      node.ExpandMode = ExpandMode.ServerSide
   End If
    nodes.Add(node)
 Next
End Sub
```

4. Now that the utility routines are taken care of, code the event handlers for the page and controls. Copy the code samples to the code behind for the default page. Create event handlers where necessary to contain the code.

   - Load the treeview during Page_Load.

```
C# Example:
```

```
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
     LoadTreeView(RadTreeView1);
  }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
    LoadTreeView(RadTreeView1)
 End If
End Sub
```

- For the treeview create a NodeClick event handler and assign the DataSource using the data table returned from GetFileList.

```
C# Example:
protected void RadTreeView1_NodeClick(object o, RadTreeNodeEventArgs e)
{
   RadGrid1.DataSource = GetFileList(this.CurrentPath);
   RadGrid1.DataBind();
}

VB Example:
Protected Sub RadTreeView1_NodeClick(ByVal o As Object,_
  ByVal e As RadTreeNodeEventArgs)
  RadGrid1.DataSource = GetFileList(Me.CurrentPath)
  RadGrid1.DataBind
End Sub
```

- Use the grid ItemDataBound event to modify controls in the grid column templates. Here we assign the icon image url based off the file name. The Size column is also tweaked slightly to ignore file sizes of "-1".

```
C# Example:
protected void RadGrid1_ItemDataBound(object sender, GridItemEventArgs e)
{
  if (e.Item is GridDataItem)
  {
     System.Web.UI.WebControls.Image icon =
        (System.Web.UI.WebControls.Image)e.Item.FindControl("icon");
    icon.ImageUrl = "Img/" +
      GetImageForExtension((string)DataBinder.Eval(e.Item.DataItem, "Name"));
    icon.AlternateText = "icon";
     icon.ImageAlign = System.Web.UI.WebControls.ImageAlign.AbsMiddle;
    icon.Style.Add("vertical-align", "middle");
    icon.BorderWidth = Unit.Pixel(0);

     GridDataItem dataItem = (GridDataItem)e.Item;

   string fileLength = (string)DataBinder.Eval(e.Item.DataItem, "Size");
   if (fileLength != "-1")
     dataItem["Size"].Text = fileLength;
   else
     dataItem["Size"].Text = " ";
  }
}
```

```
VB Example:
Protected Sub RadGrid1_ItemDataBound(ByVal sender As Object,_
  ByVal e As GridItemEventArgs)
 If TypeOf e.Item Is GridDataItem Then
   Dim icon As System.Web.UI.WebControls.Image = _
        CType(e.Item.FindControl("icon"), System.Web.UI.WebControls.Image)
    icon.ImageUrl = "Img/" + GetImageForExtension(_
       CType(DataBinder.Eval(e.Item.DataItem, "Name"), String))
    icon.AlternateText = "icon"
     icon.ImageAlign = System.Web.UI.WebControls.ImageAlign.AbsMiddle
     icon.Style.Add("vertical-align", "middle")
    icon.BorderWidth = Unit.Pixel(0)
   Dim dataItem As GridDataItem = CType(e.Item, GridDataItem)
   Dim fileLength As String = CType(DataBinder.Eval(e.Item.DataItem, "Size"), String)
   If Not (fileLength = "-1") Then
      dataItem("Size").Text = fileLength
   Else
      dataItem("Size").Text = " "
   End If
 End If
End Sub
```

- Create a NodeExpand event handler for the treeview. When the expand is clicked and there are no nodes under the clicked item, re-bind using the clicked-on node as a starting point.

```
C# Example:
protected void RadTreeView1_NodeExpand(object o, RadTreeNodeEventArgs e)
{
  if (e.NodeClicked.Nodes.Count == 0)
  {
      BindDirectory(e.NodeClicked.Value, e.NodeClicked.Nodes);
  }
}

VB Example:
Protected Sub RadTreeView1_NodeExpand(ByVal o As Object,_
  ByVal e As RadTreeNodeEventArgs)
 If e.NodeClicked.Nodes.Count = 0 Then
     BindDirectory(e.NodeClicked.Value, e.NodeClicked.Nodes)
 End If
End Sub
```

- This last server side event leads us into the client side code. The AjaxRequest method is initiated from a client side call that gets handled by the RadAjaxManager. You'll see this from the client side shortly to see how the method gets invoked, but for now know that when some client side method gets called, say from the row click of a grid or from a refresh button, then this method is called asynchronously and passed a single argument. The argument typically is the name of the command but can be anything you care to stuff into the argument string on the client side. For example the grid RowClick argument is formatted on the client as "RowClick:123" where 123 is the item index of the item clicked. In the case of the Refresh only the "Refresh" command name is passed.

```
C# Example:
public void AjaxRequest(object source,
      Telerik.WebControls.AjaxRequestEventArgs e)
{
  string[] arguments = e.Argument.Split(':');
  string commandName = arguments[0];
  switch (commandName)
  {
```

```
    case "RowClick":
      {
        int index = int.Parse(arguments[1]);
          GridDataItem item = RadGrid1.Items[index];
          Label pathLabel = item.FindControl("pathLabel") as Label;
           UpdateFrameset(details, pathLabel.Text);

        break;
      }
    case "Refresh":
      {
          LoadTreeView(RadTreeView1);
          RadGrid1.DataSource = GetFileList(this.CurrentPath);
          RadGrid1.DataBind();
        break;
      }
  }
}

VB Example:
Public Sub AjaxRequest(ByVal source As Object, _
 ByVal e As   Telerik.WebControls.AjaxRequestEventArgs)
 Dim arguments As String() = e.Argument.Split(":"C)
 Dim commandName As String = arguments(0)
 Select commandName
 Case "RowClick"
   Dim index As Integer = Integer.Parse(arguments(1))
   Dim item As GridDataItem = RadGrid1.Items(index)
   Dim pathLabel As Label = _
        CType(ConversionHelpers.AsWorkaround(item.FindControl("pathLabel"), _
     GetType(Label)), Label)
    UpdateFrameset(details, pathLabel.Text)
   ' break
 Case "Refresh"
    LoadTreeView(RadTreeView1)
    RadGrid1.DataSource = GetFileList(Me.CurrentPath)
   RadGrid1.DataBind
   ' break
 End Select
End Sub
```

Client Side Code

In the HTML for the default page add a script section just after the opening form tag. In the next step we will hook up these functions, but for now just paste them in.

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

 var isInAjaxCall;

 function RequestStart()
{
  isInAjaxCall = true;
}

 function ResponseEnd()
{
   isInAjaxCall = false;
}

 function RowClick(index)
```

```
  {
     if (!isInAjaxCall)
    {
         window["<%=RadAjaxManager1.ClientID%>"].AjaxRequest("RowClick:"  +
          this.Rows[index].ItemIndex);
    }
    else
    {
       alert("too many requests at one time");
    }
  }

   function  RefreshClick()
  {
     var slidingZone = <%= zoneLeft.ClientID %>;
       slidingZone.UnDockPane("slidingPaneOptions");
       slidingZone.DockPane("slidingPaneFiles");
     if (!isInAjaxCall)
    {
         window["<%=RadAjaxManager1.ClientID%>"].AjaxRequest("Refresh");
    }
  }

</script>
```

There are several issues to discuss about the JavaScript code above:

- It can happen that multiple Ajax requests are attempted, locking up the application. The work around for this is to use the RadAjaxManager ClientEvents-OnRequestStart and ClientEvents-OnResponseEnd event handlers to track the current status of Ajax traffic. Set an isInAjaxCall variable true when a request is sent, then check it before initiating any Ajax requests.

  Note: The events for RadAjaxManager occur in this order:

  RequestStart
  RequestSent
  ResponseReceived
  ResponseEnd

- RowClick() will be called from the rad grid client events OnRowClick. That event handler is provided the index into the row that was clicked. The AjaxRequest call passes a command name "RowClick", a colon as a delimiter, then the ItemIndex of the row. In the server AjaxRequest() event handler the argument is split apart. Note: We could also have used the __doPostBack method if we didn't mind the postback effect. The call would have been:

  ```
  __doPostBack("<%=  RadGrid1.UniqueID  %>",  "RowClicked:"  +  this.Rows[index].ItemIndex);
  ```

- The purpose behind RefreshClick() is to change the UI on the client side (hide the option pane, dock the file list), then call back to the server run code that would otherwise have been in a button click event handler.

Hookup Events and Run

1. Now that the JavaScript is in place the last step is to hook up the client side event handlers.

   - In the tag for btnRefresh add the onClick event handler:

     ```
     OnClick="RefreshClick();"
     ```

- In the Grid ClientSettings add the OnRowClick event handler.

```
<ClientEvents  OnRowClick="RowClick"  />
```

- In the RadAjaxManager add the events below. OnAjaxRequest is actually a server side handler that gets triggered on the client side and the other two are client side only.

```
OnAjaxRequest="AjaxRequest"
ClientEvents-OnRequestStart="RequestStart"
ClientEvents-OnResponseEnd="ResponseEnd"
```

2. Run the application. Test the functionality:

- Click on directories in the treeview.
- Click on directories and files in the grid (upper right hand pane). See how the field set responds.
- Open the options sliding pane and enter a new path. You can also point at a path on the network. Be aware that no error handling is in place for "file not found". Notice how the

2.6.6 Summary

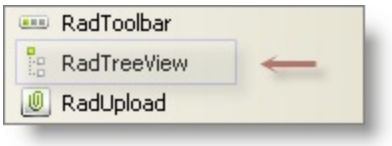
This chapter demonstrated techniques for handling screen real estate using the rad splitter, panes and split bars. The chapter explored pane and sliding pane size, docking and collapsing behaviors. To illustrate a common UI design pattern the splitter was used to create an Explorer style user interface. A more involved File Explorer example demonstrated integration between the splitter and multiple rad components as well as designing the application to be AJAX enabled using RadAjaxManager.

2.7 RadTreeView

2.7.1 Getting Started

A RadTreeView control can be used to display hierarchical data such as file directory, table of contents in a tree structure. The RadTreeView control is made up of nodes. Each node in the tree is represented by a RadTreeNode object. Nodes can be classified as root node, parent node, child node and leaf node.

- A node that is not contained by any other node and is the ancestor to all the other nodes is called a root node.
- A node that contains other nodes is called a parent node.
- The node that is contained by another node is called a child node. Child nodes in the same level are called siblings.
- A node that has no children is called a leaf node. A node can be both a parent and a child, but root, parent, and leaf nodes are mutually exclusive.



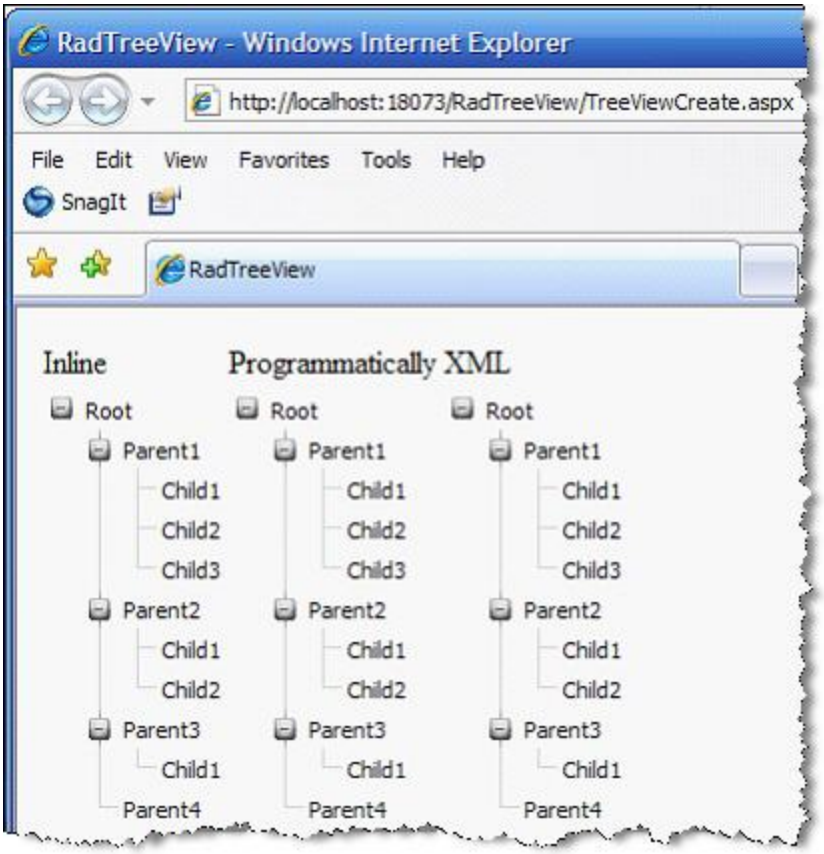
RadTreeView in the toolbox

2.7.2 Using RadTreeView in the Designer

Defining TreeView Items

You can define RadTreeView by multiple routes with essentially identical output (see figure below):

- In the designer.
- "Inline" in the HTML markup.
- Programmatically in the code-behind.
- Using an XML file to define the structure and using the *ContentFile* property or LoadContentFile() method.

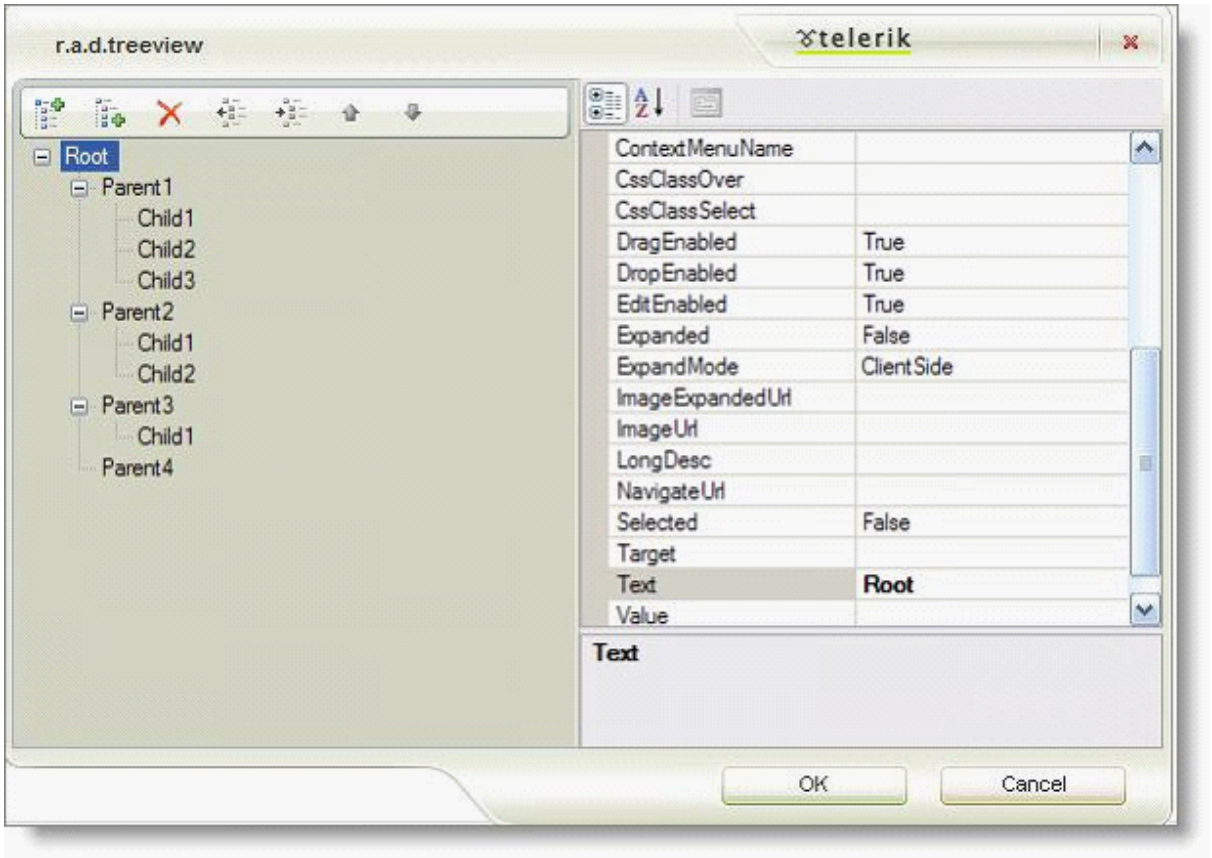


RadTreeView output by defining inline, programmatically and using XML

Lab: Defining in the Designer

The structure of a RadTreeView can be defined at design-time using the designer associated with the RadTreeView control. The nodes created at design-time will be persisted in the aspx or ascx markup files. To create a RadTreeView as design-time you need to

1. Create a empty or open an existing ASP.NET Web Site in Visual Studio IDE
2. Choose a WebForm and add a RadTreeView Control on the page
3. Click on the smart tag associated with the RadTreeView control or right click on the RadTreeView and choose Build RadTreeView. This will launch the RadTreeView designer (as shown in the following figure)
4. You can now add/edit/delete RadTreeNodes and set their properties.



Defining treeview in the designer

Defining Inline

You can also define the RadTreeView structure inline directly by editing the aspx or ascx markup files. Adding the following markup to your aspx or ascx file will create a RadTreeView same as the one above created using the designer.

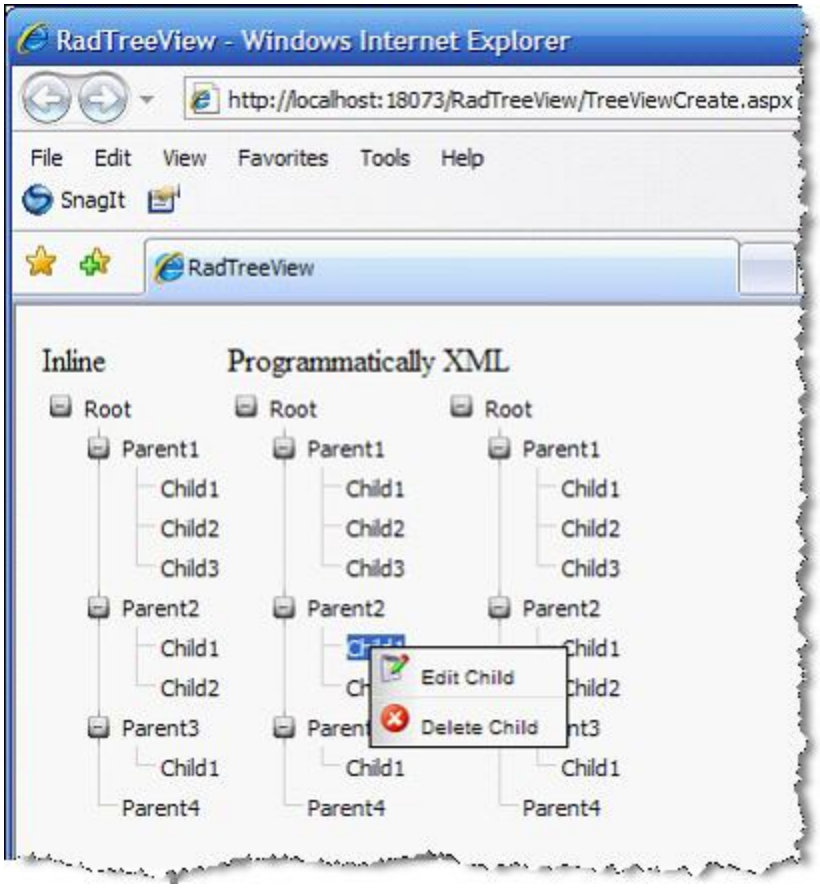
```
<%@  Register  Assembly="RadTreeView.Net2"  Namespace="Telerik.WebControls"
   TagPrefix="radT" %>

<radT:RadTreeView  ID="RadTreeView1"  runat="server">
<Nodes>
<radT:RadTreeNode  runat="server"  Text="Root">
    <Nodes>
          <radT:RadTreeNode runat="server" Text="Parent1">
             <Nodes>
                   <radT:RadTreeNode runat="server" Text="Child1">
                 </radT:RadTreeNode>
                   <radT:RadTreeNode runat="server" Text="Child2">
                 </radT:RadTreeNode>
                   <radT:RadTreeNode runat="server" Text="Child3">
                 </radT:RadTreeNode>
             </Nodes>
         </radT:RadTreeNode>
          <radT:RadTreeNode runat="server" Text="Parent2">
             <Nodes>
                   <radT:RadTreeNode runat="server" Text="Child1">
```

```
                </radT:RadTreeNode>
                 <radT:RadTreeNode runat="server" Text="Child2">
                </radT:RadTreeNode>
            </Nodes>
        </radT:RadTreeNode>
         <radT:RadTreeNode runat="server" Text="Parent3">
            <Nodes>
                 <radT:RadTreeNode runat="server" Text="Child1">
                </radT:RadTreeNode>
            </Nodes>
        </radT:RadTreeNode>
         <radT:RadTreeNode runat="server" Text="Parent4">
        </radT:RadTreeNode>
    </Nodes>
</radT:RadTreeNode>
</Nodes>
</radT:RadTreeView>
```

Defining Context Menus

You can add right click context menus for each RadTreeView node by setting the nodes *ContextMenuName* property.

- *RadTreeViewContextMenu.ContextMenu* type represents a context menu. You can have several context menus associated with a RadTreeView by setting the *ContextMenus* property to an ArrayList of ContextMenus.
- *RadTreeViewContextMenu.ContextMenuItem* type represents each individual item in the context menu. Each ContextMenuItem should be assigned a unique ID.
- The ContextMenuItem *Image* property can be used to associate an image.
- The ContextMenuItem *PostBack* property can be set to true to cause a postback when the menu item is clicked.



Using the context menu

Applying Templates

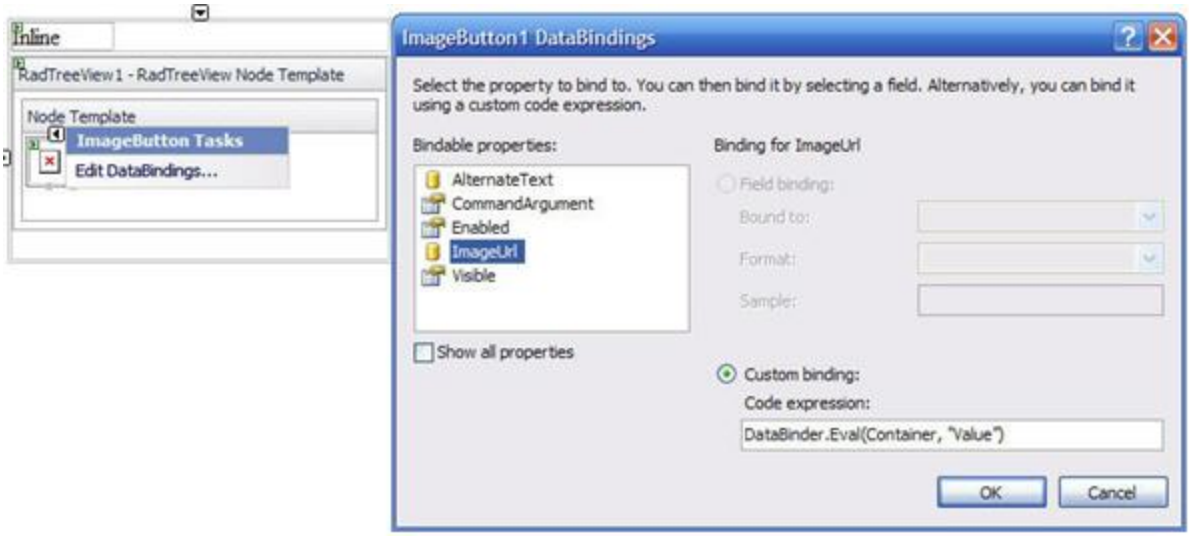
RadTreeView supports templates to customize the appearance of tree nodes by embedding other controls into a RadTreeView. There are two types of templates that can be added to a RadTreeView.

1. A global (RadTreeView Node template) template that will be applied to all RadTreeNode automatically.

2. A node template that can be applied to specify individual nodes. A node template will override the global template.

You can add and edit a global template at design-time by using the RadTreeView Smart Tag, clicking Edit Templates and choosing RadTreeView Node Template. Now, on the RadTreeView Node Template editor you can drag and drop any control from the toolbox. For example: drop an ImageButton, choose Edit DataBindings and set the ImageUrl property to databinding expression. In the following code snippet the RadTreeNode *Value* is bound to the *ImageUrl* property of the ImageButton control:

```
<%#  DataBinder.Eval(Container,  "Value")  %>
```



Binding the ImageUrl in a template

You can override the global template by defining a template for a specific node. For example, drop a Button, choose Edit DataBindings and set the *Text* property to a databinding expression:

```
<%# DataBinder.Eval(Container, "Value") %>.
```

Adding Attributes

You will very likely need to store arbitrary data in treeview nodes, for example, ID numbers corresponding to database data. The RadTreeNode.Attributes collection can be used to store any number of attributes as name-value pairs. You can add attributes to RadTreeNodes inline, at runtime or using an XML file. In the following example we have added a CustomData property to the tree nodes and since it's not a property of the RadTreeNode itself, CustomData will be treated as an attribute. We have also added a NodeTemplate with a data binding expression <%# DataBinder.Eval(Container, "Attributes['CustomData']")%> to bind the CustomData attribute to the RadTreeNode *Text*:

```
<radT:RadTreeView ID="RadTreeView1" runat="server" Skin="Inox">
    <Nodes>
          <radT:RadTreeNode runat="server" Text="Root" CustomData="RootNode">
            <Nodes>
                <radT:RadTreeNode runat="server" Text="Parent1"
                  CustomData="ParentNode1">
                  <Nodes>
                      <radT:RadTreeNode runat="server" Text="Child1"
                        CustomData="Parent1ChildNode1">
                      </radT:RadTreeNode>
                        <radT:RadTreeNode runat="server" Text="Child2"
                          CustomData="Parent1ChildNode2"   >
                      </radT:RadTreeNode>
                        <radT:RadTreeNode runat="server" Text="Child3"
                          CustomData="Parent1ChildNode3">
                      </radT:RadTreeNode>
                  </Nodes>
                </radT:RadTreeNode>
                <radT:RadTreeNode runat="server" Text="Parent2"
                  CustomData="ParentNode2">
                  <Nodes>
                      <radT:RadTreeNode runat="server" Text="Child1"
                        CustomData="Parent2ChildNode1">
                      </radT:RadTreeNode>
```

```
                            <radT:RadTreeNode runat="server" Text="Child2"
                              CustomData="Parent2ChildNode2">
                            </radT:RadTreeNode>
                        </Nodes>
                    </radT:RadTreeNode>
                </Nodes>
            </radT:RadTreeNode>
        </Nodes>
        <NodeTemplate>
              <%# DataBinder.Eval(Container, "Attributes['CustomData']")%>
        </NodeTemplate>
</radT:RadTreeView>
```

2.7.3 Using RadTreeView at Runtime

Defining TreeView Items at Runtime

Defining Programmatically

You can also programmatically add or edit or delete RadTreeNodes on the server side using the RadTreeView API.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        AddNodes();
    }
}

private void AddNodes()
{
    //Clear nodes
     RadTreeView2.Nodes.Clear();
    RadTreeNode root = new RadTreeNode("Root", "-1");
    RadTreeNode p1 = new RadTreeNode("Parent1", "1");

    RadTreeNode c11 = new RadTreeNode("Child1", "1.1");
    RadTreeNode c12 = new RadTreeNode("Child2", "1.2");
    RadTreeNode c13 = new RadTreeNode("Child3", "1.3");
     p1.Nodes.Add(c11);
     p1.Nodes.Add(c12);
     p1.Nodes.Add(c13);

    RadTreeNode p2 = new RadTreeNode("Parent2", "2");
    RadTreeNode c21 = new RadTreeNode("Child1", "2.1");
    RadTreeNode c22 = new RadTreeNode("Child2", "2.2");
     p2.Nodes.Add(c21);
     p2.Nodes.Add(c22);

    RadTreeNode p3 = new RadTreeNode("Parent3", "3");
    RadTreeNode c31= new RadTreeNode("Child1", "3.1");
     p3.Nodes.Add(c31);

    RadTreeNode p4 = new RadTreeNode("Parent4", "4");

     root.Nodes.Add(p1);
```

```
     root.Nodes.Add(p2);
     root.Nodes.Add(p3);
     root.Nodes.Add(p4);
      RadTreeView2.Nodes.Add(root);

    //Expand all nodes
      RadTreeView2.ExpandAllNodes();
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
   AddNodes
 End If
End Sub

Private Sub AddNodes()
  RadTreeView2.Nodes.Clear
 Dim root As RadTreeNode = New RadTreeNode("Root", "-1")
 Dim p1 As RadTreeNode = New RadTreeNode("Parent1", "1")
 Dim c11 As RadTreeNode = New RadTreeNode("Child1", "1.1")
 Dim c12 As RadTreeNode = New RadTreeNode("Child2", "1.2")
 Dim c13 As RadTreeNode = New RadTreeNode("Child3", "1.3")
  p1.Nodes.Add(c11)
  p1.Nodes.Add(c12)
  p1.Nodes.Add(c13)
 Dim p2 As RadTreeNode = New RadTreeNode("Parent2", "2")
 Dim c21 As RadTreeNode = New RadTreeNode("Child1", "2.1")
 Dim c22 As RadTreeNode = New RadTreeNode("Child2", "2.2")
  p2.Nodes.Add(c21)
  p2.Nodes.Add(c22)
 Dim p3 As RadTreeNode = New RadTreeNode("Parent3", "3")
 Dim c31 As RadTreeNode = New RadTreeNode("Child1", "3.1")
  p3.Nodes.Add(c31)
 Dim p4 As RadTreeNode = New RadTreeNode("Parent4", "4")
  root.Nodes.Add(p1)
  root.Nodes.Add(p2)
  root.Nodes.Add(p3)
  root.Nodes.Add(p4)
   RadTreeView2.Nodes.Add(root)
   RadTreeView2.ExpandAllNodes
End Sub
```

Defining In XML

You can also define the structure of a RadTreeView using an XML file. Here is an example of the XML file structure used to load the treeview:

```
<Tree>
   <Node Text="Root" Expanded="True" Value="-1">
      <Node Text="Parent1" Expanded="True" Value="1">
        <Node Text="Child1" Value="1.1" />
        <Node Text="Child2" Value="1.2" />
       <Node Text="Child3" Value="1.3" />
    </Node>
      <Node Text="Parent2" Expanded="True" Value="2">
        <Node Text="Child1" Value="2.1" />
        <Node Text="Child2" Value="2.2" />
    </Node>
      <Node Text="Parent3" Expanded="True" Value="3">
        <Node Text="Child1" Value="3.1" />
```

```
    </Node>
     <Node Text="Parent4" Expanded="True" Value ="4">
    </Node>
  </Node>
</Tree>
```

To associate the XML file to a RadTreeView you can either set the *ContentFile* property to the path of the XML file or you can use *LoadContentFile()* method to load the XML file. The aspx file would look like this:

```
<radT:RadTreeView ID="RadTreeView3" runat="server" ContentFile="~/tree.xml"
   Skin="Inox">
</radT:RadTreeView>
```

In the code behind use the LoadContentFile() method from the Page_Load event handler.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
          RadTreeView3.LoadContentFile("Tree.xml");
    }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
     RadTreeView3.LoadContentFile("Tree.xml")
 End If
End Sub
```

Lab: Defining Treeview at RunTime

In this lab, we'll use the two techniques just discussed to populate two RadTreeView controls with their items

1. Add a new web page (aspx) to your project.

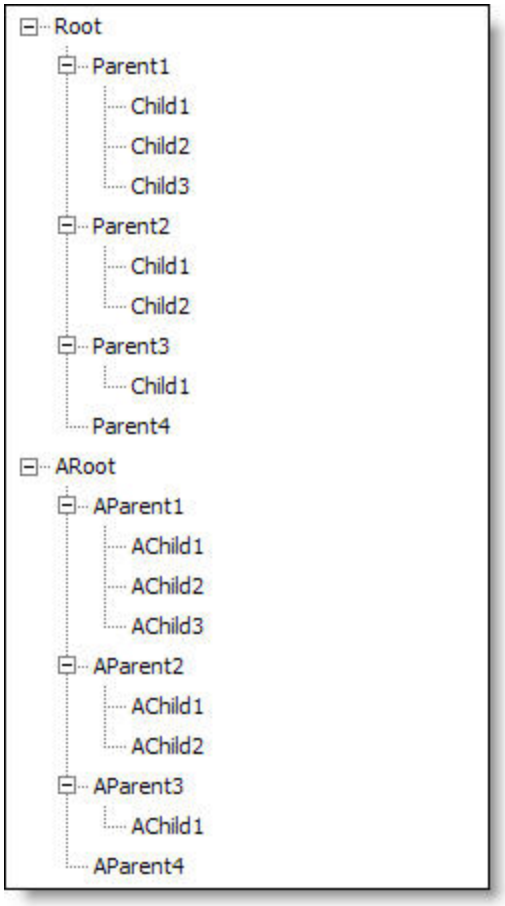
2. Add two RadTreeView components to the design surface

3. Add the code from the section Defining Programmatically to the code-behind. Since the tree defined in the XML file has a structure identical to that defined in code, you may wish to add some identifying characteristic to the node contents of one of the files (for instance, prefixing some of the node Text values with the letter "A").

4. Immediately after the call to AddNodes() in the Page_Load event, insert the code from the topic Defining In XML. Note, however, that you should change the name of the RadGrid control to be RadGrid1, rather than RadGrid3.

5. Add an XML file to your project, named "Tree.xml". Populate it with the data from the XML sample shown in the Defining In XML section.

Run the project. Your browser should display two RadTreeView controls, one defined in code and one loaded from the XML file.



Two RadTreeViews

Defining Context Menus at Runtime

Adding Context Menus Programmatically

In this example three ContextMenu objects are created and added to an array list. The ContextMenu objects have their *Name* properties assigned "RootMenu", "ParentMenu" and "ChildMenu" respectively. The array list is added to the *ContextMenus* property of the treeview. Finally the *ContextMenuName* property is assigned to each node where we want the context menus to appear.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
     AddNodes();
     AddContextMenu();
  }
}

private void AddContextMenu()
{
   ArrayList contextMenus = new ArrayList();
   ContextMenu cm1 = new ContextMenu();
   cm1.Name = "RootMenu";
   cm1.Width = 100;
```

```
   contextMenus.Add(cm1);

  ContextMenu cm2 = new ContextMenu();
  cm2.Name = "ParentMenu";
  cm2.Width = 100;
   contextMenus.Add(cm2);

  ContextMenu cm3 = new ContextMenu();
  cm3.Name = "ChildMenu";
  cm3.Width = 100;
   contextMenus.Add(cm3);

  ContextMenuItem cmi1 = new ContextMenuItem("New Parent");
  cmi1.ID = "Add";
  cmi1.PostBack = true;
  cmi1.Image = "Add.png";

  ContextMenuItem cmi2 = new ContextMenuItem("New Child");
  cmi2.ID = "AddSub";
  cmi2.PostBack = true;
  cmi2.Image = "Add.png";

  ContextMenuItem cmi3 = new ContextMenuItem("Edit Child");
  cmi3.ID = "Edit";
  cmi3.PostBack = true;
  cmi3.Image = "Edit.png";

  ContextMenuItem cmi4 = new ContextMenuItem("Delete Child");
  cmi4.ID = "Delete";
  cmi4.PostBack = true;
  cmi4.Image = "Delete.png";

 //Root Menu items
   cm1.Items.Add(cmi1);

 //Parent Menu items
   cm2.Items.Add(cmi1);
   cm2.Items.Add(cmi2);

 //Child Menu items
   cm3.Items.Add(cmi3);
   cm3.Items.Add(cmi4);

   RadTreeView2.ImagesBaseDir = "~/images/";
   RadTreeView2.ContextMenus = contextMenus;

   RadTreeView2.Nodes[0].ContextMenuName = "RootMenu";
 foreach (RadTreeNode node in RadTreeView2.Nodes[0].Nodes)
 {
    node.ContextMenuName = "ParentMenu";
   foreach (RadTreeNode cnode in node.Nodes)
   {
      cnode.ContextMenuName = "ChildMenu";
   }
 }
}
```

VB Example:

```
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
   AddNodes
```

```
    AddContextMenu
 End If
End Sub

Private Sub  AddContextMenu()
 Dim  contextMenus As ArrayList = New ArrayList
 Dim cm1 As ContextMenu = New ContextMenu
 cm1.Name = "RootMenu"
 cm1.Width = 100
  contextMenus.Add(cm1)
 Dim cm2 As ContextMenu = New ContextMenu
 cm2.Name = "ParentMenu"
 cm2.Width = 100
  contextMenus.Add(cm2)
 Dim cm3 As ContextMenu = New ContextMenu
 cm3.Name = "ChildMenu"
 cm3.Width = 100
  contextMenus.Add(cm3)
 Dim cmi1 As ContextMenuItem = New ContextMenuItem("New Parent")
 cmi1.ID = "Add"
 cmi1.PostBack = True
 cmi1.Image = "Add.png"
 Dim cmi2 As ContextMenuItem = New ContextMenuItem("New Child")
 cmi2.ID = "AddSub"
 cmi2.PostBack = True
 cmi2.Image = "Add.png"
 Dim cmi3 As ContextMenuItem = New ContextMenuItem("Edit Child")
 cmi3.ID = "Edit"
 cmi3.PostBack = True
 cmi3.Image = "Edit.png"
 Dim cmi4 As ContextMenuItem = New ContextMenuItem("Delete Child")
 cmi4.ID = "Delete"
 cmi4.PostBack = True
 cmi4.Image = "Delete.png"
  cm1.Items.Add(cmi1)
  cm2.Items.Add(cmi1)
  cm2.Items.Add(cmi2)
  cm3.Items.Add(cmi3)
  cm3.Items.Add(cmi4)
 RadTreeView2.ImagesBaseDir = "~/images/"
 RadTreeView2.ContextMenus = contextMenus
  RadTreeView2.Nodes(0).ContextMenuName = "RootMenu"
 For Each node As RadTreeNode In RadTreeView2.Nodes(0).Nodes
    node.ContextMenuName = "ParentMenu"
   For Each cnode As RadTreeNode In node.Nodes
      cnode.ContextMenuName = "ChildMenu"
   Next
 Next
End Sub
```

Adding Context Menus using XML

You can also define your context menus using XML files (see Context.xml below). The content file for the RadTreeView refers to the context menus by setting its *ContextMenuName* property (see Tree.xml below). Then the context menu is associated with a RadTreeView by setting the *ContextMenuContentFile* property (see aspx below) .

Context.xml

```
<ContextMenus>
```

```
   <Menu  Name="RootMenu"  Width="100">
      <Item  Image="Add.png"  ID="Add"   Text="New  Parent"  PostBack="True"/>
  </Menu>
   <Menu  Name="ParentMenu"  Width="100">
      <Item  Image="Add.png"  ID="Add"   Text="New  Parent"  PostBack="True"/>
       <Item  Image="Add.png"  ID="AddChild"  Text="New  Child"  PostBack="True"/>
  </Menu>
   <Menu  Name="ChildMenu"  Width="100">
      <Item  Image="Edit.png"  ID="Edit"  Text="Edit  Child"  PostBack="True"/>
       <Item  Image="Delete.png"  ID="Delete"  Text="Delete  Child"  PostBack="True"/>
  </Menu>
</ContextMenus>
```

Tree.xml

```
<Tree>
   <Node  Text="Root"  Expanded="True"  Value="-1" ContextMenuName="RootMenu">
      <Node  Text="Parent1" Expanded="True"  Value="1" ContextMenuName="ParentMenu">
        <Node  Text="Child1"  Value="1.1" ContextMenuName="ChildMenu"/>
        <Node  Text="Child2"  Value="1.2" ContextMenuName="ChildMenu"/>
        <Node  Text="Child3"  Value="1.3" ContextMenuName="ChildMenu"/>
    </Node>
      <Node  Text="Parent2"  Expanded="True"  Value="2" ContextMenuName="ParentMenu">
        <Node  Text="Child1"  Value="2.1" ContextMenuName="ChildMenu"/>
        <Node  Text="Child2"  Value="2.2" ContextMenuName="ChildMenu"/>
    </Node>
      <Node  Text="Parent3"  Expanded="True"  Value="3" ContextMenuName="ParentMenu">
        <Node  Text="Child1"  Value="3.1" ContextMenuName="ChildMenu"/>
    </Node>
      <Node  Text="Parent4"  Expanded="True"  Value  ="4" ContextMenuName="ParentMenu">
    </Node>
  </Node>
</Tree>
```

aspx

```
<radT:RadTreeView  ID="RadTreeView3"  runat="server"    Skin="Inox"
  ContentFile="Tree.xml"
  ContextMenuContentFile="ContextMenu.xml"
    ImagesBaseDir="~\images\">
</radT:RadTreeView>
```

Lab: Defining Context Menus at RunTime

Just like the RadTreeView content definition, You can define context menus at runtime either in code, or by loading them from an XML file-based definition as discussed in the two previous sections. Here's how:

1. Extend the web page you constructed in the last Lab: Defining Treeview at RunTime.

2. Add the code from the section Adding Context Menus Programmatically to the code-behind, adding the call to AddContextMenu in the Page_Load event. Also insert the code for the method call.

3. Add a new XML file named "Context.xml", and copy the contents of the file shown in Adding Context Menus using XML to that file.

4. In the Page_Load event, set the properties for the context menus of the RadTreeView control as follows:

```
RadTreeView1.ContextMenuContentFile   =   "Context.xml";
RadTreeView1.ImagesBaseDir   =   @"~\images\";
```

5. As illustrated in the previous section, you also need to set the ContextMenu names on a node-by-node basis. You can do this either by modifying the Tree.xml file you created previously, to include the ContextMenu name; or you can refactor the code to use the same code used in the programmatic context menu instantiation.

6. Run the project. Verify that you have successfully added the context menus by right-clicking on the items in the RadTreeView.

Applying Templates at Runtime

To add templates at runtime you need to define a class that implements System.Web.UI.ITemplate and then associate an instance of the class to a RadTreeView or RadTreeNode *NodeTemplate* property. This example shows two ITemplate implementations, one for images and one for buttons. The ImageTemplate is a global template, i.e. is assigned to the *NodeTemplate* property of the treeview. The ButtonTemplate is assigned to a specific node in AddNodes().

```
C# Example:
class ImageTemplate : ITemplate
{
  public void InstantiateIn(Control container)
  {
     ImageButton button1 = new ImageButton();
     button1.ID = "ImageButton1";
     button1.DataBinding += new   EventHandler(button1_DataBinding);
      container.Controls.Add(button1);
  }

  private void button1_DataBinding(object sender, EventArgs e)
  {
     ImageButton target = (ImageButton)sender;
      RadTreeNode node = (RadTreeNode)target.BindingContainer;
    string nodeValue = (string)DataBinder.Eval(node, "Value");
     target.ImageUrl = node.Value;
  }
}

class ButtonTemplate : ITemplate
{
  public void InstantiateIn(Control container)
  {
     Button button1 = new Button();
     button1.ID = "Button1";
     button1.DataBinding += new   EventHandler(button1_DataBinding);
      container.Controls.Add(button1);
  }

  private void button1_DataBinding(object sender, EventArgs e)
  {
     Button target = (Button)sender;
      RadTreeNode node = (RadTreeNode)target.BindingContainer;
    string nodeValue = (string)DataBinder.Eval(node, "Value");
     target.Text = node.Value;
  }
}

protected void Page_Load(object sender, EventArgs e)
{
   RadTreeView2.NodeTemplate = new ImageTemplate();
   AddNodes();
```

```
}

private void AddNodes()
{
  //Clear nodes
    RadTreeView2.Nodes.Clear();
   RadTreeNode root = new RadTreeNode("Animals", "~/images/animals006.gif");
   RadTreeNode p1 = new RadTreeNode("Dogs", "~/images/animals019.gif");

   RadTreeNode c11 = new RadTreeNode("Dog1", "~/images/animals020.gif");
   RadTreeNode c12 = new RadTreeNode("Dog2", "~/images/animals021.gif");
   RadTreeNode c13 = new RadTreeNode("Dog3", "~/images/animals022.gif");
   RadTreeNode c14 = new RadTreeNode("Dog4", "~/images/animals023.gif");
    p1.Nodes.Add(c11);
    p1.Nodes.Add(c12);
    p1.Nodes.Add(c13);
    p1.Nodes.Add(c14);


   RadTreeNode p2 = new RadTreeNode("Cats", "~/images/animals012.gif");
   RadTreeNode c21 = new RadTreeNode("Cat1", "~/images/animals015.gif");
   RadTreeNode c22 = new RadTreeNode("Cat2", "~/images/animals017.gif");
   RadTreeNode c23 = new RadTreeNode("Cat3", "~/images/animals018.gif");
    p2.Nodes.Add(c21);
    p2.Nodes.Add(c22);
    p2.Nodes.Add(c23);

   RadTreeNode p3 = new RadTreeNode("New Node", "Node Template");
    p3.NodeTemplate = new ButtonTemplate();

    root.Nodes.Add(p1);
    root.Nodes.Add(p2);
    root.Nodes.Add(p3);
     RadTreeView2.Nodes.Add(root);

  //Expand all nodes
    RadTreeView2.ExpandAllNodes();
}
```

VB Example:

```
Class ImageTemplate
Implements ITemplate

 Public Sub InstantiateIn(ByVal container As Control)
   Dim button1 As ImageButton = New ImageButton
    button1.ID = "ImageButton1"
     AddHandler button1.DataBinding, AddressOf button1_DataBinding
      container.Controls.Add(button1)
 End Sub

 Private Sub button1_DataBinding(ByVal sender As Object, ByVal e As EventArgs)
   Dim target As ImageButton = CType(sender, ImageButton)
   Dim node As RadTreeNode = CType(target.BindingContainer, RadTreeNode)
   Dim nodeValue As String = CType(DataBinder.Eval(node, "Value"), String)
    target.ImageUrl = node.Value
 End Sub
End Class

Class ButtonTemplate
Implements ITemplate
```

```
 Public Sub  InstantiateIn(ByVal  container As  Control)
   Dim  button1 As Button = New  Button
    button1.ID = "Button1"
     AddHandler  button1.DataBinding, AddressOf  button1_DataBinding
      container.Controls.Add(button1)
 End Sub

 Private Sub  button1_DataBinding(ByVal  sender As Object, ByVal e As  EventArgs)
   Dim  target As  Button  =  CType(sender,  Button)
   Dim  node As  RadTreeNode  =  CType(target.BindingContainer,  RadTreeNode)
   Dim  nodeValue As String  =  CType(DataBinder.Eval(node,  "Value"), String)
    target.Text  =  node.Value
 End Sub
End  Class

Protected Sub  Page_Load(ByVal  sender As Object, ByVal e As  EventArgs)
  RadTreeView2.NodeTemplate  = New  ImageTemplate
  AddNodes
End Sub

Private Sub  AddNodes()
   RadTreeView2.Nodes.Clear
 Dim  root As  RadTreeNode  = New   RadTreeNode("Animals",   "~/images/animals006.gif")
 Dim  p1 As  RadTreeNode  = New   RadTreeNode("Dogs",   "~/images/animals019.gif")
 Dim  c11 As  RadTreeNode  = New   RadTreeNode("Dog1",   "~/images/animals020.gif")
 Dim  c12 As  RadTreeNode  = New   RadTreeNode("Dog2",   "~/images/animals021.gif")
 Dim  c13 As  RadTreeNode  = New   RadTreeNode("Dog3",   "~/images/animals022.gif")
 Dim  c14 As  RadTreeNode  = New   RadTreeNode("Dog4",   "~/images/animals023.gif")
  p1.Nodes.Add(c11)
  p1.Nodes.Add(c12)
  p1.Nodes.Add(c13)
  p1.Nodes.Add(c14)
 Dim  p2 As  RadTreeNode  = New   RadTreeNode("Cats",   "~/images/animals012.gif")
 Dim  c21 As  RadTreeNode  = New   RadTreeNode("Cat1",   "~/images/animals015.gif")
 Dim  c22 As  RadTreeNode  = New   RadTreeNode("Cat2",   "~/images/animals017.gif")
 Dim  c23 As  RadTreeNode  = New   RadTreeNode("Cat3",   "~/images/animals018.gif")
  p2.Nodes.Add(c21)
  p2.Nodes.Add(c22)
  p2.Nodes.Add(c23)
 Dim  p3 As  RadTreeNode  = New  RadTreeNode("New  Node",  "Node  Template")
  p3.NodeTemplate  = New  ButtonTemplate
  root.Nodes.Add(p1)
  root.Nodes.Add(p2)
  root.Nodes.Add(p3)
   RadTreeView2.Nodes.Add(root)
   RadTreeView2.ExpandAllNodes
End Sub
```

Lab: Applying Templates at Runtime

Templates can be extremely useful when you want to accomplish a task that is not behavior inherent in the control you are using, such as displaying a picture in an imagebutton. In this lab, we'll create and add two different templates to the controls in our RadTreeView.

1. Add a new web page (aspx) to your project, and set it as the start page.

2. Add a RadTreeView control to your design surface.

3. Copy the code from the preceding section Applying Templates at Runtime, and add it to the code-behind page. **Make sure** that the two classes which descend from ITemplate (ImageTemplate and

ButtonTemplate) are defined **outside** of the main form class definition.

4. Run your project. Verify that the images and text displayed on the nodes of your RadTreeView control are what you expect them to be.

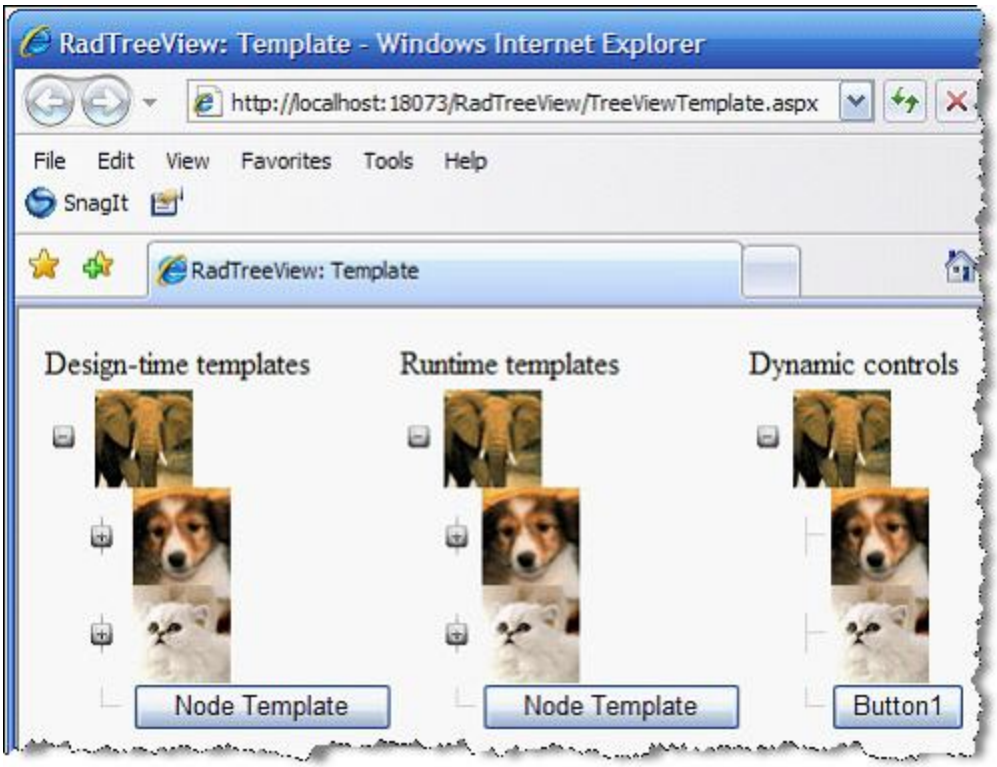
Adding Controls Dynamically

You can also add other controls to the RadTreeNode's controls collection at runtime. The following code segment shows how to add ImageButton and Button controls to different nodes in a RadTreeView.

```
C# Example:
private void AddControlNodes()
{
  //Clear nodes
    RadTreeView3.Nodes.Clear();

   RadTreeNode root = new RadTreeNode("Animals");
   ImageButton imgButton1 = new ImageButton();
   imgButton1.ImageUrl = "~/images/animals006.gif";
    root.Controls.Add(imgButton1);

   RadTreeNode p1 = new RadTreeNode("Dogs");
   ImageButton imgButton2 = new ImageButton();
   imgButton2.ImageUrl = "~/images/animals019.gif";
    p1.Controls.Add(imgButton2);

   RadTreeNode p2 = new RadTreeNode("Cats");
   ImageButton imgButton3 = new ImageButton();
   imgButton3.ImageUrl = "~/images/animals012.gif";
    p2.Controls.Add(imgButton3);

   RadTreeNode p3 = new RadTreeNode("New Node");
   Button button1 = new Button();
   button1.Text = "Button1";
    p3.Controls.Add(button1);

    root.Nodes.Add(p1);
    root.Nodes.Add(p2);
    root.Nodes.Add(p3);
    RadTreeView3.Nodes.Add(root);

  //Expand all nodes
    RadTreeView3.ExpandAllNodes();
}

VB Example:
Private Sub AddControlNodes()
   RadTreeView3.Nodes.Clear
 Dim root As RadTreeNode = New RadTreeNode("Animals")
 Dim imgButton1 As ImageButton = New ImageButton
  imgButton1.ImageUrl = "~/images/animals006.gif"
   root.Controls.Add(imgButton1)
 Dim p1 As RadTreeNode = New RadTreeNode("Dogs")
 Dim imgButton2 As ImageButton = New ImageButton
  imgButton2.ImageUrl = "~/images/animals019.gif"
   p1.Controls.Add(imgButton2)
 Dim p2 As RadTreeNode = New RadTreeNode("Cats")
 Dim imgButton3 As ImageButton = New ImageButton
  imgButton3.ImageUrl = "~/images/animals012.gif"
   p2.Controls.Add(imgButton3)
```

```
 Dim p3 As RadTreeNode = New RadTreeNode("New Node")
 Dim button1 As Button = New Button
  button1.Text = "Button1"
   p3.Controls.Add(button1)
   root.Nodes.Add(p1)
   root.Nodes.Add(p2)
   root.Nodes.Add(p3)
    RadTreeView3.Nodes.Add(root)
    RadTreeView3.ExpandAllNodes
End Sub
```

The effect using templates or programmatically added controls is the same:



Controls added to nodes

Lab: Adding nodes programmatically

In this lab, we will implement the dynamic control addition discussed in the previous section.

1. Add a new web page to your project. Set it as the start page.
2. Add a RadTreeView control to your form.
3. In the Page_Load event of the form, call the AddControlNodes method detailed in the previous section.
4. Add the AddControlNodes method.
5. Run your project. Verify that the buttons appear with the pictures and text as you assigned them.

Adding Attributes at Runtime

Adding Attributes Programmatically

You can programmatically add attributes to a RadTreeNode by accessing the Attributes collection. The following code snippet adds an attribute called "CustomData" to each created node:

```
C# Example:
private void AddNodesAndAttributes()
{
  //Clear nodes
    RadTreeView2.Nodes.Clear();
   RadTreeNode root = new RadTreeNode("Root", "-1");
    root.Attributes.Add("CustomData", "RootNode");
   RadTreeNode p1 = new RadTreeNode("Parent1", "1");
    p1.Attributes.Add("CustomData", "ParentNode1");

   RadTreeNode c11 = new RadTreeNode("Child1", "1.1");
    c11.Attributes.Add("CustomData", "Parent1ChildNode1");
   RadTreeNode c12 = new RadTreeNode("Child2", "1.2");
    c12.Attributes.Add("CustomData", "Parent1ChildNode2");
   RadTreeNode c13 = new RadTreeNode("Child3", "1.3");
    c13.Attributes.Add("CustomData", "Parent1ChildNode3");
    p1.Nodes.Add(c11);
    p1.Nodes.Add(c12);
    p1.Nodes.Add(c13);

   RadTreeNode p2 = new RadTreeNode("Parent2", "2");
    p2.Attributes.Add("CustomData", "ParentNode2");
   RadTreeNode c21 = new RadTreeNode("Child1", "2.1");
    c21.Attributes.Add("CustomData", "Parent2ChildNode1");
   RadTreeNode c22 = new RadTreeNode("Child2", "2.2");
    c22.Attributes.Add("CustomData", "Parent2ChildNode1");
    p2.Nodes.Add(c21);
    p2.Nodes.Add(c22);

    root.Nodes.Add(p1);
    root.Nodes.Add(p2);
     RadTreeView2.Nodes.Add(root);

  //Expand all nodes
    RadTreeView2.ExpandAllNodes();
}

VB Example:
Private Sub AddNodesAndAttributes()
  RadTreeView2.Nodes.Clear
 Dim root As RadTreeNode = New RadTreeNode("Root", "-1")
  root.Attributes.Add("CustomData", "RootNode")
 Dim p1 As RadTreeNode = New RadTreeNode("Parent1", "1")
  p1.Attributes.Add("CustomData", "ParentNode1")
 Dim c11 As RadTreeNode = New RadTreeNode("Child1", "1.1")
  c11.Attributes.Add("CustomData", "Parent1ChildNode1")
 Dim c12 As RadTreeNode = New RadTreeNode("Child2", "1.2")
  c12.Attributes.Add("CustomData", "Parent1ChildNode2")
 Dim c13 As RadTreeNode = New RadTreeNode("Child3", "1.3")
  c13.Attributes.Add("CustomData", "Parent1ChildNode3")
 p1.Nodes.Add(c11)
 p1.Nodes.Add(c12)
 p1.Nodes.Add(c13)
 Dim p2 As RadTreeNode = New RadTreeNode("Parent2", "2")
```

```
  p2.Attributes.Add("CustomData",  "ParentNode2")
 Dim c21 As RadTreeNode = New RadTreeNode("Child1", "2.1")
  c21.Attributes.Add("CustomData",  "Parent2ChildNode1")
 Dim c22 As RadTreeNode = New RadTreeNode("Child2", "2.2")
  c22.Attributes.Add("CustomData",  "Parent2ChildNode1")
  p2.Nodes.Add(c21)
  p2.Nodes.Add(c22)
  root.Nodes.Add(p1)
  root.Nodes.Add(p2)
   RadTreeView2.Nodes.Add(root)
   RadTreeView2.ExpandAllNodes
End Sub
```

Adding Attributes from XML

You can use attributes to attach arbitrary data to nodes from within the XML for a treeview. Add attributes to the Node tag that are not already used by the RadTreeView itself. In the example below the *CustomData* doesn't exist for a node so it is considered to be an attribute:

```
<Tree>
   <Node Text="Root" Expanded="True" Value="-1" CustomData="RootNode" >
      <Node Text="Parent1" Expanded="True" Value="1" CustomData="ParentNode1">
        <Node Text="Child1" Value="1.1" CustomData="Parent1ChildNode1"/>
        <Node Text="Child2" Value="1.2" CustomData="Parent1ChildNode2"/>
        <Node Text="Child3" Value="1.3" CustomData="Parent1ChildNode3"/>
    </Node>
     <Node Text="Parent2" Expanded="True" Value="2" CustomData="ParentNode2">
        <Node Text="Child1" Value="2.1" CustomData="Parent2ChildNode1"/>
        <Node Text="Child2" Value="2.2" CustomData="Parent2ChildNode2"/>
    </Node>
  </Node>
</Tree>
```

You can declaratively access the attribute by binding to it:

```
<radT:RadTreeView ID="RadTreeView3" runat="server"  Skin="Inox"
                  ContentFile="TreeAttributes.xml">
    <NodeTemplate>
      <%# DataBinder.Eval(Container, "Attributes['CustomData']")%>
    </NodeTemplate>
</radT:RadTreeView>
```

Lab: Adding Custom Attributes

Custom attributes are particularly useful when you'd like to keep certain information associated with a node above and beyond the properties normally associated with a node. In this lab, we'll use a "ClickCount" property to keep track of the number of times a node has been clicked.

1. Add a new web page (aspx) to your project and set it as the start page.

2. Add a RadTreeView control and an asp label control to your page. Initialize the label text to read "No nodes have been clicked"

3. Add an XML file to you project named "ClickCount.xml", and add the following text to it:

```
<?xml version="1.0" encoding="utf-8" ?>
<Tree>
   <Node Text="Root" Expanded="True" Value="-1" CustomData="RootNode" >
      <Node Text="Parent1" Expanded="True" Value="1" CustomData="ParentNode1">
        <Node Text="Child1" Value="1.1" CustomData="Parent1ChildNode1" ClickCount="0"/>
        <Node Text="Child2" Value="1.2" CustomData="Parent1ChildNode2" ClickCount="0"/>
```

```
        <Node Text="Child3" Value="1.3" CustomData="Parent1ChildNode3" ClickCount="0"/>
    </Node>
      <Node Text="Parent2" Expanded="True" Value="2" CustomData="ParentNode2">
        <Node Text="Child1" Value="2.1" CustomData="Parent2ChildNode1" ClickCount="0"/>
        <Node Text="Child2" Value="2.2" CustomData="Parent2ChildNode2" ClickCount="0"/>
    </Node>
  </Node>
</Tree>
```

4. Add the following code to the Page_Load event:

```
C# Example:
if (!IsPostBack)
{
    RadTreeView1.LoadContentFile("ClickCount.xml");
}

VB Example:
If Not IsPostBack Then
     RadTreeView1.LoadContentFile("ClickCount.xml")
End If
```

5. In the RadTreeView property sheet, double-click in the "NodeClick" event to create the method prototype. The NodeClick event should read as follows:

```
C# Example:
protected void RadTreeView1_NodeClick(object o,
    Telerik.WebControls.RadTreeNodeEventArgs e)
{
   RadTreeNode treeNode = (RadTreeNode)e.NodeClicked;
  if (!(treeNode.Attributes["ClickCount"] == null))
  {
    int currentCount = Int32.Parse(treeNode.Attributes["ClickCount"]) + 1;
    string customData = treeNode.Attributes["CustomData"];
     Label1.Text = String.Format(
      "The node named {0} has custom data {1} and has been clicked {2} times.",
          treeNode.Text, customData, currentCount.ToString());
     treeNode.Attributes["ClickCount"] = currentCount.ToString();
  }
}

VB Example:
protected Sub RadTreeView1_NodeClick(ByVal o As Object, _
  ByVal e As   Telerik.WebControls.RadTreeNodeEventArgs)
 Dim treeNode As RadTreeNode = CType(e.NodeClicked, RadTreeNode)
 If Not (treeNode.Attributes("ClickCount") is Nothing) Then
   Dim currentCount As Integer = Int32.Parse(treeNode.Attributes("ClickCount")) + 1
   Dim customData As String = treeNode.Attributes("CustomData")
    Label1.Text = String.Format(_
  "The node named {0} has custom data {1} and has been clicked {2} times.",_
     treeNode.Text, customData, currentCount.ToString)
     treeNode.Attributes("ClickCount") = currentCount.ToString
 End If
End Sub
```

This code will increment the custom data for the node's ClickCount, and display the current count each time the node is clicked.

6. Make sure that the "AutoPostback" attribute of the RadTreeView is set true.

7. Run your project. Click on various nodes. Notice that the count is updated and the custom data is displayed each time you click on a leaf node.

Data Binding

RadTreeView supports data binding and can be populated from any IListSource such as the DataSet, DataTable or DataView or IEnumerable such as arrays, ArrayList or ObjectDataSource. The ability for RadTreeView to create a hierarchy based off the data varies on the type of datasource:

- RadTreeView will automatically create the hierarchy when bound to XmlDataSource or SiteMapDataSource.
- When bound to data sources such as DataTable, DataView, SqlDataSource, AccessDataSource the *DataFieldID* and *DataFieldParentID* properties must be set to establish the hierarchy.
- RadTreeView can create only root nodes when bound to any class that implements IEnumerable or an ObjectDataSource. When binding to IEnumerables you need to set the DataSource property and call the DataBind() method.

Here are some of the RadTreeView properties specific to Data binding:

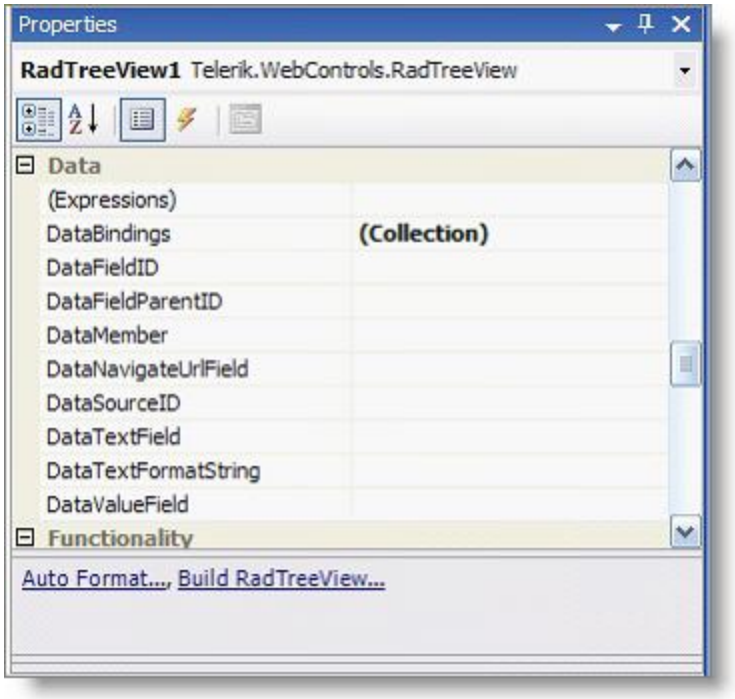
DataTextField: Specifies the field in the data source that provides the user-readable *Text* for the tree nodes.

DataValueField: Specifies the field in the data source that provides the underlying *Value* for the tree nodes.

DataNavigateUrlField: Specifies the field in the data source that provides the *NavigateUrl* for the tree nodes.

DataFieldID: Specifies the field in the data source that holds the *ID* value.

DataFieldParentID: Specifies the field in the data source that holds the *ParentID* and is used to establish hierarchy.



RadTreeView Data Properties

ArrayList

To bind a RadTreeView to an ArrayList you need to set the DataSource property to an ArrayList instance and call the DataBind() method. When bound to an ArrayList only the root nodes are created as there is no hierarchy.

```
C# Example:
private ArrayList GenerateArrayList()
{
   ArrayList cityList = new ArrayList();
   cityList.Add("European cities");
   cityList.Add("North American cities");
   cityList.Add("South American cities");
   cityList.Add("Asian cities");
   cityList.Add("African cities");

  return cityList;
}

protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
      RadTreeView1.DataSource = GenerateArrayList();
     RadTreeView1.DataBind();
  }
}

VB Example:
Private Function GenerateArrayList() As ArrayList
 Dim cityList As ArrayList = New ArrayList
  cityList.Add("European cities")
  cityList.Add("North American cities")
```

```
 cityList.Add("South American cities")
 cityList.Add("Asian cities")
 cityList.Add("African cities")
 Return cityList
End Function

protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
    RadTreeView1.DataSource = GenerateArrayList
    RadTreeView1.DataBind
 End If
End Sub
```

SqlDataSource or AccessDataSource

To bind a RadTreeView to a SqlDataSource you need to set the DataSourceID to the *ID* property of the SqlDataSource control. To establish a hierarchy, you need to set the DataFieldID and the DataFieldParentID fields to the specific data source columns as shown below in code behind.

The aspx markup sets up the SqlDataSource and references it in the RadTreeView *DataSourceID* property:

```
<radT:RadTreeView ID="RadTreeView1" runat="server"
   DataSourceID="SqlDataSource1">
</radT:RadTreeView>
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
    ConnectionString="<%$ ConnectionStrings:CS1 %>"
   SelectCommand="SELECT * FROM [Table1]"
    ProviderName="<%$ ConnectionStrings:CS1.ProviderName %>">
</asp:SqlDataSource>
```

In this example the code behind determines how the database fields are used. This could also be done entirely through setting properties on the RadTreeView.

```
protected void Page_Load(object sender, EventArgs e)
{
  if (!IsPostBack)
  {
     RadTreeView1.DataFieldID = "ID";
     RadTreeView1.DataFieldParentID = "ParentID";
     RadTreeView1.DataTextField = "Text";
     RadTreeView1.DataNavigateUrlField = "URL";
     RadTreeView1.DataBind();

     RadTreeView1.ExpandAllNodes();
  }
}
```

ObjectDataSource

To bind a RadTreeView to an ObjectDataSource you need to set the *DataSourceID* to the ID of the ObjectDataSource control. The ObjectDataSource *TypeName* and *SelectMethod* should be set to the business object that needs to be bound.

Note: When bound to an ObjectDataSource the treeview cannot create hierarchy.

Note: See the MSDN for more information about ObjectDataSource: http://msdn2.microsoft.com/en-us/library/9a4kyhcx.aspx.

The aspx markup defines ObjectDataSource and the RadTreeView DataSourceID points to the

ObjectDataSource. Notice the ObjectDataSource *TypeName* and *SelectMethod* properties.

```
<radT:RadTreeView  ID="RadTreeView1"  runat="server"
    DataSourceID="ObjectDataSource1">
</radT:RadTreeView>

<asp:ObjectDataSource  ID="ObjectDataSource1"  runat="server"
   SelectMethod="GetCities"
   TypeName="City">
</asp:ObjectDataSource>
```

The GetCities() method returns the generic list of City objects referred to in ObjectDataSource *SelectMethod*. The City object defined below is referred to in the ObjectDataSource *TypeName* property.

```
C# Example:
public class City
{
  private String m_name;
  private String m_url;

  public City()
  {
  }

  public String Text
  {
    get { return m_name; }
    set { m_name = value; }
  }

  public String Url
  {
    get { return m_url; }
    set { m_url = value; }
  }

  public static List<City> GetCities()
  {
     List<City> cityList = new List<City>();

    City city1 = new City();
    city1.Text = "San  Francisco";
    city1.Url = "http://en.wikipedia.org/wiki/San_Francisco";
     cityList.Add(city1);

    City city2 = new City();
    city2.Text = "Boston";
    city2.Url = "http://en.wikipedia.org/wiki/Boston";
     cityList.Add(city2);

    City city3 = new City();
    city3.Text = "Seattle";
    city3.Url = "http://en.wikipedia.org/wiki/Seattle";
     cityList.Add(city3);

    City city4 = new City();
    city4.Text = "Toronto";
    city4.Url = "http://en.wikipedia.org/wiki/Toronto";
     cityList.Add(city4);

    return cityList;
```

```
  }
}

VB  Example:
Public class City
 Private m_name As String
 Private m_url As String

 Public Sub New()
 End Sub

 Public Property Text() As String
   Get
     Return m_name
   End Get
   Set
      m_name = value
   End Set
 End Property

 Public Property Url() As String
   Get
     Return m_url
   End Get
   Set
      m_url = value
   End Set
 End Property

Public Shared Function GetCities() As ListOf(City)
  Dim cityList As ListOf(City)
  Dim city1 As City = New City
   city1.Text = "San Francisco"
    city1.Url = "http://en.wikipedia.org/wiki/San_Francisco"
   cityList.Add(city1)
  Dim city2 As City = New City
   city2.Text = "Boston"
    city2.Url = "http://en.wikipedia.org/wiki/Boston"
   cityList.Add(city2)
  Dim city3 As City = New City
   city3.Text = "Seattle"
    city3.Url = "http://en.wikipedia.org/wiki/Seattle"
   cityList.Add(city3)
  Dim city4 As City = New City
   city4.Text = "Toronto"
    city4.Url = "http://en.wikipedia.org/wiki/Toronto"
   cityList.Add(city4)

  Return cityList

End Function

End class
```

SiteMapDataSource

To bind a RadTreeView to a SiteMapDataSource you need to set the *DataSourceID* to the ID of the SiteMapDataSource control. When bound to a SiteMapDataSource, the treeview hierarchy is established automatically and the sitemap fields for url, title, and description are automatically bound to *NavigateUrl*, *Text*, and *ToolTip* properties respectively. The aspx markup defines the RadTreeView,

SiteMapDataSource and the relationship between the two:

```
<radT:RadTreeView   ID="RadTreeView1"   runat="server"   DataSourceID="SiteMapDataSource1">
</radT:RadTreeView>
<asp:SiteMapDataSource   ID="SiteMapDataSource1"   runat="server"   />
```

This sitemap.xml provides the url, title and descriptions:

```
<?xml  version="1.0"  encoding="utf-8"  ?>
<siteMap   xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0"   >
    <siteMapNode  url="http://www.telerik.com"  title="telerik"
       description="telerik  home  page">
       <siteMapNode  url="http://www.telerik.com/radcontrols"
       title="r.a.d.controls"   description="r.a.d.controls"   >
         <siteMapNode  url="http://www.telerik.com/radtreeview"
       title="r.a.d.treeview"   description="r.a.d.treeview   control"/>
         <siteMapNode  url="http://www.telerik.com/radgrid"
       title="r.a.d.grid"   description="r.a.d.grid   control"/>
         <siteMapNode  url="http://www.telerik.com/radeditor"
       title="r.a.d.editor"   description="r.a.d.editor   control"/>
         <siteMapNode  url="http://www.telerik.com/radcalendar"
       title="r.a.d.calendar"   description="r.a.d.calendar   control"/>
      </siteMapNode>
      <siteMapNode  url="http://www.telerik.com/radajax"
       title="r.a.d.ajax  controls"  description="r.a.d.ajax  controls"  />
   </siteMapNode>
</siteMap>
```

XmlDataSource

To bind a RadTreeView to a XmlDataSource you need to set the *DataSourceID* to the ID of the XmlDataSource control. When bound to a XmlDataSource, the hierarchy is established automatically. However, properties such as Text and NavigateUrl need to be set explicitly either declaratively or in code behind. The aspx markup defines the RadTreeView, the XMLDataSource and the relationship between the two. **Note**: XPath="/Items/Item" is set to ignore the root node.

```
<radT:RadTreeView  ID="RadTreeView1"  runat="server"
    DataSourceID="XmlDataSource1"
   DataTextField="Text"
    DataNavigateUrlField="Url">
</radT:RadTreeView>

<asp:XmlDataSource   ID="XmlDataSource1"   runat="server"
    DataFile="~/App_Data/Cities.xml"
   XPath="/Items/Item">
</asp:XmlDataSource>
```

The xml file below defines the treeview structure and defines the Text and Url attributes.

```
<?xml  version="1.0"  encoding="utf-8"  ?>
<Items  Text="">
   <Item  Text="European  cities"  Url=""   >
      <Item  Text="Sofia"  Url="http://en.wikipedia.org/wiki/Sofia"  />
      <Item  Text="Berlin"  Url="http://en.wikipedia.org/wiki/Berlin"  />
     <Item  Text="Paris"  Url="http://en.wikipedia.org/wiki/Paris"  />
   </Item>
   <Item  Text="North  American  cities"  Url="">
      <Item  Text="Boston"  Url="http://en.wikipedia.org/wiki/Boston"  />
      <Item  Text="San  Francisco"  Url="http://en.wikipedia.org/wiki/San_Francisco"  />
```

```
      <Item Text="Seattle" Url="http://en.wikipedia.org/wiki/Seattle" />
      <Item Text="Toronto" Url="http://en.wikipedia.org/wiki/Toronto" />
  </Item>
  <Item Text="South American cities" Url="">
      <Item Text="Rio de Janeiro" Url="http://en.wikipedia.org/wiki/Rio_De_Janeiro" />
     <Item Text="Buenos Aires" Url="http://en.wikipedia.org/wiki/Buenos_aires" />
  </Item>
  <Item Text="Asian cities" Url="">
      <Item Text="Tokyo" Url="http://en.wikipedia.org/wiki/Tokyo" />
      <Item Text="Seul" Url="http://en.wikipedia.org/wiki/Seul" />
      <Item Text="Beijing" Url="http://en.wikipedia.org/wiki/Beijing" />
      <Item Text="Tehran" Url="http://en.wikipedia.org/wiki/Teheran" />
  </Item>
  <Item Text="African cities" Url="">
      <Item Text="Kano" Url="http://en.wikipedia.org/wiki/Kano" />
       <Item Text="Johannesburg" Url="http://en.wikipedia.org/wiki/Johannesburg" />
     <Item Text="BeninCity" Url="http://en.wikipedia.org/wiki/Benin" />
  </Item>
</Items>
```

Lab: Connecting to a Data Source

Knowing how to add content manually is important, and sometimes it's even what the application requires; but much of the time, you'll probably display RadTreeView contents that are based on a database, and possibly even selection criteria. In this lab, we'll cover how to connect to a hierarchical database structure.

1. Add a new web page (aspx) to your project.

2. Add the following components to your design surface: a RadTreeView, a SqlDataSource, and two label and Textbox pairs. Format them in a table so that the appearance is clean, if you'd like.

3. Connect the SqlDataSource to the Northwind database. Select all the columns (*) in the Employees table.

4. Connect the RadTreeView to the SqlDataSource.

5. Set the properties of the RadTreeView as follows:

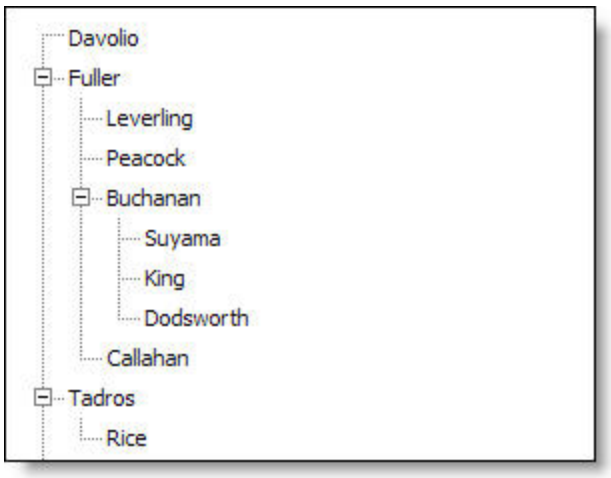
- DataFieldID: EmployeeID
- DataTextField: LastName
- DataFieldParentID: ReportsTo

The body of your markup should look something like this:

```
<body>
     <form id="form1" runat="server">
    <div>
        <table width="100%">
           <tr>
                <td style="width: 40%">
                     <radT:RadTreeView ID="RadTreeView1" runat="server"
                          DataSourceID="SqlDataSource1"    DataFieldID="EmployeeID"
                          DataFieldParentID="ReportsTo"    DataTextField="LastName">
                    </radT:RadTreeView>
               </td>
                <td style="width: 60%">
                   <table>
                       <tr>
                            <td style="width: 40%">
```

```
                                Employee ID</td>
                            <td style="width: 60%">
                                <asp:TextBox ID="TextBox1"
                                    runat="server"></asp:TextBox></td>
                        </tr>
                        <tr>
                            <td>
                                Last Name</td>
                            <td>
                                <asp:TextBox ID="TextBox2"
                                    runat="server"></asp:TextBox></td>
                        </tr>
                    </table>
                </td>
            </tr>
        </table>

    </div>
        <asp:SqlDataSource ID="SqlDataSource1" runat="server"
            ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
            SelectCommand="SELECT * FROM [Employees]"></asp:SqlDataSource>
    </form>
</body>
```

6. If you run your project at this point, you should get a simple hierarchical tree that displays the reporting structure of the Northwind Trading Company:



Northwind Reporting Structure

The Employee ID and name fields you added will be used in a later exercise (Lab: Server Side Events).

Drag and Drop

RadTreeView supports the following types of drag-and-drop:

1. Drag and drop within a single RadTreeView.

   - Single node drag-and-drop – To enable single node drag-and-drop functionality you should set DragAndDrop property to True.
   - Multiple nodes drag-and-drop - To enable dragging multiple nodes you should set DragAndDrop and MultipleSelect properties to True.

2. Drag and drop in-between nodes – To enable this feature you should set the

DragAndDropBetweenNodes property of the RadTreeView to True.

3. Drag and drop between trees - When RadTreeView detects a valid drag-and-drop operation (the selected node is dropped onto another node) it generates a NodeDrop server-side event. The event handler for NodeDrop can then perform specific action, by fetching both the source and destination nodes. You can control dragging in the following way:
   - Drag in one direction only – To enable this you should set DragAndDrop property to True and handle the OnNodeDrop for the source tree only.
   - Drag in both directions - To enable this you should set DragAndDrop property to True and handle the OnNodeDrop for both trees.
4. Drag and drop to any HTML element

The following code snippet shows how to handle drag and drop in-between RadTreeView nodes. From the markup you can see that the DragAndDrop and DragAndDropBetweenNodes properties are set to True and the OnNodeDrop event is handled with a server-side event handler:

```
<radt:radtreeview  id="RadTreeView1"  runat="server"
   AutoPostBack="True"
   DragAndDrop="True"
    OnNodeDrop="RadTreeView1_NodeDrop"
  DragAndDropBetweenNodes="True">
</radt:radtreeview>
```

The code behind handles the NodeDrop event. The DragPosition property is used to determine the relationship between the dropped source node and the destination node its being dropped on.

```
C# Example:
protected void  RadTreeView1_NodeDrop(object o, RadTreeNodeEventArgs e)
{
   RadTreeNode sourceNode = e.SourceDragNode;
   RadTreeNode destNode = e.DestDragNode;
    RadTreeViewDropPosition dropPosition = e.DropPosition;

  if (destNode != null)
  {
    if  (sourceNode.TreeView.SelectedNodes.Count  <= 1)
    {
        PerformDragAndDrop(dropPosition, sourceNode, destNode);
    }
    else if  (sourceNode.TreeView.SelectedNodes.Count  > 1)
    {
      //When multiple nodes are selected perform the drop on
      //each of the selected nodes
      foreach (RadTreeNode node in  sourceNode.TreeView.SelectedNodes)
      {
          PerformDragAndDrop(dropPosition, node, destNode);
      }
    }

     destNode.Expanded = true;
       sourceNode.TreeView.ClearSelectedNodes();
  }
}

private void  PerformDragAndDrop(RadTreeViewDropPosition  dropPosition,
```

```
RadTreeNode sourceNode, RadTreeNode destNode)
{
  if (!CheckNodeExists(sourceNode, destNode))
  {
    switch (dropPosition)
    {
      case RadTreeViewDropPosition.Over: // child
        if (!sourceNode.IsAncestorOf(destNode))
        {
            sourceNode.Owner.Nodes.Remove(sourceNode);
           destNode.Nodes.Add(sourceNode);
        }
        break;

      case RadTreeViewDropPosition.Above: // sibling - above
          sourceNode.Owner.Nodes.Remove(sourceNode);
         destNode.InsertBefore(sourceNode);
        break;

      case RadTreeViewDropPosition.Below: // sibling - below
        if (!sourceNode.IsAncestorOf(destNode))
        {
            sourceNode.Owner.Nodes.Remove(sourceNode);
           destNode.InsertAfter(sourceNode);
        }
        break;
    }
  }
}

//Check if the source and the destination is under the same parent
//This check is handy when you want to allow drag and drop
//only when the source and destination have different parents.
private bool CheckNodeExists(RadTreeNode sourceNode, RadTreeNode destNode)
{
   RadTreeNode destParentNode = destNode.Parent;
  if (destParentNode != null)
  {
    foreach (RadTreeNode childnode in destParentNode.Nodes)
    {
      if (childnode.Value == sourceNode.Value)
        return true;
    }
  }
  return false;
}

VB Example:
Protected Sub RadTreeView1_NodeDrop(ByVal o As Object, ByVal e As RadTreeNodeEventArgs)
 Dim sourceNode As RadTreeNode = e.SourceDragNode
 Dim destNode As RadTreeNode = e.DestDragNode
 Dim dropPosition As RadTreeViewDropPosition = e.DropPosition
 If Not (destNode Is Nothing) Then
   If sourceNode.TreeView.SelectedNodes.Count <= 1 Then
       PerformDragAndDrop(dropPosition, sourceNode, destNode)
   Else
     If sourceNode.TreeView.SelectedNodes.Count > 1 Then
       For Each node As RadTreeNode In sourceNode.TreeView.SelectedNodes
           PerformDragAndDrop(dropPosition, node, destNode)
       Next
     End If
```

```
    End If
     destNode.Expanded = True
      sourceNode.TreeView.ClearSelectedNodes
 End If
End Sub

Private Sub  PerformDragAndDrop(ByVal dropPosition As  RadTreeViewDropPosition,_
 ByVal sourceNode As  RadTreeNode, ByVal destNode As  RadTreeNode)
 If Not  CheckNodeExists(sourceNode,  destNode) Then
   Select  dropPosition
   Case  RadTreeViewDropPosition.Over
     If Not  sourceNode.IsAncestorOf(destNode) Then
         sourceNode.Owner.Nodes.Remove(sourceNode)
        destNode.Nodes.Add(sourceNode)
     End If
     ' break
   Case  RadTreeViewDropPosition.Above
       sourceNode.Owner.Nodes.Remove(sourceNode)
      destNode.InsertBefore(sourceNode)
     ' break
   Case  RadTreeViewDropPosition.Below
     If Not  sourceNode.IsAncestorOf(destNode) Then
         sourceNode.Owner.Nodes.Remove(sourceNode)
        destNode.InsertAfter(sourceNode)
     End If
     ' break
   End Select
 End If
End Sub

Private Function  CheckNodeExists(ByVal sourceNode As RadTreeNode, _
 ByVal destNode As  RadTreeNode) As Boolean
 Dim  destParentNode As  RadTreeNode  =  destNode.Parent
 If Not  (destParentNode  Is Nothing) Then
   For Each childnode As  RadTreeNode In  destParentNode.Nodes
     If  childnode.Value  =  sourceNode.Value Then
       Return True
     End If
   Next
 End If
 Return False
End Function
```

Lab: Drag and Drop

With the maturation of browser technology and client-side scripting, one of the powerful features that has crossed over from the Win Forms world is the ability to drag and drop browser objects. In this lab, we'll see how relatively simple it is to implement that feature using the RadTreeView component.

1. Add a new web page (aspx) to your project.

2. Add a RadTreeView component to the design surface.

3. Using the steps discussed in previous labs (eg, Lab: Defining Treeview at RunTime). connect your RadTreeView content to the "ClickCount .xml" content file.

4. Set the DragAndDrop property of your RadTreeView component to True.

5. Run your project. Select any node by click-and-hold with your mouse, and scroll the mouse away from the treeview. Notice that the element you clicked on gets a shadow copy trailing along after the

mouse cursor. You can drop the element anywhere you like (but since we haven't handled the node drop event yet, nothing spectacular happens).

Mouse during Drag and Drop

6. Close your browser.

7. In the event handler section of your RadTreeView's property sheet, double-click the NodeDrop event to create the method prototype.

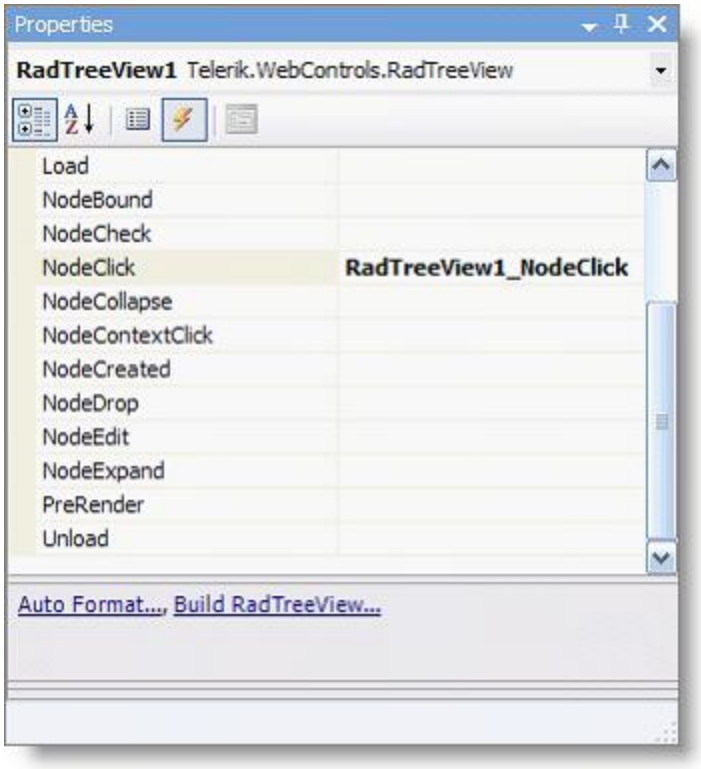
8. Copy the three methods discussed in the previous section into the code behind file (overwriting the prototype NodeDrop method - we used that as a convenient shortcut to wire the event handler to the component).

9. Finally, set the "DragAndDropBetweenNodes" property to true. This enables you to place a node element between two other node elements; when you begin manipulating the node elements in the next step, you will notice that when the mouse cursor is placed between to adjacent siblings, a horizontal line will appear on the screen marking the projected placement position.

10. Run your project again. Verify that you are now able to pick up node elements and move them around, adding them as child elements of other nodes, promoting them as siblings of their parents,and in short, being able to manipulate the node elements however you wish using the simple point-and-click mouse driven interface.

Server Side Events

RadTreeView provides API for handling server-side postback events as well as client-side JavaScript events.



RadTreeView Events

NodeClick: RadTreeView NodeClick event fires when you click on a RadTreeNode with *PostBack* property set to true. Alternatively you can set the RadTreeView's *AutoPostBack* property true to force postback for all tree nodes.

```
<radt:radtreeview  id="RadTreeView1"  runat="server"
    OnNodeClick="RadTreeView1_NodeClick"
   AutoPostBack="True">
</radt:radtreeview>
```

NodeExpand: The NodeExpand event fires when you click on the "+" icon. To fire the event you should set the *ExpandMode* of each RadTreeNode. There are three ExpandModes:

- ExpandMode.**ClientSide** - nodes are already added to the treeview when the page is displayed. This is the default ExpandMode.
- ExpandMode.**ServerSide** - nodes are added upon clicking the "+" icon. RadTreeNodes are added on the server-side and therefore a postback is required.
- ExpandMode.**ServerSideCallBack** - nodes are added upon clicking the "+" icon. Tree-Nodes are added on the client-side and therefore no postback is required. You should take into account that nodes added on the client-side are not persisted on the server. Therefore, these nodes cannot take part in any server-side events. In other words you cannot access these nodes and their attributes on the server-side.

NodeCollapse: The NodeCollapse event is fired upon collapsing RadTreeNodes (clicking the "-" icon in front of the nodes).

NodeEdit event is fired when you edit the text of a RadTreeNode and before you can edit a node's text AllowNodeEditing property on the RadTreeView should be set to True and also the node's EditEnabled property should be set to True.

The following code snippet shows how to change the edited node's text to the new text and disable editing.

```
C# Example:
protected void RadTreeView1_NodeEdit(object o, RadTreeNodeEventArgs e)
{
   if (e.NewText.Length > 0)
  {
     e.NodeEdited.Text = e.NewText;
      e.NodeEdited.EditEnabled = false;
  }
}

VB Example:
Protected Sub RadTreeView1_NodeEdit(ByVal o As Object, _
  ByVal e As RadTreeNodeEventArgs)
 If e.NewText.Length > 0 Then
    e.NodeEdited.Text = e.NewText
    e.NodeEdited.EditEnabled = False
 End If
End Sub
```

NodeDrop event is fired when drag-and-drop functionality is enabled and a node is dragged and dropped. To enable drag-and-drop you need to set DragAndDrop property to True.

Lab: Server Side Events

A lab session on the RadTreeView component wouldn't be complete without a brief look at responding to events. In this lab, we'll take up where we left off in the lab on connecting to a data source, and extend the example to read and display employee information based on the node that was clicked.

1. Set the AutoPostback property of the RadTreeView control to true.

2. Set the DataValueField property to point to the EmployeeID field.

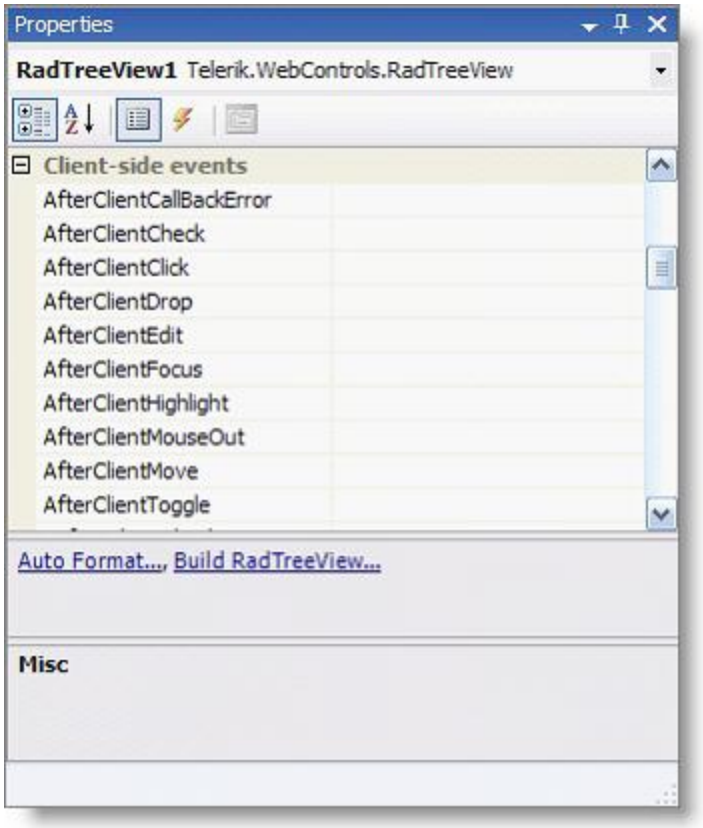
3. On the property sheet of the RadTreeView, double-click the NodeClick event to create the method prototype.

4. Add the following code to the NodeClick event handler:

```
C# Example:
protected void RadTreeView1_NodeClick(object o,
    Telerik.WebControls.RadTreeNodeEventArgs e)
{
     TextBox1.Text = (string)e.NodeClicked.Value;
     TextBox2.Text = (string)e.NodeClicked.Text;
}

VB Example:
Protected Sub RadTreeView1_NodeClick(ByVal o As Object, _
  ByVal e As Telerik.WebControls.RadTreeNodeEventArgs)
  TextBox1.Text = CType(e.NodeClicked.Value, String)
  TextBox2.Text = CType(e.NodeClicked.Text, String)
End Sub
```

2.7.4 Client Scripting with RadTreeeView

Client Events



RadTreeView events in the Properties Window

BeforeClientClick is fired prior to node clicking - when end users use the mouse to click or use keyboard to navigate to a particular node. The event is called just prior to server-side postback or URL redirection. The postback can be cancelled by returning false in the BeforeClientClick event handler.

AfterClientClick is fired after a node has been clicked - when end users use the mouse or the keyboard to click a certain node. The event is also called just prior to server-side postback or URL redirection.

BeforeClientDoubleClick is fired upon double-clicking on a node. The postback can be cancelled by returning false in the BeforeClientDoubleClick event handler.

```
<radt:radtreeview   id="RadTreeView1"   runat="server"
           OnNodeClick="RadTreeView1_NodeClick"
          AutoPostBack="True"
           BeforeClientClick="BeforeClickHandler"
           AfterClientClick="AfterClickHandler">
</radt:radtreeview>

<script   language="javascript">
function   BeforeClickHandler(node)
{
   alert("You clicked " +   node.Text);
   return false;
}
function   AfterClickHandler(node)
```

```
{
  alert("You clicked " +  node.Text);
}
</script>
```

BeforeClientContextClick is called when the user clicks on a context menu item. The event receives three parameters: the instance of the node, the text of the context menu item that is selected and the ID of the ContextMenuItem. The postback can be cancelled by returning false in the BeforeClientContextClick event handler. The following event handler displays a confirmation when users select delete on the context menu item and enables editing the node when edit context menu item is chosen.

```
<script  language="javascript">
function  BeforeClientContextHandler(node,  itemText,  itemID)
{
   if (itemID == "Delete")
   {
        var name=confirm("Are you sure to Delete " + node.Text)
       if (name==true)
          return true;
       else
         return false;
   }

   if (itemID == "Edit")
   {
       node.EditEnabled = true;
      return false;
   }
   else
     return true;
}
</script>
```

BeforeClientContextMenu is called just prior to opening RadTreeView's context menu. This function receives two parameters - the instance of the current node and the event object of the browser. This event can be used disable context menu for certain nodes in the RadTreeView by returning false.

```
function  BeforeClientContextMenuHandler  (node,  e)
{
   alert("Context  Menu  clicked");
   if  (node.Category  ==  "RootNode")
      return false;
}
```

BeforeClientDrag, BeforeClientDrop, AfterClientDrop, and **AfterClientMove** are related to client-side drag & drop and are available when RadTreeView DragAndDrop property is set to True.

BeforeClientDrag accepts a single parameter - the instance of the node being dragged.

BeforeClientDrop accepts three parameters sourceNode, destNode and events. The first parameter sourceNode is the instance of the node being dragged. The second parameter destNode is set to the instance of the node the source node is being dropped to. It the node is dropped on a Html element this parameter will be null. The third parameter events is set to the instance of the browser events.

AfterClientDrop fires immediately after a tree node has been dropped.

AfterClientMove is executed each time the user moves the mouse while dragging the node. The event handler receives a single parameter - the client-side events instance.

2.7.5 Summary

This chapter covered practical aspects of using the RadTreeView control. You should be able to link your RadTreeView control to a number of different data sources, and display the data in a number of different ways, whether that is simply setting the text value of the node, or putting data into a templated control (text, images, or any other controls as dictated by the requirements of your application). You also discovered how to create context menus for your RadTreeView elements, and several methods for how to associate different context menus with different nodes. You also learned how to use node attributes to store custom data, how to deal with issues surrounding drag-and-drop, and how to hook into and respond to the RadTreeView events exposed on the client side.

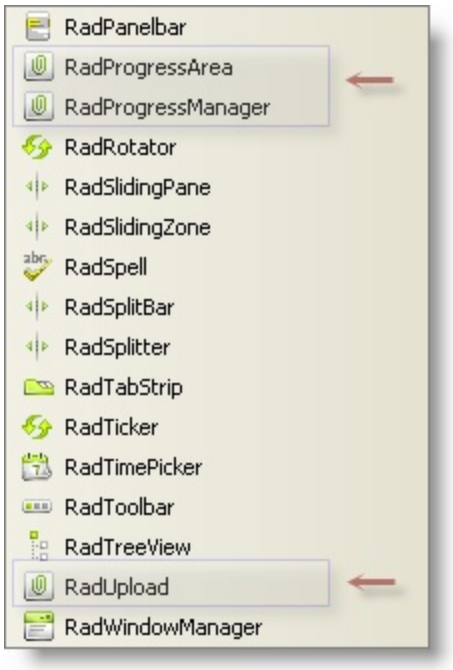
2.8 RadUpload

2.8.1 Getting Started

HtmlInputFile control in ASP.NET allows file upload from a browser to the Web server. While uploading files the ASP.NET runtime caches the file content in memory before saving the contents on the Web server. So, applications that use large file uploads end up using lots of server resources which degrades performance and makes the application less scalable. To overcome this inefficiency several component vendors have come up with their own version of a file upload product and one such product is the Telerik RadUpload.

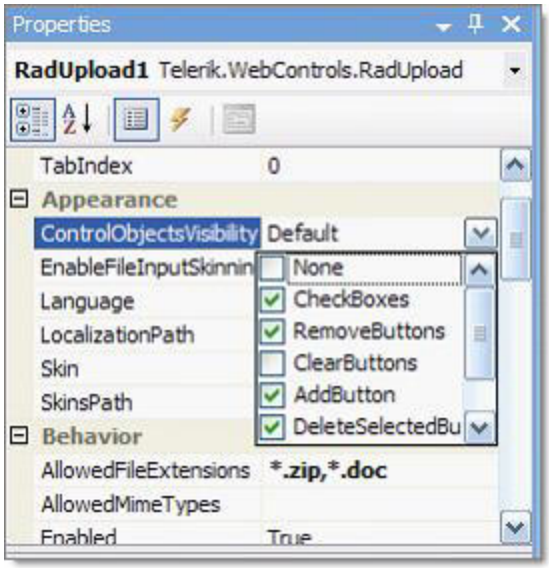
The Telerik RadUpload consists of a RadUpload, RadProgressArea, RadProgressManager controls. RadUpload is a specialized file-upload component, employing a highly efficient HttpModule (RadUploadHttpModule) that allocates a minimum amount of server memory and processes the upload stream in chunks and uses temporary folders on the server to perform optimized upload. The RadUpload control is also completely configurable and supports single and multi-file uploads

RadProgressArea can be used with a RadUpload control or with a standard HtmlInputControl to show progress. RadProgressArea contains a ProgressTemplate property, which can be used to fully customize its content and style. RadProgressManager control is used to monitor the form submission progress and update one or more progress areas in a page. Both RadProgressArea and RadProgressManager provide rich API for client-side customization.



RadUpload controls in the toolbox

RadUpload while performing optimized file upload also supports single or multiple file uploads. RadUpload allows you to add additional client-side file selectors at runtime. The *InitialFileInputsCount* property specifies the initial number of file selectors to render and you can add additional file selectors up to the maximum specified in the *MaxFileInputsCount* property. RadUpload can automatically validating the size, extension and the MIME type of the uploaded files. The valid files can be accessed using the *UploadedFiles* property and are stored in a folder specified by the *TargetFolder* property. The invalid files are still uploaded on the server and can be accessed using the *InvalidFiles* property. The RadUpload appearance can be customized by applying skins and the *ControlObjectsVisibility* property.



ControlObjectsVisibility property values

2.8.2 Configuring RadUploadHttpModule

To take advantage of the server memory optimization functionality of the RadUpload you must register the RadUploadHttpModule in your application's web.config file.

```
<configuration>
  ...
   <system.web>
     ...
        <httpModules>
             <add name="RadUploadModule" type="Telerik.WebControls.RadUploadHttpModule,
              RadUpload.Net2, Version=2.2.1.0, Culture=neutral,
              PublicKeyToken=b4e93c26a31a21f0" />
        </httpModules>
     ...
   </system.web>
  ...
</configuration>
```

Note: Make sure you have the correct type information depending upon the version of the RadUpload.Net2 assembly you are using.

When using the RadUploadHttpModule, the request stream is processed in chunks with variable size and saves them in a temporary folder, thus minimizing the usage of the server memory. By default, the RadUploadHttpModule selects different chunk sizes depending upon the size of the uploaded files and balances between memory usage and performance. To override the default request chunk size you can add the following text into the <appSettings> section of the application's web.config file.

```
<add  key="Telerik.RadUpload.ChunkSize"  value="20000"  />
```

The *value* is the chunk size in bytes.

The RadUploadHttpModule saves the uploaded files in the system's temp folder by default. To override

the location for temporary files you can add the following text into the <appSettings> section of the application's web.config file.

```
<add  key="Telerik.RadUpload.TempFolder"  value="c:\temp"  />
```

2.8.3 Configuring RadUploadProgressHandler

To use the RadProgressArea and the RadProgressManager you need to register the RadUploadProgressHandler in your application's web.config file.

```
<configuration>
  ...
   <system.web>
     ...
     <httpHandlers>
          <add  verb="*"  path="Telerik.RadUploadProgressHandler.aspx"
            type="Telerik.WebControls.RadUploadProgressHandler,
           RadUpload.Net2,  Version=2.2.1.0,  Culture=neutral,
           PublicKeyToken=b4e93c26a31a21f0" />
        ...
     </httpHandlers>
     ...
   </system.web>
   ...
</configuration>
```

Note: Make sure you have the correct type information depending upon the version of the RadUpload.Net2 assembly you are using.

2.8.4 Configuring large file uploads

.NET framework by default limits the HTTP request size to 4MB by specifying maxRequestLength="4096" in machine.config. To upload larger files, you must change the application's web.config and override the maxRequestLength.

```
<configuration>
  ...
   <system.web>
     ...
     <!-- Max Request Length is 2GB (2,147,483,648 bytes) -->
      <httpRuntime  maxRequestLength="2097151" />
     ...
   </system.web>
   ...
</configuration>
```

The value of the attribute specifies the maximum request length in kilobytes.

2.8.5 Upload File Validation

- **Automatic Validation**: RadUpload has few properties that can be set to automatically validate the size, the extension and the MIME type of the uploaded files. *AllowedFileExtensions* can be set to enable file extension validation. *AllowedMimeTypes* can be set to enable MIME type validation. *MaxFileSize* can be set to enable the file size validation.

- **Client-side Validation**: RadUpload allows file extension validation on the client with the client-side function *ValidateExtensions*. The client-side validation is done using a ASP.NET CustomValidator control. But it's important to not set the *ControlToValidate* property to the RadUpload control.
- **Custom Validation**: RadUpload provides the *ValidatingFile* event, which can be used to define custom validation logic and to override the internal validation if needed. When the file is validated, the *IsValid* property of the event arguments must be set according the result of the validation. Set *SkipInternalValidation* property to True to disable the internal validation for the specific file.

2.8.6 Saving Upload Files

- **Automatic Saving**: RadUpload allows you to automatically save the uploaded files to an existing virtual path that is specifying in the *TargetFolder* property. To save the files to a physical path you can specify the path in the *TargetPhysicalFolder* property. *OverwriteExistingFiles* property can be set to True to overwrite existing files with the same name.
- **Manual Saving**: RadUpload allows you to save the uploaded files in the web server or in any other location. You can store the uploaded files in a relational database or in other data store for later retrieval. The *UploadedFiles* property provides a collection of *UploadedFile* and you can iterate through the collection and use the UploadedFile.SaveAs() method to save the files in web server or use the *UploadedFile.Content* property to save the contents in a relational database.

Sample 1: Simple file upload with validation

SimpleUpload.aspx

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

<%@ Register Assembly="RadUpload.Net2" Namespace="Telerik.WebControls"
   TagPrefix="radU" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">

<script>
function Validate(source, arguments)
{
   //Clear the statusLabel
     document.getElementById("statusLabel").innerHTML="";
   //Validate the file extension
     arguments.IsValid = <%= RadUpload1.ClientID %>.ValidateExtensions();
   if (!arguments.IsValid )
      alert("File extension validation failed");
}
</script>

<head runat="server">
     <title>Telerik: RadUpload</title>
</head>
<body>
    <form id="form1" runat="server">
```

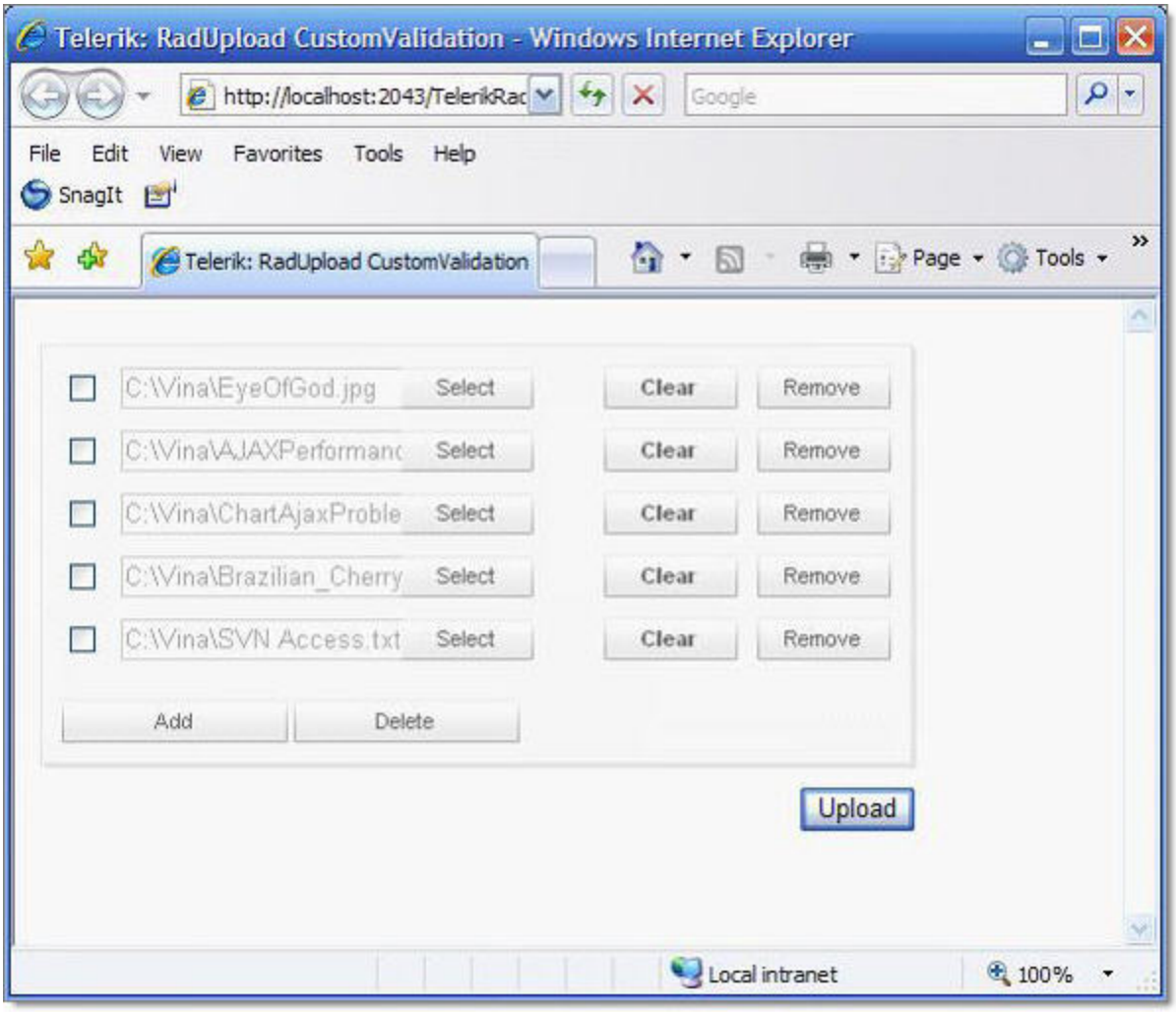
```
        <div>
            <table>
                <tr>
                    <td>
                        <radU:RadUpload ID="RadUpload1" runat="server"
                        LocalizationPath="RadControls/Upload//Localization"
                           SkinsPath="~/RadControls/Upload/Skins"
                            ControlObjectsVisibility="All"
                           TargetFolder="~/MyUploadedFiles"
                            AllowedFileExtensions=".doc"   InitialFileInputsCount="3" />
                    </td>
                </tr>
                <tr>
                    <td align="right">
                        <asp:Label ID="statusLabel" runat="server"></asp:Label>
                        <asp:Button ID="Button1" runat="server"
                         Text="Upload"  OnClick="Button1_Click" />
                    </td>
                </tr>
            </table>
             <asp:CustomValidator ID="CustomValidator1" runat="server"
                 ClientValidationFunction="Validate"
                Display="Dynamic"
                ErrorMessage="Invalid  file  extension"
                OnServerValidate="CustomValidator1_ServerValidate">
             </asp:CustomValidator>
        </div>
    </form>
</body>
</html>
```

SimpleUpload.aspx.cs

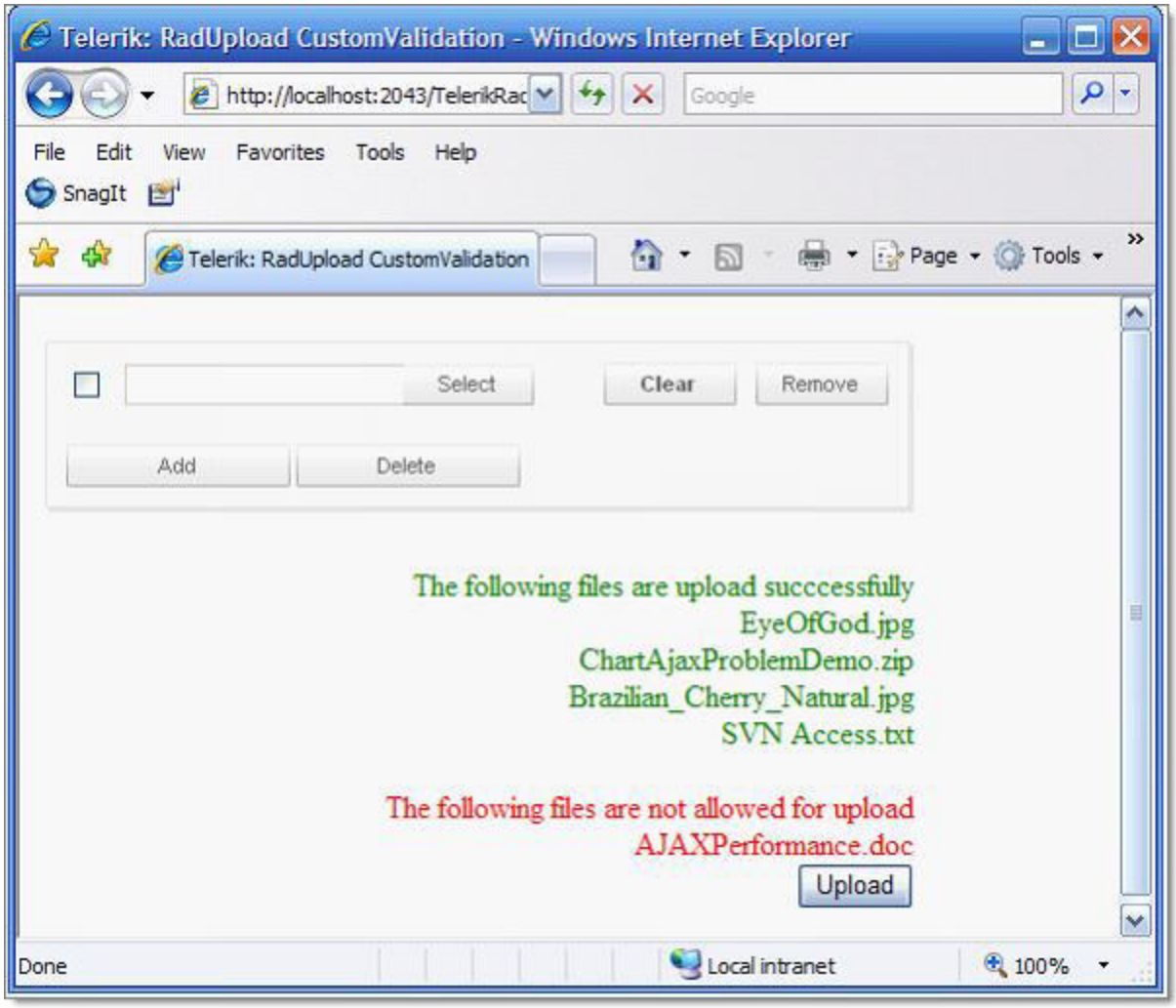
```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

using System.Drawing;
using Telerik.WebControls;
using Telerik.WebControls.RadUploadUtils;

public partial class SimpleUpload : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
        statusLabel.Text = "";
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
```

```
            if (RadUpload1.InvalidFiles.Count == 0)
            {
                foreach (UploadedFile file in RadUpload1.UploadedFiles)
                {
                    file.SaveAs(@"c:\MyUploadDir\" + file.GetName(), true);
                }
                if (RadUpload1.UploadedFiles.Count > 0)
                    DisplayMessage("Upload successful", false);
            }
        }
        catch (Exception ex)
        {
            DisplayMessage("Upload failed" + ex.Message, true);
        }

    }

    protected void CustomValidator1_ServerValidate(object source,
        ServerValidateEventArgs args)
    {
        if (RadUpload1.InvalidFiles.Count == 0)
        {
            CustomValidator1.ErrorMessage = "";
            args.IsValid = true;
        }
        else
        {
            CustomValidator1.ErrorMessage = "File validation failed";
            args.IsValid = false;
        }
    }

    private void DisplayMessage(string text, bool isError)
    {
        this.statusLabel.Font.Bold = true;
        this.statusLabel.Visible = true;
        if (isError)
        {
            this.statusLabel.ForeColor = Color.Red;
        }
        else
        {
            this.statusLabel.ForeColor = Color.Green;
        }
        this.statusLabel.Text = text;
    }
}
```

Sample 2: Simple file upload with Custom validation



Upload Custom Validation project running



Application after upload

CustomValidationUpload.aspx:

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

<%@  Register  Assembly="RadUpload.Net2"  Namespace="Telerik.WebControls"
  TagPrefix="radU"  %>
<!DOCTYPE  html  PUBLIC  "-//W3C//DTD  XHTML  1.0  Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html   xmlns="http://www.w3.org/1999/xhtml">

<head  runat="server">
     <title>Telerik:  RadUpload  CustomValidation</title>
</head>
<body>
     <form  id="form1"  runat="server">
        <div>
```

```
            <table>
                <tr>
                    <td>
                          <radU:RadUpload ID="RadUpload1" runat="server"
                           LocalizationPath="RadControls/Upload//Localization"
                              SkinsPath="~/RadControls/Upload/Skins"
                             ControlObjectsVisibility="All"
                             TargetFolder="~/MyUploadedFiles"
                              OnValidatingFile="RadUpload1_ValidatingFile"
                               MaxFileSize="1024" />
                    </td>
                </tr>
                <tr>
                     <td align="right">
                           <asp:Label ID="statusLabel" runat="server"></asp:Label>
                           <asp:Label ID="errorLabel" runat="server"></asp:Label>
                           <asp:Button ID="Button1" runat="server" Text="Upload"
                            OnClick="Button1_Click" />
                    </td>
                </tr>
            </table>
        </div>
    </form>
</body>
</html>
```

CustomValidationUpload.aspx.cs:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

using System.Drawing;
using Telerik.WebControls;
using Telerik.WebControls.RadUploadUtils;

public partial class CustomValidationUpload : System.Web.UI.Page
{
    protected void Page_Load(object sender, EventArgs e)
    {
         statusLabel.Text = "";
         errorLabel.Text = "";
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        try
        {
            if (RadUpload1.UploadedFiles.Count > 0)
            {
                 String message = "</br>The following files are upload succcessfully</br>";
                foreach (UploadedFile file in RadUpload1.UploadedFiles)
                {
```

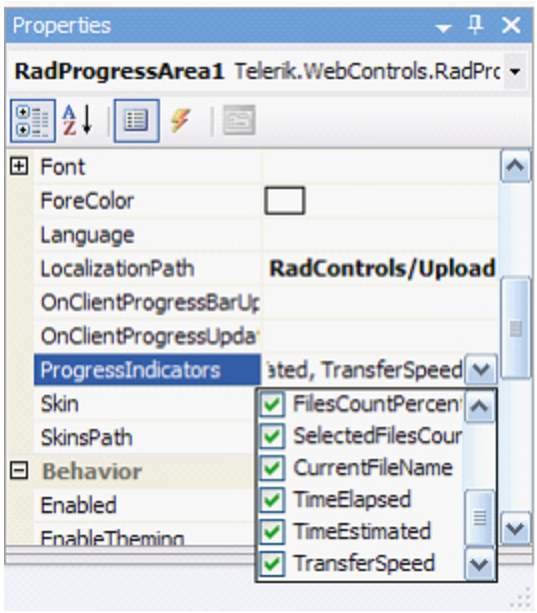
```
                    message += file.GetName() + "</br>";
                    file.SaveAs(@"c:\MyUploadDir\" + file.GetName(), true);
                }
                 DisplayMessage(message, false);
            }

            if (RadUpload1.InvalidFiles.Count > 0)
            {
                 String error = "</br>The following files are not allowed for upload</br>";
                foreach (UploadedFile ifile in RadUpload1.InvalidFiles)
                {
                    error += ifile.GetName() + "</br>";
                }
                 DisplayMessage(error, true);
            }
        }
        catch (Exception ex)
        {
             DisplayMessage("Upload failed" + ex.Message, true);
        }
    }

    private void DisplayMessage(string text, bool isError)
    {
        if (isError)
        {
            this.errorLabel.ForeColor = Color.Red;
            this.errorLabel.Text = text;
        }
        else
        {
            this.statusLabel.ForeColor = Color.Green;
            this.statusLabel.Text = text;
        }
    }

    protected void RadUpload1_ValidatingFile(object sender, ValidateFileEventArgs e)
    {
        int maxZip = 10485760; //10 MB
        int maxText = 4096; //4 KB
        int maxJpg = 512000; // 500KB

        switch(e.UploadedFile.GetExtension().ToLower())
        {
            case ".zip":
               if (e.UploadedFile.ContentLength > maxZip)
                   e.IsValid = false;
                e.SkipInternalValidation = true;
               break;
            case ".txt":
                if (e.UploadedFile.ContentLength > maxText)
                    e.IsValid = false;
                 e.SkipInternalValidation = true;
                break;
            case ".jpg":
                if (e.UploadedFile.ContentLength > maxJpg)
                    e.IsValid = false;
                 e.SkipInternalValidation = true;
                break;
        }
    }
```

```
}
```

2.8.7 Progress Monitor

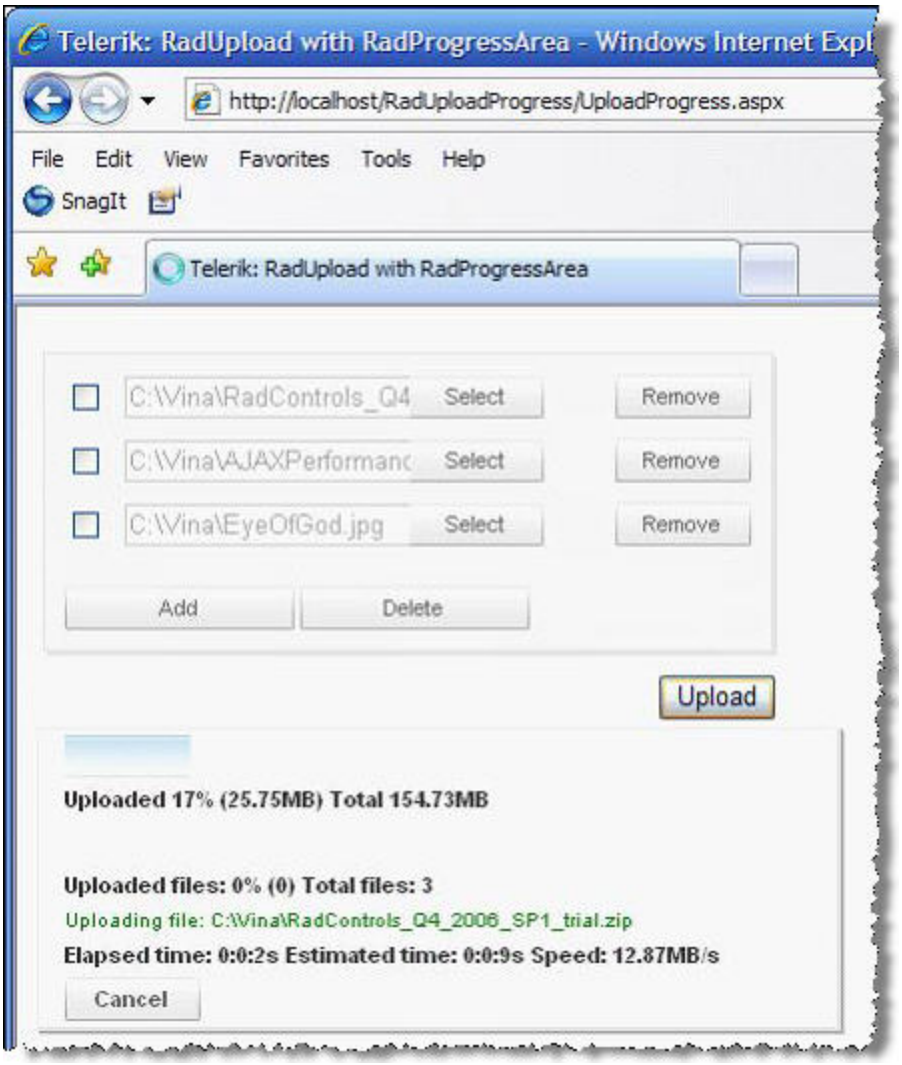
RadProgressArea provides UI for displaying the progress of file uploads or any other long running process on the web server. RadProgressManager makes AJAX calls to the containing application and updates progress information in the RadProgressArea controls on the page. To add a progress indicator to your file upload add a RadProgressManager and a RadProgressArea control to your page.

RadProgressArea has two properties that allow customization. The *ProgressIndicators* property can be used to change the visibility of the standard progress indicators, such as: TotalProgressBar, TimeElapsed, TimeEstimated or TransferSpeed. The *DisplayCancelButton* property can be used to change the visibility of the Cancel button, used to cancel the file upload.



ProgressIndicators property values

Sample 3: Simple file upload with RadProgressArea



Running project showing progress

UploadProgress.aspx:

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

<%@ Register Assembly="RadUpload.Net2" Namespace="Telerik.WebControls" TagPrefix="radU" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html   xmlns="http://www.w3.org/1999/xhtml">

<script>
function Validate(source, arguments)
{
    //Clear the statusLabel
      document.getElementById("statusLabel").innerHTML="";
    //Validate the file extension
     arguments.IsValid = <%= RadUpload1.ClientID %>.ValidateExtensions();
```

```
    if (!arguments.IsValid )
       alert("File extension validation failed");
}
</script>

<head  runat="server">
      <title>Telerik:  RadUpload  with  RadProgressArea</title>
</head>
<body>
     <form id="form1" runat="server">
        <div>
            <table>
                <tr>
                    <td>
                          <radU:RadUpload ID="RadUpload1" runat="server"
                           LocalizationPath="RadControls/Upload//Localization"
                              SkinsPath="~/RadControls/Upload/Skins"
                              TargetFolder="~/MyUploadedFiles"
                              AllowedFileExtensions=".zip,.doc,.jpg"
                             InitialFileInputsCount="3"
                              OverwriteExistingFiles="True"/>
                    </td>
                </tr>
                <tr>
                     <td align="right">
                           <asp:Label ID="statusLabel" runat="server"></asp:Label>
                           <asp:Button ID="Button1" runat="server" Text="Upload"
                            OnClick="Button1_Click" />
                    </td>
                </tr>
            </table>
               <asp:CustomValidator ID="CustomValidator1" runat="server"
                  ClientValidationFunction=""
                 Display="Dynamic"
                  ErrorMessage="Invalid file extension"
                    OnServerValidate="CustomValidator1_ServerValidate">
             </asp:CustomValidator>
               <radU:RadProgressManager ID="RadProgressManager1" runat="server" />
               <radU:RadProgressArea ID="RadProgressArea1" runat="server"
                DisplayCancelButton="True">
             </radU:RadProgressArea>
        </div>
    </form>
</body>
</html>
```

UploadProgress.aspx.cs:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
```

```
using  System.Drawing;
using  Telerik.WebControls;
using   Telerik.WebControls.RadUploadUtils;

public  partial class  UploadProgress  :  System.Web.UI.Page
{
    protected void  Page_Load(object  sender,  EventArgs  e)
    {
         statusLabel.Text = "";
    }

    protected void  Button1_Click(object  sender,  EventArgs  e)
    {
        try
        {
            //Upload  only  if  all  the  files  validate
            if  (RadUpload1.InvalidFiles.Count  == 0)
            {
                foreach  (UploadedFile  file in   RadUpload1.UploadedFiles)
                {
                     file.SaveAs(@"c:\MyUploadDir\"  +  file.GetName(), true);
                }

                if  (RadUpload1.UploadedFiles.Count  > 0)
                    DisplayMessage("Upload  successful", false);
            }
        }
        catch  (Exception  ex)
        {
             DisplayMessage("Upload  failed"  +  ex.Message, true);
        }


    }

    protected void   CustomValidator1_ServerValidate(object  source,
       ServerValidateEventArgs  args)
    {
        if  (RadUpload1.InvalidFiles.Count  == 0)
        {
             CustomValidator1.ErrorMessage = "";
            args.IsValid = true;
        }
        else
        {
             CustomValidator1.ErrorMessage = "File  validation  failed";
            args.IsValid = false;
        }
    }

    private void  DisplayMessage(string  text,  bool  isError)
    {
        this.statusLabel.Font.Bold   = true;
        this.statusLabel.Visible   = true;
        if  (isError)
        {
            this.statusLabel.ForeColor   =   Color.Red;
        }
        else
        {
```

```
            this.statusLabel.ForeColor  =  Color.Green;
        }

        this.statusLabel.Text  =  text;
    }
}
```

2.8.8 Lab: Upload with progress area

This lab will demonstrate RadUpload configuration. The application will upload a single file at a time and display a progress area.

1. Create a new web site: File | New | Web site | ASP.NET Web Site and name the site "UploadLab". Note the location of the file path for later reference.
2. Add a new web configuration file to the web site named "web.config". Add the following inside the System.Web tags to enlarge the maximum request length, and add the assemblies for progress/ upload modules and handlers:

```
<!--Rad Upload related configuration begin-->

<!-- Max Request Length is 2GB (2,147,483,648 bytes) -->
<httpRuntime  maxRequestLength="2097151"/>

<httpHandlers>
    <add verb="*" path="Telerik.RadUploadProgressHandler.aspx"
     type="Telerik.WebControls.RadUploadProgressHandler, RadUpload.Net2,
</httpHandlers>

<httpModules>
   <add name="RadUploadModule"
       type="Telerik.WebControls.RadUploadHttpModule, RadUpload.Net2,
    Version=2.2.0.0, Culture=neutral, PublicKeyToken=b4e93c26a31a21f0"/>
</httpModules>

<!--Rad Upload related configuration end-->
```

Note: Check the Version number and PublicKeyToken above. These both must agree with the actual assembly DLL that you have in the references for the site.

Note: To check the version, the control itself will print that on the page in design mode. You can also right click the RadUpload.Net2.dll and check the version in the property pages. The public key token is more likely to stay static, but you can check that on RadUpload.Net2.dll by running the strong naming tool on .NET SDK command line, e.g. **sn -T RadUpload.Net2.dll.**

3. In the Internet Information Services control panel snap-in, create a new virtual directory with alias "UploadLab" and point the directory at the web site you just created in Visual Studio. Leave the other settings at their defaults.
4. Reopen the project using the Local IIS option. If prompted to migrate the web site to .NET 2.0, select OK.
5. On the default page drop a RadUpload control. Set the *ControlObjectsVisibility* property to "None". This will leave only the select button and input box visible on the screen.
6. Add a standard Button and set *ID* to "btnLoad" and the *Text* to "Upload".
7. Add a standard Label and set *ID* to "lblStatus" and set the *Text* to blank.

8. Add a RadProgressArea control. Set the *ProgressIndicators* property to "TotalProgressBar, TotalProgressPercent".

9. Add a RadProgressManager control and leave the default properties.

10. In the code behind for the default page, at the top of the page class add a constant to contain the path to upload files to:

```
C# Example:
const string UploadPath = @"C:\MyUploads\";

VB Example:
Const UploadPath As String = "C:\MyUploads\"
```

11. In the code behind for the Page_Load event handler in the default page verify the upload directory is created:

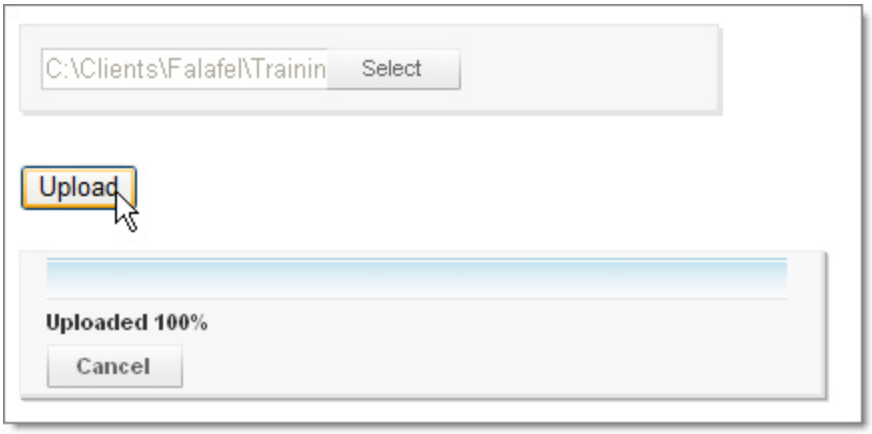
```
C# #Example:
protected void Page_Load(object sender, EventArgs e)
{
  if (!Directory.Exists(UploadPath))
      Directory.CreateDirectory(UploadPath);
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not Directory.Exists(UploadPath) Then
     Directory.CreateDirectory(UploadPath)
 End If
End Sub
```

12. Create an event handler for the btnUpload OnClick:

```
C# Example:
protected void btnUpload_Click(object sender, EventArgs e)
{
  try
  {
    if (RadUpload1.InvalidFiles.Count == 0)
    {
      foreach (UploadedFile file in RadUpload1.UploadedFiles)
      {
          file.SaveAs(UploadPath + file.GetName(), true);
      }
       lblStatus.Text = "Upload successful.";
    }
    else
    {
       lblStatus.Text = string.Format(
        "There were {0} invalid files. Download was terminated.",
          RadUpload1.InvalidFiles.Count);
    }
  }
  catch (Exception ex)
  {
     lblStatus.Text = "Error uploaded: " + ex.Message;
  }
}

VB Example:
```

```
Protected Sub  btnUpload_Click(ByVal  sender As Object, ByVal e As  EventArgs)
 Try
   If  RadUpload1.InvalidFiles.Count  =  0 Then
     For Each file As  UploadedFile In  RadUpload1.UploadedFiles
         file.SaveAs(UploadPath + file.GetName, True)
     Next
      lblStatus.Text = "Upload successful."
   Else
      lblStatus.Text = String.Format(_
      "There were {0} invalid files. Download was terminated.", _
       RadUpload1.InvalidFiles.Count)
   End If
 Catch ex As  Exception
    lblStatus.Text = "Error uploaded: " + ex.Message
 End Try
End Sub
```

13.Run the application. Try uploaded a relatively large file so you have time to see the progress bar. Check the c:\MyUploads directory for the uploaded file. **Note:** If you don't see the progress bar, be sure you're running this as an IIS website, not using the built-in web server.



Running application uploading a file

2.8.9 File Download

Now that we have seen how to upload files with the RadUpload control let's see how we can do file download a file from web server to the client. There is no "RadDownload" component per se, but here is how you can get it done in code. For file download you need to set the content type of the HTTP response to "application/octet-stream" and stream the file contents as shown below.

```
String  fileName  =  "mydownload.jpg";
byte[]  contents;

...
  //Load  contents  into  byte  array
    HttpContext.Current.Response.ContentType  = "application/octet-stream";
     HttpContext.Current.Response.AddHeader("Content-Disposition",
     "attachment;  filename=" + fileName);
     HttpContext.Current.Response.Clear();
      HttpContext.Current.Response.OutputStream.Write(contents, 0,  contents.Length);
     HttpContext.Current.Response.End();
```

2.8.10 Troubleshooting

If you run into issues while working with the RadUpload, RadProgressArea controls check the following checklist published by Valeri Hristov, Telerik development team to see if you are running into some of the known limitations.

Application configuration

1. Do you use the integrated web development server in VS2005 (Cassini) to host your application? We still do not support this environment and if you want to benefit from RadMemoryOptimization and RadProgressArea you should host your application in IIS.
2. Add RadUploadProgressModule registration in web.config
3. Add RadUploadProgressHandler registration in web.config
4. Add RadProgressManager to the page
5. Do you submit the page using AJAX button (the button is either in Ajax / Update Panel, or has command / trigger assigned)? Files can be uploaded only with a real postback, not with Ajax / Atlas call. The following topic contains a workaround for RadAjax: Uploading Files with Atlas or Ajax.
6. Check if you are using an URL rewriter:
   - "Telerik.RadUploadProgressHandler.aspx" path must be excluded from rewriting.
   - The URL rewriter must persist the custom query string parameters, added by RadProgressManager.
7. Do you use other HttpModules for file upload in your application? The upload HttpModules are incompatible by their nature. You should use only one upload HttpModule in an application.
8. Check if the application trace is enabled. If it is, disable it. RadUploadHttpModule is not compatible with the application trace.
9. Check if your application is running in a web farm. We still do not support this environment for progress monitoring. However, RadMemoryOptimization will be operational in a web farm.

IIS Configuration

Virtual Directory properties --> Application Settings --> Configuration --> Application Mappings --> Add / Edit Application Extension Mapping
Open .ASPX extension and uncheck "Check that file exists". The checkbox is unchecked by default, but it is possible to be checked by some hosting providers or custom configurations. If you have no access to these settings, you can put an empty page named Telerik.RadUploadProgressHandler.aspx in your application root.

Browser issues

Opera stops the execution of the client-side script when the page is submitted, thus preventing the progress area from proper operation. We are looking forward for browser updates which will enable executing client-side scripts when a submit button is clicked.

2.8.11 Summary

This section explained how to configure your application to use RadUpload, how to validate the upload, perform custom validation, deal with large file uploads, and how to display progress during the upload. Even though not currently handled by a Telerik control the rudiments of performing a file download were

described for the sake of completeness. This section also described common troubleshooting scenarios and solutions.

Day

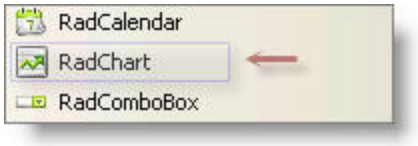
3 Day 3

3.1 RadChart

3.1.1 Getting Started

The Telerik RadChart control provides a nice selection of different chart types that can be used to easily display data in a graphical format on your web forms. There are a vast number of properties which control every imaginable aspect of the chart's visual presentation - most of which are supported by one of the most comprehensive wizards you've ever seen.

In the Telerik controls section of your Visual Studio toolbar, the icon to add a chart to your page or user control looks like this:



The RadChart component

In the first section, we'll take an in-depth look at the chart wizard, since you can use it to control so much of the component's behavior. Although at its core it is "just" a means of setting properties, in this case the wizard is used to group and organize the properties into related areas and present then visually and in the context of other properties that are closely related. Although you can accomplish these same settings using the regular property sheet, rest assured that in most cases, using the wizard will be a much more efficient way of getting the job done.

After that, we'll explore how to use the drill-down capability, and see how you can use the information conveyed in the postback event to determine what happened on the client-side of the application. We'll wrap up by taking a look at how we can use Telerik AJAX features to update other sections of the form.

With those objectives in mind, set up a new Visual Studio project, connect it to a data source, and explore the wizard.

1. Create a new ASP.Net project named "Charting"

2. Create a RadControls folder within the project

3. Copy the \Chart directory to this RadControls directory. The \Chart directory is located under the directory where you installed your rad controls. In a typical installation, this will be something like "C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls". **Note: There is an identically-named "Chart" folder located at the same directory level as the RadControls folder** which contains a lot more files. This is **not** the folder you should copy - it contains numerous examples and images which are not needed for the development of your project.

4. Drop a RadChart component onto your design surface.

In order to work through the examples, you'll need to connect to a datasource. We'll use the Northwind database as the basis for samples in the RadChart labs.

5. Drop a SqlDataSource onto your design surface. Use the smart tag's "Configure Data Source" option to set up a connection to an available Northwind database, and include the select statement shown below. When you're done, the markup file should look something like this:

```
<asp:SqlDataSource   ID="SqlDataSource1"   runat="server"
       ConnectionString="<%$   ConnectionStrings:NorthwindConnectionString   %>"
```

```
      SelectCommand='select  categoryname,  count(*)  "ItemCount"
                from categories, products, [order details]
               where  categories.categoryId  =  products.categoryId
               and  products.productId  =  [order  details].productId
                group by categoryname'>
</asp:SqlDataSource>
```

Important Note: The select statement in the example above CANNOT stretch across multiple lines as shown - it is only shown in that manner so that you can see the entire contents of the select statement here. Also notice that what would normally be double-quotes around the select command have been replaced with single quotes. We do this because the SQL syntax requires the double quotes around the substituted column name "ItemCount".

6. On the RadChart control, set the DataSourceID property to point to the newly-creates SqlDataSource.

As a point of reference, the data returned from the Northwind database we'll be using for these materials looks like this:

Categoryname	Count
Beverages	404
Condiments	216
Confections	334
Dairy Products	366
Grains/Cereals	196
Meat/Poultry	173
Produce	136
Seafood	330

3.1.2 Exploring the RadChart Wizard at Design Time

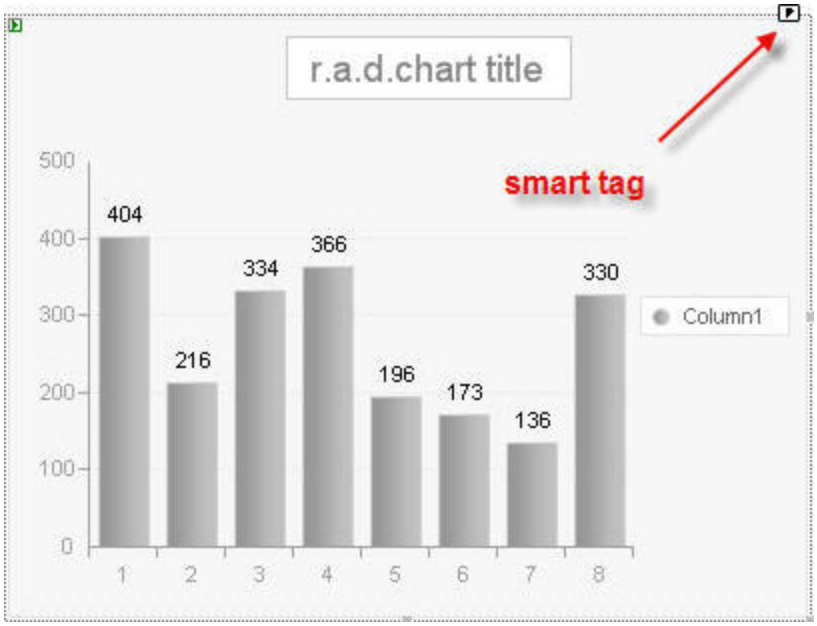
RadChart is a comprehensive chart generation component which supports thirteen types of business charts, using a design-time wizard supporting over 95% of the control's available settings. Modifications made through the wizard are displayed on the design surface as they are made, allowing you to immediately see the impact of the changes you are making. This "live component" aspect of the RadChart is a tremendous relief from the old way of CCR (change - compile - run) testing that used to be the only way to inspect the results of your changes.

Don't like your changes, and want to start over? No problem, Telerik has also incorporated a "reset" button into the wizard, so if you want to go back to your starting point (from the time you opened up the chart wizard), you can do so with one click of the "Reset" button. You can accomplish the same thing by clicking the "Cancel" button, except the wizard will disappear.

Lab: Thirteen chart types, no waiting

With your SqlDataSource hooked up to a Northwind database, and your RadChart hooked up to the SqlDataSource, you should already see the first fruits of Telerik's work at making the design-surface control respond in real time to reflect property changes. Although it's pretty bland thus far with its grey-

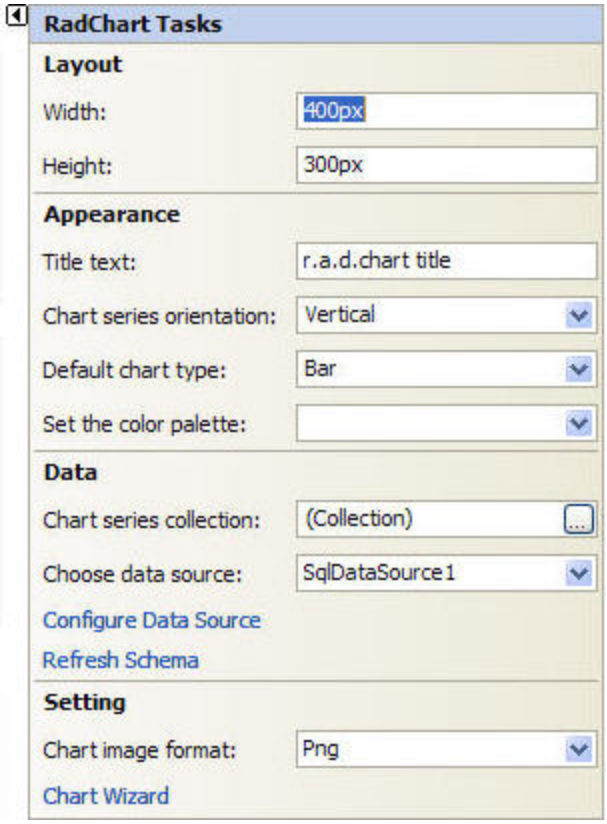
tone color scheme, the data being displayed in the chart is reflecting the actual database contents - those numbers and bar heights you see displayed are real!



A Very Plain Chart with its Smart Tag

In this lab we'll spend a fair amount of time exploring the chart wizard, first looking at the variety of ways that RadChart can display your data.

1. In case you're not familiar with the smart tag, it's highlighted in the graphic above. Click on the smart tag, and you'll see the following menu:



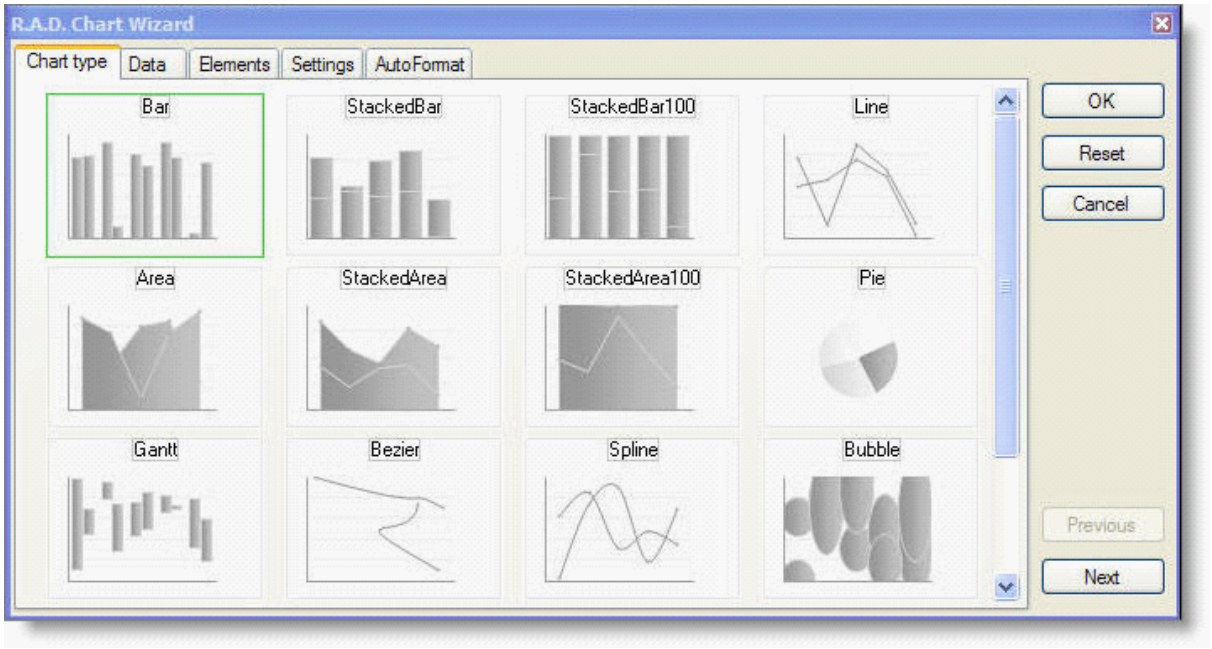
The smart tag menu

Even from the smart tag menu, you'll notice that there is a fair amount of customization that you can do to the chart.

2. On the context menu, change the following items:

- Set the width to 320
- Set the height to 240px (thus preserving the ratio)
- Set the title to "Order Items Shipped - by Category"
- Set the image format to "gif"

As you make each of these changes (except the last - image format is an output rendering setting), you will see the corresponding change take place in the chart displayed on the design surface. Now, click on the "Chart Wizard" link at the bottom of the context menu. Here's what you'll see:



RadChart Wizard displaying the ChartType tab

This is pretty basic stuff, but helpful nevertheless - it's a nice visual reminder of what the different charts (which are also available from the dropdown list in the context menu) look like. Click around on the various charts and watch how the data we're using is reflected in the chart being displayed.

3. Select the "Pie Chart" type before proceeding to the next section.

Lab: Charts without a Data Source

Let's say you do a survey of your friends and ask them which Wrigley's gum flavor they prefer - Spearmint, Doublemint, or JuicyFruit? And you want to put a pie chart showing the results on your web site, without the hassle of creating and maintaining a table in some database, or even programmatically (in the code)? One nice feature about the RadChart Wizard is that it will help you put in and organize data you wish to display in just this fashion.

Activate the RadChart wizard either by clicking on the chart's smart tag and then on the "Chart Wizard" link at the bottom of the context menu; or, you can select the chart on the design surface, and you will find a "Chart Wizard" link active at the bottom of the properties window. To make life just a bit easier still, you can right-click on the chart itself, and find a "Chart Wizard" link in the context menu.

In this lab, you'll enter some data that doesn't come from a data source, and do some customization of the chart's visual appearance.

1. On the RadChart wizard, click on the "Data" tab. You will see that this tab has three subtabs, and that the data returned from the query is displayed in a tabular form on the "DataSource" tab. Now, although you set up the connection to the Northwind database in the Getting Started section, this lab will use data that you'll enter by hand; so, from the "Choose data source" drop down, select the "(None)" entry. Now click on the "Series" subtab.

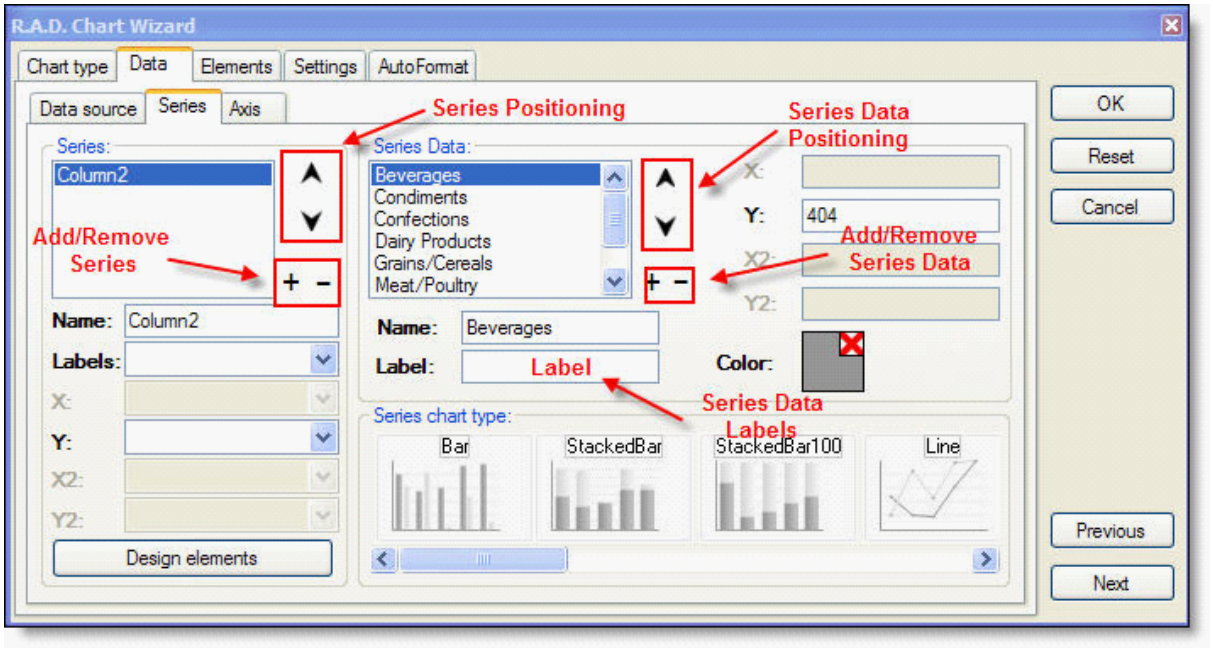
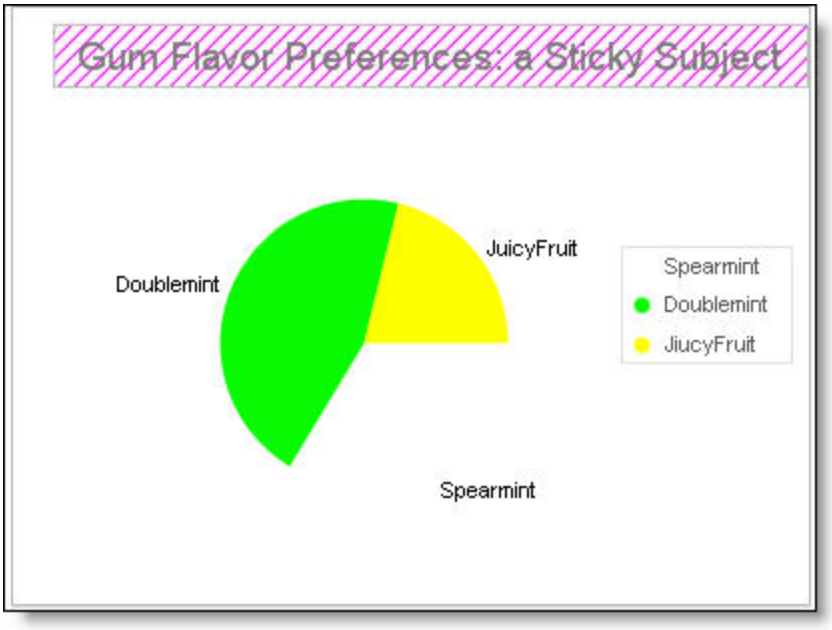


Chart Wizard Data Series SubTab

2. Add a "Series" entry by clicking on the "+" sign in the "Add/Remove Series Data" section. In the "Series Data Label" box, enter "Spearmint". In the entry box for the "Y" value, enter the number of your friends who prefer Spearmint over any other Wrigley's flavor. Finally, click in the middle of the "Color" area, and pick a color from the pop-up menu. White would be a nice color for Spearmint.

3. Add two more flavors: Doublemint (green) and JuicyFruit (Yellow). As each flavor of gum is added to the chart, a new segment appears in the pie, its area reflecting the percentage of the total area represented by the total number of friends' choices you have entered thus far.

4. Click on the "Elements" tab, My first reaction was "Wow". There are sure a lot of options available. Let's start by giving the chart a title - no, make that a **distinctive** title. Click on the "Title" tab. In the input box for the title, enter "Gum Flavor Preferences: a Sticky subject". In the Fill Style dropdown, select "Hatch". In the Hatch Style dropdown, select Wide Upward Diagonal. And (of course) to make this visible, you'll need to introduce some color, since a white-on-white hatching is pretty boring - so in the color selector next to the fill style dropdown, select a color. The medium fuchsia color is nice...



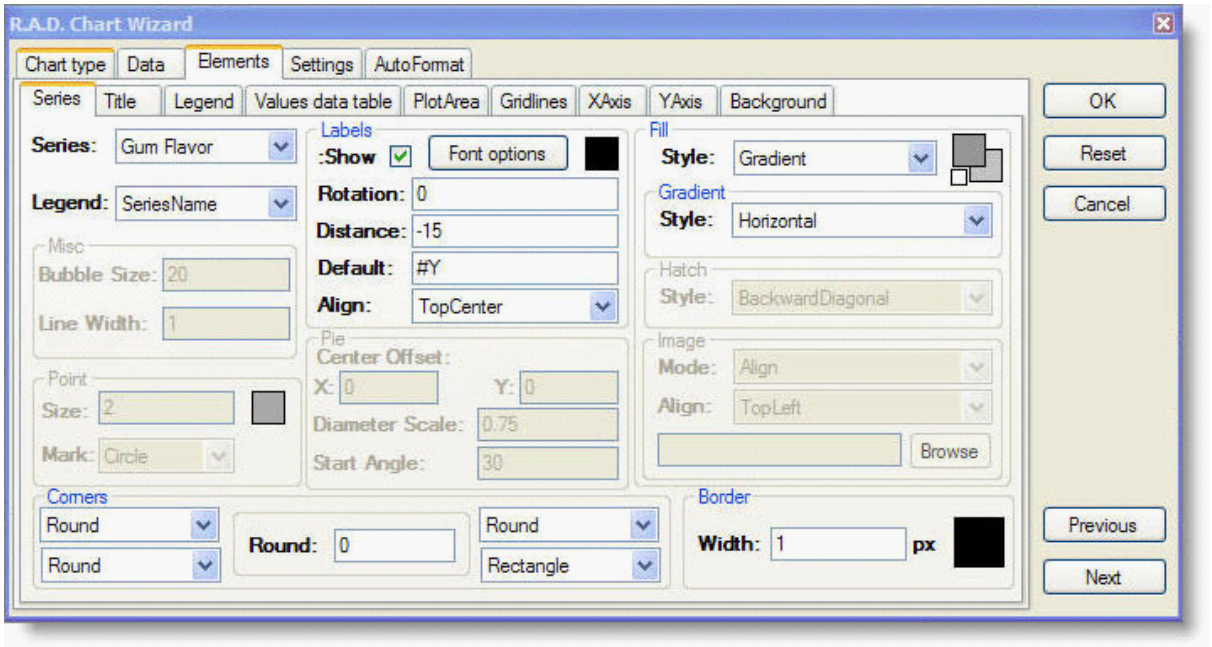
2006 Silicon Valley Ugly Pie Chart contest - Honorable Mention

As you clicked on the "Pie" chart type, you may have noticed that a legend appeared on the chart, fully populated with the names of the various product categories. The reason for this is that of the available chart types, the pie chart is the only one capable of displaying just one series. Because of that fact, the wizard is smart enough to figure out where the series data is coming from, and make appropriate entries in the legend,

In the context of a pie chart, which can display results from only one series, the "Add/Remove Series" and "Series Positioning" buttons have no real application - you may ignore them for pie charts, and we'll discuss them later where they are more applicable. Likewise, the three areas for manipulating series data don't have any real relevance when your data is going to come from a data source. Oh, you can add new items or rearrange existing items if you'd like, and the changes will be reflected on the design surface; but as soon as you run the project, you'll find that the RadChart component is going to jilt you, and your feelings will only end up getting hurt.

The Elements Wizard

You probably noticed (once again) that as you did the work in the previous section, the changes were appearing on the design surface just as you made them. While you've still got a fairly simple set of data to work with, now is probably a great time to step through the various tabs, twiddle with the settings, and see what they do to effect the chart display. Click on the next tab over in sequence on the wizard - "Elements" - and you'll see one of the most comprehensive wizard interfaces that you've probably ever seen.



The Chart Wizard Elements Interface (Series Tab)

There's a lot just on the Series sub tab, and there are eight other tabs. We're going to spend some time stepping through each of the tabs and attempt to expand on what settings on each tab are responsible for. Here are some of the Elements wizard highlights:

- Series Tab
- Title Tab
- Legend Tab
- Values Data Table Tab
- Plot/Area Tab
- Gridlines Tab
- X-Axis and Y-Axis Tabs
- Background Tab

Series Tab

Series: This drop-down list selects which of your data series the settings you are making will apply to. We have not yet ventured into entering multiple series on a chart; when we do, just be aware that the settings made on this tab apply on a series-by-series basis, and that this drop-down is where you select which series you are working with.

Legend: The Legend dropdown lets you select the source of the text in the legend box from either the "Series Name" (which will result in only one entry in the legend box), the "Item Labels" (which will result in one entry for each item in the series), or "Nothing", effectively rendering the legend box not visible if you are using only one series. Note: As of this writing, entries in the legend when "Item Labels" is selected are coming from the "Item Name" entry for each item instead. Further, the although the item name can be entered on the wizard's Data/Series tab, it does not persist as expected. The way to work around this is to use the series editor from the property page, and from there the "Items" editor. Changes

made to the Name value there will be persisted and displayed in the legend.

Misc: This area contains two input boxes, labeled "Bubble Size" (applicable only to Bubble charts), and "Line Width" (applicable to the Line, Bezier and Spline chart types).

Point: The settings in this area are available to set the size, color and style of markers used to mark data points for the following types of charts: line, spline, bezier, all of the area-type charts, and the point chart. Note that the "Color" setting available here applies to all of the chart types except the "Point" chart. Individual points in a point chart take their color from whatever color is set for the item on the "Data - Series" page.

Labels: The "Show" checkbox sets whether or not the series labels are visible next to the data. The "Font Options" button brings up a traditional font selection dialog for selecting font, style, size, effect and script, while the color is set using the color swatch next to the button. "Rotation" will angle the label text (on many chart styles, though not on pie charts), and may be entered as a negative number. The "Distance" setting may also be entered either positive or negative, and controls the distance offset (in pixels) that the label will be placed from the edge of the corresponding chart element. There is some RadChart behavior here of which you should be aware, in that the labels will not display above the top of the Y axis. If you have left the Y-axis in "Autoscale" mode (on the Data/Axis tab), then the label for the maximum Y-value (and for any other Y value sufficiently close to the max value) will be displayed down into the corresponding chart element, despite whatever positive distance offset you may apply. The way to work around this is to select a Y axis maximal value (on the aforementioned Data/Axis tab) sufficient to leave enough "headroom" above the ChartSeriesItem to display the label with the same offset as the rest of the items.

The "**Default**" selection, though presented as a textbox, really out to be a combobox, because there are three entries predefined by Telerik which may be entered here. These are:

- **#Y - represents the value of the item**
- **#%** - the percentage of the total sum of all items
- **#SUM** - the sum of all item values
- In addition, you may enter a standard numeric format string as a suffix to any of the above - for example, to display the value of the item using a currency format, you would enter "#Y{C}".

The "Align" setting dictates where the label will appear in relation to the displayed ChartSeriesItem (this has no application to the Pie Chart type).

Pie: (Applicable only to the Pie Chart type) Setting the X and Y Center Offset will move the center of the pie chart left and right, up and down. This might be useful, for instance, if you have a legend that will contain longer item labels within it, and you want to nudge the chart around a little bit to accommodate the legend display. Note, however, that there is not a tremendous amount of leeway to do this nudging before you start taking slices off the edge of the pie, and that's where the next setting for Diameter Scale can come in handy: this setting allows you to proportionally shrink (or expand) the size of the chart being displayed. The final setting in the Pie section is for the Start Angle, which dictates where the segments will start being filled in. The default is at zero degrees, which means that the first pie segment is drawn from a straight line running from the center point of the circle straight to the right (due East for you compass aficionados), and proceeding downward or clockwise from there. The "Start Angle" offset causes the pie segment to advance clockwise around the circle

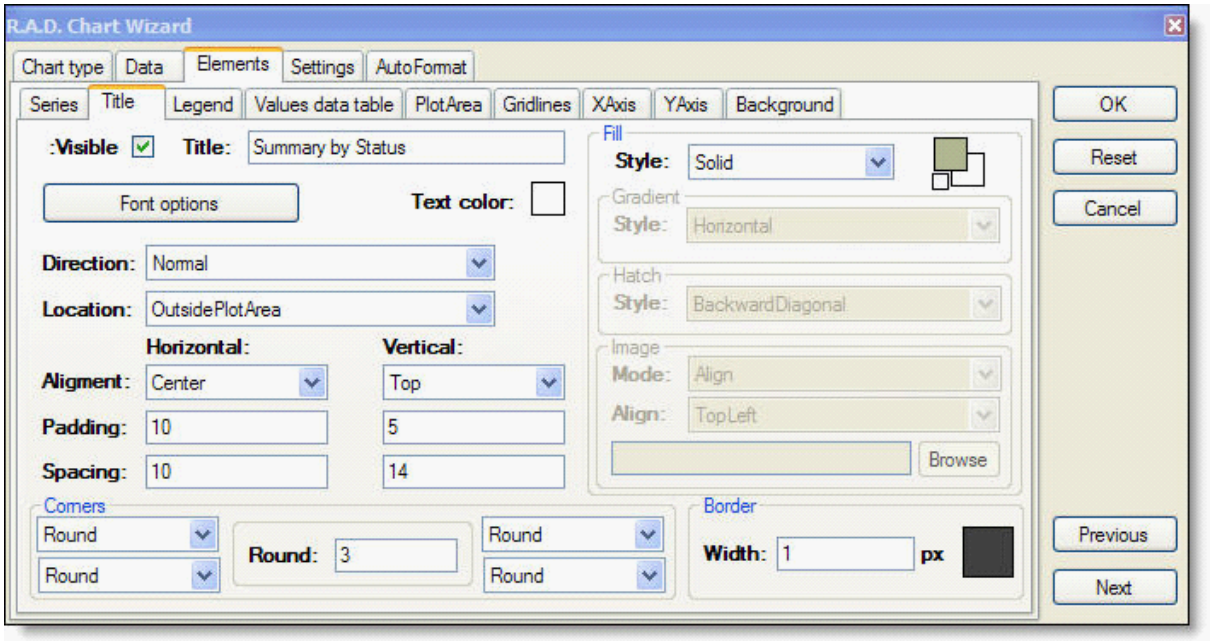
Fill: The entries in this area set the fill properties of the chart. You can select from solid, gradient, hatch, or image fills. The various other boxes in this area will be activated for input based on the fill style selected. Colors for solid, gradient and hatch are selected using the selector to the right of the style dropdown. There are five different gradient patterns to choose from, and over fifty different hatching

patterns (multiplied by the nearly limitless number of color possibilities available). One of the most intriguing fill possibilities, however, is the "Image" fill. When you first select the Image fill style, the chart rendered on the design surface will disappear for lack of an image to process. Selecting an image remedies this. If the image you select is large enough, the "Mode" and "Align" controls will have no effect, and the image will be displayed inside the chart. However, if the image is small enough, you can have a little fun with how it displays in your fill area. One smallish image you could use is in the Telerik NET2\img directory, and is named designerBox.jpg. Select that image and play with the various tiling settings.One word of caution: at the time of this writing, selecting the "image" fill style and exiting the wizard is the last time you'll see the wizard for this chart - a reported bug causes you not to be able to re-open the wizard from either the context menu or the property page. You can, however, still access and set properties using the regular property editor.

Border: This area lets you set the border width and color for the area-type charts, and also applies to the markers inside the legend when the legend is based on the series name (as opposed to the Item Labels).

Title Tab

We visited this tab briefly when building up that award-winning look in the last section; but let's take a little closer look at what's available here.



The Chart Wizard Elements Interface (Title Tab)

Visible: Sets visibility of the Title area of the chart.

Title: Maintains the caption for the overall chart.

Font Options: Sets the font, style, size, script, effect and color used for the title.

Direction: The "Normal" direction for the title runs left-to-right parallel to the horizontal bottom of the chart area; but you can use this drop down list to select a title that runs bottom-to-top, with the tops of the letters facing left; or top-to-bottom, with the tops of the letters facing right.

Location: Your choices here are "Inside Plot Area" and "Outside Plot Area". This is a little misleading in the case of the "Left" and "Right" direction settings, because all it really ends up meaning is that the

vertically-oriented title will start or end inside or outside the plot area.

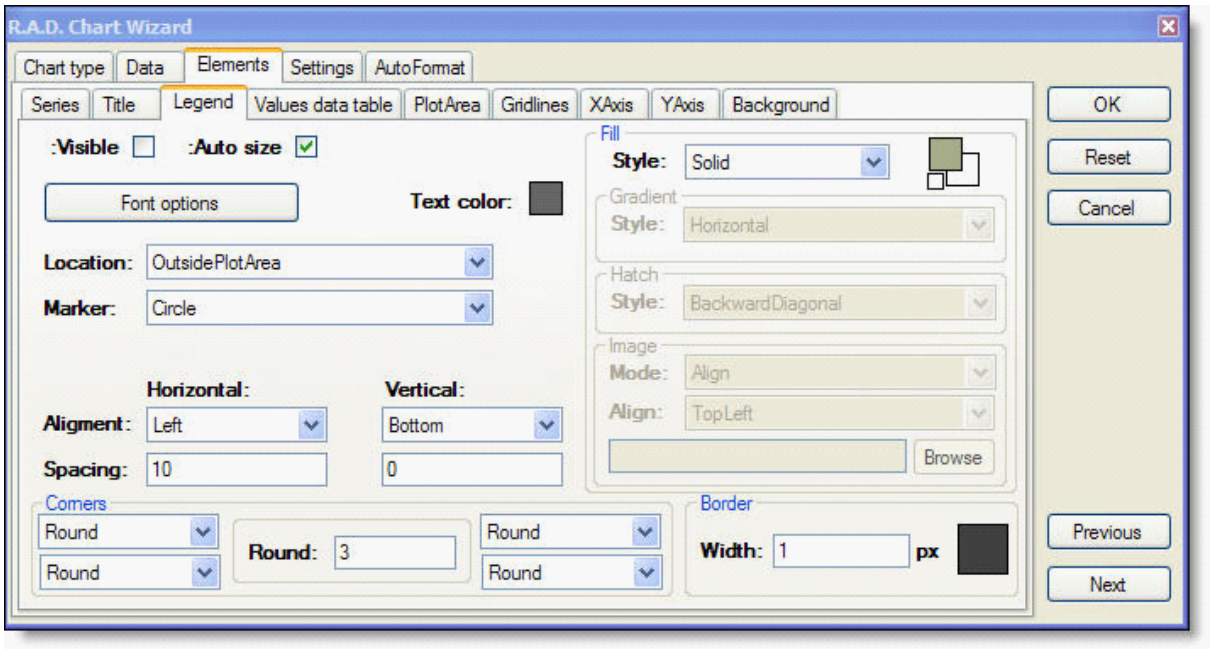
Alignment, Padding and Spacing: "Padding" refers to the amount of space between the characters that make up the title and the borders of the title container, while "Spacing" refers to extra space wrapping the outside of the title bar. "Alignment" is relative to the rectangle allotted to the chart component as a whole. There is a fair amount of interplay between these three settings: for example, in an area 400 pixels wide, for a message that takes up 350 pixels of width, with horizontal centering and 0 spacing, the title will appear exactly centered. If you change the spacing to 10, the title will get nudged to the right; change it to 60, and the title will start getting pushed off the right side of the chart, truncating the end of the title bar. The order of precedence is that the padding values are applied first. The major alignment values are then applied, followed by the spacing parameters (which can also be negative, by the way).

Corners: There are four corners on the title, and each can be designated to be either rectangular or rounded. The rounding parameter for each of the corners is determined by the positionally corresponding setting. The amount of rounding is set with the "Round" parameter, and tops out at about 15 (setting values over 15 has no additional effect).

Fill and Border: These settings are directly analogous to the settings of the same name in the "Elements - Series" subtab. Refer to that section for additional description. Note that the same caution about using an image fill type also applies here as there.

Legend Tab

The settings on the legend tab are virtually identical to those for the title tab. Two new settings are introduced.



The Chart Wizard Elements Interface (Legend Tab)

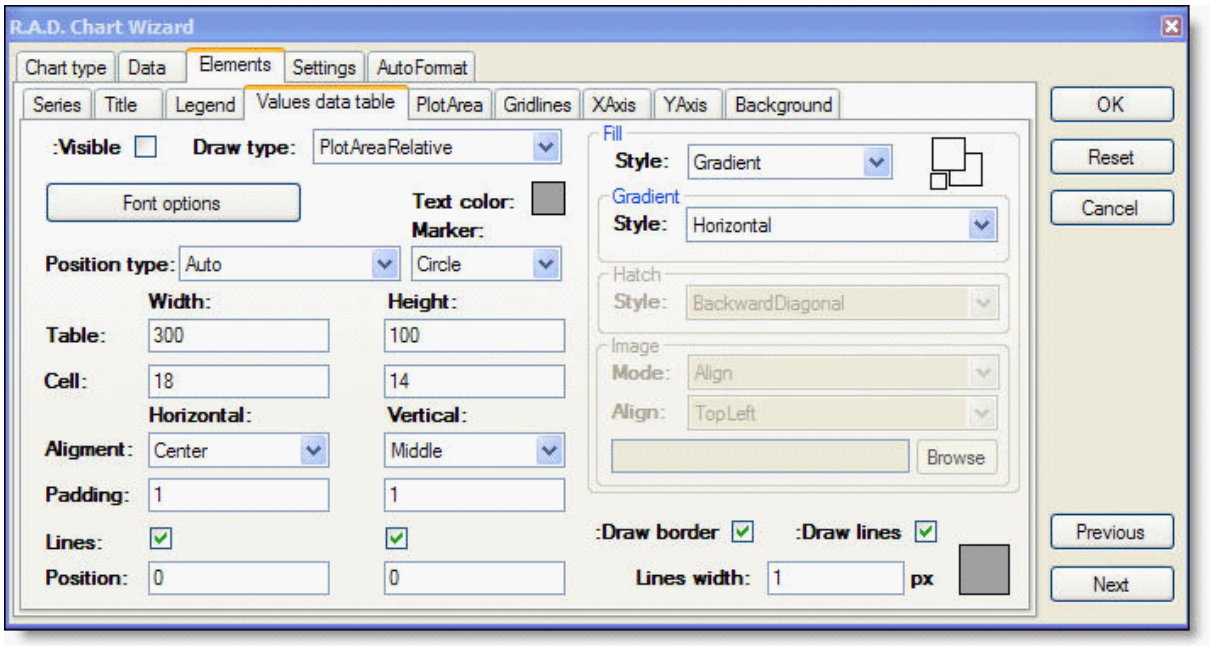
Auto Size: This controls whether the legend box is automatically sized by logic built into the RadChart component. If Auto sizing is disabled, the legend box will expand to fill the entire area allotted to the RadChart component, and there are no wizard settings available to adjust the legend's appearance. In order to set the appearance, you need to exit the wizard, and in the property editor expand "Legend", then "Legend Frame". There, you will find settings for X and Y offsets, as well as Width and Height of the

Legend Frame.

Marker: This setting (limited to Rectangle, Circle and Triangle) indicates the shape of the marker inside the legend adjacent to the legend item, when the legend is based on series name. When the series is based on Item Labels, this setting has no impact on the markers - they appear rectangular whatever the marker setting.

Values Data Table Tab

The "Values Data Table" is displayed beneath the plot area, and contains the data which make up the series. Again, many of the settings on this legend should be getting to be familiar, and the reader is referred back to the discussion under the "Title" tab. However, several new settings are introduced here.



The Chart Wizard Elements Interface (Data Tab)

Visible: Stating the obvious. All you really need to be aware of here is that pie charts won't display a values data table (one way to see this very graphically is to check the visible checkbox , then click on the "Chart Type" tab of the wizard - the micro-charts displayed there will all display a tiny version of the data table, except for the pie chart).

Draw Type: This setting controls how the data table is drawn. The "AutoSize" setting renders a nice, compact table, while "PlotAreaRelative" moves the series name outside the edge of the plot area, and spreads the series items evenly across the width of the chart, with the value centered below the corresponding point plotted in the chart area. The two other options (CellFixedSize and TableFixedSize) will take their width and height parameters from the corresponding entry boxes in the middle of the left-hand panel. If the "draw type" is set to TableFixedSize, and the table width and height are set to 300 and 100 as shown in the example above, the values data table will be drawn according to those dimensions. Note that it is possible to specify a table size that exceeds the size allocated to the chart component, so be careful when setting your chart sizes.

Lines: These two checkboxes separately control whether the horizontal and vertical internal lines (those separating the table cells) are drawn. Note that unchecking both of these boxes has the same net effect as unchecking the "Draw Lines" box described below; however, if the "Draw Lines" box is unchecked, no checked setting in either of the "Lines" boxes will have any effect.

Draw Border: This setting controls whether the data table edging is displayed (left and right sides, and bottom - the top is dictated by the bottom of the plot area).

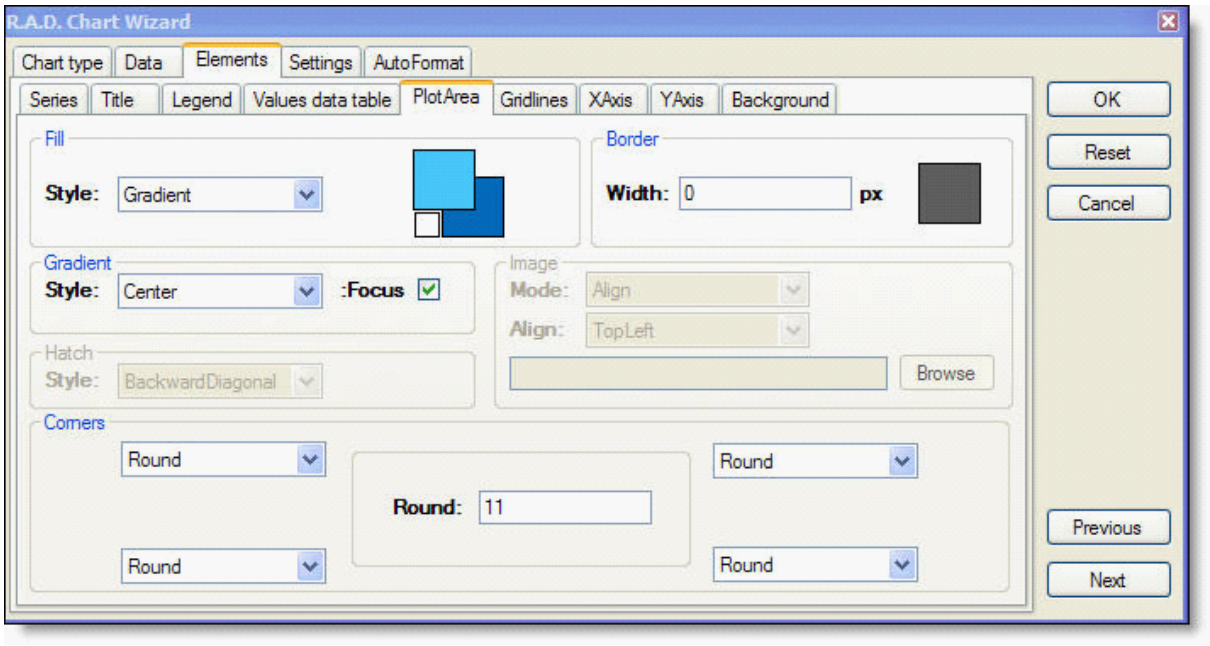
Draw Lines: This setting controls whether the lines internal to the table (the ones separating the cells) are displayed. If this box is checked, drawing of the lines is further controlled by whether the "Lines" boxes are checked (see above); if unchecked, no lines will be drawn regardless of the checked state of the "Lines" boxes.

Lines width and color: These control the width and color of the lines that are drawn as borders and internal separation lines for the table.

Fill, gradient, hatch and image: These settings are directly analogous to the settings of the same name in the "Elements - Series" subtab. Refer to that section for additional description. Note that the same caution about using an image fill type also applies here as there.

Plot/Area Tab

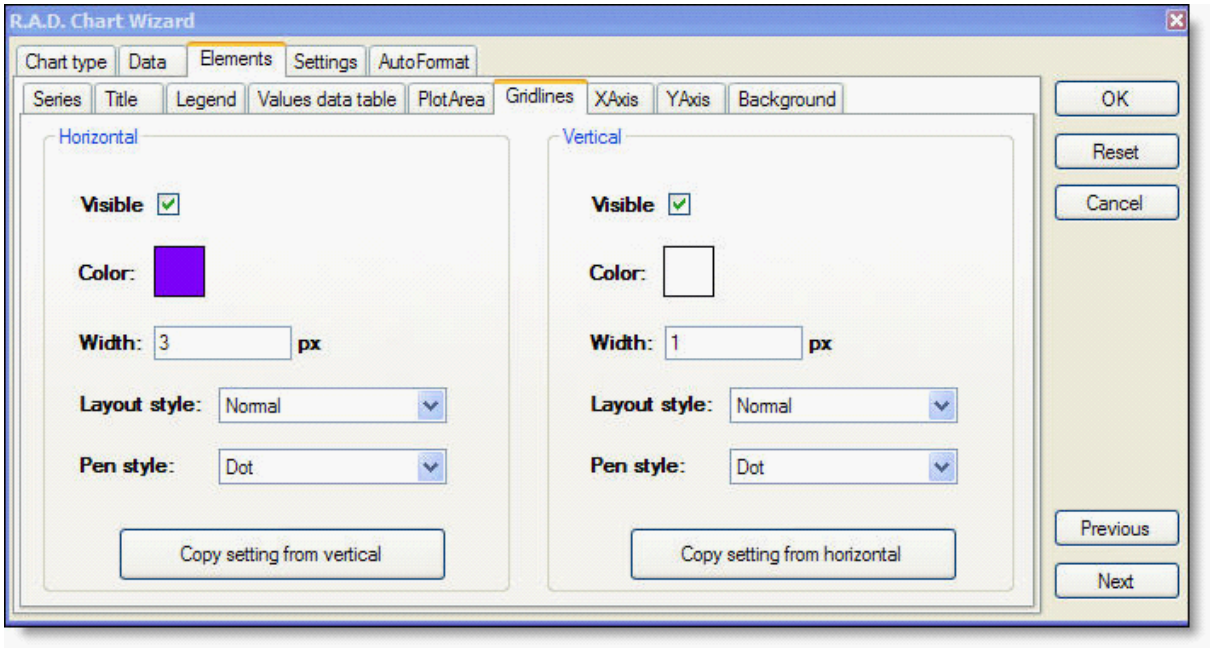
The settings in this table control the visual appearance of the main plot area. If you have been reading the text of this manual straight through, all of the settings in this tab are familiar to you - the only difference is that they apply to the plot area, so there will be no further discussion of the settings function in this section.



The Chart Wizard Elements Interface (Plot/Area Tab)

Gridlines Tab

The settings on this tab control the display properties of the vertical and horizontal gridlines in most charts (pie charts clearly are not going to have any gridlines).



The Chart Wizard Elements Interface (Gridlines Tab)

Note that the settings available are given for both horizontal and vertical gridlines.

Visible: Controls whether the gridlines are visible or not.

Color: Select from the standard 48-color popup palette, or specify your own custom color.

Width: Specifies the width of the gridline, in pixels.

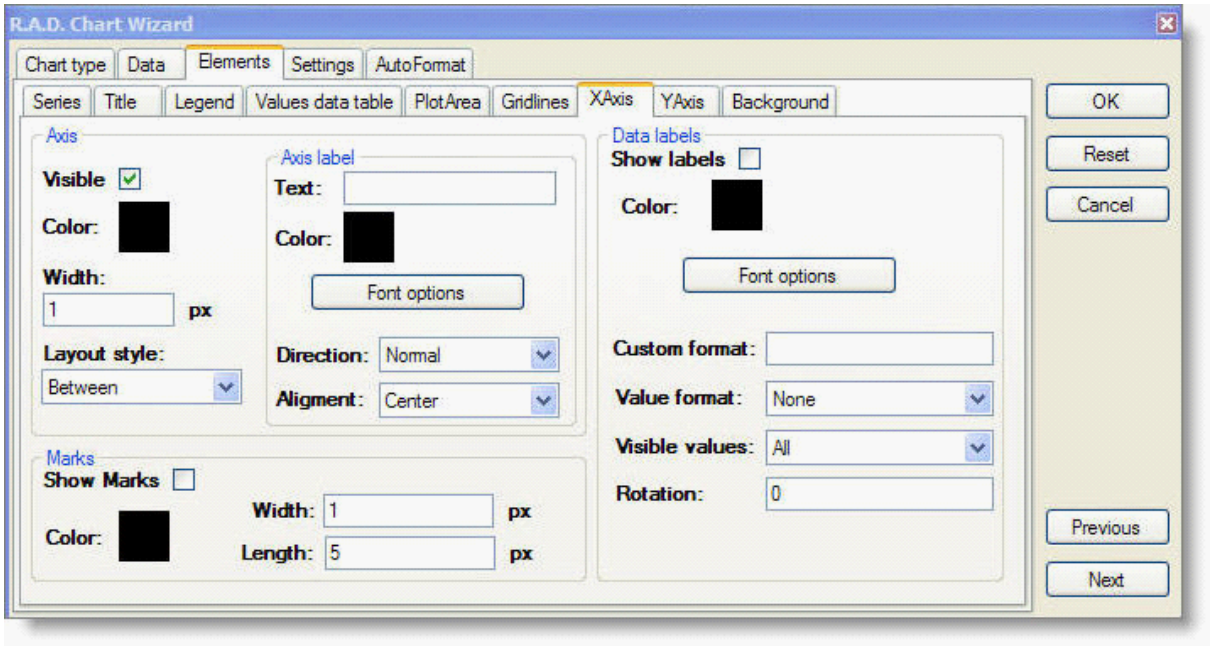
Layout Style: The choices available here are "Normal" and "Expanded". The "Expanded" selection removes every other gridline (that is to say, there are half as many gridlines displayed).

Pen Style: This setting controls the type of line used to draw the gridlines - solid, dashed, dotted, and a couple different combinations of dashes and dots.

Copy Settings..: These buttons are convenient to keep the settings for your vertical and horizontal gridlines consistent with each other.

X-Axis and Y-Axis Tabs

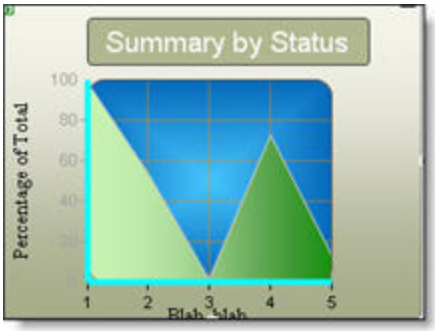
These two tabs, which contain nearly identical settings controlling the appearance of the horizontal (X-Axis) and Vertical (Y-Axis), will be covered in the same section. Once again, these settings apply to all but the pie chart type.



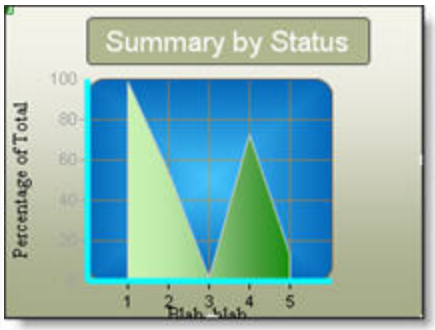
The Chart Wizard Elements Interface (Axis Tabs)

Axis: The settings in this area refer to the line drawn along the bottom (X-axis) or left-hand side (Y-axis) of the chart, and control the visibility, color and width of the axis line.

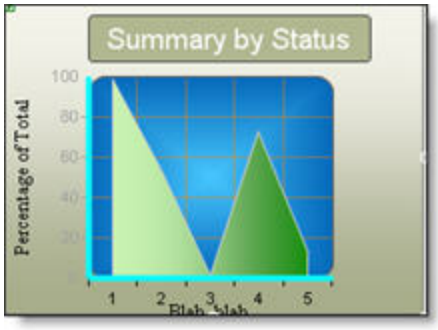
Layout Style: This is the only setting that is available on the X-Axis tab that is not also available on the Y-Axis tab. This setting controls the "padding" of the data along the X-Axis. The differences are best illustrated with these images:



Normal



Inside



Between

As you can see, the "Normal" setting starts and ends your data at the extreme edged of the chart, while

the "Inside" setting provides a full unit of space between the edge of the chart and your displayed data, while "Between" is... well, between the other two in the amount of space it provides.

Axis Label: The axis label settings control the display of the labels along the X and Y axis edges. While the charts in the "Layout style" section above are conveniently in sight, note the positioning of the label settings with respect to the margins - the Y-Axis label is completely visible, while the X-Axis label is partially obscured. We point this out here because although the label settings are controlled on this tab, their spacing from the edge of the chart is not (and in this case, the Y-Axis label spacing has been corrected, while the spacing for the X-Axis label has not). This issue will be addressed under the heading of the "Background tab" section, which follows this section on the X- and Y-axis. Note also that this display issue is also controlled somewhat by the "Data labels" setting, described below.

The axis label section contains the following settings:

- **Text:** This is the text that is displayed along the selected axis.
- **Color:** This setting controls the color of the text which is displayed.
- **Font options:** This button brings up the familiar font dialog, which permits you to select the characteristics of the font used to display the axis label text.
- **Direction:** The settings available here are normal, left and right for each axis. In the case of each axis, the "normal" setting causes the label text to be printed horizontally, while the "left" and "right" settings cause the tops of the letters comprising the title to be oriented toward the left and right (also the same regardless of the selected axis). The "normal" terminology, when applied to the X-axis title, works well; but in this writer's opinion, at any rate, "normal" for the Y-axis should really be applied to the direction labeled "left", which is what is displayed in the charts shown in the "Layout Style" section immediately preceding this one.
- **Alignment:** This setting describes the alignment of the axis label with respect to the axis which it labels. The available settings are "Top", "Middle" and "Bottom" for the Y-axis; and "Left", "Center" and "Right" for the X-axis.

Data Labels: The characteristics of the data labels properties are controlled in this section.

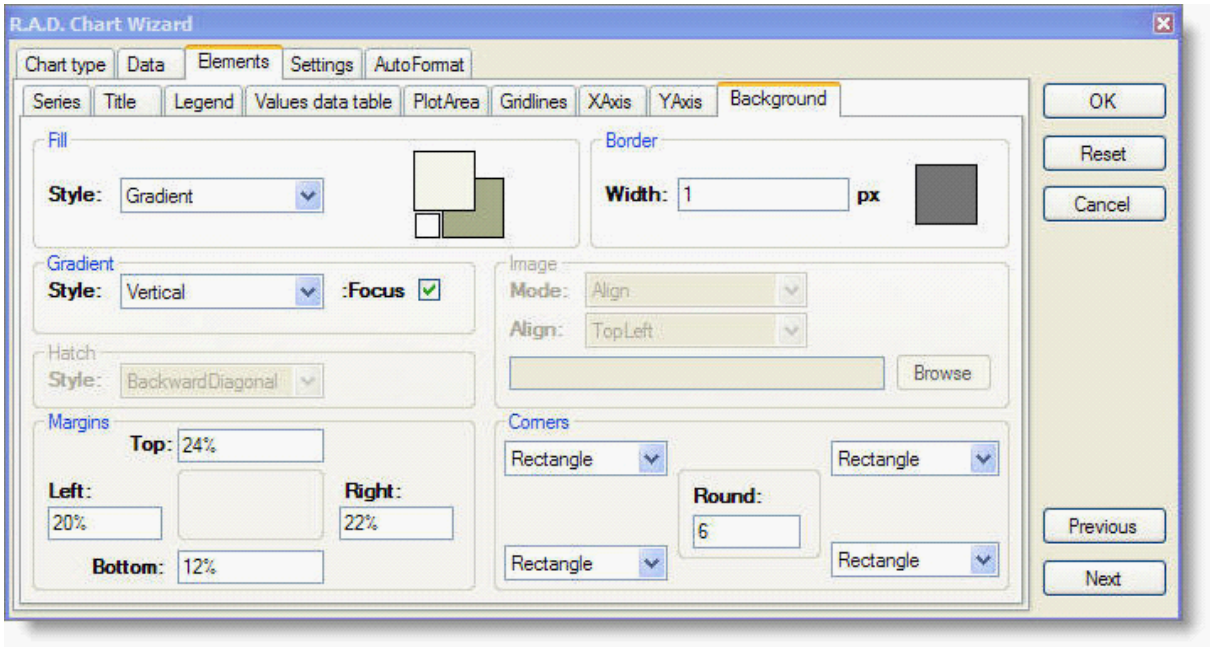
- **Show Labels:** This setting controls whether the labels are displayed or not. This setting, in some respects, interacts with the axis labels, in that there is a certain default amount of distance (in pixels) between the edge of the chart proper, and the edge of the entire component. It is this limiting factor which causes the axis labels to be obscured by the edge of the chart. One means of making more room available for the axis label is to not show the data labels; the best way to see this is to create a simple chart, give the X-axis a label, and then toggle the "Show Labels" setting on and off while watching the real-time update on the design surface.
- **Color and Font Options:** These control the obvious settings for the data labels.
- **Custom format:** This text field allows you to enter a format string for the label data. There are some odd interactions between the entry in this field, and the "Value format" dropdown which appears immediately below it, so sometimes it takes a little trial-and-error to get the labels displayed just the way you'd like. For instance, selecting "Currency" from the drop-down causes the entry in "Custom format" to be ignored completely; however, selecting "Short Date" causes each label to display the contents of the "custom format" field. This is an idiosyncrasy in the 2006 Q4 release which may be addressed in a future release; in the meanwhile, clearing the "Custom Format" field remedies the problem and should be your standard practice.
- **Value format:** Selecting an option from this dropdown will apply the selected format to the data in the labels. As noted in the previous section, there are some odd interactions between the setting selected here, and the value (if any) in the custom format box, so you should make it a

habit to clear that box when selecting an item from the dropdown.

- **Visible values:** Your choices here are "All", "Positive" and "Negative".
- **Rotation:** This setting describes the angle of rotation (expressed in degrees) of the label text. The angle can be either positive or negative.

Background Tab

The settings made on this tab control the appearance of the background area of the chart component which are not otherwise controlled by the other settings available for specific sub-areas.



The Chart Wizard Elements Interface (Background Tab)

Fill, Gradient, Hatch, Border, Image: By now, these settings and their application should be familiar to you - the only difference being the area of the display that the settings are applied to (in this case, the main background area).

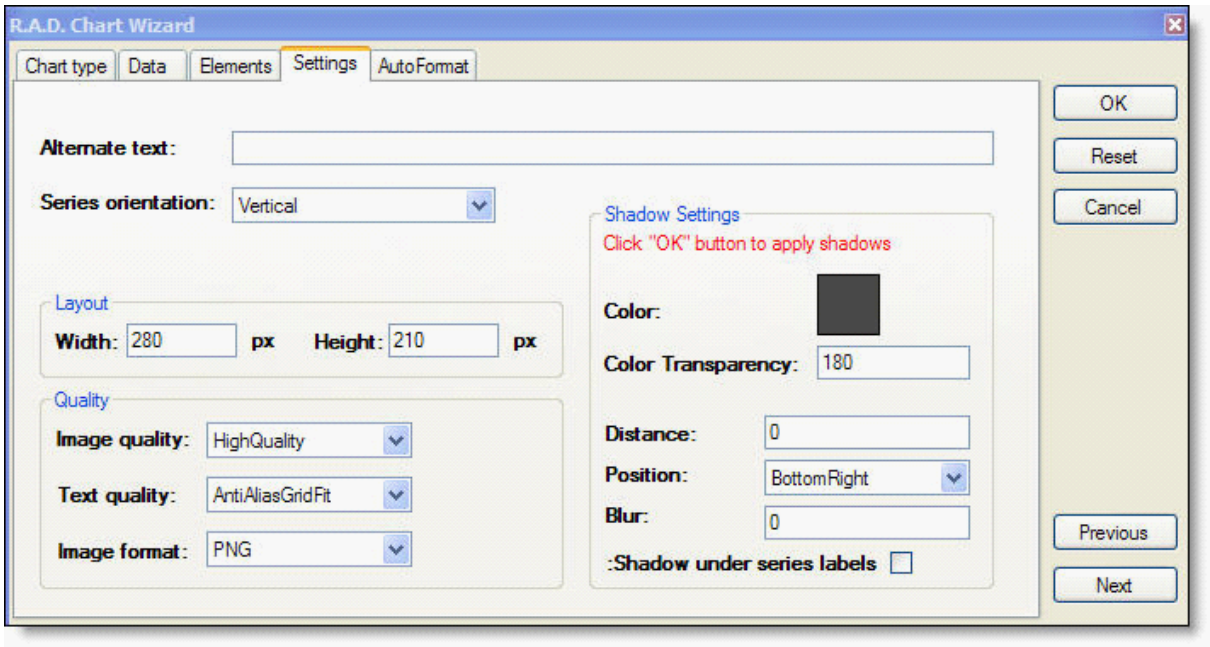
Margins: These settings, for the top, bottom, left and right margins, can be expressed either in terms of a percentage of the total width or height of the chart component which will be reserved as a margin around the chart area; or in terms of the number of pixels that will provide the margin.

Previously, in the Axis: Axis Label section, we mentioned that the appearance of the axis labels was controlled in part by settings in the "Background" section. The margin setting is that setting. Providing a sufficiently wide margin will permit the axis label to be displayed in a readable fashion. Depending on the overall chart size, the default values set by the RadChart component may not provide enough room for the label to be displayed acceptably; however, increasing the margin size here can provide enough room for both axis labels and data labels to be readably displayed.

Corners: This setting is the same as that for series, title, legend, and plot/area, except that it applies to the outside container of the chart component itself, rather than to an individual element.

The Settings Wizard

Another tab on the chart wizard is the "Settings" tab, which provides the ability to set some miscellaneous settings for the chart component.



RadChart wizard displaying the Settings tab

Alternate Text: This is the text displayed when the chart cannot be displayed.

Series Orientation: Toggling this selection between vertical (the default) and horizontal basically toggles the display of the X and Y axis. Note that in doing so, however, the original orientation of the axis labels is preserved. This means that you get the (probably) undesirable result that the bottom axis label now runs vertically centered, and the label on the left-hand side runs horizontally - and likely both run off the edge of the chart. This is easily remedied by going back to the respective axis sub-tab under the elements tab, and correcting the label direction (but don't forget that X- and Y-axis have now traded places).

Layout: This area allows you to set the height and width of the chart area.

Image Quality and Text Quality: Rendering speed versus rendering quality is a tradeoff decision that depends on many factors, and the right choice will not be the same for every site. The properties available in these two dropdowns determine the drawing method used when rendering the chart

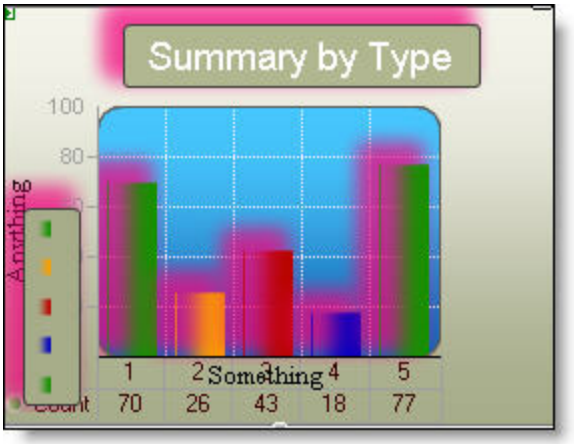
Shadow Settings: These settings allow you to apply a shadowing effect to the title, legend, and elements of the main chart area.

- **Color:** Sets the color used in the shadow
- **Color Transparency:** This value is set on a scale from 0 to 255, with 0 being the most transparent (extremely light, 0 is probably nonexistent), and 255 the most color that can be displayed.
- **Distance:** This indicates the distance (in pixels) that the shadow extends out from the object whose shadow is being cast.
- **Position:** This setting indicates the position of the shadow relative to the object for which it is

being cast, The choices available are the four sides and the four corners of the objects casting shadows.

- **Blur:** This number ranges from 0 (which provides a very distinct edge to the shadow), and ranges upwards from there - the higher the number, the blurrier the edge of the shadow. Be judicious in experimenting with this number and keep it relatively low (the default value is 0). Larger numbers (400, for example) take an inordinately long time to render, and only succeed in creating a big smear effect on the chart.

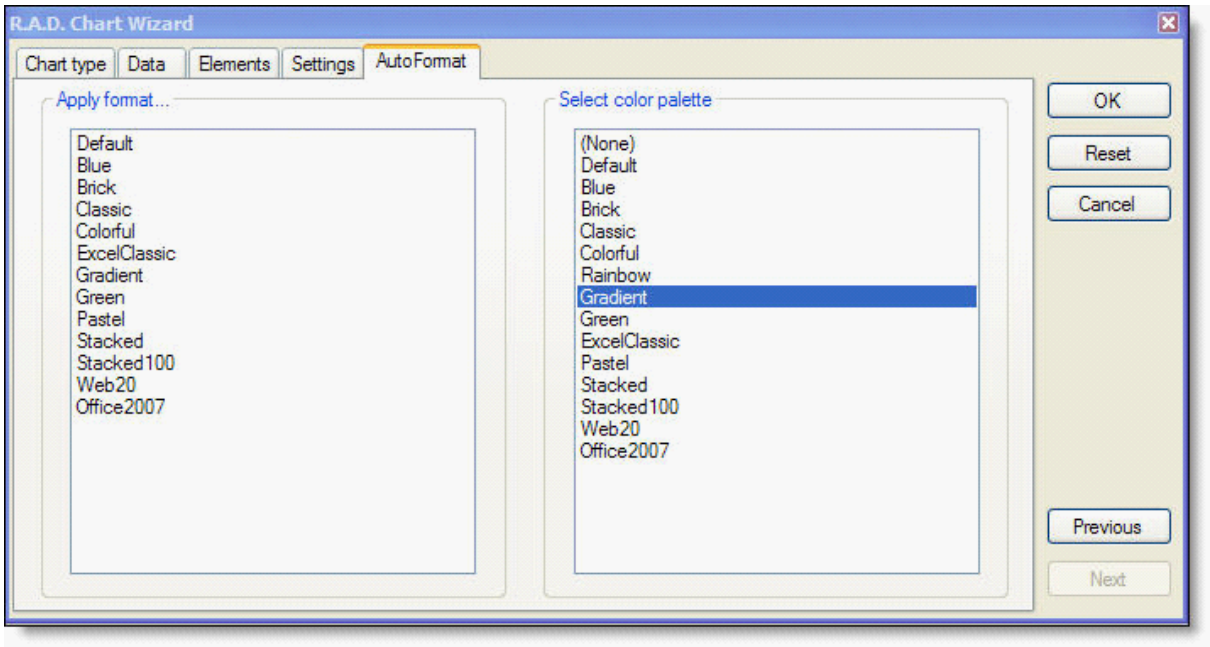
With a little judicious tinkering, you can produce some really hideous looking charts... and if you're seeing this in gray tones rather than full color, consider yourself fortunate...



Honorable mention - 2006 international hideous chart competition

The Autoformat Wizard

The final tab on the RadChart wizard is the "AutoFormat" tab.



The AutoFormat wizard tab

This tab provides a quick and convenient means of selecting from a variety of predetermined styles provided by Telerik. These format and palette selections provide not only the color palette, but also supply some of the other basic formatting elements, such as whether grid lines are displayed and the shape and display of border corners.

Summary

That about wraps it up for the RadChart wizard. The chart is substantially about appearance, and the appearance is controlled by the properties. Of course, these are all available for setting in the regular property window; but learning to navigate through the tabs of the RadChart wizard will make the task of setting these properties immensely easier.

3.1.3 Lab: Combining Chart Types

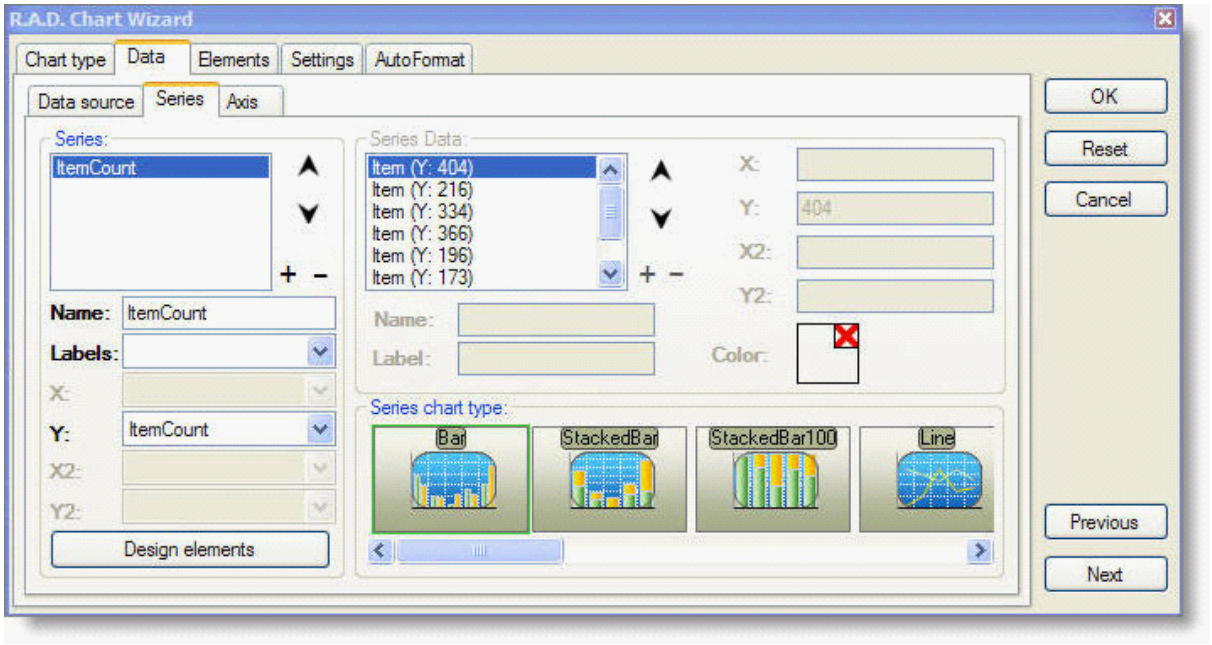
Sometimes you have a need to combine a couple of very different types of data on the same chart - for instance, a chart might present the number of tons of produce harvested from a farm versus the profit realized on the crop. One way to address this would be to provide two separate charts and present them side by side - but even better would be to combine the data into just one chart - and that's what we'll do in this section.

Combining chart data is done on the data - series tab of the chart wizard. Let's take another look at what's available there.

1. If you've been making changes (as suggested) to your property settings as we've worked through the elements of the chart wizard, the easiest thing to do at this point is to delete everything from your design surface, and return to the Getting Started section. Follow the steps there to reinitialize your connection to the Northwind database.

2. Open the chart wizard to the Data - Series tab. Your initial connection will cause your series tab to

look like this.



Selecting a Series Type from the Chart Wizard

Notice that there is a border drawn around the "Bar" chart type, which is how the initial series will be displayed.

3. Next, add some data for a second series which corresponds with the first (remember that the data for the first series was summarized for your convenience in the Getting Started section; that data is also available in the "Series Data" listbox, and it's also summarized in detail on the "Data - Data Source" tab). Suppose, for example, that you wish to plot profit against unit sales. There are eight different sales categories, so you'll want to supply eight profit amounts.

4. Start by adding the new series. Click on the series add icon, and add a series named "Profit".

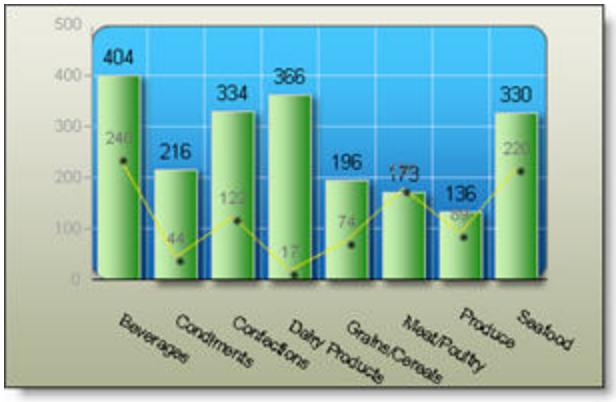
5. Add items to the series. click on the series data add icon, enter the name in the label, and the value in the box for the "Y" value. Possible values would be: 240, 44, 122, 17, 74, 180, 89, 220 (make sure that there are eight values, to correspond to the eight values returned from the Northwind table).

6. In the "Series Chart Type" display, select the "Line" chart type.

7. On the "Data - Axis" tab, uncheck the X-Axis Autoscale box. In the Binding - Labels Column dropdown, select "CategoryName".

8. Select "Elements - X-Axis". In the data labels section, make sure that "Show Labels" is checked. Because of the length of the labels, set the "Rotation" angle to 30 .

If you close the wizard and inspect your chart on the design surface at this point, you'll see a chart that looks something like this:



Superimposed chart series

3.1.4 Lab: Responding to Events

There is only one client-side event generated by the chart component, and that's the chartClick event. The event handler for the chartClick event receives a ChartClickEventArgs parameter, which provides access to four properties of interest:

- **IsLegendItem** tells you is the item that was clicked on is a legend item or not.
- **LegendItem** is the entire ChartLegendItem object that was clicked on.
- **Series** is the entire ChartSeries object that was clicked on.
- **SeriesItem** is the entire ChartSeriesItem object that was clicked on.

Making use of the information returned permits you to do some interesting things. Let's take a look at a couple of them.

1. Start once again with the basic lab starting point described in the Getting Started section; or simply use the chart from the Combining Chart Types section (note, however, the returned ChartClickEventArgs will only return the data for the first series). Set the chart type to be a pie chart.

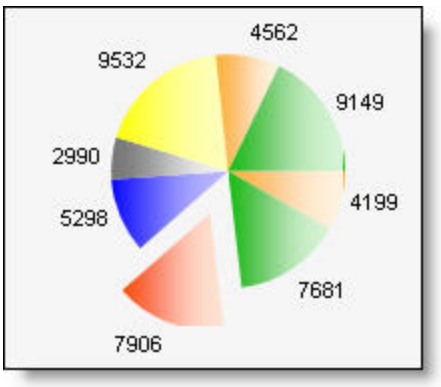
2. Select the chart component on the design surface, then open the events area of the properties window. Double-click in the "click" event area to create the ClickChartEvent stub.

3. One of the nice features of the pie chart is the ability to "explode" the section of the chart that has been selected. Add the following code to the click event in the code behind:

```
if   (!args.SeriesItem.Exploded)
{    // Allow only one exploded item at a time.
     // If the selected item isn't exploded already, bring everything else back in...
     foreach (ChartSeriesItem item in args.Series.Items)
         item.Exploded = false;
     // ... and explode the selected item outward
     args.SeriesItem.Exploded = true;
}
```

The code first checks to see if the item that was clicked is already exploded. If it's not, we make sure

that each segment of the pie chart is unexploded, and then set the exploded property for the item that was clicked. Compile and run your project, and click a couple of different segments of the pie chart to observe the behavior.



An exploded pie chart

Of course, you can do a lot more with the information that is passed in with the event arguments. Let's say that you wanted to see a breakdown of the products sold within the selected category sorted in descending order of the total sales amount, and have the results displayed in a grid (a RadGrid, of course!). How would you go about that?

4. Add a RadGrid component to your design surface. Make the initial state be visible=false.

5. Add the following code to the end of the chartClick event we started above:

```
C# Example:
System.Text.StringBuilder sb = new System.Text.StringBuilder();
sb.Append("select ProductName, sum([order details].UnitPrice * Quantity) \"Sales\" ");
sb.Append("from categories, products, [order details] ");
sb.Append("where categories.categoryid = products.categoryId ");
sb.Append("and products.productId = [order details].productId ");
sb.Append("and categoryname='" + args.SeriesItem.Label + "' ");
sb.Append("group by ProductName ");
sb.Append("order by sum([order details].UnitPrice * Quantity) desc");

ConnectionStringSettingsCollection connectionStrings =
    ConfigurationManager.ConnectionStrings;
string _connectionString =
    Convert.ToString(connectionStrings["NorthwindConnectionString"]);
SqlConnection conn = new SqlConnection(_connectionString);
SqlDataAdapter da = new SqlDataAdapter(sb.ToString(), conn);
da.SelectCommand.CommandType = CommandType.Text;
DataSet ds = new DataSet();
da.Fill(ds);
RadGrid1.DataSource = ds.Tables[0];
RadGrid1.DataBind();
RadGrid1.Visible = true;

VB Example:
Dim sb As System.Text.StringBuilder = New System.Text.StringBuilder
sb.Append("select ProductName, sum([order details].UnitPrice * Quantity) ""Sales"" ")
sb.Append("from categories, products, [order details] ")
sb.Append("where categories.categoryid = products.categoryId ")
sb.Append("And products.productId = [order details].productId ")
sb.Append("And categoryname='' + args.SeriesItem.Label + '' ')
sb.Append("group by ProductName ")
sb.Append("order by sum([order details].UnitPrice * Quantity) desc")
```

```
Dim  connectionStrings As  ConnectionStringSettingsCollection  =  _
    ConfigurationManager.ConnectionStrings
Dim  _connectionString As String = _
     Convert.ToString(connectionStrings("NorthwindConnectionString"))
Dim conn As  SqlConnection  = New   SqlConnection(_connectionString)
Dim da As  SqlDataAdapter  = New  SqlDataAdapter(sb.ToString,  conn)
da.SelectCommand.CommandType   =   CommandType.Text
Dim ds As DataSet = New  DataSet
da.Fill(ds)
RadGrid1.DataSource  =  ds.Tables(0)
RadGrid1.DataBind
RadGrid1.Visible  = True
```

6. There's one other thing you need to do to get this to work, and that's to make sure that the category name is available for use by the SQL query we constructed - it will be passed in as the args.SeriesItem.Label, but only if you reference it in the chart component. In order to do that, open up the chart wizard, go to the "Data - Series" tab, and select "CategoryName" in the Series Labels dropdown. Now when you run your project and click on one of the pie segments, the RadGrid will be populated with the summary of sales by item from the selected category.

A related way to display detail data is to take advantage of the "ImageMap" property available on various artifacts of the RadChart component. For instance, let's say that we wanted to display the same basic data as in the preceding example, except that we want to show it in a popup window. We would accomplish this by setting the value of the ImageMap property for each item in the series.

7. Disable the ClickChartEvent by deleting the onClick event handler in the property window,

8. The next step is to hook up a form-specific PreRender event, to update the ItemMap property of each ChartSeriesItem (the ItemMap property functions in the same manner as the ImageMap property for Series and Legends). Add the following to your code:

```
C#  Example:
protected override void  OnInit(EventArgs  e)
{
     InitializeComponent();
    base.OnInit(e);
}
private void  InitializeComponent()
{
    this.PreRender  += new   EventHandler(ProblemDemo_PreRender);
}

void  ProblemDemo_PreRender(object  sender,  EventArgs  e)
{
    if  (!IsPostBack)
    {
        foreach  (ChartSeriesItem  item in  RadChart1.Series[0].Items)
        {
             item.ItemMap.Attributes = "Target=_blank";
            item.ItemMap.HRef = "DetailDisplay.aspx?itemID="  +  item.Label;
             item.ItemMap.ToolTip = "This  is  for  "  +  item.Label;
        }
    }
}

VB  Example:
Protected  Overloads  Overrides Sub  OnInit(ByVal e As  EventArgs)
  InitializeComponent
  MyBase.OnInit(e)
```

```
End Sub

Private Sub  InitializeComponent()
 AddHandler  Me.PreRender, AddressOf  ProblemDemo_PreRender
End Sub

Sub  ProblemDemo_PreRender(ByVal  sender As Object, ByVal e As  EventArgs)
 If Not  IsPostBack Then
   For Each item As  ChartSeriesItem In  RadChart1.Series(0).Items
       item.ItemMap.Attributes = "Target=_blank"
       item.ItemMap.HRef = "DetailDisplay.aspx?itemID=" + item.Label
      item.ItemMap.ToolTip = "This is for " + item.Label
   Next
 End If
End Sub
```

The first two methods take care of telling ASP.Net that you want to add an event handler to the form's PreRender event. The last method steps through the items in the ChartSeries collection, setting properties as you see them above. Setting the **Attributes** to Target=_blank tells the application that whatever it is you're about to do, you want it to be displayed in a new Internet Explorer window (note that you don't want to get too fancy in the new window, because it will share the same session state as your original window). On this line also, you can set characteristics of the new window such as height, width, chrome, and several other characteristics. The **HRef** setting is where you set the page that you want to open up. You could as easily have set this to "http://www.telerik.com", for instance, if that suited your purposes. Finally, the **ToolTip** setting is available to set the text of the bubble help that pops up when the mouse cursor passes over the related area of the chart.

9. Finally, add a new web form to your project, named "DetailDisplay.aspx". Since this is just to prove the concept, you can put something really mundane on it, like "This is a proof of concept".

10. Now run your project. Note that as your cursor passes over the bars in the chart, you see bubble help pop up with the name of the bar you're passing over. Click on one of the bars - you should see a new browser window open up with the very exciting text message we entered in the last step. Also, if you look at the url, you should see the parameter we passed in containing the itemId. Knowing this, you also know that you're on your way - you can use that url parameter to display specific data from a database, display a graphic or video image, open a pdf file - the point is, the information is passed directly as a result of the HRef setting you made in your application.

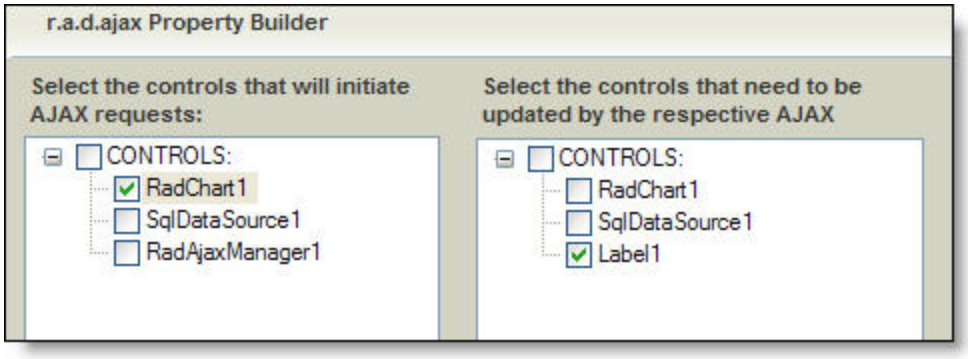
3.1.5 Lab: AJAX Enabling the RadChart control

One of the great strides forward in web technology is the advent of AJAX, and the ability to communicate with the application on the server and update your forms without the need for a full postback. Telerik has made this even easier with their AJAX-oriented controls. In this section, we'll demonstrate how to use an AjaxManager component to update another control on the form in response to a click event on the RadChart.

1. Start once again with the basic lab starting point described in the Getting Started section. Set the chart type to be a bar chart.

2. Add two components to the form: a regular label, and a Telerik AjaxManager component.

3. Set the configuration of the AjaxManager component so that it causes events fired from the chart to affect the label. To do so, open the smart tag on the AjaxManager, and select "Configure Ajax Manager". In the first column, check the box for the controlling component (the chart). In the middle column, select the label. You may leave the third column blank. When you're done, the AJAX property builder should look like this:



radAjax Property Builder

Close the property builder and the smart tag.

4. In order to make sure the label data stays available in the control (at least when you're getting that data from a database connection), you need to do something that this author finds to be just a bit counterintuitive: you need to set the "EnableViewState" property of the chart control to false. If you leave it true, it will work the first time - and not after that. If you're dealing with data that is added at design time, you don't have the same issue; if you're going to add data items programmatically, you'll want to check the behavior for yourself.

5. Because in this lab we're going to extract the label of the series item that was clicked and display it in the text of the label we added, we need to make sure it's available. Open up the chart wizard, and select the Data - Series tab. In the "Series - label" dropdown, select "categoryname". On the Elements - Series tab, make sure that the "Labels - Show" box is un-checked (that's because if you show this label, it gets kind of jammed up at the top of each item, and doesn't look very good).

6. We also need to add the event handler to handle populating the label text. Open the event handler section of the chart's property sheet, and double-click the "Click" event to create the method prototype. add the following code to the click event:

```
protected void RadChart1_Click(object sender, Telerik.WebControls.ChartClickEventArgs args)
{
     Label1.Text = args.SeriesItem.Label;
}
```

7. Run the project. Click on various bars in the chart, and notice that the label text gets updated with the label corresponding to the bar that you clicked.

Of course, this is about the simplest use you can make of this powerful capability, and there are many more uses to which it could be put. For instance, you could respond to a selection by populating a grid or fields on a form based on the selection made by a user.

3.1.6 Summary

This chapter covered the RadChart component. In it, we took an extensive look through the chart wizard.

The wizard is your friend. It organizes and presents the various properties together in logical groupings, and makes properties easier to find. It enables and disables settings groups in a context-sensitive manner (that is to say, for instance, that if a fill style is "Gradient", picking a hatch style is irrelevant. In the regular property sheets, the hatch style is still active; in the wizard, it's disabled). Many property settings result in an immediate update of the component displayed on the design surface, much more so than changing settings in the property editor.

We then looked at combining different chart types, superimposing a line chart onto a bar chart. We saw how the chart could respond to click events in several different ways, both with a postback (populating the elements of a grid or expanding a slice of the pie chart), and without a postback (by hooking a url link up to a series item, then posting a new page based on the item that was clicked). Finally, we looked at the RadChart interacting with RadAjax on the web page.

3.2 RadEditor

3.2.1 Getting Started

RadEditor is a feature intense control from Telerik that acts as a Word®-like, WYSIWYG editor for managing rich text and html content live, right within your website. This control has so many aspects it comes with its own end-user manual (see the Telerik Support | Documentation area for help file downloads).

RadEditor in the toolbox

Just to name a very few features:

- Configurable everything: toolbars, localization, skinning, control configuration file, style sheets for the control and its content.
- Save to a database using the *SubmitClicked* server side event or save the content right inside the control in the page aspx file itself.
- Built in Ajax spell checking or use a RadSpell control. We will cover RadSpell in more depth in a later chapter so we won't go into it here except to mention that its supported.
- File managers that allow the user to upload files and media to the server, to add files to editor content and to be able to delete files. The ability to specify paths, file filters and maximum file size is entirely configurable for each type of supported media (e.g. documents, Flash, media, images, template, links).
- The ability to paste from MS Word with or without formatting. You may need a way to strip out MS Words well known verbose markup before accepting the content into your system.
- Built-in modules for displaying word count statistics, tag and property inspectors and XHTML validator to name a few.
- Localization support for the editor UI features, i.e. dialogs, drop down lists, tool bar titles, submit and cancel buttons, and all the file manager dialogs. See the Telerik RadControls Editor directory in the Localization folder to see what is covered for a specific locale. Right to Left (RTL) layout also supported.
- A huge server API that provides access to all the RadEditor collections including fonts, modules, toolbars, links, colors, snippets, and css classes.
- A client scripting API that provides rapid response directly in the browser to not only RadEditor features but the underlying DOM as well.

3.2.2 Using RadEditor in the Designer

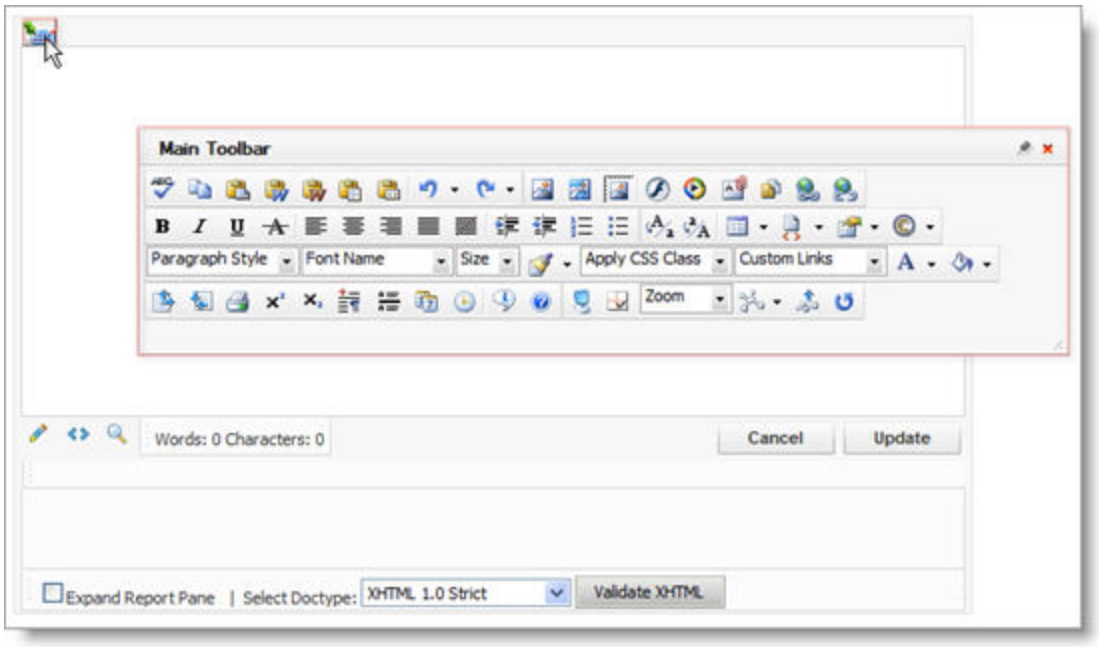
Toolbars

Toolbar Mode

RadEditor comes with a set of toolbars that control everything from font selection to inserting MS Word documents to executing your own custom commands. The *ToolBarMode* is a pivotal property because it

determines how the screen real estate will be used.

- **Default:** The tool bars appear just above the content area. You can still drag the tool bars outside the editor.
- **Floating**: If you want to disconnect the toolbar from the editing area either to preserve the dimensions of the content area or to allow the buttons to be available during scrolling use the Floating mode. The user can also "pin" the toolbar using the pin button in the upper right hand corner.



Toolbar in Floating mode

- **PageTop**: In this mode the user clicks on the editor and the toolbar displays at the top of the page. Use PageTop when you have multiple editors but only want to use the real estate at the top of the screen for each.
- **ShowOnFocus**: The toolbar appears right above the editor when the editor gets focus.

ToolsFile XML

The entire set of toolbars can be completely customized by way of a file called ToolsFile.xml. The file contains all the modules, links, tool bars snippets, colors, in fact just about everything you can see in the editor. You can assign the xml file by way of the *ToolsFile* property which defaults to \RadControls \Editor\ToolsFile.xml.

Note: You can also assign ToolsFile in code in case you want to build the xml dynamically. Using the XmlDocument Load() method in this case has the same result as assigning the *ToolsFile* property, but you can build the XML completely at runtime using XmlDocument methods.

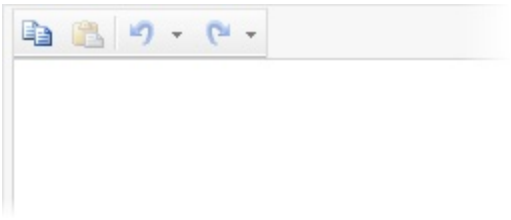
```
XmlDocument toolsFile = new XmlDocument();
toolsFile.Load(Server.MapPath("~/RadControls/Editor/ToolsFile.xml"));
RadEditor1.LoadToolsFile(toolsFile);
```

You might want to limit the number of choices for the user as in the example below where only "MainToolbar" is enabled and only four buttons on the toolbar show. This sample also sets dockable to false.

```
<root>
  <modules>...
  <tools name="MainToolbar" isribbon = "false" dockable="false" enabled="true">...
  <tools name="InsertToolbar" isribbon = "false" enabled="false">...
  <tools name="Formatting" isribbon = "false" enabled="false">...
  <tools name="Insert"  isribbon = "false" enabled="false">...
  <tools name="DropdownToolbar" isribbon = "false" enabled="false">...
  <tools name="DialogToolbar" dockingZone="Top" enabled="false" isribbon = "false">...
  <tools name="EnhancedEditToolbar" dockingZone="Bottom" dockable="true" enabled="true">...
  <links>...
  <snippets>...
  <symbols></symbols>
  <fontNames> </fontNames>
  <fontSizes></fontSizes>
  <colors>...
  <paragraphs>...
  <classes></classes>
  <dialogParameters></dialogParameters>
  <languages></languages>
  <contextMenus></contextMenus>
</root>
```

Setting toolbar Enabled attribute

The resulting toolbar looks like this:



MainToolBar with limited number of buttons

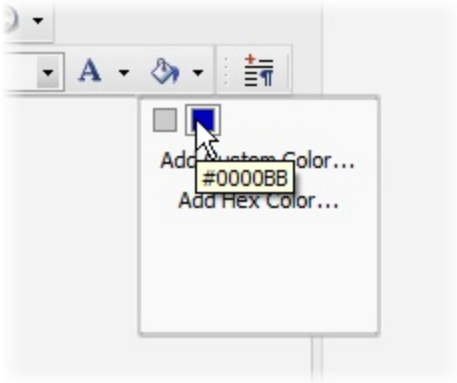
You may have noticed the "isribbon" attribute. When "isribbon" is true the toolbars take on the Office 2007 "ribbon bar" look:

"IsRibbon" is true for these toolbars

To make specific choices available to the user, edit elements of ToolsFile.xml that control color, font names, snippets, links etc.. Here are two colors added that show up in the color drop down:

```
<fontNames> </fontNames>
<fontSizes></fontSizes>
<colors>
  <color value="#CCCCCC" />
  <color value="#0000BB" />
</colors>
<paragraphs>
</paragraphs>
<classes></classes>
<dialogParameters></dialogParameters>
```

Add two new color elements



New color elements in the Dropdowns toolbar

Another handy ability of the ToolsFile is to associate context menus with various elements of HTML. For example, you can have "justify" context menus available when you right click a paragraph element:

```
<contextMenu  forElement="P">
   <tool name="JustifyLeft" />
   <tool name="JustifyCenter" />
   <tool name="JustifyRight" />
   <tool name="JustifyFull" />
</contextMenu>
```

To find definitions for all the elements and attributes in ToolsFile.xml look either at the online help or refer to the comments at the end of ToolsFile.Xml itself.

Other Toolbar Properties

The *EnableClientSerialize* property when true saves toolbar and modules settings to a cookie and restores them automatically during the next session.

ToolsHeight, ToolsWidth set the respective height and width for the toolbar and may come into play based on the Toolbar Mode. By default the toolbars will be the width of the content area, but if the toolbars are Floating over a narrow editable area you may need to explicitly set *ToolsWidth.*

ConfigFile XML

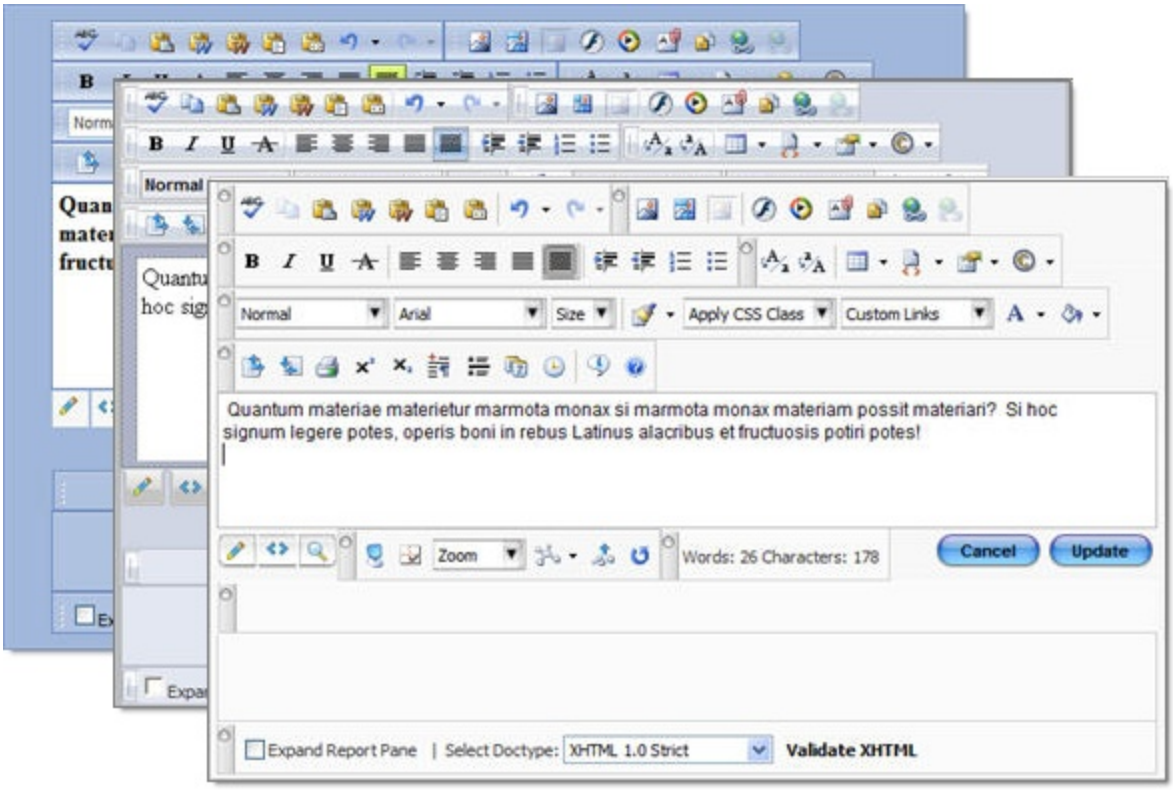
While ToolsFile configures tools, modules, links and other items in the editor, ConfigFile.xml has settings for all the properties of the Editor itself. You can think of ConfigFile.xml as the superset of properties for the editor because it defines where ToolsFile.xml, skins and other customization entities are located. Comments in the head of in ConfigFile.xml list what elements and attributes are appropriate. You can override settings made in ConfigFile.xml in the actual RadEditor tag.

```
<configuration>
  <property name="ToolsFile">~/RadControls/Editor/ToolsFile.xml</property>
  <property name="SkinsPath">~/RadControls/Editor/Skins</property>
  <property name="Skin">Default</property>
  <property name="AllowCustomColors">true</property>
  <property name="SaveAsXhtml">false</property>
  <property name="Width">630px</property>
  <property name="Height">400px</property>
  <property name="ToolsHeight">70px</property>
  <property name="StripAbsoluteImagesPaths">true</property>
  <property name="StripAbsoluteAnchorPaths">true</property>
  <property name="ConvertFontToSpan">false</property>
  <property name="FocusOnLoad">false</property>
  <property name="AllowThumbGeneration">false</property>
  <property name="ThumbSuffix">Thumbnail</property>
  <property name="SaveInFile">false</property>
  <property name="CausesValidation">true</property>
  <property name="Editable">true</property>
  <property name="HasPermission">true</property>
  <property name="MaxImageSize">204800</property>
  <property name="MaxFlashSize">102400</property>
  <property name="MaxMediaSize">5242880</property>
  <property name="MaxDocumentSize">512000</property>
```

Sampling of properties available in ConfigFile.xml

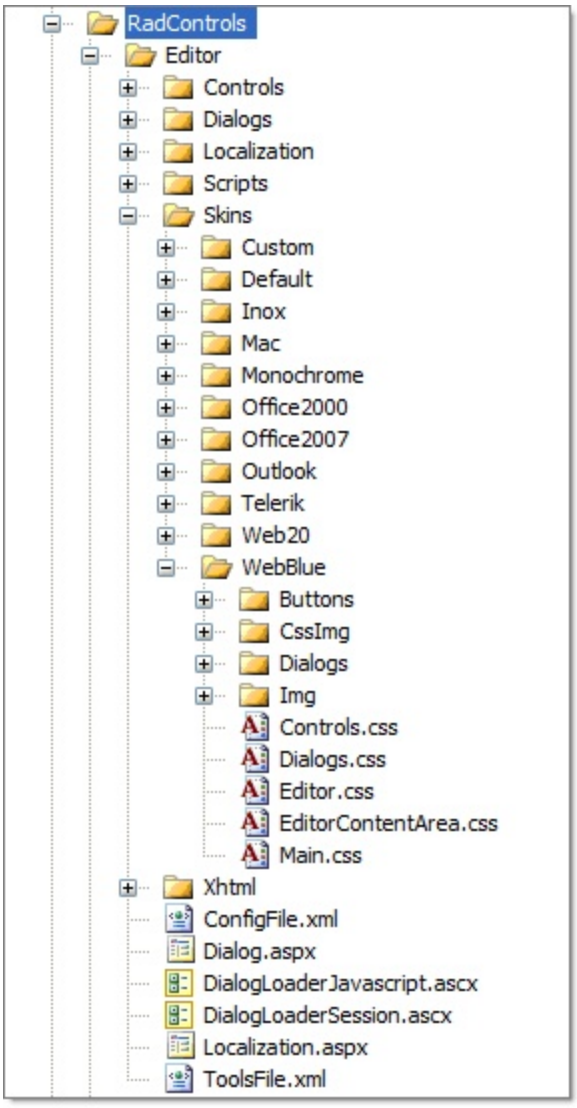
Look and Feel

Another way to customize RadEditor at design time is through Skins and style sheets. Skins can be changed right in the designer by selecting from the drop down in the properties window.



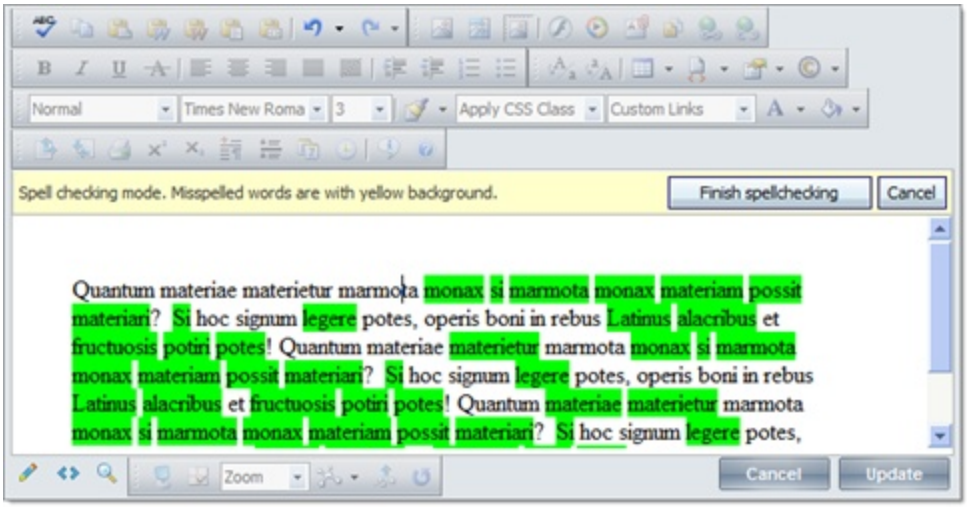
Skins for Web20, Inox and Mac

As with the other Telerik controls RadEditor gets skinning information by setting up a RadControls directory in the project and copying the files from the Telerik installation directory. There are more support directories for RadEditor than for the typical Telerik control. Style sheets are defined for every portion of the RadEditor anatomy including where EditorContentArea.css defines content area appearance.



RadControls Editor directories

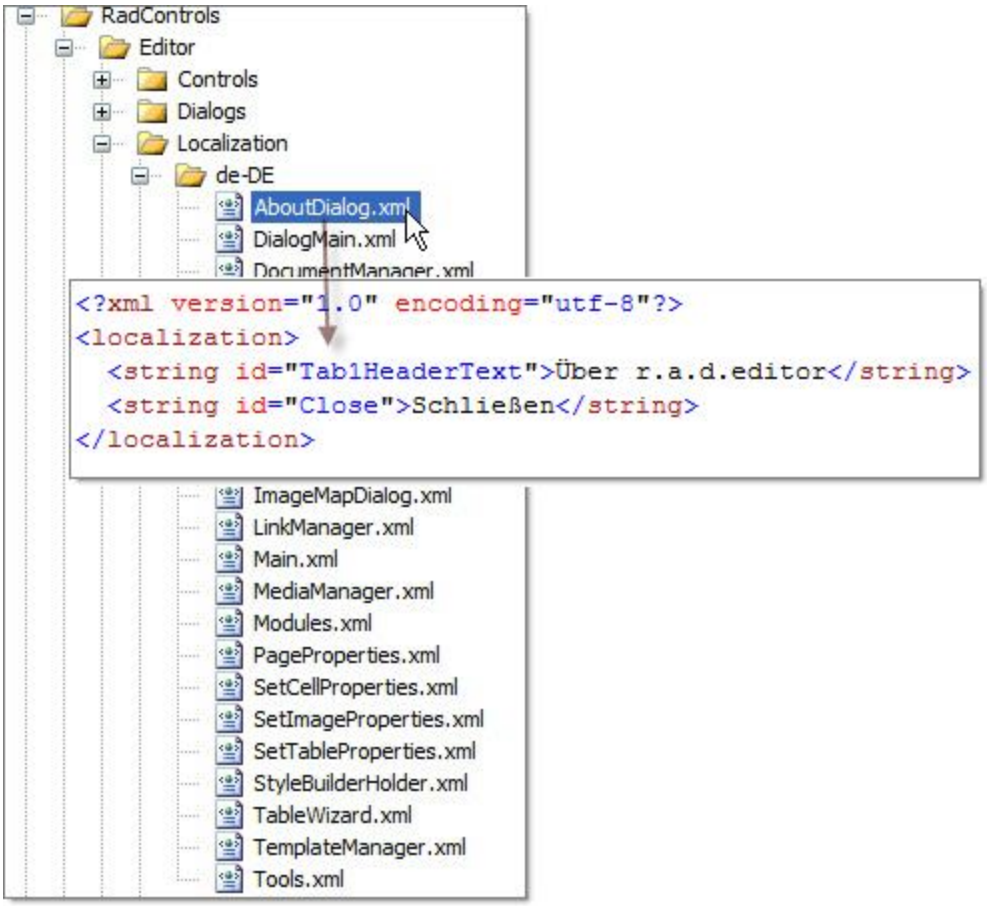
We can change a class in EditorContentArea.css called "RadEContent" to pad and margin a full 20 pixels and set the "RadEWrongWord" class background color to Lime. If you notice in the figure below the spell checking area has a message that misspelled words have a yellow background. This message can be fixed as well by editing Main.xml in the localization folder under "en-us".



EditorContentArea.css changes

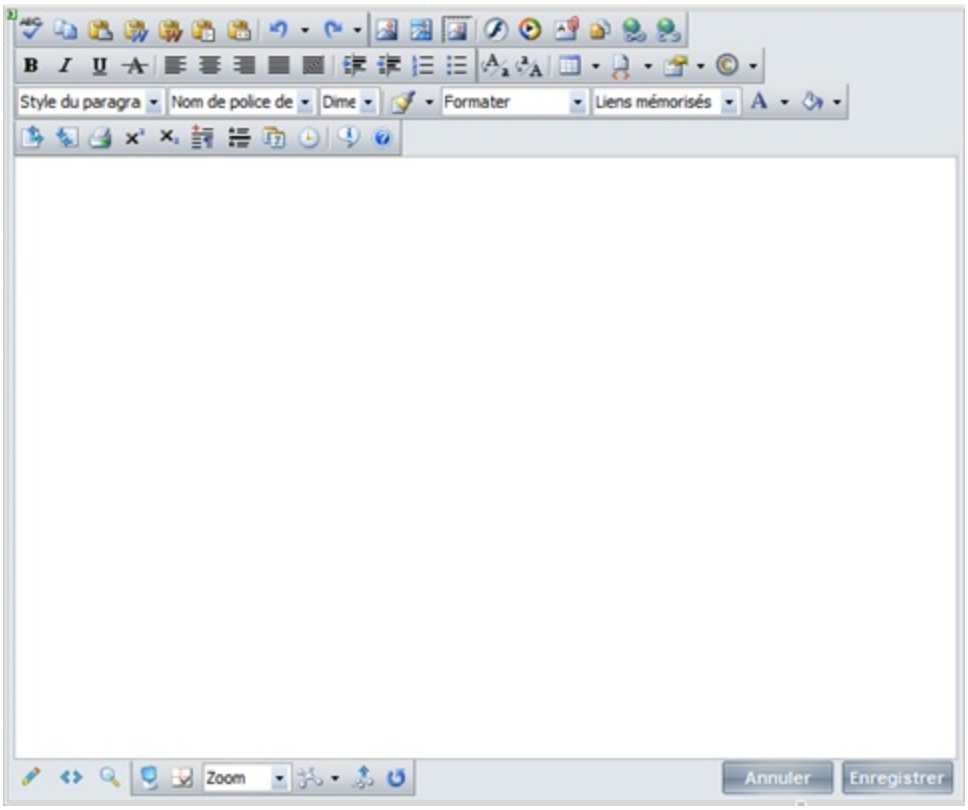
Localization

Related to appearance is the ability to localize the editor for a specific culture. The control ships with the default of US English, French and German. The folder for each language contains a series of XML files that contain the translation for various parts of the editor UI.



Opening the AboutDialog German translation

Setting the *Language* property of the editor to a culture code causes the editor to make use of the translations. You can make the change in the designer and see it reflected immediately. In the figure below Language has been set to French with the culture code of "fr-FR".



Editor at design time with Language "fr-FR"

RadEditor also supports right-to-left (RTL) text layout by way of markup and changes to the style sheets. Briefly, you need to set the HTML body tag "dir" as being right-to-left:

```
<body dir="rtl">
```

...and set the EditorContentArea.css RadEContent class alignment and text direction:

```
text-align: right;
direction:rtl;
```

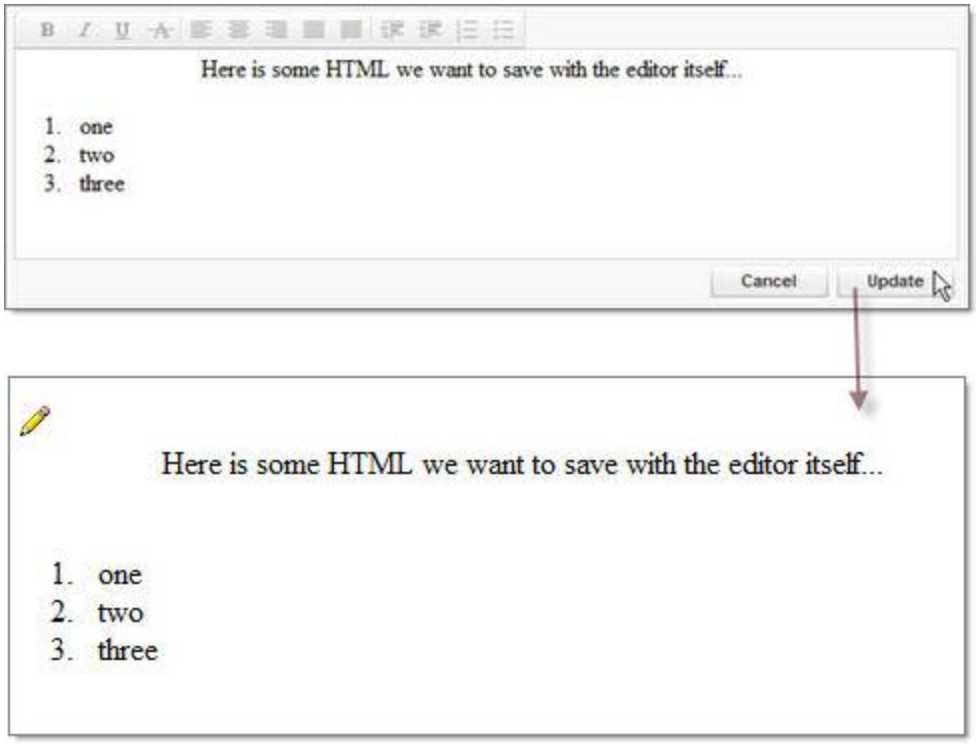
Similar steps need to be taken for the dialog and editor style sheets. The complete directions are in http://www.telerik.com/help/radeditor/v7_NET2/editorRTLsupport.html.

Saving To File

RadEditor has a handy ability to save content directly in the aspx or ascx file that contains the editor. This can be used where the site administrator wants to set up relatively static content but doesn't want to connect to the database. All it requires is that the *SaveInFile* property be set "true".

```
<radE:RadEditor ID="RadEditor1" runat="server" SaveInFile="true">
</radE:RadEditor>
```

Once the user/administrator edits the file and hits the Update button the editor changes to preview mode with the editing pencil displaying in the upper left hand corner of the page.



SaveInFile property in action

Now the markup will have content between the RadEditor tags:

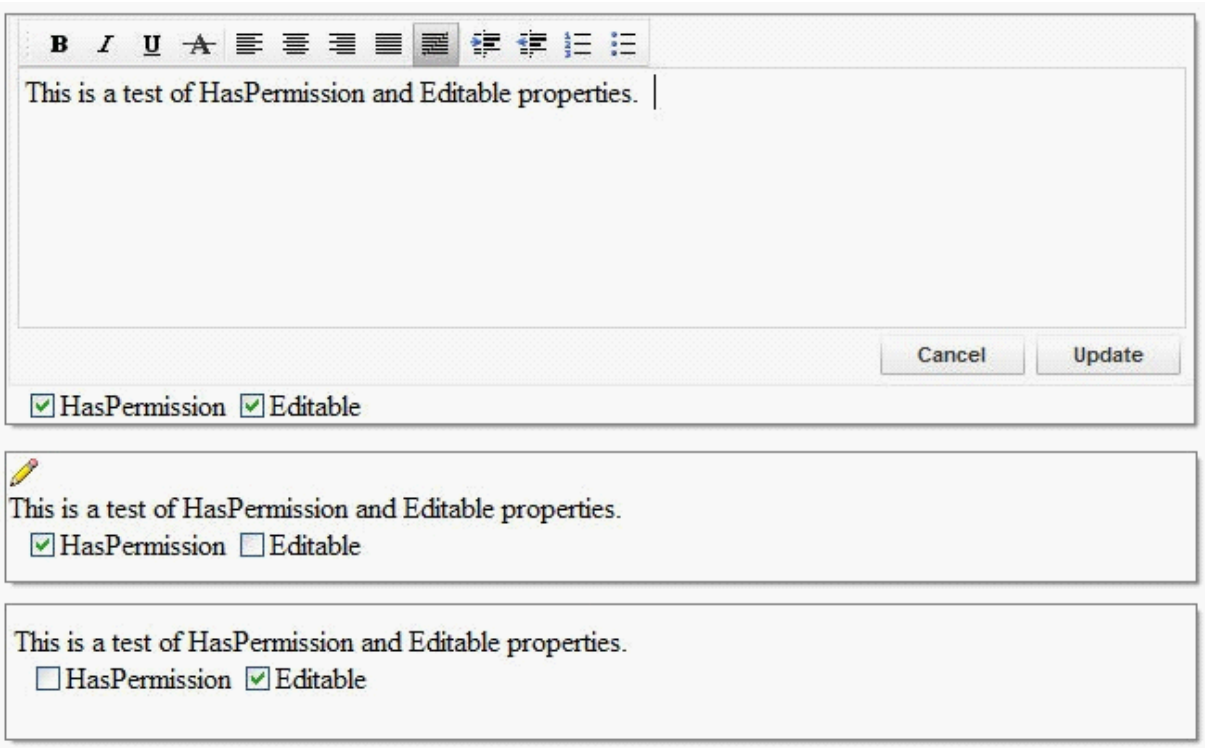
```
<radE:RadEditor  ID="RadEditor1"  runat="server"  SaveInFile="true">
   <p  align="center">
     Here  is  some  HTML  we  want  to  save  with  the  editor  itself...<br>
  </p>
  <ol>
     <li>one</li>
     <li>two</li>
      <li>three</li></ol>
</radE:RadEditor>
```

We will look next at the properties for editing and permissions that you might use along with *SaveInFile* to display the editor differently for administrators vs. end users.

Important Note from Telerik help: "Sometimes when you simultaneously browse and edit your web application, Visual Studio .NET might lock the file, and the editor will not be able to save the content directly to the page. When the project is deployed on a web-server and is not accessed directly by Visual Studio .NET or other IDE this problem does not appear."

Editing and Permissions

When using the *SaveInFile* property you will want to control who can edit the page. *HasPermission* when false prevents editing or the display of the edit pencil icon. Editable when true displays the editor already in edit mode.



The effect of HasPermission and Editable properties

File Managers

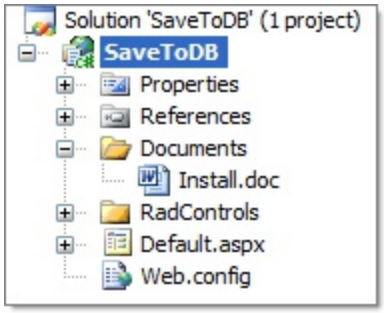
RadEditor has the facility to insert portions of external content to the content area. Documents, Flash, media, images, template, and links all have manager dialogs that take care of selecting and inserting these types of content. See "InsertToolbar" in ToolsFile.xml for the available list. File managers also allow the user to upload material to the server.

To use any of the file managers you need to set up paths for where the files will go in ConfigFile.xml or directly in the markup for the editor. This markup defines the locations for documents that may be viewed, uploaded, and where documents may be deleted from. The *DocumentsFilters* indicate that only *.doc and *.txt types will be shown in the list. The *DeletedDocumentsPaths* prevents users from deleting files except where designated, and *MaxDocumentSize* helps prevent abuses of the system by capping the number of bytes that can be transferred. A similar pattern of paths, filters and max size properties govern each file type for flash media, images, templates and links.

Note: The tilde as per convention indicates the root path in your web application.

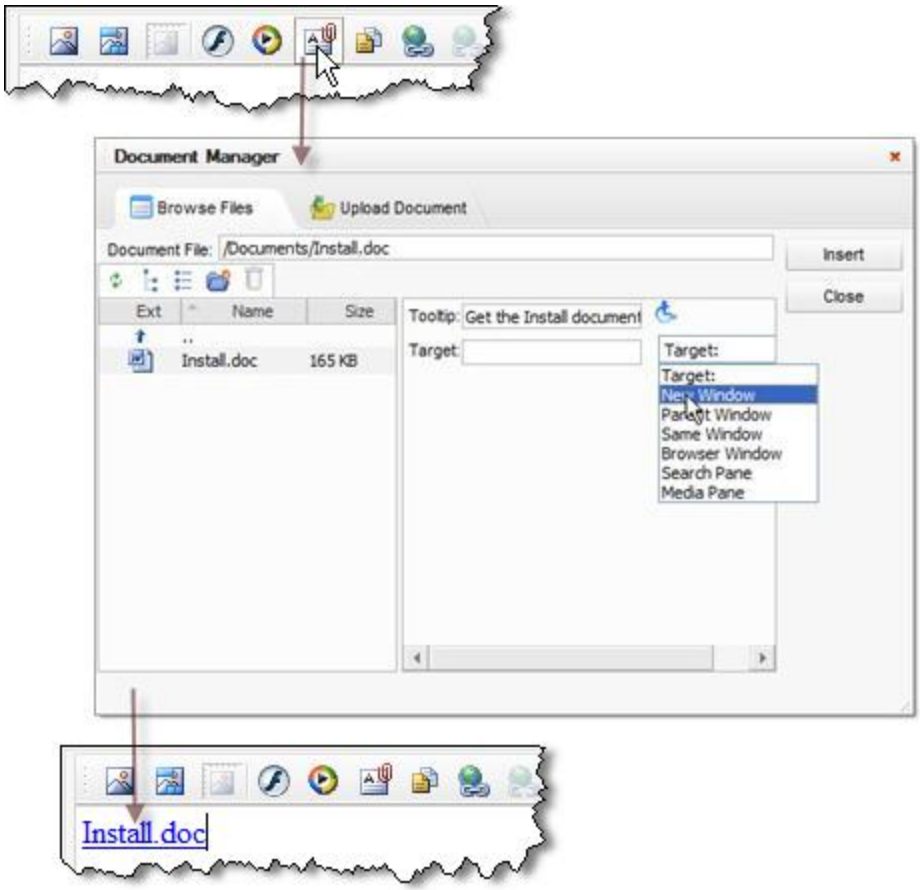
```
<radE:RadEditor  ID="RadEditor1"  runat="server"
   Height="200px"
    documentspaths="~/Documents"
    uploaddocumentspaths="~/Documents"
    deletedocumentspaths="~/Documents"
   DocumentsFilters="*.doc,  *.txt"
   MaxDocumentSize  ="512000">
</radE:RadEditor>
```

The directory structure of the project in this case looks like this figure. Notice the location of the documents directory.



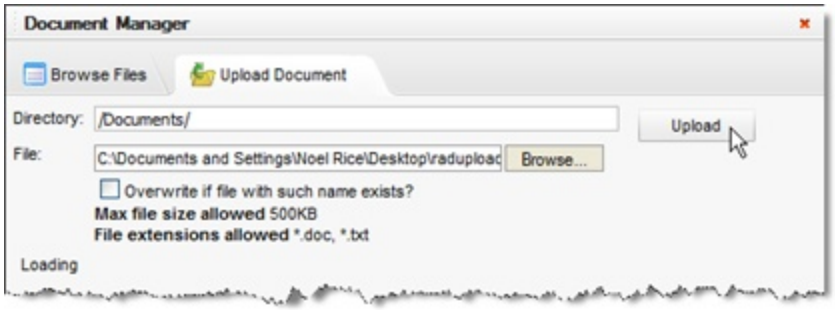
Project structure with documents directory

Now when the editor button is clicked the Document Manager dialog will display and list files from the documents path. To include a document in the content area the user selects the document, optionally types a Tooltip and selects a Target from the Target drop down list. When the user clicks the Insert button a link to the document along with tooltip and target information are created in the document.



Using the document manager

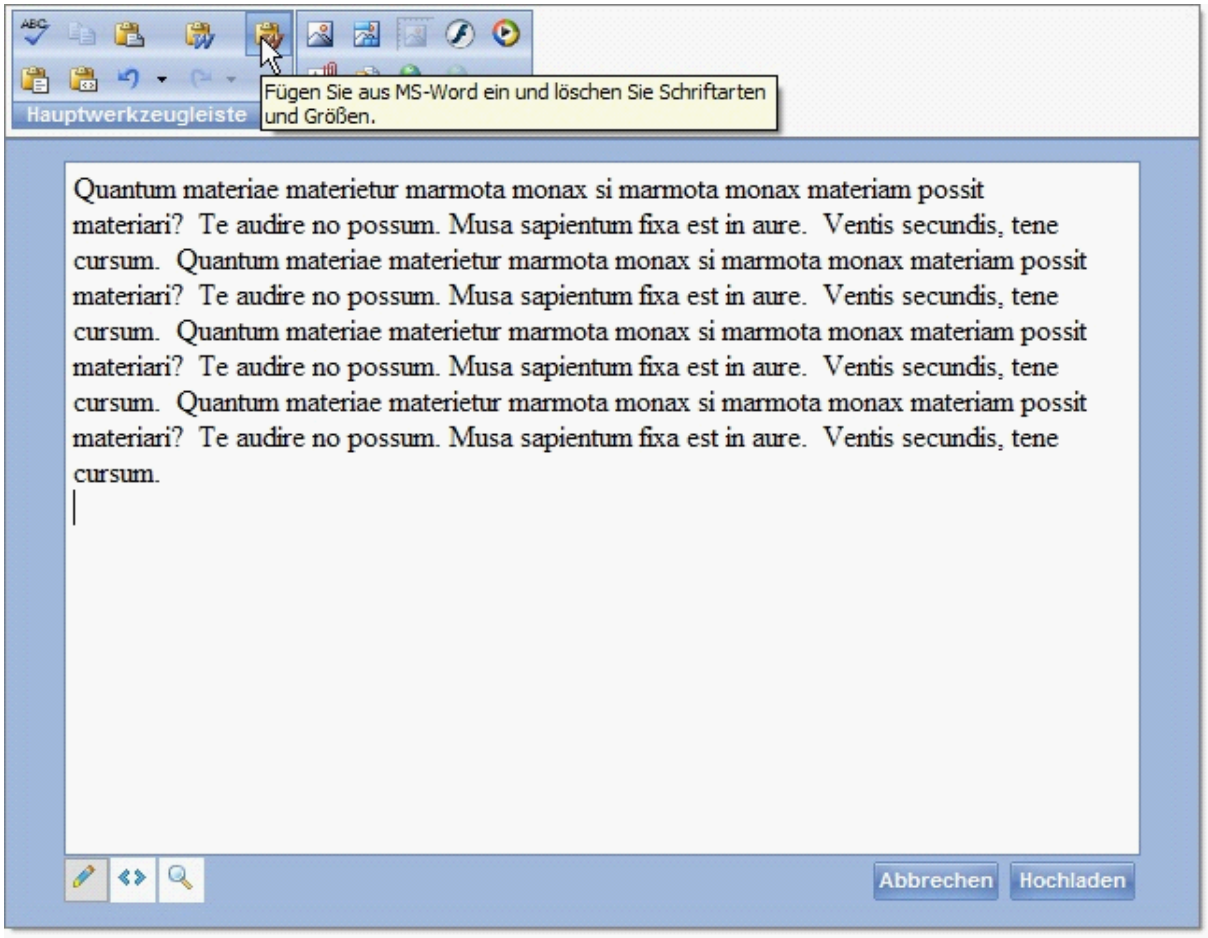
White the Browse Files tab lets your users take advantage of files on the server, the Upload Document tab lets users copy files from their local system up to the server. The user clicks the Browse button to find files on their hard drive, then click the Upload button to make a copy to the designated directory.



Uploading a file to the server

Lab: Using RadEditor

1. Create a new web application "GettingStarted".
2. Create a "RadControls" directory in the project. From the Telerik install directory copy the \RadControls\Editor directory to the RadControls of your project.
3. Drag a RadEditor from the toolbox to the default page of the project.
4. In ToolsFile.XML (found in \radcontrols\editor), set all modules *Enabled* to "false". Except for "MainToolBar" and "InsertToolbar", set all the other toolbars *Enabled* to "false". Where the *Enabled* attribute doesn't exist for a toolbar, add it. Set the *IsRibbon* property to "True" for "MainToolBar" and "InsertToolbar".
5. In the Properties Window set the *Skin* property to "Web20", *ToolbarMode* to "ShowOnFocus" and the *Language* to "de-DE".
6. Run the application. Notice how the additional menus and modules have vanished.



Editor with customized toolbars and skin

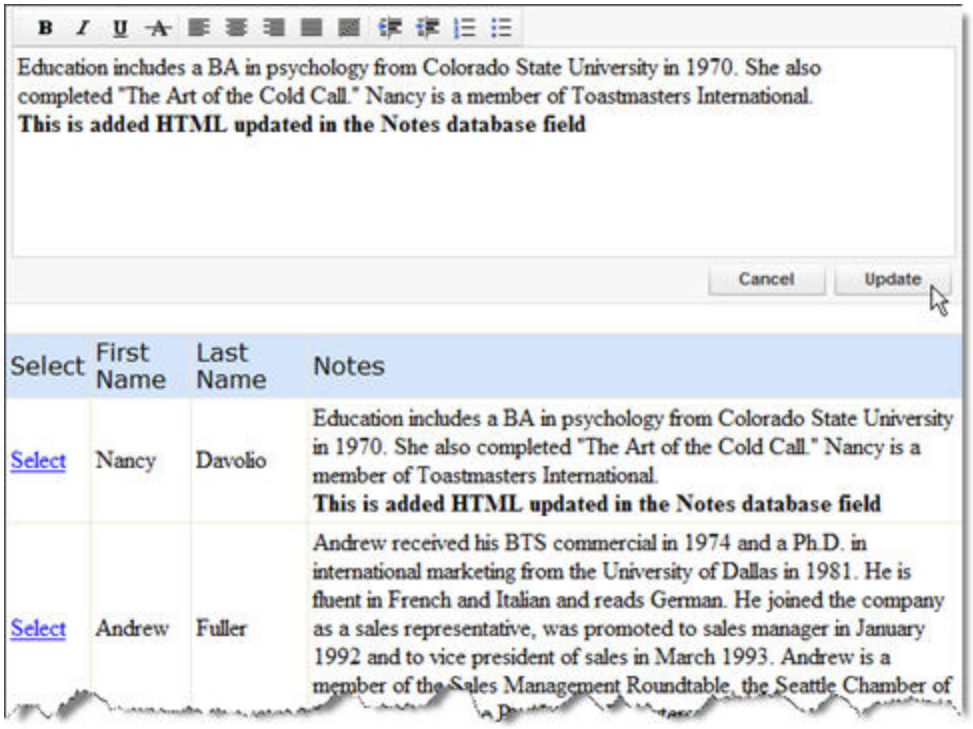
3.2.3 Using RadEditor at Runtime

Lab: Updating a Database

This lab will demonstrate retrieving and updating to a database from RadEditor and will also show the RadEditor working within a RadAjaxPanel.

RadEditor can be used as a source to update a database field containing rich text. The easiest way to do this is to hook up the SubmitClicked event handler and add your update code there. You could also include your update logic in an external button. This example will allow you to select a record from a DataGrid for editing in a RadEditor.

Note: DataGrid was chosen instead of GridView here because of how GridView handles column display. Where the DataGrid example below shows the Notes column exactly as the seen in the RadEditor, GridView shows the *markup* in the column so you see something like "This is added HTML updated...". Rather than sort out grid display issues this example focuses on the *SubmitClicked* event triggered from the Update button.



Updating grid data from RadEditor

1. Create a new application SaveToDB.
2. Add a RadAjaxPanel to the default page. The remaining items should be added onto the RadAjaxPanel.
3. Add a RadEditor, SqlDataSource, and a DataGrid.
4. The RadEditor will have minimal setup, but you may want to edit your ToolsFile.xml to disable all toolbars except one for formatting as in the figure above. This will leave you with the maximum screen real estate. Likewise you can set the *ShowHtmlMode* and *ShowPreviewMode* properties to false. Set the *Height* to "200px".
5. Configure the SqlDataSource to point at an instance of the Northwind database and set the select command to

```
SELECT EmployeeID, LastName, FirstName, Notes FROM Employees order by LastName
```

Note: See the appendix "MSDE" for information about installing the Northwind database using MSDE .

6. For the DataGrid set the *AutoGenerateColumns* property to "False" and point *DataSourceID* at the SqlDataSource ID. Add columns to the grid:
    - A ButtonColumn with *CommandName*, *HeaderText* and *Text* properties set to "Select".
    - Four BoundColumn with *DataField* properties set to "EmployeeID", "FirstName", "LastName" and "Notes" respectively. Set the *Visible* property of the EmployeeID column to false. Set the *HeaderText* property for the three visible text columns to "First Name, "Last Name" and "Notes" respectively.

The resulting markup in the aspx within the RadAjaxPanel should look like this:

```
<radA:RadAjaxPanel ID="RadAjaxPanel1" runat="server" Height="200px" Width="100%">
   <radE:RadEditor ID="RadEditor1" runat="server"
      OnSubmitClicked="RadEditor1_SubmitClicked"
```

```
      Height="200px"  showhtmlmode="false"  showpreviewmode="false">
   </radE:RadEditor>
  <br />
   <asp:datagrid  id="Grid1"  Runat="server"
       HeaderStyle-Font-Name="Verdana"  BorderWidth="1px"  CellPadding="3"
      AutoGenerateColumns="False"  Width="640px"
        DataSourceID="SqlDataSource1"
       OnSelectedIndexChanged="Grid1_SelectedIndexChanged1"    >
         <HeaderStyle   Font-Names="Verdana"  ForeColor="Black"  BackColor="#D3E5FA">
       </HeaderStyle>
       <Columns>
             <asp:ButtonColumn  CommandName="Select"  HeaderText="Select"
              Text="Select" >
              <ItemStyle Width="50px" />
            </asp:ButtonColumn>
             <asp:BoundColumn  DataField="EmployeeID"  Visible="False" />
             <asp:BoundColumn  DataField="FirstName"  HeaderText="First  Name" />
             <asp:BoundColumn  DataField="LastName"  HeaderText="Last  Name" />
             <asp:BoundColumn  DataField="Notes"  HeaderText="Notes" />
       </Columns>
    </asp:datagrid>  
<asp:SqlDataSource   ID="SqlDataSource1"   runat="server"
    ConnectionString="<%$  ConnectionStrings:NorthwindConnectionString2  %>"
   SelectCommand=
    "SELECT  EmployeeID,  LastName,  FirstName,  Notes  FROM  Employees  order  by  LastName">
</asp:SqlDataSource>
</radA:RadAjaxPanel>
```

7. In the code behind for the default page we first setup properties that handle retrieving values from the grid columns for the EmployeeID and the Notes text:

```
private  string  Notes
{
  get
  {
      return  Grid1.SelectedItem.Cells[4].Text;
  }
}

private  int  EmployeeID
{
  get
  {
      return  int.Parse(Grid1.SelectedItem.Cells[1].Text);
  }
}
```

8. We have only two event handlers left to code. One to get the data from the grid to the editor and the other to put the editor html to the database and rebind the grid. Create an event handler for the SelectedIndexChanged event. Assign the Notes property we just created to the RadEditor *Html* property.

```
protected  void  Grid1_SelectedIndexChanged1(object  sender,  EventArgs  e)
{
   RadEditor1.Html  =  this.Notes;
}
```

9. Create an event handler for the RadEditor SubmitClicked event and enter the code below. Here we act only if RadEditor *Html* has something in it and we have a grid row to update to. If there is something to save we create a command that performs an update of 'Notes" in the Employees table where the EmployeeID matches the record selected in the grid. Finally, we rebind the grid to display the changes to Notes.

```
protected void RadEditor1_SubmitClicked(object sender, EventArgs e)
{
  const string commandText =
    "update Employees set Notes = @Notes where EmployeeID = @EmployeeID";

  if ((RadEditor1.Html != string.Empty) && (Grid1.SelectedItem != null))
  {
    SqlConnection connection =
        new SqlConnection(SqlDataSource1.ConnectionString);
    SqlCommand command =
       new SqlCommand(commandText, connection);
      command.Parameters.AddWithValue("Notes", RadEditor1.Html);
      command.Parameters.AddWithValue("EmployeeID", this.EmployeeID);

    connection.Open();
     command.ExecuteNonQuery();
    connection.Close();

    Grid1.DataBind();
  }
}
```

10. Run the application and test editing the content, changing formatting, saving and retrieving.

Note: If you get an error that indicates the editor is in read-only mode, make sure you're trying to set the *Html* property, not the read-only *Text* or *Xhtml* properties.

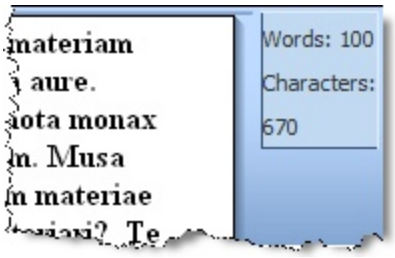
Using the RadEditor API

You can build all the characteristics of an editor at runtime on the server. The server side API has methods that parallel everything you see in ToolsFile.xml. As you would expect you can set the top level properties of RadEditor like *Skin*, *ShowHtmlMode* or *ToolbarMode*. If you want to rearrange the toolbars, add custom links or snippets, the API handles that as well. The API here is quite extensive so we will first take a look at a representative pattern where one of the RadEditor collections is cleared, new collection members are created, configured and added back to the collection.

```
C# Example:
RadEditor1.Modules.Clear();
Module module = new Module("RadEditorStatistics");
module.DockingZoneId  = "Right";
module.IsDockable  = false;
RadEditor1.Modules.Add(module);

VB Example:
RadEditor1.Modules.Clear
Dim module As Module = New  Module("RadEditorStatistics")
module.DockingZoneId  =  "Right"
module.IsDockable  = False
RadEditor1.Modules.Add(module)
```

This snippet instantiates a predefined module for editor statistics, assigns it to dock on the right hand side, *IsDockable* is set false to prevent it being dragged from that position, and the module is added to

the Modules collection.

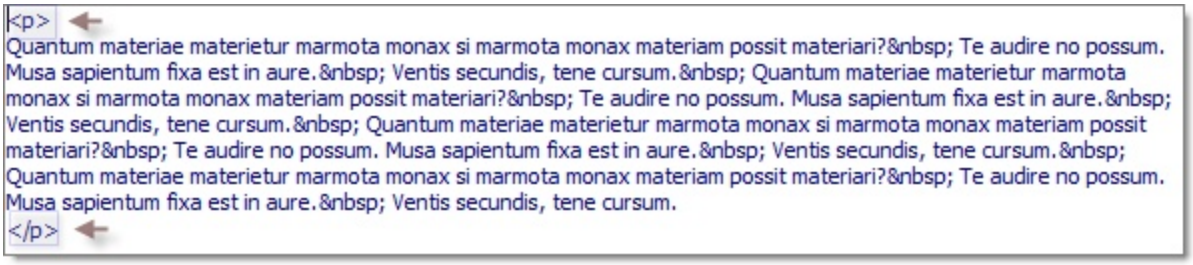


Statistics module docked to the upper right

The next example creates a context menu that is attached to a paragraph HTML element, allowing the user to justify paragraphs.

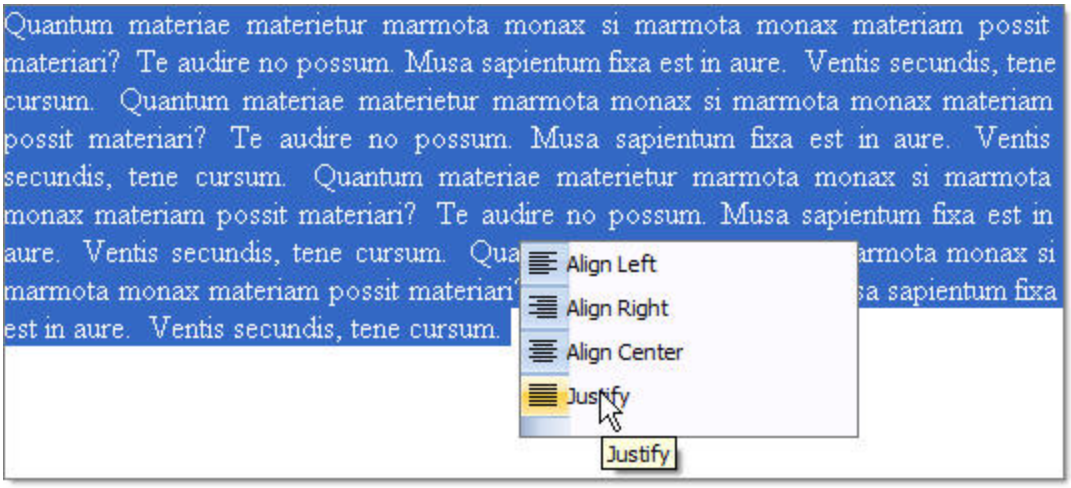
```
C# Example:
ContextMenu contextMenu = new ContextMenu("p");
contextMenu.Tools.Add(new ToolbarButton("JustifyLeft"));
contextMenu.Tools.Add(new ToolbarButton("JustifyRight"));
contextMenu.Tools.Add(new ToolbarButton("JustifyCenter"));
contextMenu.Tools.Add(new ToolbarButton("JustifyFull"));
RadEditor1.ContextMenus.Add(contextMenu);

VB Example:
Dim contextMenu As ContextMenu = New ContextMenu("p")
contextMenu.Tools.Add(New ToolbarButton("JustifyLeft"))
contextMenu.Tools.Add(New ToolbarButton("JustifyRight"))
contextMenu.Tools.Add(New ToolbarButton("JustifyCenter"))
contextMenu.Tools.Add(New ToolbarButton("JustifyFull"))
RadEditor1.ContextMenus.Add(contextMenu)
```

The effect in the editor is if I have a "<p>" tag in the HTML mode of the editor...



Defining an HTML paragraph

...in design mode I can right click the paragraph and run the ToolbarButton commands from the context menu:

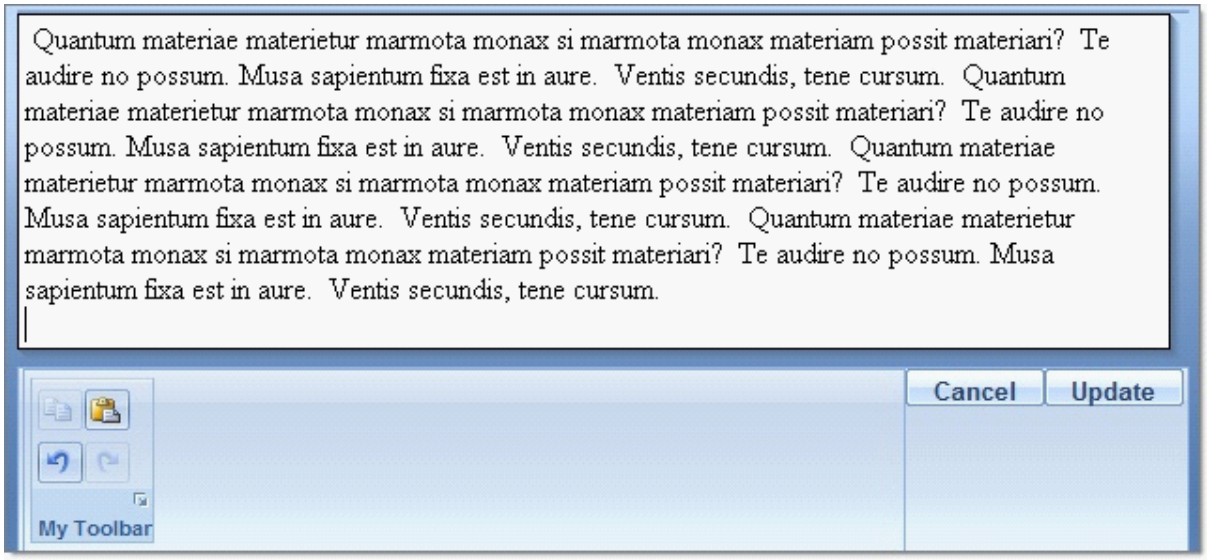


Running the context menu

One last example before we try these out in a lab. This snippet adds a toolbar with a set of buttons and a separator. Notice that the toolbar *IsRibbon* property is set to true, the *DockingZoneId* is set to "Bottom" (the possible settings are "Top", "Bottom", "Right", "Left" and "Middle") and IsDockable is set false to prevent the toolbar from being dragged. Other than that, the entire pattern of clear, create and add is just about the same.

```
C# Example:
RadEditor1.Toolbars.Clear();
Telerik.WebControls.RadEditorUtils.Toolbar toolbar = new Toolbar("My Toolbar");
toolbar.IsRibbon = true;
toolbar.DockingZoneId = "Bottom";
toolbar.IsDockable = false;
toolbar.Tools.Add(new ToolbarButton("Copy"));
toolbar.Tools.Add(new ToolbarButton("Paste"));
toolbar.Tools.Add(new ToolbarSeparator());
toolbar.Tools.Add(new ToolbarButton("Undo"));
toolbar.Tools.Add(new ToolbarButton("Redo"));
RadEditor1.Toolbars.Add(toolbar);

VB Example:
RadEditor1.Toolbars.Clear
Dim toolbar As Telerik.WebControls.RadEditorUtils.Toolbar = New Toolbar("My Toolbar")
toolbar.IsRibbon = True
toolbar.DockingZoneId = "Bottom"
toolbar.IsDockable = False
toolbar.Tools.Add(New ToolbarButton("Copy"))
toolbar.Tools.Add(New ToolbarButton("Paste"))
toolbar.Tools.Add(New ToolbarSeparator)
toolbar.Tools.Add(New ToolbarButton("Undo"))
toolbar.Tools.Add(New ToolbarButton("Redo"))
RadEditor1.Toolbars.Add(toolbar)
```

The resulting toolbar displays at the bottom of the editor as a ribbon Office2007 style toolbar that is not draggable.



New toolbar in the editor

Lab: Adding Toolbars, Menus and Modules

1. Create a new application "ServerAPI".
2. Create a "RadControls" directory in the project. From the Telerik install directory copy the \RadControls\Editor directory to the RadControls of your project.
3. Drag a RadEditor from the toolbox to the default page of the project. At this point the markup should be very simple because the rest of the work will be done in the code behind. In fact you can get away with markup that is this simple:

```
<form id="form1" runat="server">
   <radE:RadEditor ID="RadEditor1" runat="server">
   </radE:RadEditor>
</form>
```

4. In the Page_Load code behind for the default page add the following code to define the context menus, toolbars, modules and links:

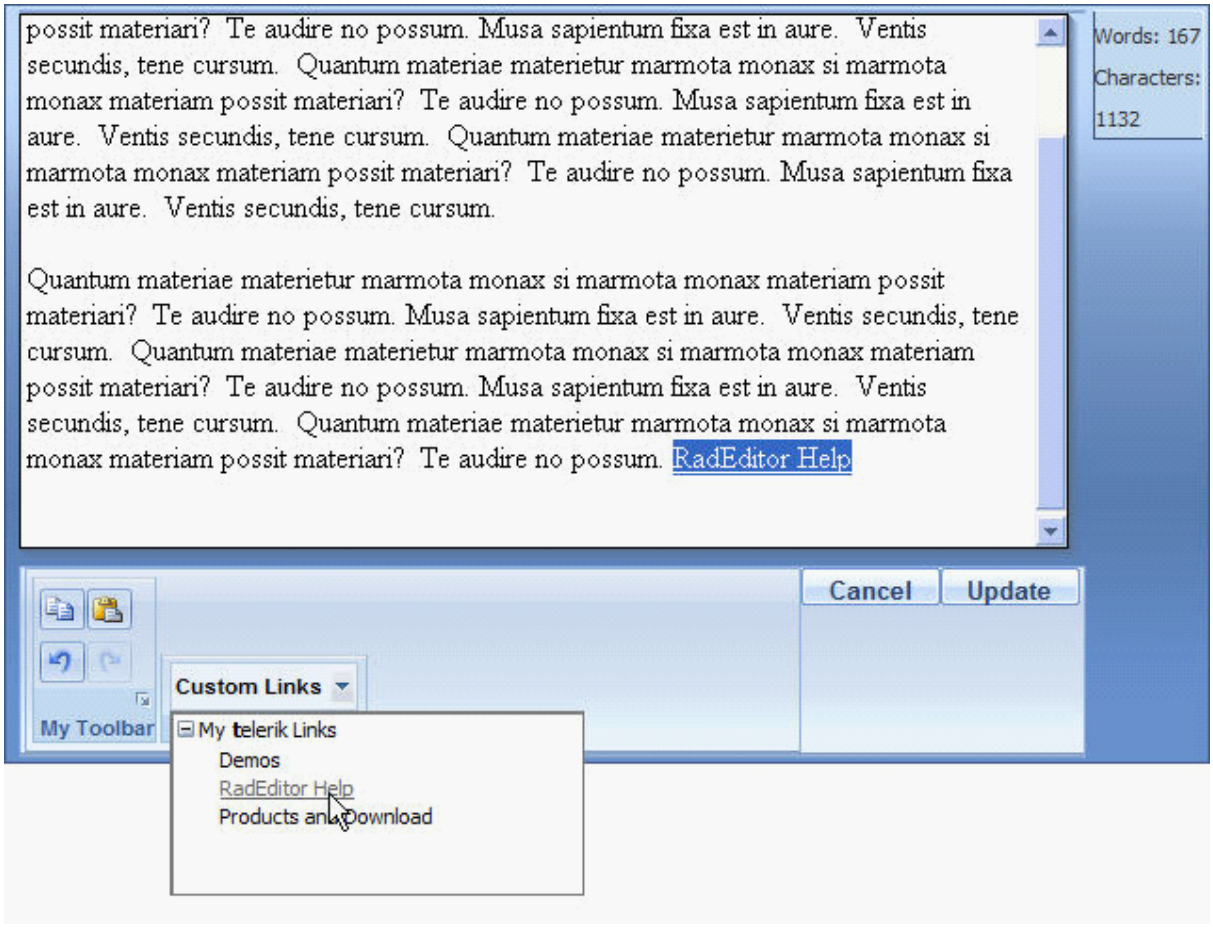
```
C# Example:
if (!Page.IsPostBack)
{
   RadEditor1.EnableClientSerialize = false;
   RadEditor1.Skin = "Office2007";
   RadEditor1.ShowHtmlMode = false;
   RadEditor1.ShowPreviewMode = false;
   RadEditor1.ToolbarMode = EditorToolbarMode.Default;

   RadEditor1.Modules.Clear();
   Module module = new Module("RadEditorStatistics");
   module.DockingZoneId = "Right";
   module.IsDockable = false;
   RadEditor1.Modules.Add(module);

   ContextMenu contextMenu = new ContextMenu("p");
   contextMenu.Tools.Add(new ToolbarButton("JustifyLeft"));
   contextMenu.Tools.Add(new ToolbarButton("JustifyRight"));
```

```
    contextMenu.Tools.Add(new  ToolbarButton("JustifyCenter"));
    contextMenu.Tools.Add(new  ToolbarButton("JustifyFull"));
     RadEditor1.ContextMenus.Add(contextMenu);

     RadEditor1.Links.ChildLinks.Clear();
   Link link =
      RadEditor1.Links.Add("My  <b>t</b>elerik  Links", "http://www.telerik.com");
    link.Add("Demos",
     "http://www.telerik.com/demos/aspnet/Editor/Examples/Overview/DefaultCS.aspx");
    link.Add("RadEditor  Help",
     "http://www.telerik.com/help/aspnet/editor/");
    link.Add("Products  and  Download",
     "http://www.telerik.com/client.net/my-licenses/my-purchases.aspx");

     RadEditor1.Toolbars.Clear();
     Telerik.WebControls.RadEditorUtils.Toolbar  toolbar  = new  Toolbar("My  Toolbar");
    toolbar.IsRibbon  = true;
    toolbar.DockingZoneId  = "Bottom";
    toolbar.IsDockable  = false;
    toolbar.Tools.Add(new  ToolbarButton("Copy"));
    toolbar.Tools.Add(new  ToolbarButton("Paste"));
    toolbar.Tools.Add(new  ToolbarSeparator());
    toolbar.Tools.Add(new  ToolbarButton("Undo"));
    toolbar.Tools.Add(new  ToolbarButton("Redo"));
     RadEditor1.Toolbars.Add(toolbar);

     Telerik.WebControls.RadEditorUtils.Toolbar  tbLink  = new  Toolbar("DropdownToolbar");
    tbLink.IsRibbon  = true;
    tbLink.DockingZoneId  = "Bottom";
    tbLink.IsDockable  = false;
    tbLink.Tools.Add(new  ToolbarDropDown("InsertCustomLink"));
     RadEditor1.Toolbars.Add(tbLink);
}

VB  Example:
If Not  Page.IsPostBack Then
  RadEditor1.EnableClientSerialize  = False
 RadEditor1.Skin  =  "Office2007"
 RadEditor1.ShowHtmlMode  = False
 RadEditor1.ShowPreviewMode  = False
 RadEditor1.ToolbarMode  =  EditorToolbarMode.Default
  RadEditor1.Modules.Clear
 Dim  module As Module = New   Module("RadEditorStatistics")
 module.DockingZoneId  =  "Right"
 module.IsDockable  = False
  RadEditor1.Modules.Add(module)
 Dim  contextMenu As  ContextMenu  = New  ContextMenu("p")
  contextMenu.Tools.Add(New   ToolbarButton("JustifyLeft"))
  contextMenu.Tools.Add(New   ToolbarButton("JustifyRight"))
  contextMenu.Tools.Add(New   ToolbarButton("JustifyCenter"))
  contextMenu.Tools.Add(New   ToolbarButton("JustifyFull"))
   RadEditor1.ContextMenus.Add(contextMenu)
   RadEditor1.Links.ChildLinks.Clear
 Dim  link As Link = _
     RadEditor1.Links.Add("My  <b>t</b>elerik  Links",  "http://www.telerik.com")
 link.Add("Demos",  _
       "http://www.telerik.com/demos/aspnet/Editor/Examples/Overview/DefaultCS.aspx")
 link.Add("RadEditor  Help",  _
     "http://www.telerik.com/help/aspnet/editor/")
 link.Add("Products  and  Download",  _
      "http://www.telerik.com/client.net/my-licenses/my-purchases.aspx")
```

```
  RadEditor1.Toolbars.Clear
 Dim toolbar As Telerik.WebControls.RadEditorUtils.Toolbar = _
   New Toolbar("My Toolbar")
 toolbar.IsRibbon = True
 toolbar.DockingZoneId = "Bottom"
 toolbar.IsDockable = False
  toolbar.Tools.Add(New ToolbarButton("Copy"))
  toolbar.Tools.Add(New ToolbarButton("Paste"))
  toolbar.Tools.Add(New ToolbarSeparator)
  toolbar.Tools.Add(New ToolbarButton("Undo"))
  toolbar.Tools.Add(New ToolbarButton("Redo"))
   RadEditor1.Toolbars.Add(toolbar)
 Dim tbLink As Telerik.WebControls.RadEditorUtils.Toolbar = _
   New Toolbar("DropdownToolbar")
 tbLink.IsRibbon = True
 tbLink.DockingZoneId = "Bottom"
 tbLink.IsDockable = False
  tbLink.Tools.Add(New ToolbarDropDown("InsertCustomLink"))
   RadEditor1.Toolbars.Add(tbLink)
End If
```

5. Run the application.

A couple of new things have popped up in the code. The first is the creation of a set of links. These are hierarchical in nature so we have a main link for the Telerik site and a series of sub links underneath it. Further down towards the end of the snippet a toolbar is defined to contain the predefined "InsertCustomLink" tool that displays all the links defined above. The running application looks like this:



The running application configured completely with the API

3.2.4 Client Scripting with RadEditor

Using JavaScript you can get at the guts of RadEditor, its document, content and tools, not to mention editor state and visibility to events. In this section you'll have a brief tour of selected client functions, we will see the available events that fire client side and finally put these all together in a lab.

Tour of client functions

If you simply want a reference to the editor itself use the ClientID property emitted from the server, then call client side methods from the editor object.

```
var myEditor = <%=RadEditor1.ClientID %>;
var logHtml = myEditor.GetHtml(false);
myEditor.SetHtml(logHtml + "<div>" + message + "</div>");
```

If you need to access multiple editors on a single page Telerik has a helpful client side global called RadEditorGlobalArray:

```
var allEditors = RadEditorGlobalArray;

for (i=0; i<allEditors.length; i++)
{
   alert("Editor: " + allEditors[i].Id);
}
```

You can access the document object within the editor then use standard JavaScript to operate on the document. For example you can iterate all links within a document:

```
var editor= <%=RadEditor1.ClientID%>
var links = editor.Document.body.getElementsByTagName("A");
for (var i = 0; i < links.length; i++)
   { alert(links.href);
}
```

The JavaScript document object has an execCommand() function that performs a number of formatting tasks that you can call (see MSDN, Google for commands available to execCommand). So combining the ability to get at all editors and run commands against each editor document, this code snippet sets the current selection for each editor to bold font:

```
function SetAllEditorsBold()
{
   var allEditors = RadEditorGlobalArray;

   for (i=0; i<allEditors.length; i++)
  {
    allEditors[i].Document.execCommand("Bold", false, null);
  }
}
```

To reach one of the objects owned by the editor use one of the "get" methods:

Function	**Example**

GetContentArea()	This example retrieves the editor content object and alters its style:
	`function ClientLoad(editor)`
	`{`
	`  var style = editor.GetContentArea().style;`
	`  style.backgroundColor = "WhiteSmoke";`
	`  style.color= "DarkGray";`
	`  style.fontFamily= "Arial";`
	`  style.fontSize= "15";`
	`}`

GetToolByName()	If you want to alter the dimensions of a tool in the editor, use the GetToolByName() function, then operate on the tools properties:
	`function OnClientLoad(editor)`
	`{`
	`  editor.GetToolByName("FontSize").PopupWidth = 400;`
	`  editor.GetToolByName("FontSize").PopupHeight = 400;`
	`  editor.GetToolByName("ApplyClass").PopupWidth = 400;`
	`  editor.GetToolByName("ApplyClass").PopupHeight = 400;`
	`}`

Function	Example

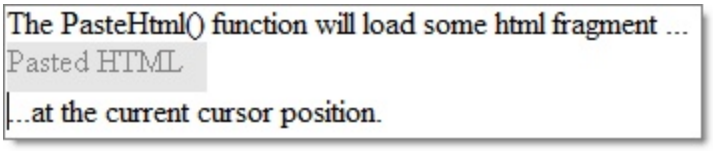
GetSelection()
 	
GetSelectionHtml()
	Get the selected text or HTML. This example takes the selected HTML from one editor and pastes it inside a div to a second editor.
	`function GetSelection()`
	`{`
	`  var selection = <%=reEdit.ClientID%>.GetSelectionHtml();`
	`  var logHtml = <%=reLog.ClientID%>.GetHtml(false);`
	`  <%=reLog.ClientID%>.SetHtml(logHtml +`
	`  '<div style="font-weight: bold">' + selection + '</div>');`
	`}`

To put html back into the editor use SetHtml(). This snippet replaces the entire document HTML using existing text contained in "logHtml" adds another text variable "message", surrounded by a div:

```
<%=reLog.ClientID %>.SetHtml(logHtml + "<div>" + message + "</div>");
```

Or you can insert just an HTML fragment to the current cursor position using PasteHtml():

```
function InsertDiv()
{
    var editor = <%=reEdit.ClientID %>;
    editor.PasteHtml(
      '<div style="width:100px; height:25px; background-color:#CCCCCC; ' +
       'filter:alpha(Opacity=50); overflow:auto">Pasted HTML</div>');
}
```



Pasting a div to the document

Client Events

There are just a handful of events for RadEditor, but they cover a lot of ground. Particularly the OnClientLoad is used for most initialization tasks and OnClientCommandExecuting/Executed are fired on most actions within the editor. Each event is passed a reference to the editor and may also be passed parameters with other qualifying information.

OnClientCancel, OnClientSubmit: These events fire before the user cancels or submits the current edits. These events are cancelable by returning false. For example you can prompt the user "Do you want to cancel your changes?" and discontinue the event if the user says no:

```
function   ClientCancel(editor)
{
     return confirm("Are you sure you want to cancel your changes?");
}
```

OnClientCommandExecuting: Just about everything that happens in the editor involves a command of some sort, e.g. changing fonts, user pressing enter, printing... This event lets you know that a command is about to occur, what its parameters are, and allows you to cancel it. This example gets the parameter value if any and runs a log() function:

```
function  ClientCommandExecuting  (editor,  commandName,  tool)
{
     var selValue =
        (tool && tool.GetSelectedValue) ? tool.GetSelectedValue() : "none";
      Log("OnClientCommandExecuting. Command name:" +
         commandName +
         ". Selected Value:" +
         selValue);
}
```

OnClientCommandExecuted: The signature for this function is identical OnClientCommandExecuting but occurs after the command has executed and does not allow cancellation by returning false.

OnClientLoad: This function presents an chance to initialize the editor. In some cases this event may occur slightly too early in relation to initialization of objects within the page. In these cases Telerik support recommends using the setTimeout() JavaScript function to move the initialization to a later event. In this next snippet the style can be set directly, but other initialization is moved to the window onLoad:

```
function   ClientLoad(editor)
{
   var  style  =  editor.GetContentArea().style;
   style.backgroundColor  =  "WhiteSmoke";
   style.color=  "DarkGray";
   style.fontFamily=  "Arial";
   style.fontSize=  "15";

   var  SetWindowFunction  =  function  ()
  {
    window.setTimeout(function()
        {
          // do some initialization here
        }, 10);
  }
   window.onload  =  SetWindowFunction;
}
```

OnClientModeChange: This event occurs when the editor mode changes between design, html and preview. Use the editor GetMode() function to return the current mode as an integer value: 1=design, 2=Html, 3=Preview.

Lab: Logging RadEditor Events

This lab will actually have two editors running. One at page top in preview mode will log events as they happen in a second editor. The lab will demonstrate events currently available for RadEditor and will initialize the editor size and content area style. The lab will also cover retrieving and setting HTML in the editor content area. One of the frequently asked questions on the Telerik forum is "how do I set the initial editor mode to design/html/preview". We will handle setting the "log" editor to preview mode during the client load event.

1. Create a new application "ClientEvents"
2. Create a "RadControls" directory in the project. From the Telerik install directory copy the \RadControls\Editor directory to the RadControls of your project.
3. Drop a RadEditor control from the toolbox to the default web page. That will register the RadEditor assembly for us. Now replace the HTML markup in the aspx for the editor with the snippet shown below. Notice the *ID* is "reLog" because it will be used to log events from a second editor that we will define in a moment. The OnClientLoad event for reLog is defined so that we can set the editor to preview mode when the application starts up.

```
<radE:RadEditor  ID="reLog"  runat="server"
   Editable="True"
    EnableServerSideRendering="True"
   EnableTab="True"
    HasPermission="True"
    RenderAsTextArea="False"
   Skin="WebBlue"
    ConvertTagsToLower="True"
    ConvertToXhtml="False"
    ShowHtmlMode="False"
    ShowPreviewMode="False"
  ShowSubmitCancelButtons="False"
  OnClientLoad="ClientLoad"
  >
</radE:RadEditor>
```

4. Add a break element to separate the two editors, then add a second RadEditor with ID "reEdit" to the aspx markup. This RadEditor tag will define the remaining client events:

```
<br/>

<radE:RadEditor  ID="reEdit"  runat="server"
    ConvertTagsToLower="True"
    ConvertToXhtml="False"
    DocumentsFilters="*.*"
    EnableClientSerialize="True"
    EnableContextMenus="True"
   EnableDocking="True"
    EnableEnhancedEdit="True"
    EnableHtmlIndentation="True"
    EnableServerSideRendering="True"
   EnableTab="True"
    OnClientCancel="ClientCancel"
     OnClientCommandExecuted="ClientCommandExecuted"
  OnClientCommandExecuting="ClientCommandExecuting"
    OnClientModeChange="ClientModeChange"
    OnClientSubmit="ClientSubmit"
```

```
   PassSessionData="True"
   RenderAsTextArea="False"
   Skin="WebBlue"
   SpellEditDistance="1"
    TemplateFilters="*.html,*.htm"
   ToolbarMode="Default"
   Height="200px">
</radE:RadEditor>
```

5. Just inside the form tag add this JavaScript code that defines the event handlers for the client side events:

```
<script   type="text/javascript">
//<![CDATA[

   function  ClientLoad(editor)
  {
     var  style  =  editor.GetContentArea().style;
     style.backgroundColor  =  "WhiteSmoke";
     style.color=  "DarkGray";
     style.fontFamily=  "Arial";
     style.fontSize=  "15";

     var  SetWindowFunction  =  function  ()
    {
      window.setTimeout(function()
          {
              // 1=editor, 2=HTML, 3=Preview
              editor.SetMode(3);
             editor.SetSize(650, 100);
          }, 1);
    }
     window.onload  =  SetWindowFunction;

  }

   function  ClientCommandExecuting  (editor,  commandName,  tool)
  {
      var  selValue  =
          (tool  &&  tool.GetSelectedValue)  ?  tool.GetSelectedValue()  :  "none";
      if  (commandName  !=  "Enter")
     {
         Log("OnClientCommandExecuting. Command name:" +
          commandName +
          ". Selected Value:" +
          selValue);
     }
  }

   function  ClientCommandExecuted(editor,  commandName,  tool)
  {
      var  selValue  =
          (tool  &&  tool.GetSelectedValue)  ?  tool.GetSelectedValue()  :  "none";
      if(commandName  !=  "Enter")
     {
         Log("OnClientCommandExecuted. Command name:" +
          commandName +
          ". Selected Value:" +
          selValue);
     }
```

```
  }

   function  ClientModeChange(editor)
  {
      window.setTimeout(  function()
          {
             var mode = editor.GetMode();
             var modeName = "";
               switch (mode)
              {
                   case 1: modeName = "Design"; break;
                   case 2: modeName = "HTML"; break;
                   case 3: modeName = "Preview"; break;
              }
             Log("OnClientModeChange: " + mode + ": " + modeName);
          }
          ,10
      );
  }

   function  ClientSubmit(editor)
  {
       Log("OnClientSubmit");
      return confirm( "Are you sure you want to submit?");
  }

   function  ClientCancel(editor)
  {
       Log("OnClientCancel");
       return confirm("Are you sure you want to cancel your changes?");
  }

   function  Log(message)
  {
     var  logHtml  =  <%=reLog.ClientID %>.GetHtml(false);
     <%=reLog.ClientID %>.SetHtml(logHtml + "<div>" + message + "</div>");
  }
//]]>
</script>
```

Let's take these one at a time:

- **ClientLoad** retrieves the content area for the edit grid and manipulates the style. But then we want to change the mode and size of the editor, but it's too soon so the setTimeout() function allows us to piggy-back off the window onload event. Once onload runs, the timeout is fired a millisecond later, the mode for the topmost editor is changed to Preview and the size is changed.
- **ClientCommandExecuting** first determines if tool and GetSelectedValue are defined. If they do then GetSelectedValue() is called. For example if the user clicks on the Foreground Color button and choose the color White, the selected value will be "#FFFFFF". The command and selected value are displayed to the topmost editor, except if the user has just typed "Enter". This demonstrates conditional code based on editor command and also prevents focus from moving away from reEdit.
- **ClientCommandExecuted** is identical to ClientCommandExecuting but occurs just after.
- **ClientModeChange** signals the user has just changed between design/html/preview modes. Here we call editor GetMode() function, map the return value to a string and display the result.

- **ClientSubmit** and **ClientCancel** can be tested by hitting the update or cancel buttons at the bottom of the editor. If the user cancels the dialog the event does not continue.
- Finally, the **Log**() function gets the html in the topmost "log" editor, adds a message to display in a div and sets the html in the log editor.

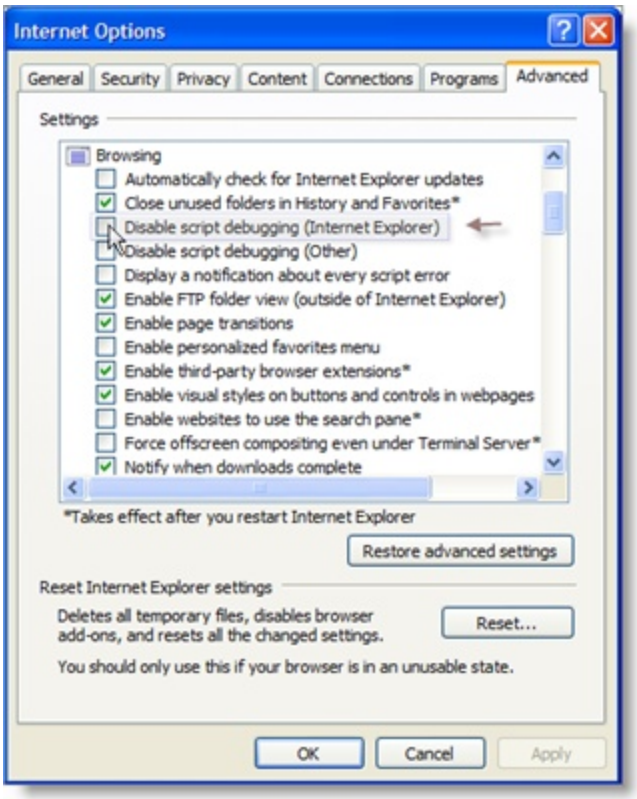
6. Run the application.

Lab: Using the Script Explorer

Visual Studio 2005 has a sometimes overlooked feature that allows you to debug into JavaScript and to examine markup emitted by aspx controls. It's worth looking at here because the settings on the editor radically change the character and amount of markup sent to the browser.

Note: The figures for this example are taken from running in IE 7.

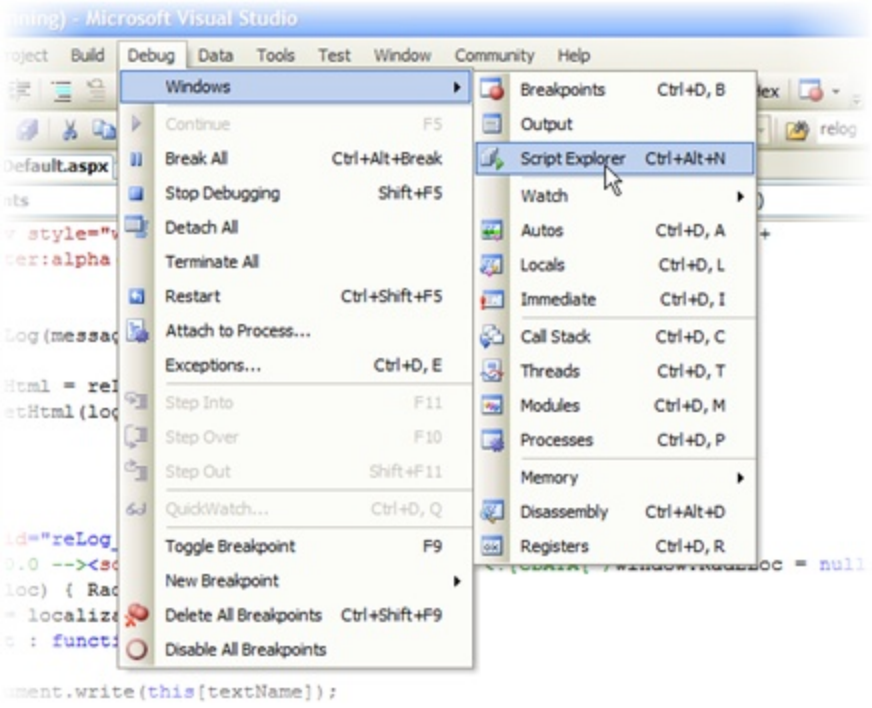
1. First check that script debugging is available in your browser. In IE 7 this is done from Internet Options | Advanced | uncheck "Disable Script Debugging (Internet Explorer).



Verifying that script debugging is available

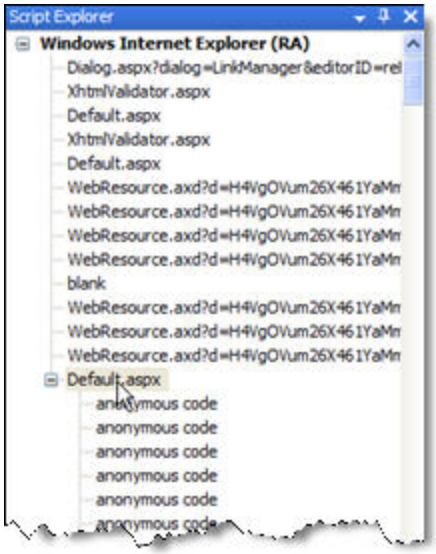
2. Run the application from the last lab "Logging RadEditor Events".

3. From Visual Studio run Debug | Windows | Script Explorer.



Running Script Explorer

4. This will display the Script Explorer window with a list of pages currently loaded including our page. Double click "Default.aspx" to begin debugging.



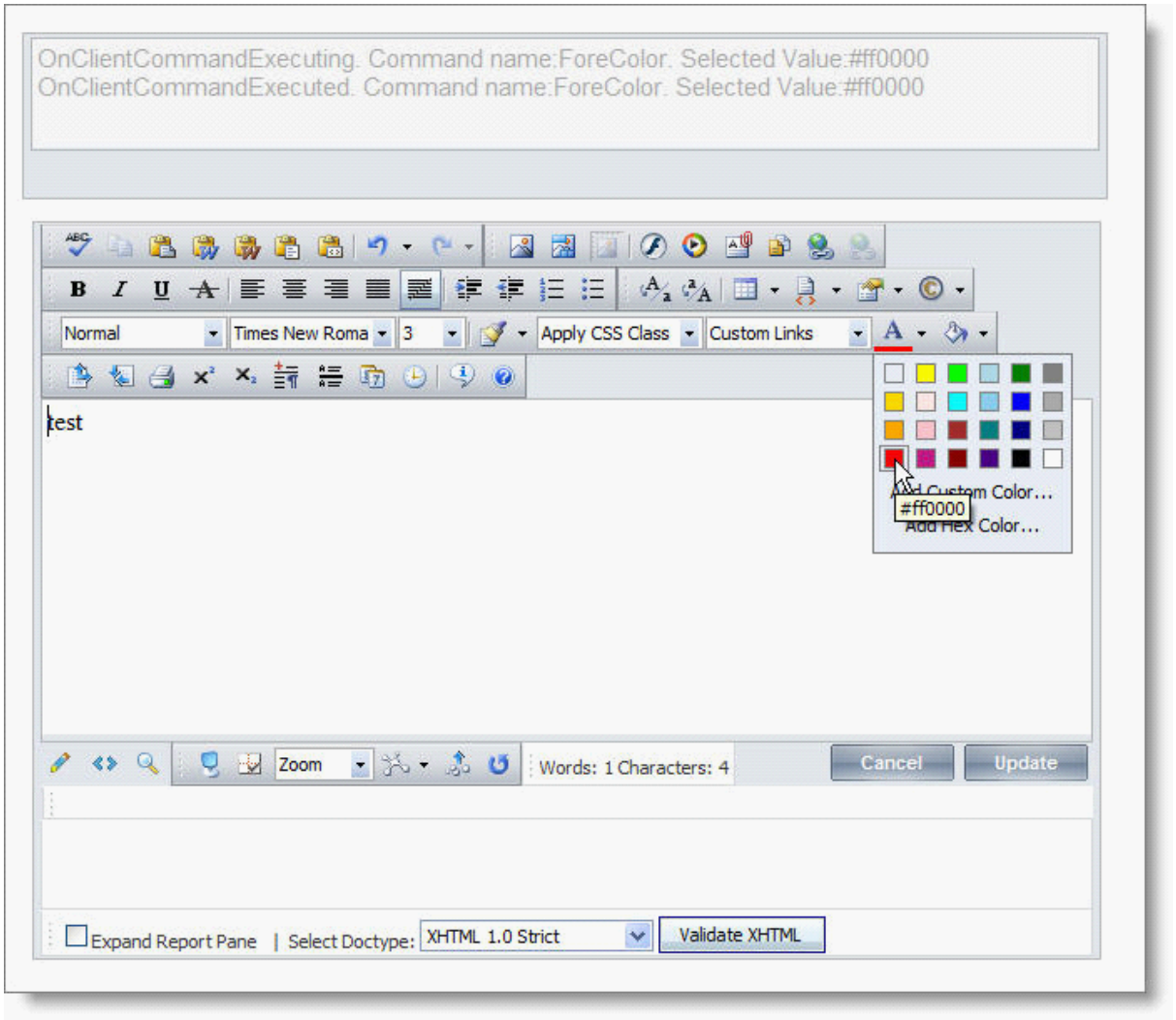
Selecting the page in Script Explorer

5. When the debugging window for the page displays you might at first think it's default.aspx but a closer looks shows that all server side code has been evaluated. For instance the Log() function no longer contains "<%=reLog.ClientID %>". These have all been processed on the server and emitted as "reLog". Lower on the page you can see the editor control "reLog" is now a collection of HTML and JavaScript. "reEdit" has most of the bells and whistles turned on and if you search for it lower on the page you'll find pages of markup that supports its runtime functionality.

```
  function Log(message)
  {
    var logHtml = reLog.GetHtml(false);
    reLog.SetHtml(logHtml + "<div>" + message + "</div>");
  }
//]]>
</script>

      <div id="reLog_wrapper" style="display:inline-block;">
    <!-- 7.0.0 --><script type="text/javascript">/*<![CDATA[*/window.RadELoc = null;
if (window.loc) { RadELoc = window.loc; }
window.loc = localization_en_US = {
    showText : function(textName)
    {
        document.write(this[textName]);
    }
```

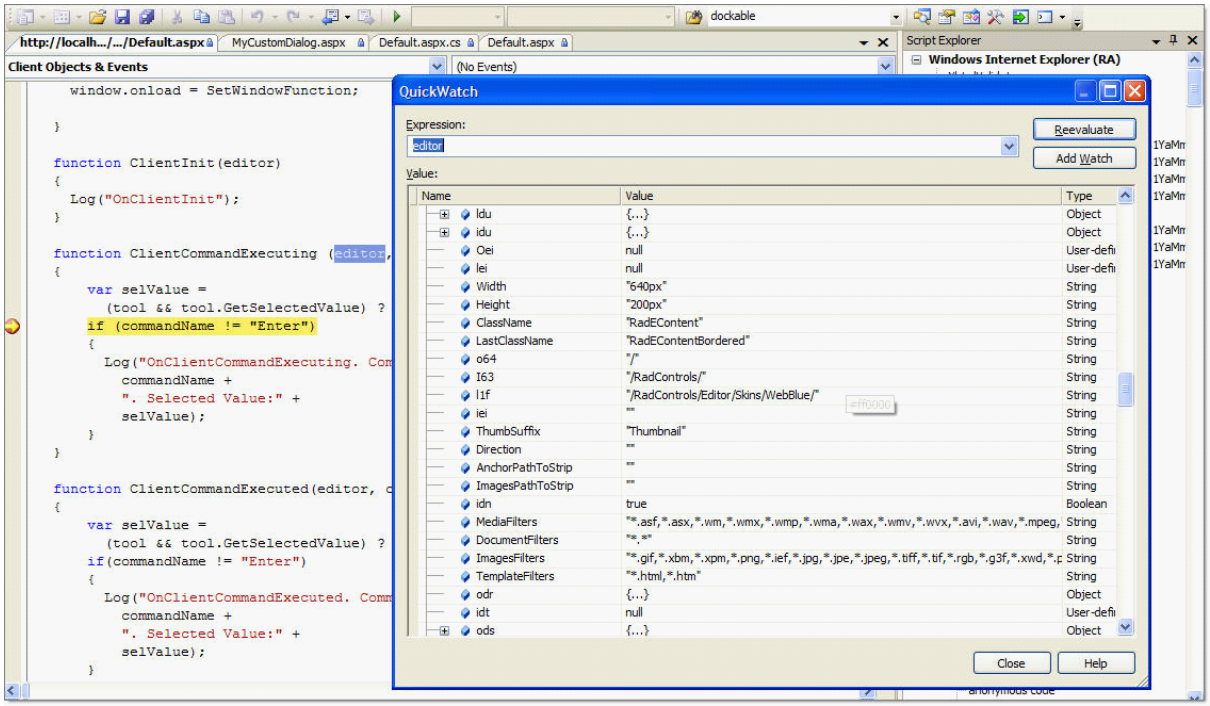
RadEditor at runtime on the client

6. Put a break point in the ClientCommandExecuting event handler on the line "if (commandName != "Enter")". In the bottom editor "reEdit" type some text, select it, click the foreground color button and click the color red:



Running the ForeColor command

7. When the debugger stops on the breakpoint you can use the standard Visual Studio debugging facilities Add Watch and QuickWatch to explore everything in your JavaScript code. The figure below shows the "editor" parameter in the QuickWatch window. The naming convention for many of the non-public JavaScript members are quite short and cryptic, probably due to the effort to keep the data load to the browser down.



Evaluating the editor in QuickWatch

8. After experimenting with the debugger, stop the application.

9. Set the *RenderAsTextArea* property of reEdit to true.

10. Re-run the application again and use the script explorer to see how reEdit is emitted. By comparing the two in this case you'll find that reEdit as a text area is about 30% "less filling". The 30% isn't intended to be a rule, but an instance where changing RadEditor properties changes the editor payload and capabilities out on the browser.

3.2.5 Summary

This chapter covered some useful aspects of this extensive component including:

- Ways of getting data in and out of the editor.
- Methods of persisting the content.
- Multiple levels of customization using ToolsFile.xml and ConfigFile.xml.
- Control of appearance through skins.
- Localization of the editor, dialogs and tools.
- Control of editing permissions.
- Client methods and events which included a short tour of the Script Explorer.
- File managers.

Some of the things we did not cover are:

- Defining editable and non-editable areas.

- Printing.
- Content Filters.
- Customized dialogs and modules.

See the online help and the RadEditor overview at http://www.telerik.com/demos/aspnet/Editor/Examples/Overview/DefaultCS.aspx for more ideas and techniques.

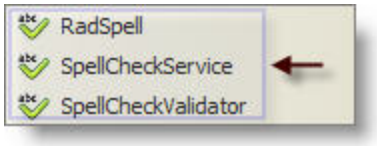
3.3 RadSpell

3.3.1 Getting Started

RadSpell is a full featured, performant, localizable and highly customizable spell checker for web applications. RadSpell can be used on any client or server side editable element anywhere in a web application including:

- In Standard ASP.NET pages
- Non ASP.NET pages (e.g. ASP, HTML)
- In user controls
- In templates
- With the RadEditor
- In Master/Content page scenarios
- With any other components on the page that allow javascript access to getting and setting text.

RadSpell consists of the RadSpell control that performs the actual spell checking, a convenient SpellCheckValidator that adapts the spell checker to standard ASP.NET validation functionality and a spell check service component that performs client side spell checking without a graphical user interface.

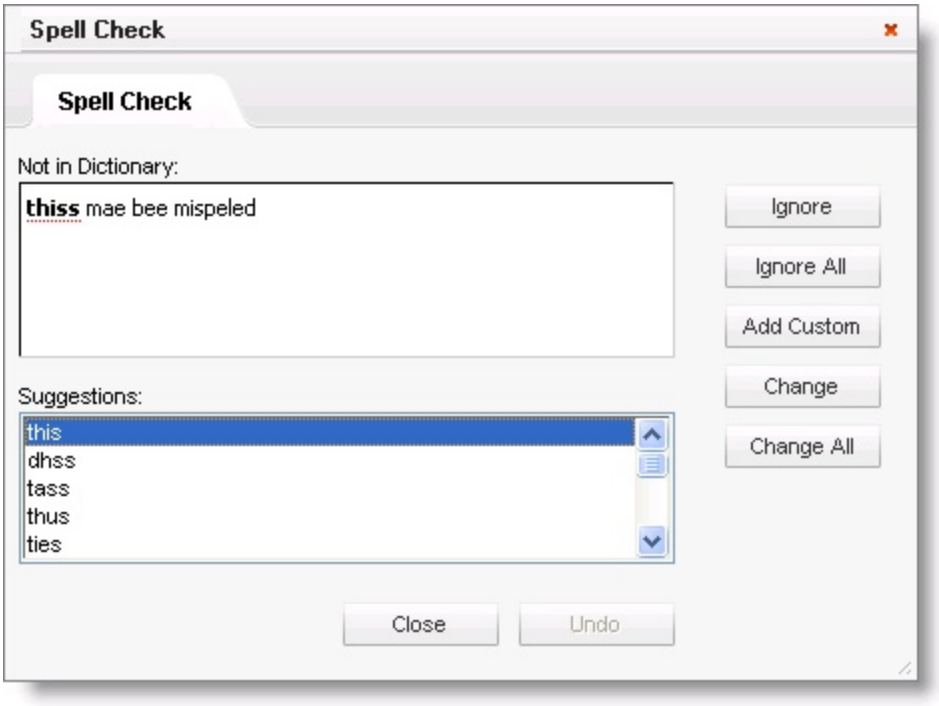


RadSpell controls in the toolbox

At its simplest, RadSpell is much like a standard ASP.NET validity checking component. You simply drop the component from the toolbox to the page and set the *ControlToCheck* Property. At runtime the component appears as a button or link (or the button can be hidden altogether) as in the figure below.

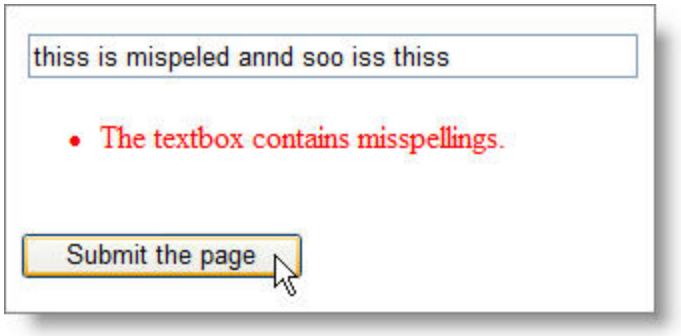
RadSpell at runtime

Clicking the component or calling a client side method to begin spell checking displays a dialog as shown in the figure below.



The RadSpell dialog

Adding a Telerik SpellCheckValidator and a standard ValidationSummary control allows you to prevent form submission when controls are misspelled, and to automatically display an error message in the ValidationSummary.



SpellCheckValidator preventing form submission

RadSpell is configurable in a number of ways:

- The spell checking provider defaults to TelerikProvider. TelerikProvider and PhoneticProvider are the same and use a phonetic algorithm to provide "sounds like" suggestions that work well for English language text. EditDistanceProvider works better for non-English languages than the phonetic provider. MicrosoftWordProvider uses COM interop to communicate with the Microsoft Word spell checker and is less performant than the other two options because of the extra communication layer.
- Localization: both the spell checker user interface and the words being checked can be localized. The control supports 22 languages and ships with English, German and French dictionaries ready to go. Localization materials for other languages are available for download in the Telerik Client.net area of the website.
- Dictionaries are customizable. Telerik provides a utility for importing and editing dictionaries, so you

can have dictionaries not only localized for particular languages, but for various fields of knowledge such as medical or legal. The user can optionally be supplied with a "Add to Custom Dictionary" button option on the spell checker.

- By default RadSpell can point at most ASP.NET and HTML controls that contain text and spell check without further configuration. RadSpell also has the ability to define a custom text source that expands the possibilities to include any control on a web page that provides JavaScript access to text. This could include a Macromedia Flash object, or the Internet Explorer API.
- The text that RadSpell pays attention to can be tailored through properties that specify words or word fragments that can be ignored. Form example, you can decide that email addresses and upper case words should not be checked, but that file names should always be checked.
- Appearance: RadSpell is skinnable and comes with a set of predefined skins, many of which match other Telerik controls.

3.3.2 Using RadSpell at design time

This section covers how to use the RadSpell and SpellCheckValidator controls in the design environment with no code including:

- Basic usage
- Commonly used properties
- Controlling the appearance of the spell check button
- Controlling the language and appearance of the spell check dialog
- Localizing the spell check
- Creating and using dictionaries and custom dictionaries.

Lab: RadSpell and SpellCheckValidator basics

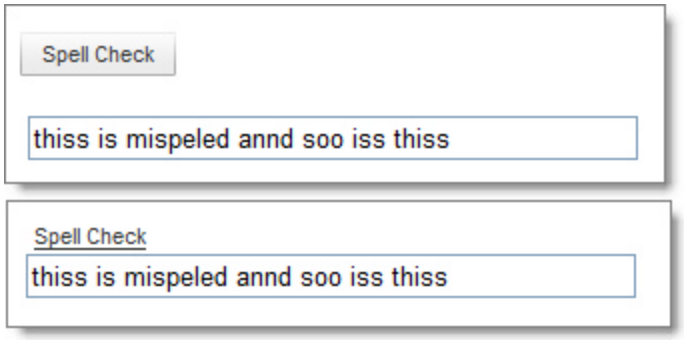
In this lab we will spell check a simple textbox with the minimum properties necessary to demonstrate RadSpell use We will also see how the ButtonType property changes the runtime appearance of RadSpell. And finally we will hook up the SpellCheckValidator to demonstrate how to prevent page submission when a control contains misspelled words.

1. Create a new web application project "GettingStarted".
2. Create a RadControls directory in the project and copy the Spell folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls
3. Drop a RadSpell and standard TextBox controls on the default page. Leave the default *ID* properties.
4. In the TextBox control *Text* property enter "thiss is mispeled annd soo iss thiss" or the misspelling of your choice.
5. In the RadSpell *ControlToCheck* property drop down the list and select the textbox control.
6. Run the application. Click the "Spell Check" button. Notice that the changes made to the text in the

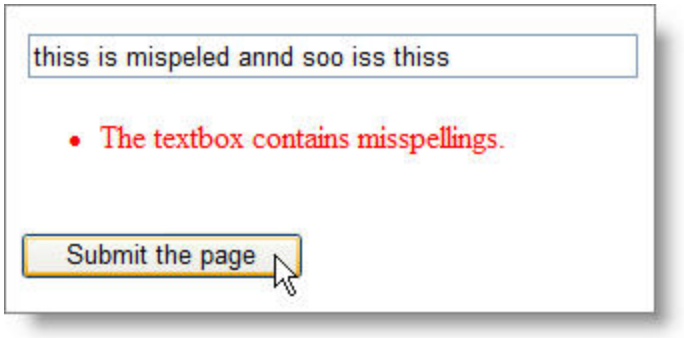
Spell Check dialog are updated in the textbox control. Canceling before completing the spell check will trigger a confirmation dialog allowing you to cancel all changes made up to that point. The Spell Check dialog undo button allows you to back out changes one at a time.

7. Experiment with the value of the *ButtonType* property and re-run the application. PushButton and ImageButton will look similar but have slightly different runtime behavior. Link button will show as an HTML link (see figure below). ButtonType of "None" can be used when spell checking is triggered by JavaScript or by the SpellCheckValidator.



Spell Checker with ButtonType of PushButton and LinkButton

8. Stop the application.
9. Now add a SpellCheckValidator. Set the *ControlToValidate* property to the RadSpell control (not the textbox itself) from the drop down list in the properties window. Set the *ErrorMessage* property to "The textbox contains misspellings".
10. Add a ValidationSummary control to the form. Leave the default properties.
11. Add a standard Button control to the form. Change the *Text* property to "Submit the page".
12. Set the RadSpell *ButtonType* property to None.
13. Run the application. Notice the "Spell Check" button does not display. Click the submit button to see the SpellCheckValidator working together with RadSpell and standard ValidationSummary controls.



Using RadSpell, SpellCheckValidator and ValidationSummary together

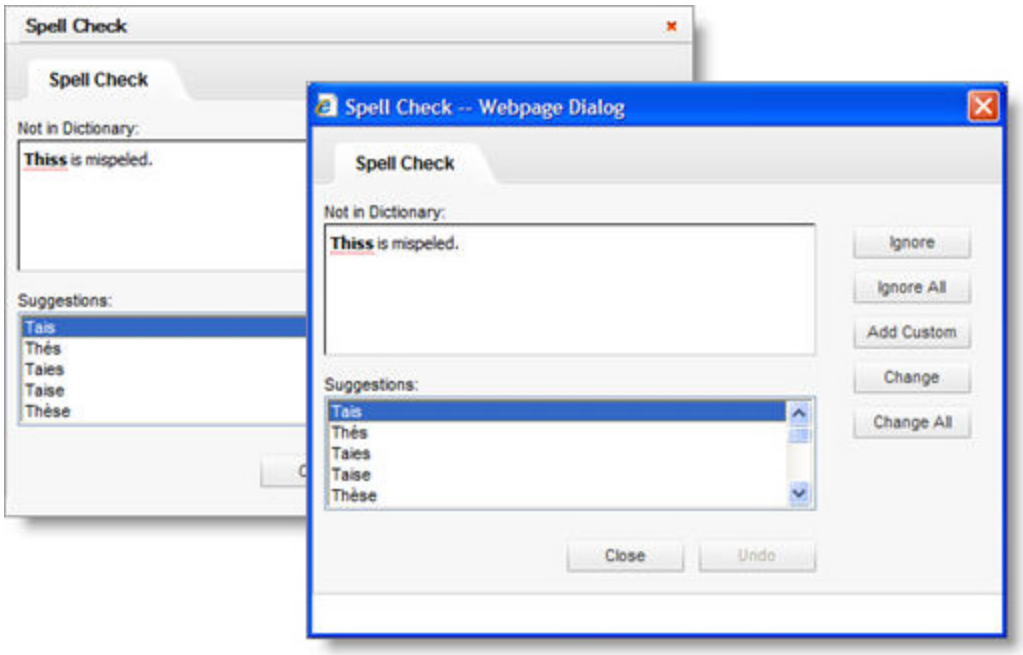
Other Important Properties

FragmentIgnoreOptions: Set this property to bypass common word fragments. The values for this property are None, FileNames, Urls, EmailAddresses, All. For example if the value of FragmentIgnoreOptions is Urls and EmailAddress then "support@telerik.com" and "http://www.telerik.

com" are omitted from the spell check.

WordIgnoreOptions: Set this property to bypass common patterns that should not be checked. The set includes None, UPPERCASE, WordsWithCapitalLetters, RepeatedWords, WordsWithNumbers. For example if WordIgnoreOptions is set to UPPERCASE and WordsWithCapitalLetters the words "Washington" and "ERROR" would not be flagged as misspellings.

UseClassDialogs: If true the spell check dialog is similar to those displayed by JavaScript functions window.open and window.showModalDialog. Defaults to false.



UseClassDialogs is true for the dialog shown on top

Lab: Localization

RadSpell is currently localizable to 20 different languages and ships with support for English, French and German. The two key properties for localization are *Language* and *LanguageDictionary*. *Language* controls the localization language for the spell checker dialog user interface. *LanguageDictionary* is the language for the dictionary that will be used to perform the spell check.

This lab will demonstrate:

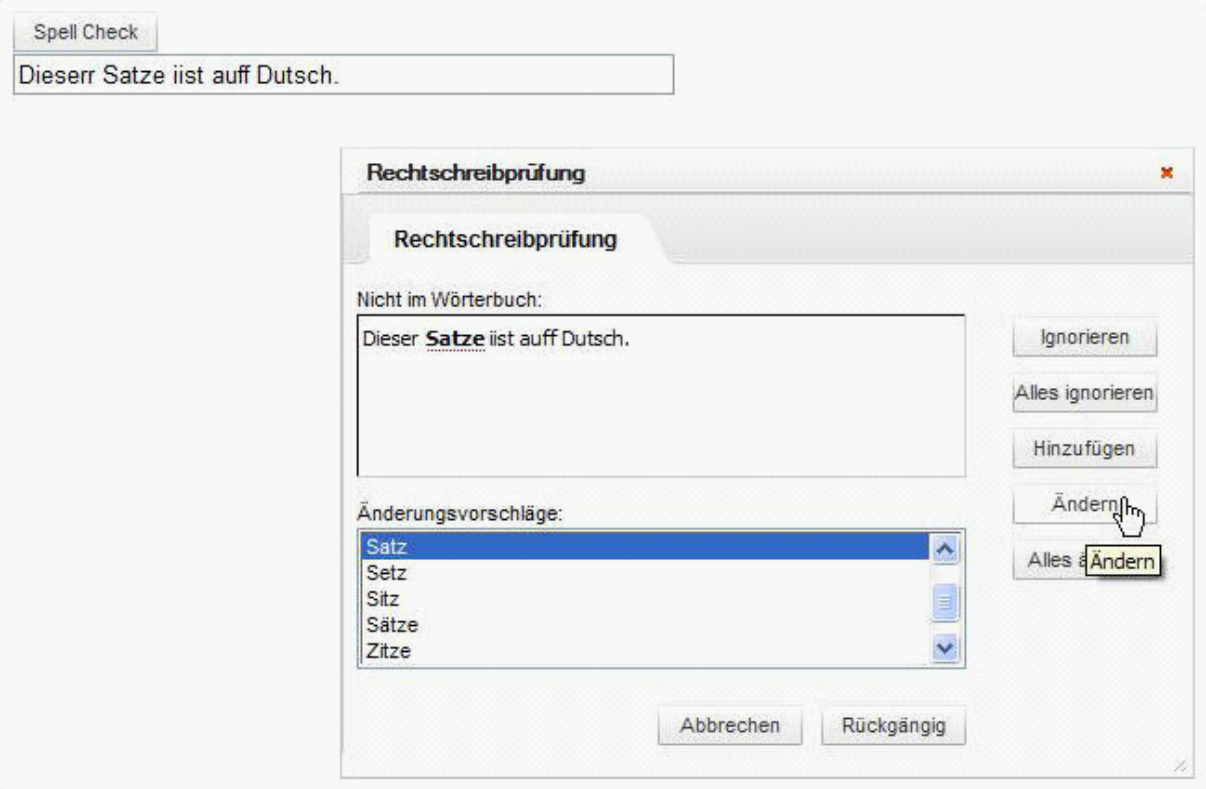
- Localizing the spell check to a single language
- Allowing the user to select from multiple languages to spell check.
- How to localize the language to spell check.
- How to localize the spell check dialog UI itself.
- How to change the spell check provider to a provider appropriate to non-English languages.

1. Create a new project "Localization".
2. Create a RadControls directory in the project and copy the Spell folder from installation RadControls

directory. For a default installation you find the RadControls directory in:

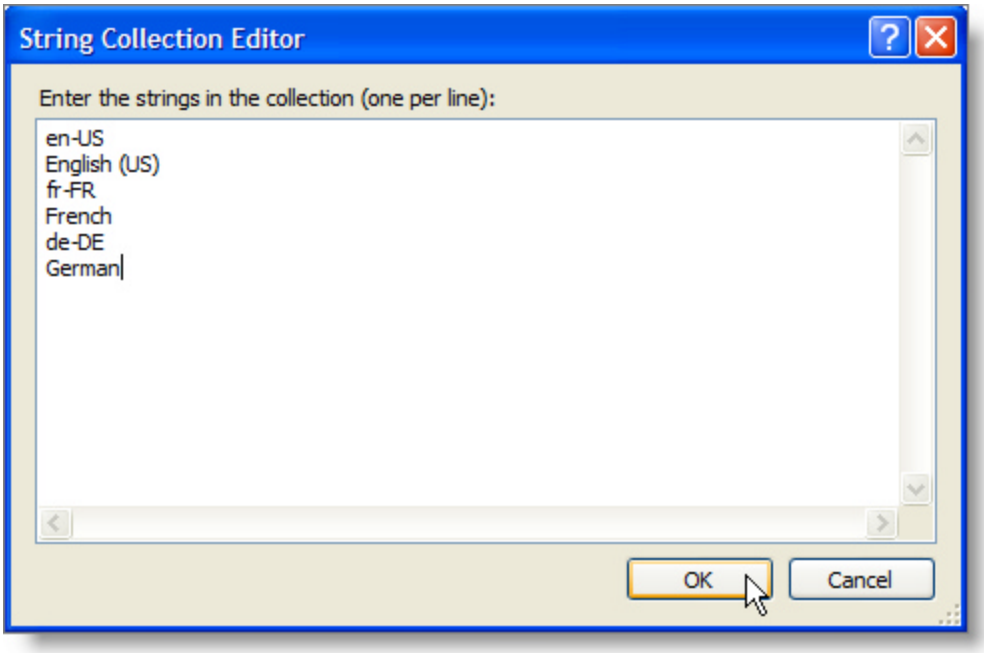
C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls

3. Add a RadSpell control to the default page.
4. Add a standard TextBox control to the page. In the *Text* property enter "Dieser Satz ist auf Deutsch" ("This sentence is in German" according to Google Translate).
5. Set the RadSpell *ControlToCheck* property to the TextBox control using the drop down list in the Properties window.
6. Set the RadSpell *SpellCheckProvider* property to "EditDistanceProvider" and the *EditDistance* property to "2". As the *EditDistance* property value increases, more suggestions are provided in the spell check dialog list but the dialog will take longer to display.
7. Set the RadSpell *Language* and *LanguageDictionary* properties to "de-DE".
8. Run the application. Clicking the "Spell Check" button will trigger a message box informing you (in German) that the spell check is complete. Try altering the TextBox text to introduce misspellings and run the spell check a second time to observe the behavior.



Spell checking using German for Language and LanguageDictionary

9. Stop the application.
10. In the RadSpell remove the settings for *Language* and *LanguageDictionary*. In the RadSpell *SupportedLanguages* property editor enter the culture codes and descriptions for US English, French and German as shown in the figure below. Each culture code is followed by an arbitrary description for the language. Note: currently this feature supports changing the dictionary language. The spell check dialog UI will still be controlled by the *Language* property.



Listing the supported languages to allow at runtime

11. Run the application again. The spell check in German will return immediately with a "Spell Check Complete" dialog. Both the English and French spell checks will flag every word as a misspelling.

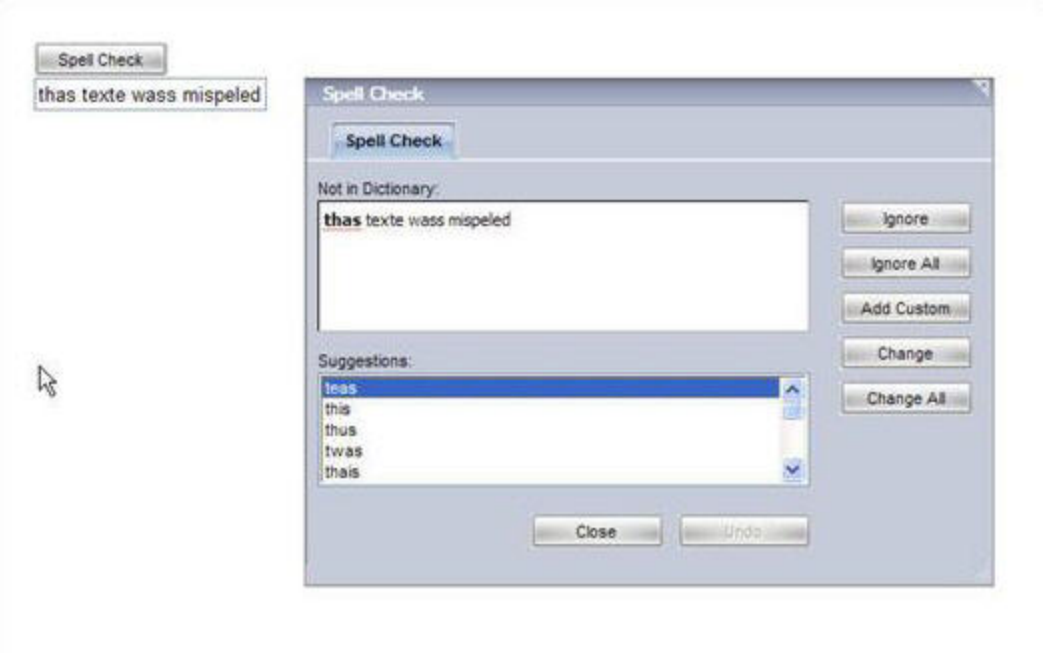
Lab: Customizing Look and Feel

The appearance of the RadSpell dialog can be configured by:

- Changing the Skin property to apply a whole set of styles to the dialog user interface elements. Some skins for RadSpell are common across the entire RadControl set. See http://www.telerik.com/demos/aspnet/Spell/Examples/Skin/DefaultCS.aspx for a complete graphical list.
- Editing the XML that controls the wording in the user interface. The file "SpellDialog.xml" contains strings localized for a given culture that can be edited.

This lab demonstrates configuring skins and customizing text for the spell check dialog.

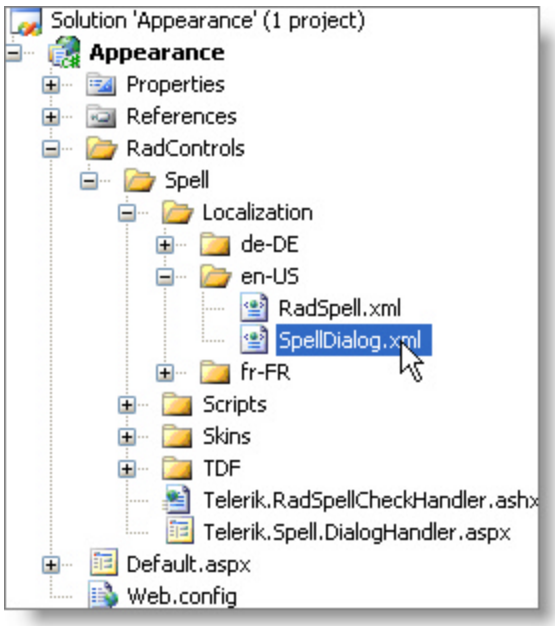
1. Create a new web application project "Appearance".
2. Copy the RadControls Spell directory to the project.
3. Add RadSpell and standard TextBox controls to the page. Set the TextBox *Text* property to "thas texte wass mispeled". Set the RadSpell *ControlToCheck* property to the TextBox control.
4. Set the *Skin* property to "Inox".
5. Run the application. Notice that both the "Spell Check" button and the dialog appearance have changed to fit the styles defined in the "Inox" skin.



Spell check dialog with Inox skin

6. Stop the application.

7. In the Solution Explorer, open the SpellDialog.xml file found in \RadControls\Spell\Localization\en-US.



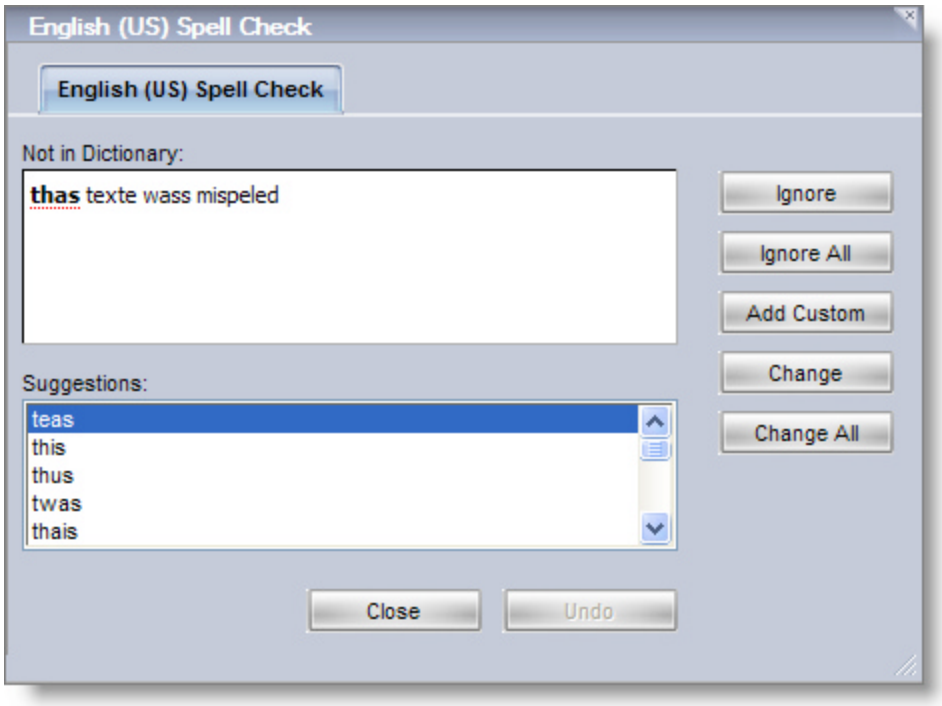
Selecting the SpellDialog.xml localization file

8. Change the value for the "Title" element from "Spell Check" to "English (US) Spell Check".

```
<?xml version="1.0" encoding="utf-8" ?>
<localization>
  <string id="Title">English (US) Spell Check</string>
  <string id="NoPermission">
    You don't have permission to access this page or your cookie has expired!<br />Please, refresh the page
  </string>
  <string id="Tab1HeaderText">Spell Check</string>
  <string id="ProgressMessage">Spell Checking in progress.....</string>
  <string id="Confirm">Do you want to apply or cancel the changes to the text so far?</string>
  <string id="ChangesMade">You have made changes to the text.</string>
  <string id="SpellCheckComplete">The Spell Check is complete!</string>
  <string id="Nosuggestions">No suggestions</string>
  <string id="Undo">Undo</string>
  <string id="UndoEdit">Undo Edit</string>
  <string id="Ignore">Ignore</string>
  <string id="IgnoreAll">Ignore All</string>
  <string id="Cancel">Close</string>
  <string id="AddCustom">Add Custom</string>
  <string id="Change">Change</string>
  <string id="ChangeAll">Change All</string>
  <string id="Help">Help</string>
  <string id="NotInDictionary">Not in Dictionary:</string>
  <string id="Suggestions">Suggestions:</string>
  <string id="AddWord1">Are you sure you want to add '</string>
  <string id="AddWord2">' to the custom dictionary?</string>
</localization>
```

Editing the "Title" element

9. Run the application. The dialog UI language for the title will **not** have changed. This is because the default localization files are resourced. You will need to change the Language property to "en-US" specifically to use the localization file in your project. Make the change to the RadSpell Language property and rerun the application. You should now see the change in UI language.

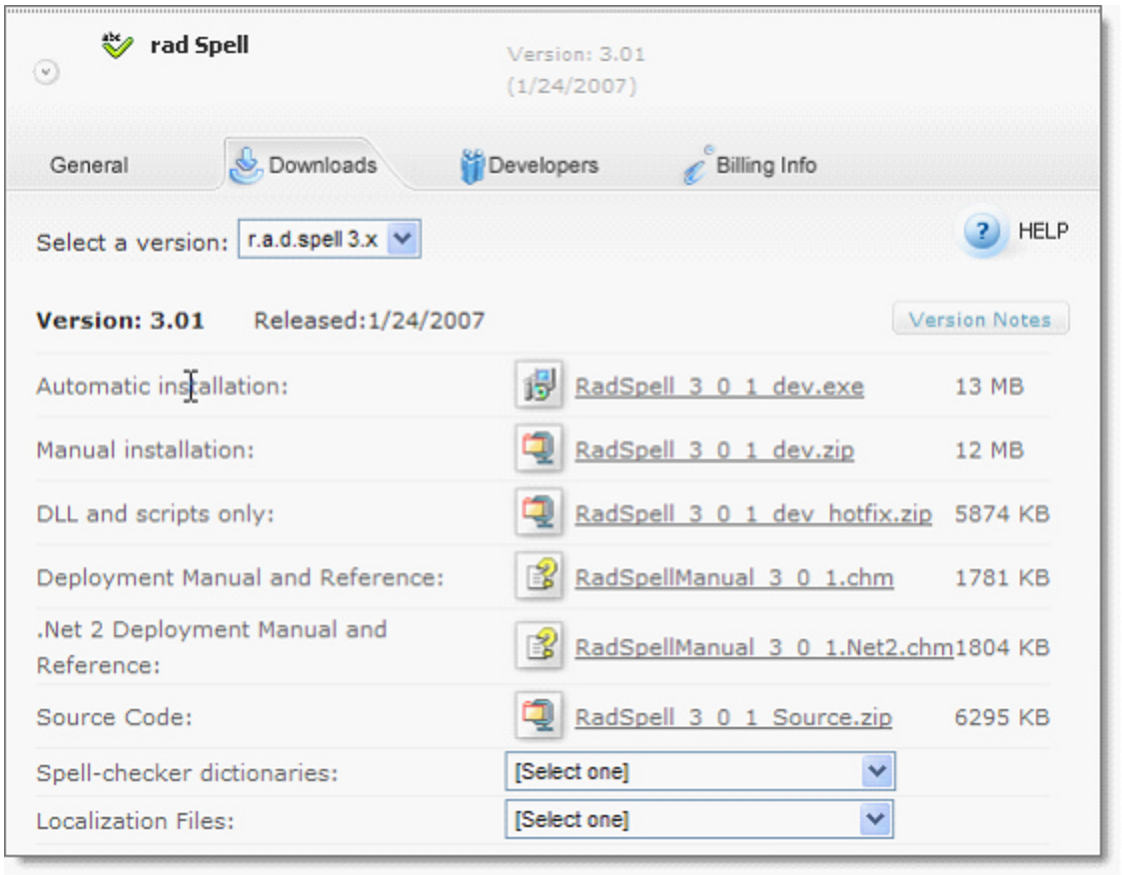


Spell check dialog with changed title text

Lab: Using dictionaries

Localized dictionaries are text files with a "tdf" (Telerik dictionary file) extension. If the language you want to localize to is not English, French or German you have two choices:

- Download a user-created dictionary from the Telerik site. The dictionaries and localized dialog xml files are available from Telerik help files can be downloaded from client.net menu item | My Licenses | My Products | <product> | Downloads tab (see figure below). Here you can also locate installation files and manuals for any of the RadControls.
- Create a dictionary for the language or subject specialty. The Using RadSpell at Runtime section covers how to create and edit new dictionaries.



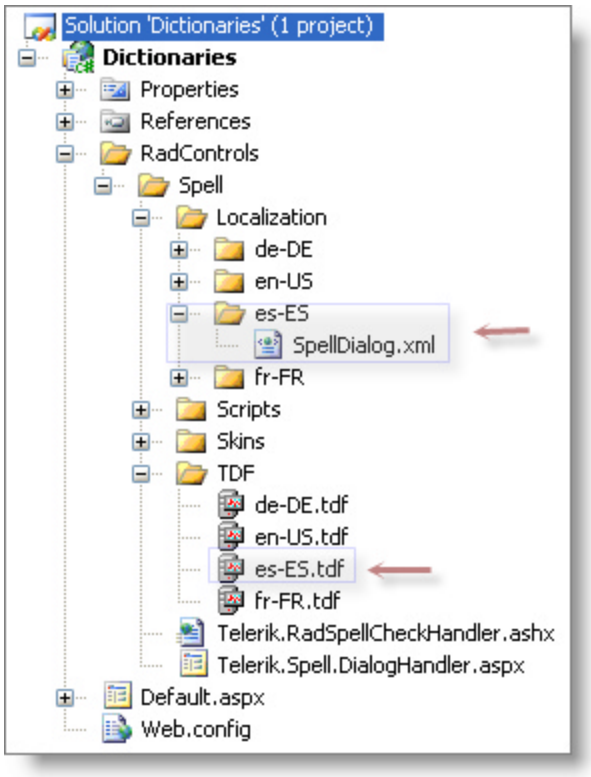
Client.NET downloads

In this lab we will localize the spell check to Spanish using a downloaded Spanish dictionary and dialog localization file. A copy of the files used in this lab are available in the \rad spell\data directory.

1. Create a new web application project "Dictionaries".
2. Copy the RadControls Spell directory to the project.
3. Add RadSpell and standard TextBox controls to the page. Set the TextBox *Text* property to "thas texte wass mispeled". Set the RadSpell *ControlToCheck* property to the TextBox control. In the TextBox *Text* property enter "Esto es una oración en españole". According to Google Translate, this means "This is a sentence in Spanish", but Espanol is misspelled.
4. Create a new directory "es-ES" in the \RadControls\Spell\Localization directory. Copy SpellDialog.

xml from to the directory. Here is where the project will find the Spanish text for the dialog itself. Note: You can find a copy of SpellDialog.xml in \rad spell\data

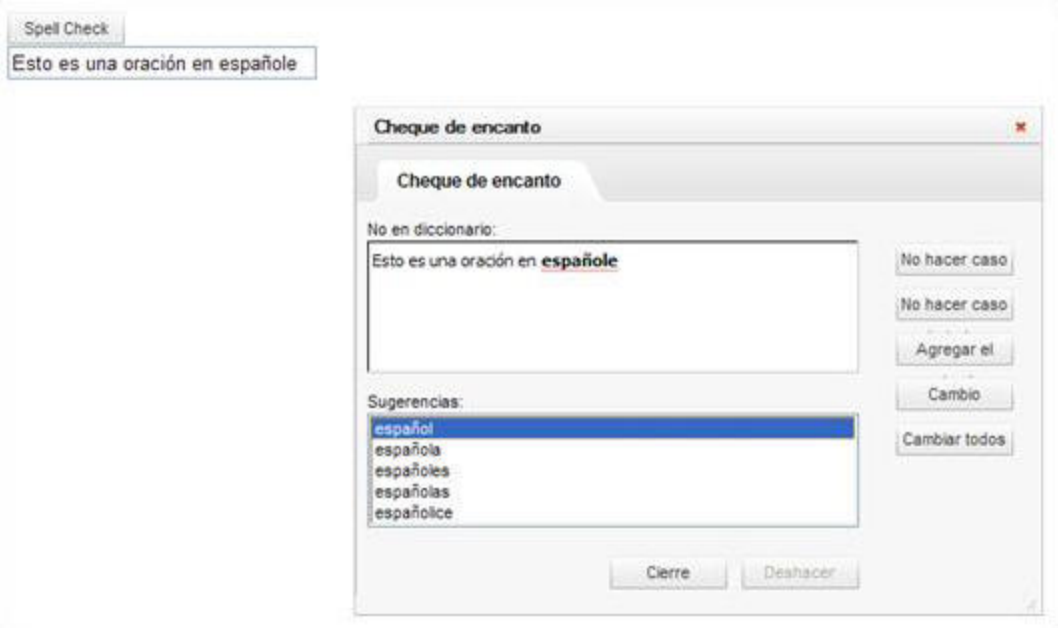
5. Copy the dictionary file "es-ES.tdf" to the \RadControls\Spell\TDF directory. Your project should now look like the figure below. Note: you can find a copy of es-ES.tdf in \rad spell\data.



Adding dialog xml and dictionary

6. Set the RadSpell *Language* and *LanguageDictionary* properties to "es-ES".

7. Run the application. Notice that both the dialog user interface and the suggestions from the dictionary are in Spanish.



The running project using the new dictionary

Custom Dictionaries

The user can add their own words to a custom dictionary at run time. This is a text file that resides in the same directory as the TDF files but is named according to the pattern <Language><CustomDictionarySuffix>.txt, for example "en-US-Custom.txt".

The properties that govern custom dictionary behavior are:

AllowAddCustom: This is true by default and makes the "Add Custom" button visible (see figure below).

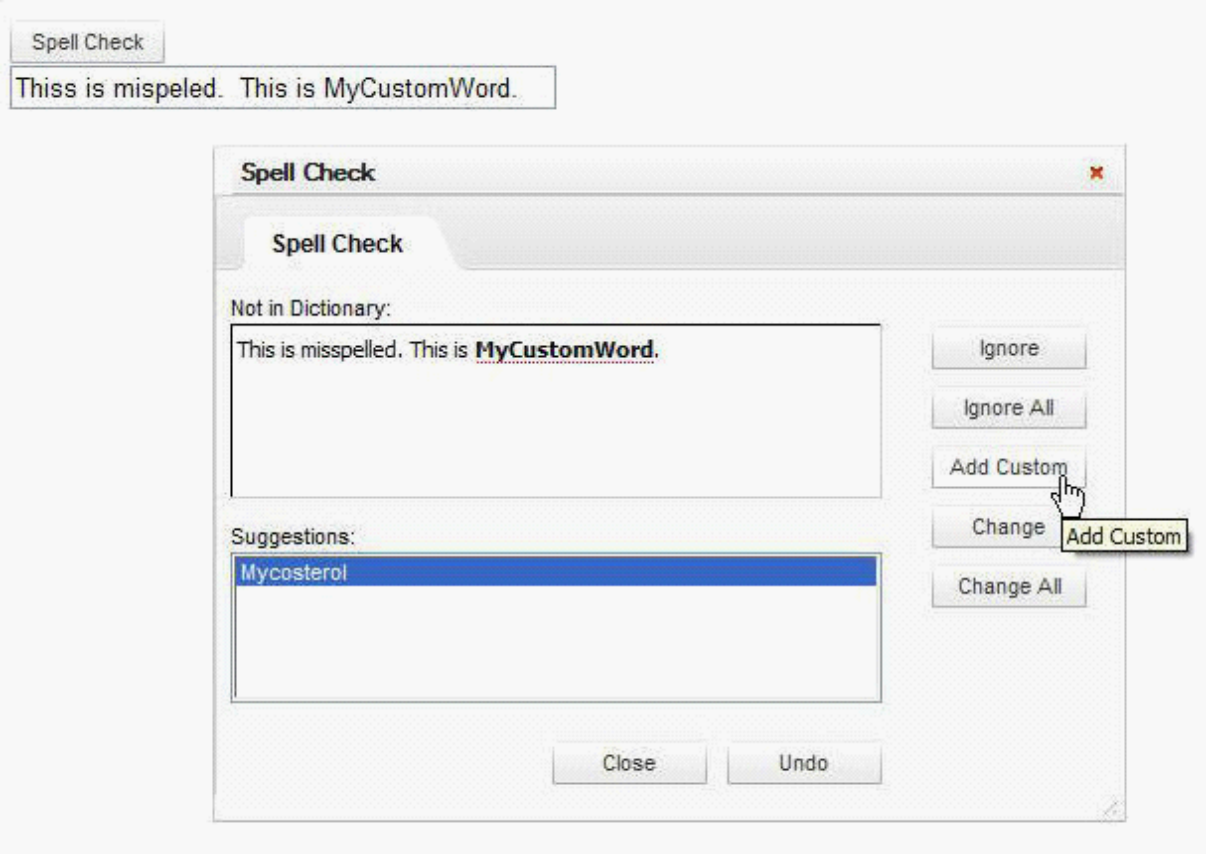
CustomDictionarySuffix: This defaults to "-Custom". If you want to allow custom dictionaries per user signed on, you could change the CustomDictionarySuffix to the user name or id. If user "Mary" is signed on, then the custom dictionary is named "en-US-Mary.txt".

CustomDictionarySourceTypeName: If you needed a more robust custom dictionary storage mechanism, for example to Oracle or MS SQL server, you can implement your own class. That class must implement the ICustomDictionarySource interface from the Telerik.WebControls namespace.

DictionaryPath: The directory that contains dictionary files.

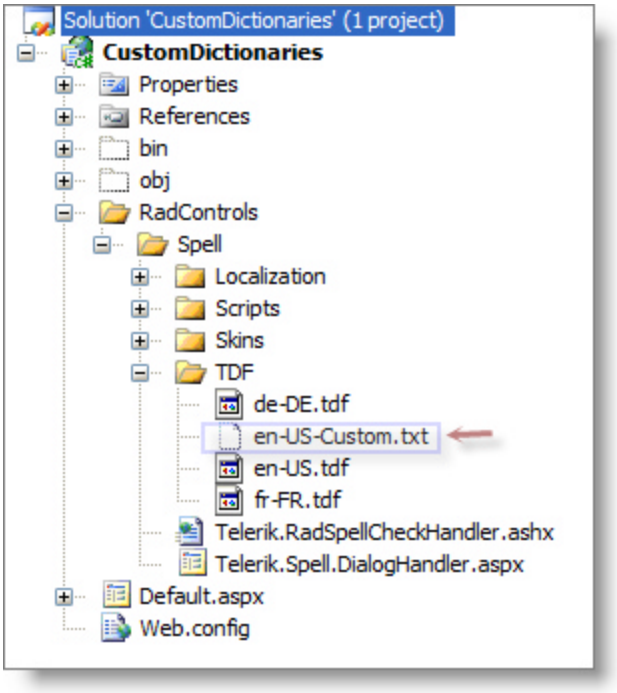
DictionaryLanguage: The language for the custom dictionary.

In the figure below the user adds the word "MyCustomWord" to the default custom dictionary.



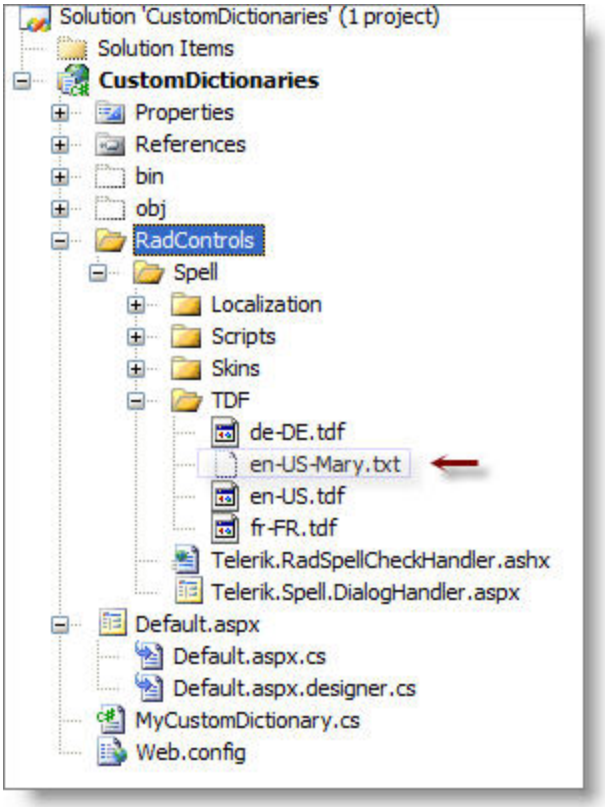
Adding a custom word

A file with the culture-suffix.txt of "en-US-Custom.txt" is created in the TDF directory and used for subsequent spell checks.



Custom dictionary is created automatically

In the figure below the CustomDictionarySuffix has been changed to "-Mary".



3.3.3 Using RadSpell at run time

This section demonstrates:

- How RadSpell can be used on a variety of different control types at runtime.
- Creating and maintaining new dictionaries for localization or special purpose (e.g. medical, legal, or technical dictionaries).

Lab: Assigning the control to be spell checked

You can assign the control to be spell dynamically at runtime. On the server this involves setting the *ControlToCheck* property and letting the RadSpell user interface handle the rest. This lab shows how ControlToCheck can changed server side by selecting from a list of the controls on the page.

1. Create a new web application "ControlToCheck".
2. Add a RadComboBox control to the page and set the *ID* property to "rcbControls".
3. Add a RadSpell control to the page. Leave the properties at their defaults.
4. Set up the RadControls directory in the project. See the directions for setting up the RadControls directory in the "RadSpell at design time" section.
5. Drop a standard ASP.NET TextBox on the page and set the *ID* property to "tbTest". Set the *Text* property to "thiss TextBox mae bee mispeled".
6. Drop an HTML TextArea on the page and set the *ID* property to "taTest". Right click and select "Run as Server Control". The reason for running this as a server control is only so we can iterate the page controls and find it. In the html Source view for the page enter the following text in the tag:

```
<textarea id="taTest" cols="20" rows="2" runat="server">thiss textarea mae bee mispeled</textarea>
```

7. Drop a standard ASP.NET button on the page and set the *ID* property to "btnTest". Set the *Text* property to "thiss button text mae bee mispeled".
8. In the code behind for the page write a private method to iterate the controls on the page and populate the combo box. The method should only list controls you want spell checked -- not the form, combo box or the spell checker itself. Pass the ListControls() method a reference to the RadComboBox the controls will be listed in, and the parent of the control to iterate.

```
C# example:
private void ListControls(RadComboBox radComboBox, Control parent)
{
    foreach (Control control in form1.Controls)
    {
        if ((control == null) ||
             (control.ID == null) ||
             (control is RadSpell) ||
              (control == radComboBox))
            continue;

          radComboBox.Items.Add(
            new    Telerik.WebControls.RadComboBoxItem(control.ID));
    }
}

VB Example:
private Sub ListControls(ByVal radComboBox as RadComboBox, _
   ByVal parent as Control)
```

```
 for Each control as Control in  form1.Controls
   if (control is Nothing) OrElse _
    (control.ID is Nothing) OrElse _
   (typeof control is RadSpell) OrElse _
    (control = radComboBox) Then
     ' continue
   End if
    radComboBox.Items.Add( _
     New    Telerik.WebControls.RadComboBoxItem(control.ID))
 Next
End Sub
```

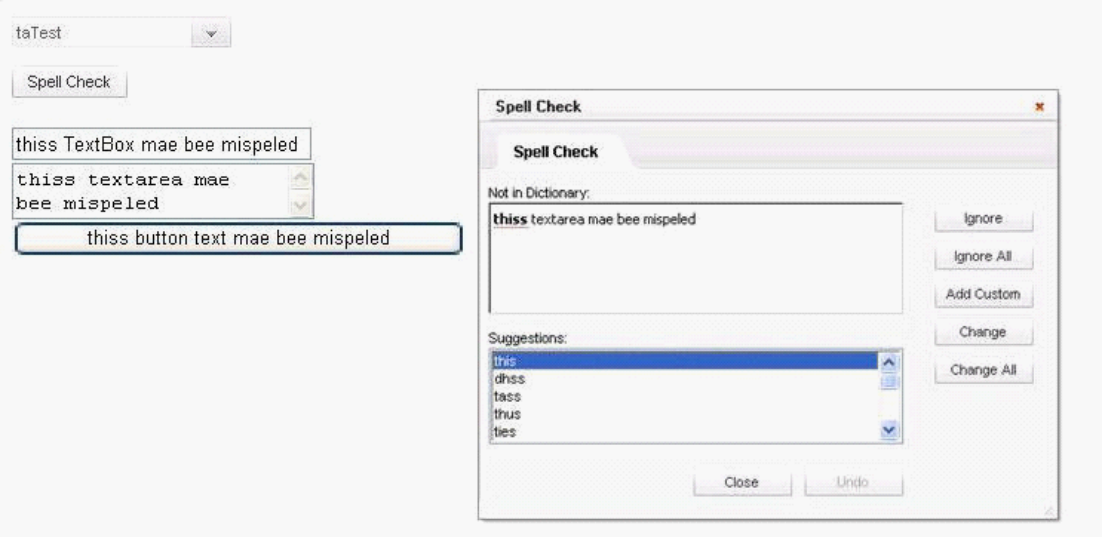
9. Call the method, passing a reference to the form.

```
C# example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
         ListControls(rcbControls, form1);
    }
}

VB example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
    ListControls(rcbControls, form1)
 End If
End Sub
```

10. Add an event handler for the RadComboBox SelectedIndexChanged event and simply assign the text for the selected controls ID to the *ControlToCheck* property of the RadSpell.

```
C# example:
protected void   RadComboBox1_SelectedIndexChanged(
    object  o,  RadComboBoxSelectedIndexChangedEventArgs  e)
{
     RadSpell1.ControlToCheck = e.Text;
}

VB example:
protected Sub  RadComboBox1_SelectedIndexChanged(  _
  ByVal o As Object, ByVal e As    RadComboBoxSelectedIndexChangedEventArgs)
  RadSpell1.ControlToCheck  =  e.Text
End Sub
```

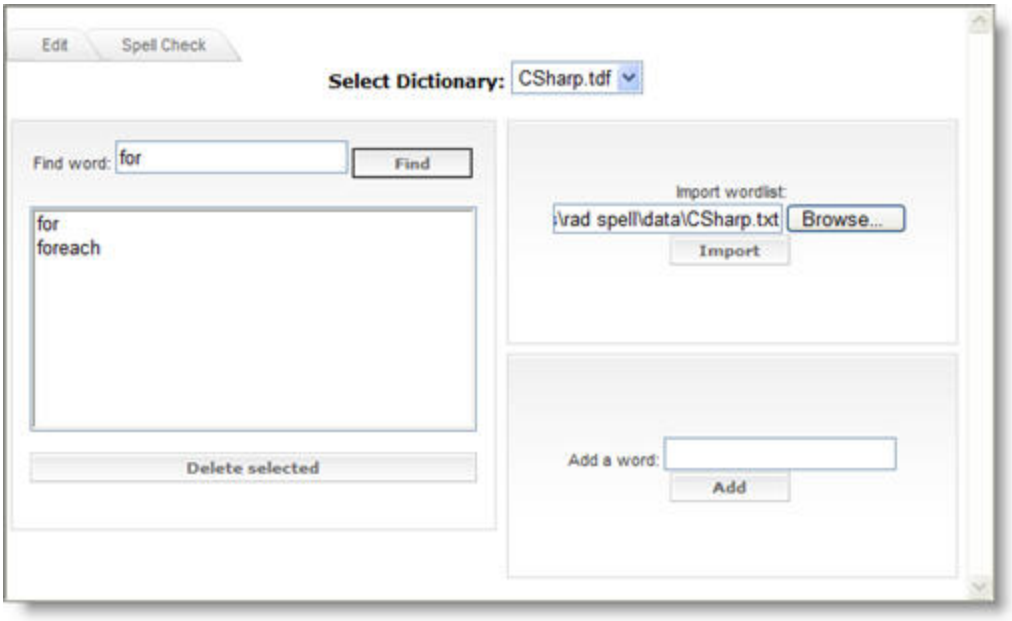
11. Run the project. Select each of the controls and run the spell check via the built-in button. Notice that the spell check works with the HTML control as well as the standard ASP.NET controls.



The running project with spell checker dialog displayed.

Lab: Creating and editing new dictionaries

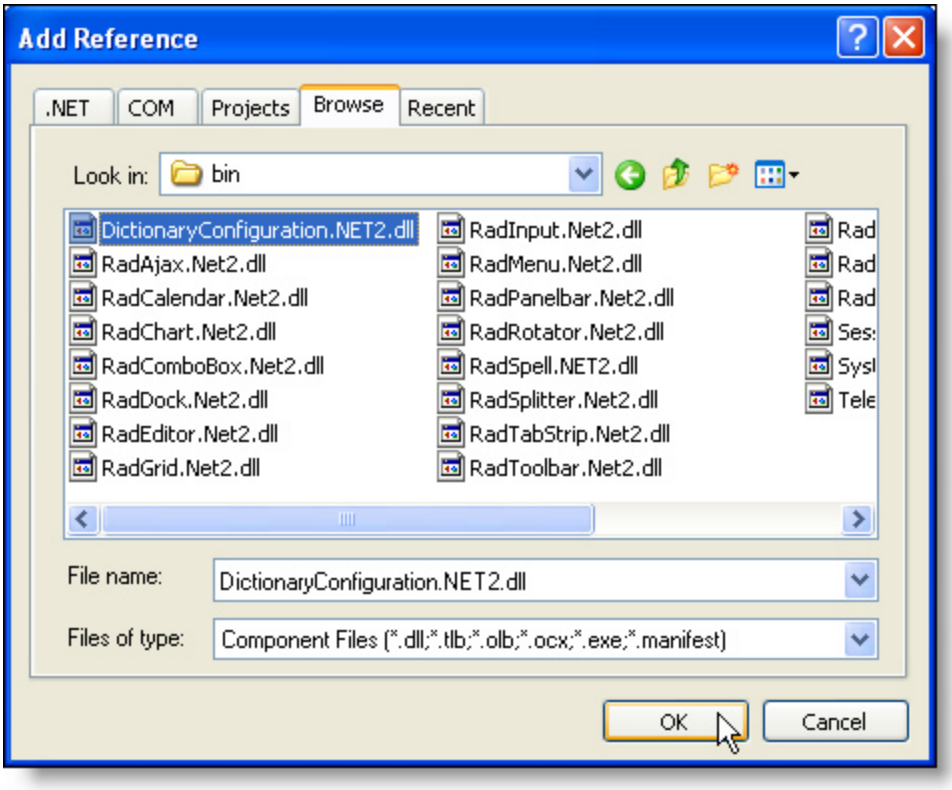
If the provided RadSpell dictionaries aren't sufficient you can create your own. You can augment an existing dictionary with words it doesn't have or create special purpose dictionaries. The lab you're about to begin employs a Telerik provided user control that handles adding, finding and deleting words in one of the dictionaries for the project. The user control also lets you import a text file with a list of words to a dictionary file. The lab project demonstrates loading a word list text file of C# key words to a dictionary and also allows the new dictionary to be tested with RadSpell. The lab data file CSharp.txt contains a list of C# reserved words, but you can load any set of words that is sorted alphabetically (in a pinch you can use Excel for this purpose) and has no leading or trailing spaces.



Importing a wordlist to a dictionary

1. Create a new web application project "DictionaryConfigurator".

2. Add a reference for DictionaryConfiguration.NET2.dll to the project. This assembly is found in the bin folder of the installation directory. In a default installation the path would be C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\bin.



Adding the DictionaryConfiguration reference

3. Copy the RadControls Spell directory to the project.

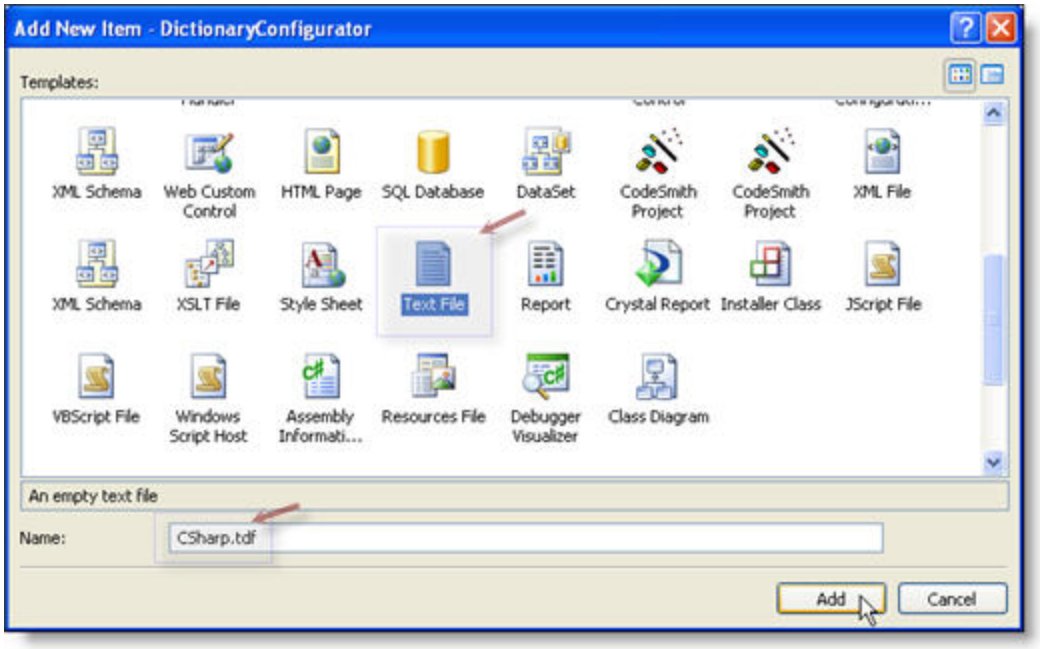
4. From the Telerik installation directory add EditDictionary.ascx to your project.

Note: In a default installation, EditDictionary.ascx is found in:

C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\Spell\DictionaryConfiguration.

5. Add a new empty dictionary to the project. This will be a blank dictionary file that will be filled later by importing a word list text file.

- Right click the TDF directory and select Add | New Item | Text File.
- Enter the file name as "CSharp.tdf".
- Click the Add button.



Adding a new blank dictionary

6. Add and populate a RadMultiPage control.
   - Add a RadMultiPage control to the page.
   - Set the *SelectedIndex* property to "0".
   - Drag EditDictionary.ascx from the Solution Explorer to the PageView on the MultiPage.
   - Drop a second PageView control into the RadMultiPage.
   - To the second PageView add the literal text "Select Dictionary:", a RadComboBox, a RadSpell, and a standard TextBox control. Set the *ID* property of the RadComboBox to "rcbDictionaries" and the *AutoPostBack* property to "true".
7. Add a RadTabStrip above the RadMultiPage. Create two new tabs "Edit" and "Spell Check". Set the *AutoPostBack* property to true. Set the *MultiPageID* property to the RadMultiPage control.
8. In the codebehind for the default page add "Using" entries for System.IO, DictionaryConfiguration and Telerik.WebControls.

```
C# Example:
using System.IO;
using DictionaryConfiguration;
using Telerik.WebControls;

VB Example:
Imports System.IO
Imports DictionaryConfiguration
Imports Telerik.WebControls
```

9. Add a private member of type Dictionaries to the page. This class automatically finds and lists the TDF files it finds in the project.

```
C# Example:
private Dictionaries _dictionaries = new Dictionaries(HttpContext.Current);

VB Example:
```

```
Private _dictionaries As Dictionaries = New Dictionaries(HttpContext.Current)
```

10. In the Page_Load use the Dictionaries object to return an array list with the names of the dictionary files.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
           rcbDictionaries.DataSource = _dictionaries.DictionaryNames();
         rcbDictionaries.DataBind();
    }
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 If Not IsPostBack Then
     rcbDictionaries.DataSource = _dictionaries.DictionaryNames
    rcbDictionaries.DataBind
 End If
End Sub
```

11. Create a helper method that sets the *DictionaryLanguage* of the RadSpell based on the currently selected dictionary. This just gets the current file name from the drop down list and strips the "TDF" file extension -- the remaining text is the dictionary name.

```
C# Example:
private void SetLanguage()
{
     RadSpell1.DictionaryLanguage =
      Path.GetFileNameWithoutExtension(rcbDictionaries.Text);
}

VB Example:
Private Sub SetLanguage()
  RadSpell1.DictionaryLanguage = _
      Path.GetFileNameWithoutExtension(rcbDictionaries.Text)
End Sub
```

12. Create event handlers for the RadTabStrip TabClick event and the RadComboBox SelectedIndexChanged event. Call SetLanguage() from both event handlers.

```
C# Example:
protected void   rcbDictionaries_SelectedIndexChanged(object o,
     Telerik.WebControls.RadComboBoxSelectedIndexChangedEventArgs  e)
{
     SetLanguage();
}

protected void  RadTabStrip1_TabClick(object sender, TabStripEventArgs e)
{
     SetLanguage();
}

VB Example:
Protected Sub   rcbDictionaries_SelectedIndexChanged(ByVal o As Object, _
  ByVal e As     Telerik.WebControls.RadComboBoxSelectedIndexChangedEventArgs)
  SetLanguage
End Sub

Protected Sub  RadTabStrip1_TabClick(ByVal sender As Object, _
  ByVal e As  TabStripEventArgs)
```

```
  SetLanguage
End Sub
```

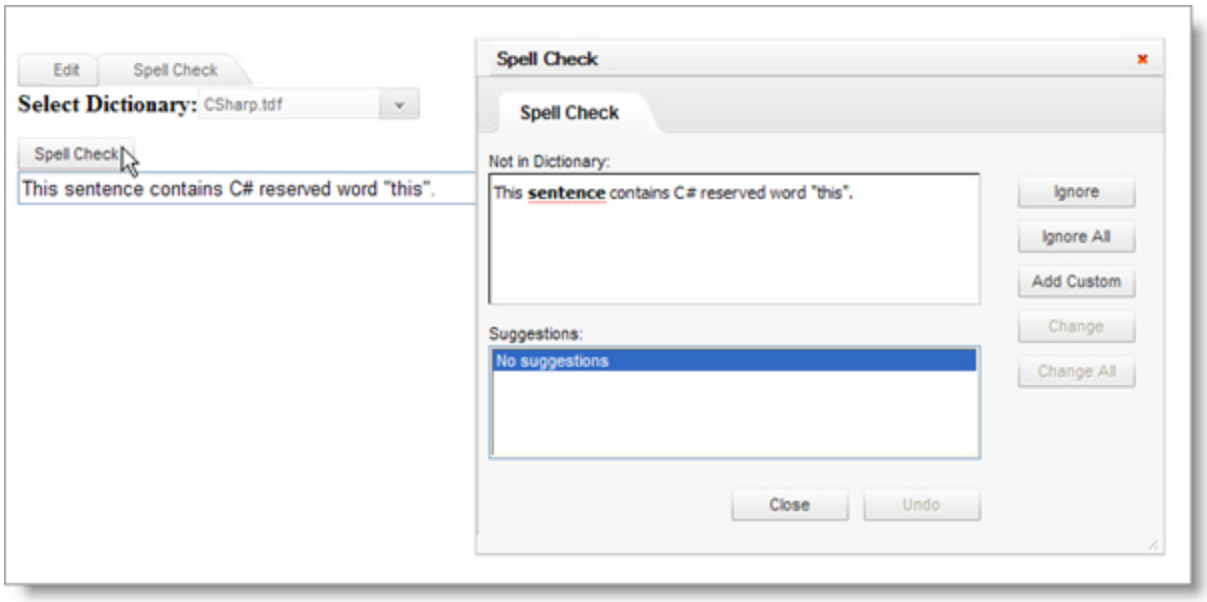
13. Run the application. Notice the RadComboBox lists CSharp.tdf and the other dictionaries for English, French and German.

14. Import CSharp.txt to CSharp.TDF. This step populates the dictionary file with words from the text file.

- Click the "Browse" button next to the "Import WordList text entry field.
- Select the text file "CSharp.txt", found in the \RadSpell\Data directory, two directories above this project directory.
- Click the "Import button.

15. Test the dictionary

- Type "for" in the "Find Word" field and click the "Find" button. The entries "for" and "foreach" show up.
- Search for the word "true". "true" is not in CSharp.txt and so isn't found in the dictionary. Using the "Add a word" entry, type "true" and click the "Add" button. Search again on "true" and it will appear. Select the word "true" and click the "Delete Selected" button.
- Change the dictionary from the drop down list to "en-US.tdf". Enter the word "for" in the "Find Word" field again and click "Find". A lengthy list of common English words starting with "for" are listed.
- Click the "Spell Check" tab.
- In the text box enter 'This sentence contains C# reserved word "this".' When CSharp.tdf is the language in the RadComboBox, all words except "this" are flagged as misspellings.



Spell check using the CSharp dictionary

When en-US.TDF is the selected dictionary the entire spell check completes successfully.

3.3.4 Client Scripting with RadSpell

This section will focus on controlling spell check functionality from the client:

- Triggering spell checking from the client
- Accessing RadSpell properties on the client
- Working with RadSpell client events
- Getting maximum flexibility from the RadSpell control by setting custom text sources

Client side basics

To trigger a spell check from the client side you need a reference to the RadSpell control. You get that from outputting the RadSpell ClientID to a parameter in special client function GetRadSpell(). Here's what it looks like:

```
var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
```

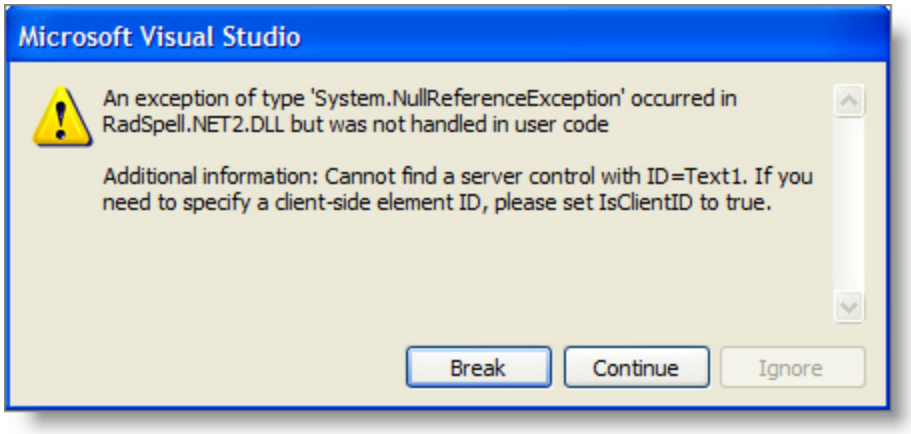
Call the StartSpellCheck() client side method to commence spell checking:

```
RadSpell1.StartSpellCheck();
```

Now you can trigger a spell check from any client side event. Here the onblur event triggers a spell check when the user tabs off of a text box:

```
<input id="Text1" type="text"
   onblur="GetRadSpell('<%= RadSpell1.ClientID %>').StartSpellCheck()" />
```

If you want to reference a control that is not a server control (not designated "runat='server'") you need to set the IsClientID property on RadSpell to "True". Otherwise the following error is displayed:



The error when IsClientID = "false"

Here's another example where the spell check is triggered by a standard HTML button:

```
<body>

  <script type="text/javascript">
    //<!--
    function spellCheck()
    {
      var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
      RadSpell1.StartSpellCheck();
    }
    //-->
  </script>

  <form id="form1" runat="server">
    <div>
      <radS:RadSpell ID="RadSpell1" runat="server" ButtonType="None"
        ControlToCheck="TextBox1" IsClientID="True"></radS:RadSpell>
      <input id="Button1" type="button" value="Spell Check"
        onclick="spellCheck()" />
      <br />
      <asp:TextBox ID="TextBox1" runat="server"
        Text="Thsi iss mispeld in a TextBox"></asp:TextBox>
  </form>
</body>
```

Triggering the spell check in client script

You can also change properties of RadSpell in client script. The DictionaryLanguage is changed to French in this example:

```
<script   type="text/javascript">
  //<!--
   function spellCheck()
  {
     var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
     RadSpell1.DictionaryLanguage = 'fr-FR';
     RadSpell1.StartSpellCheck();
  }
  //-->
</script>
```

To change properties of the spell check dialog, use the client side function GetDialogOpener(). This example changes the language of the dialog UI to French and displays in a "classic" window.

```
<script   type="text/javascript">
  //<!--
   function spellCheck()
  {
     var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
     RadSpell1.DictionaryLanguage = 'fr-FR';
     var dialog = RadSpell1.GetDialogOpener();
     dialog.UseClassicDialogs = true;
     dialog.Language = 'fr-FR';
     RadSpell1.StartSpellCheck();
  }
  //-->
```

```
</script>
```

Lab: Trigger spell checking on the client

1. Create a new project "ClientBasics".
2. Create a RadControls directory in the project and copy the Spell folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls

3. Add a RadSpell control to the default page.
4. Add an HTML TextBox control to the page. In the *Text* property enter "thsi iss mispeld".

```
<input id="Text1" type="text" value="Thsi iss mispeld" />
```

5. Just below the <body> tag add the following JavaScript code:

```
<script type="text/javascript">
//<!--
   function spellCheck()
  {
     var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
     RadSpell1.ControlToCheck = "Text1";
     var dialog = RadSpell1.GetDialogOpener();
     dialog.UseClassicDialogs = true;
      RadSpell1.StartSpellCheck();
  }
//-->
</script>
```

This sets the control to be spell checked, sets the spell check dialog to use the classic dialogs display style and initiates the spell check.

6. Run the application and verify that the Text1 control is being checked and that the dialog is of JavaScript "classic" style.

Events

Events fire on the client side when the spell check is started, finished, canceled and when the dialog is closing. Here is an example that simply logs the fact that the events occurred to a scrolling textbox:

```
<body>
   <script type="text/javascript">
  //<!--
   function spellCheck()
  {
     var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
      RadSpell1.StartSpellCheck();
  }
   function log(text)
  {
      var taEvents = document.getElementById("taEvents");

     taEvents.value += text + "\n";
      taEvents.scrollTop = taEvents.scrollHeight;
  }
  //-->
</script>

   <form id="form1" runat="server">
     <div>
```

```
        <radS:RadSpell ID="RadSpell1" runat="server" ButtonType="None"
          ControlToCheck="Text1" IsClientID="True"
          OnClientCheckCancelled="log('Cancelled')"
          OnClientCheckFinished="log('Finished')"
          OnClientCheckStarted="log('Started')"
         OnClientDialogClosing="log('Closing')"
       ></radS:RadSpell>
        <input id="Button1" type="button" value="Spell Check" onclick="spellCheck()" /><br />
       <input id="Text1" type="text" value="Thsi iss mispeld" /><br />
      <br />
       <textarea id="taEvents"></textarea>
  </form>
</body>
```

Each of the events takes a "sender", the RadSpell object itself, and "args" which varies depending on the event being fired. The following code snippet fires as the spell check begins, logs the *ID* of the RadSpell object and the name of the control being checked as well as the dictionary language being checked against. Notice the return value is based on the DictionaryLanguage being English -- If the DictionaryLanguage is not English the dialog will be prevented from displaying.

```
<script type="text/javascript">
//<!--
  . . .
function checkStarted(sender, args)
{
   log("spell: " + sender.Id + " started for: " + sender.ControlToCheck);
   var RadSpell1 = GetRadSpell(sender.Id);
    log(sender.DictionaryLanguage);
   return sender.DictionaryLanguage == "en-US";
}
  . . .
<radS:RadSpell ID="RadSpell1" runat="server" ButtonType="None"
   ControlToCheck="Text1" IsClientID="True"
      OnClientCheckStarted="checkStarted"
</radS:RadSpell>
  . . .
```

There are some events not yet surfaced as properties at the time of this writing:

- OnClientWordCorrected
- OnClientWordIgnored
- OnClientCustomWordAdded

You can still get these events by using the AttachEvent() method passing the name of the client side event and the handler function that will respond to the event:

```
. . .
function wordIgnored(sender, args)
{
   log("WordIgnored");
}
. . .
</body>

<script type="text/javascript">
  //<!--
    <%= RadSpell1.ClientID %>.AttachEvent('OnClientWordIgnored', 'wordIgnored');
  //-->
</script>

</html>
```

Note: You can find out the property names and values for an object using the JavaScript below. This general purpose routine allows you to explore the contents of "args".

```
function  getProperties(obj)
{
  for (prop in obj){
    log(prop + ": " + obj[prop] + "\n");
  }
}
```

Be aware that you are likely to return fairly cryptic results from this function, but at least you have a method of backing out what the object does. In the case of "args" for *OnClientWordIgnored* it returns the word ignored, the position of the word in the total text, and if the "Ignore All" button was clicked.

Custom text sources

The ability to define a custom text source significantly expands the flexibility of the RadSpell control. With a custom text source you have the ability to:

- Get the text to be checked from any source that can be accessed in JavaScript
- Return the spell checked results to any location accessible by JavaScript.
- Spell checking selected text
- Check multiple controls on a page with a single spell check dialog.

The custom text source is defined by calling the *SetTextSource()* method on the client side. SetTextSource requires you define a JavaScript object that contains GetText() and SetText(param) methods. Review the example below and note the following:

- The MyTextSource object has GetText() and SetText() functions that read and write the value for a TextArea.
- The spellCheck() function creates an instance of MyTextSource() and passes it in a call to SetTextSource().
- The definition of RadSpell doesn't have the *ControlToCheck* property set. If SetTextSource() wasn't being called an error would be raised.

```
<body>
     <script type="text/javascript">
    //<!--
       function MyTextSource()
      {
           this.GetText = function()
          {
                 return document.getElementById('TextArea1').value;
          }

           this.SetText = function(newValue)
          {
                 document.getElementById('TextArea1').value = newValue;
          }
      }

       function spellCheck()
      {
               var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
```

```
            var source = new MyTextSource();
             RadSpell1.SetTextSource(source);
             RadSpell1.StartSpellCheck();
      }
    //-->
    </script>

   <form id="form1" runat="server">

    <div>
        <radS:RadSpell ID="RadSpell1" runat="server" ButtonType="None">
       </radS:RadSpell>
        <textarea id="TextArea1" cols="40" rows="5">Thiis es mispeld</textarea>
         <button onclick="spellCheck(); " class="Button">Spellcheck</button>
  </form>
</body>
```

This example is functionally equivalent to simply setting the ControlToCheck, but with this technique we can place the corrected spell check results into another control. Taking the previous example as a starting point, a second text area is added and the SetText() function is redefined so that text returned from the spell checker is inserted to the second control:

```
function  MyTextSource()
{
     this.GetText = function()
    {
           return document.getElementById('TextArea1').value;
    }

     this.SetText = function(newValue)
    {
           document.getElementById('TextArea2').value = newValue;
    }
}
```

Extending this example a little further you can check all the text for multiple controls in a single shot. In this example we put several HTML elements for TextAreas in an array as a parameter to our text source object. In the GetText() the text for each text area is rolled into an array, concatenated to a single string and passed to the spell checker. Notice that the delimiter for the JavaScript join() function is a unique tag that also contains a line break ("
") tag so that each controls text will display on its own line. When the spell checker is done and passes the text back to SetText() the string is split apart on the same delimiter and placed back into the correct controls. Online Telerik examples also use a HtmlElementTextSource helper class for better encapsulation and flexibility.

```
function   MultipleTextSource(sources)
{
   this.sources = sources;

   this.GetText = function()
  {
    var texts = [];
    for (var i = 0; i < this.sources.length; i++)
    {
            texts[texts.length] = this.sources[i].innerHTML;
    }
      return  texts.join("<someUniqueTag><br/></someUniqueTag>");
  }
```

```
   this.SetText = function(text)
  {
      var texts = text.split("<someUniqueTag><br/></someUniqueTag>");
     for (var i = 0; i < this.sources.length; i++)
     {
            this.sources[i].innerHTML = texts[i];
     }
  }
}

function  multipleCheck()
{
   var sources =
  [
      document.getElementById('textarea1'),
      document.getElementById('textarea2'),
      document.getElementById('textarea3')
  ];

   var spell = GetRadSpell('<%= RadSpell1.ClientID %>');
    spell.SetTextSource(new  MultipleTextSource(sources));
   spell.StartSpellCheck();
}
```

Another custom text source example demonstrates access to other features available on the client. Internet Explorers Selection and Range objects allow text to be selected on the client at runtime, passed to the spell checker for modification and then inserted right back to the selection location.

```
function   SelectedTextSource(source)
{
   this.GetText = function()
  {
     source.setActive();
     var range = document.selection.createRange();
    return range.text;
  }

   this.SetText = function(text)
  {
     source.setActive();
     var range = document.selection.createRange();
    range.text = text;
  }
}

function  selectionCheck()
{
   var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');
      var source = document.getElementById('TextArea1');
    RadSpell1.SetTextSource(new  SelectedTextSource(source));
   RadSpell1.StartSpellCheck();
}
```

See the DHTML Methods reference on MSDN for further information on methods setActive(), createRange() and TextRange objects at:

http://msdn2.microsoft.com/en-us/library/ms533053.aspx

Also see the Telerik online examples for getting and setting text from a Macromedia Flash object.

Lab: Setting a simple Custom Text Source

1. Create a new project "SimpleCustomTextSource".
2. Create a RadControls directory in the project and copy the Spell folder from installation RadControls directory. For a default installation you find the RadControls directory in:

   C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls

3. Add a RadSpell control to the default page. Set the *ButtonType* property to "None". Do **not** set the *ControlToCheck* property.
4. In the code source for the default page add an HTML TextArea and button.

```
<textarea id="TextArea1" cols="40" rows="5">Thsi iis mispeld</textarea>
<br />
<button onclick="spellCheck();" class="Button">Spell Check</button>
```

5. Just below the <body> tag add the JavaScript tag:

```
<script type="text/javascript">
  //<!--


  //-->
</script>
```

6. In the JavaScript tag add code to define your custom text source object:

```
function MyTextSource()
{
    this.GetText = function()
    {
        return document.getElementById('TextArea1').value;
    }

    this.SetText = function(newValue)
    {
        document.getElementById('TextArea1').value = "Spell checked: " + newValue;
    }
}
```

7. In the JavaScript tag add code to create an instance of your custom text source object, assign it to the RadSpell control and initiate the spell check.

```
function spellCheck()
{
  var RadSpell1 = GetRadSpell('<%= RadSpell1.ClientID %>');

   RadSpell1.SetTextSource(new MyTextSource());
   RadSpell1.StartSpellCheck();
}
```

8. Run the application.

3.3.5 Summary

This section covered common scenarios for RadSpell including basic setup and usage, localizing dictionaries and the spell check dialog, custom user dictionaries, creating your own dictionaries for specific languages or special purposes, integrating with standard ASP.NET validation, and skinning. We also looked at how RadSpell can be used in the client to trigger spell checking, respond to events and

the many uses of assigning custom text sources.

3.4 RadWindow

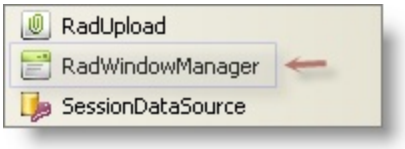
3.4.1 Getting Started

If you've used ASP.NET and JavaScript to simply build a dialog and return a value, you're going to love RadWindow. In the past this task cost some involved design and coding or at the very least, time consuming research.

Here are a few things you can accomplish with RadWindow:

- Tailor the exact appearance and language of all windows. This level of control encompasses alert/confirm/prompt dialogs, splash screens or any other window you care to define.
- Separate window handling functionality from window content.
- Drag windows around independently on the page.
- Enjoy cross browser compatibility. Forget about productivity sapping issues such as the "bleed through" of windowed items requiring IFrame "Shims" in IE. These problems are handled automatically for you.
- Make popup windows modal.
- Minimize windows, even putting the minimized icons in a "tray" area like a Windows application.
- Open windows within windows.

Compared with the power you get with RadWindow, the control is very easy to use. The RadWindowManager is a container for one or more RadWindow objects and handles the interaction with each window.



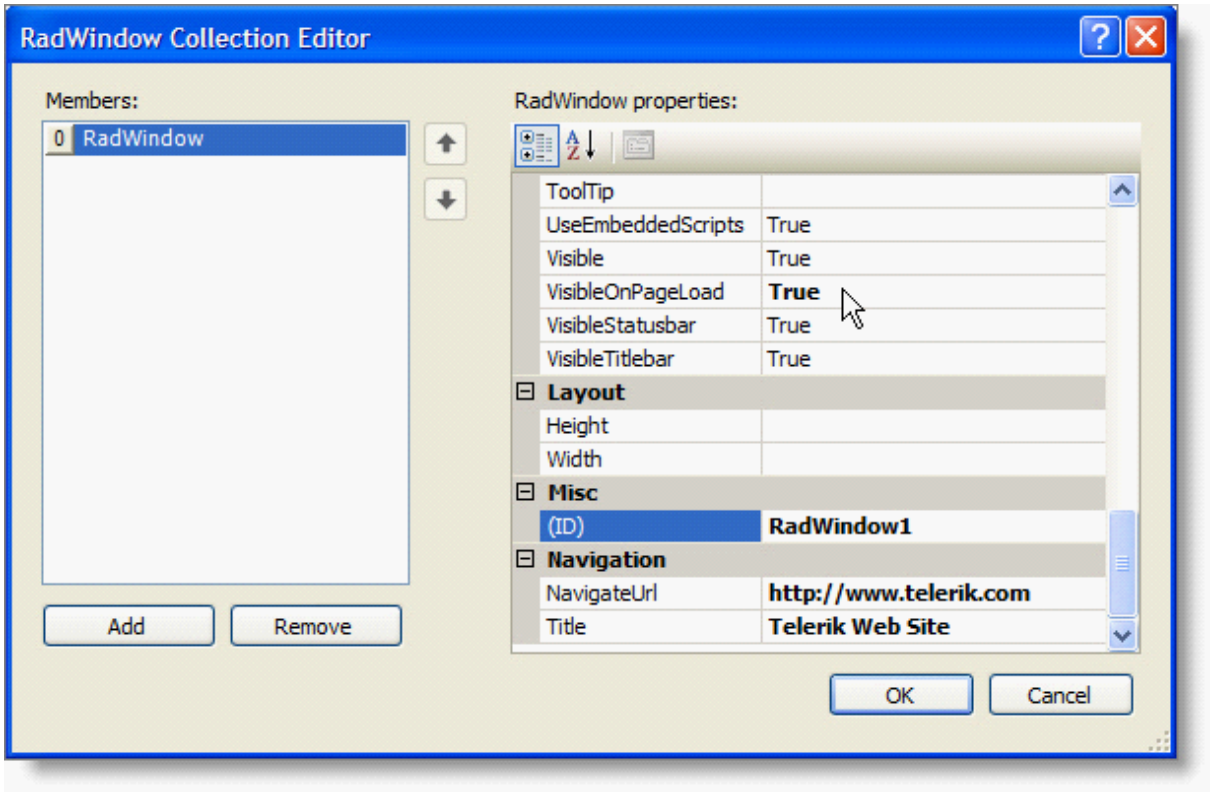
RadWindow in the toolbox

Lab: A Quick Tour

In this lab you get a quick sampling of RadWindow capabilities. You will see how to:

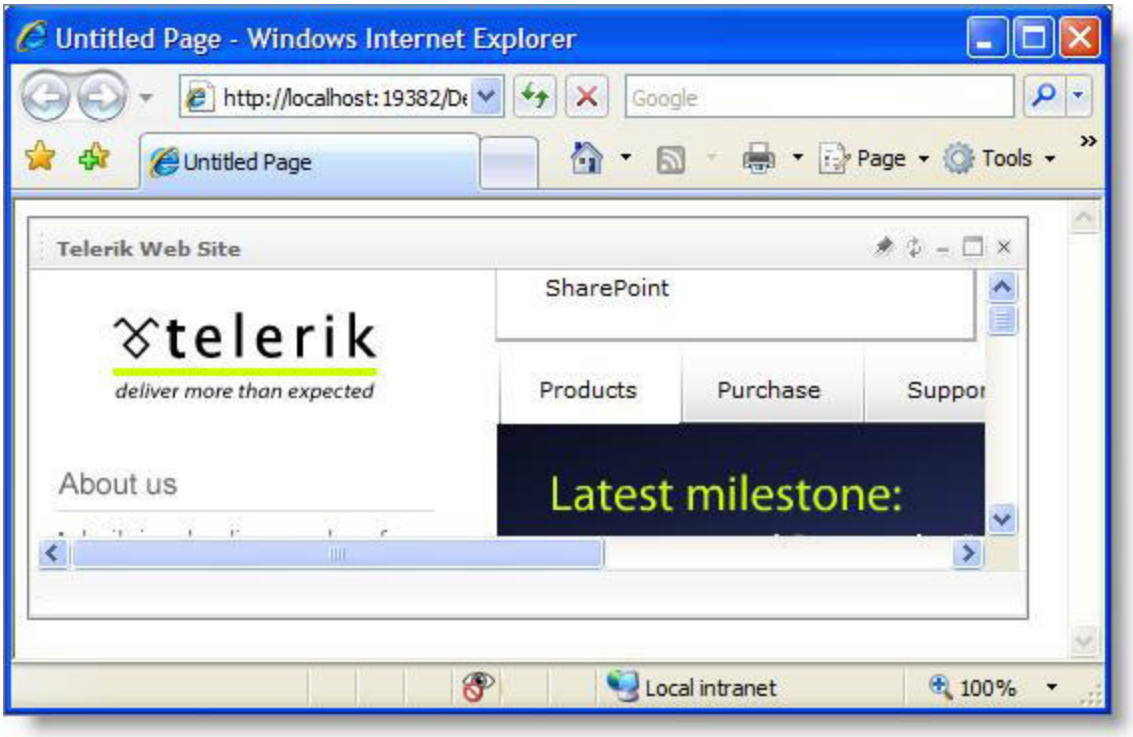
- Start a window on page load.
- Use the RadWindow Collection Editor.
- Associate a control with a window so that clicking the control causes the window to pop up.
- Set RadWindow properties to alter the appearance and behavior of the window.
- Display a modal window.
- Define multiple windows.
- Apply a skin to the windows.

1. Create a new web application "GettingStarted".
2. Create a new RadControls directory in your project. Copy the "Window" directory from the Telerik RadControls directory to the RadControls directory in your project.
3. Drop a RadWindowManager onto the default page.
4. Create the window object:
   - Click the ellipses for the *Windows* property to display the RadWindow Collection Editor.
   - Click the Add button to create a single window. This will create a RadWindow instance with an ID of "RadWindow1".
   - In the properties for the window set the *NavigateUrl* property to "http://www.telerik.com"
   - Set the *Title* property to "Telerik Web Site".
   - Set the *VisibleOnPageLoad* property to "True".



Working in the RadWindow Collection Editor

5. Run the application. You should see the window popup immediately. Experiment with the window, moving it on the form, using the "pin" button, minimizing and finally closing the window.



The popup window running in the browser

6. Stop the application.
7. Add a Label control from the Standard section of the tool box.
   - Set the *ID* property to "lblTelerik".
   - Set the *Font* | *Underline* property to "True".
   - Set the *ForeColor* property to "Blue".
   - Set the *Text* property to "Show Telerik Site".
8. Edit the properties for the window:
   - Go back to the *Windows* property collection editor and select "RadWindow1".
   - Set the *VisibleOnPageLoad* property back to "False".
   - Find the *OpenerElementID* property and set it to "lblTelerik".
9. Run the Application. This time the popup will not show automatically. Click the label on the page and the popup will display.

   Note: By the way, if you want to have the cursor change when mousing over the label you can wrap the label tag in a div (see example below). The important thing here isn't the label appearance but to show how the OpenerElementID uses standard controls, or HTML elements for that matter as triggers for opening RadWindows.

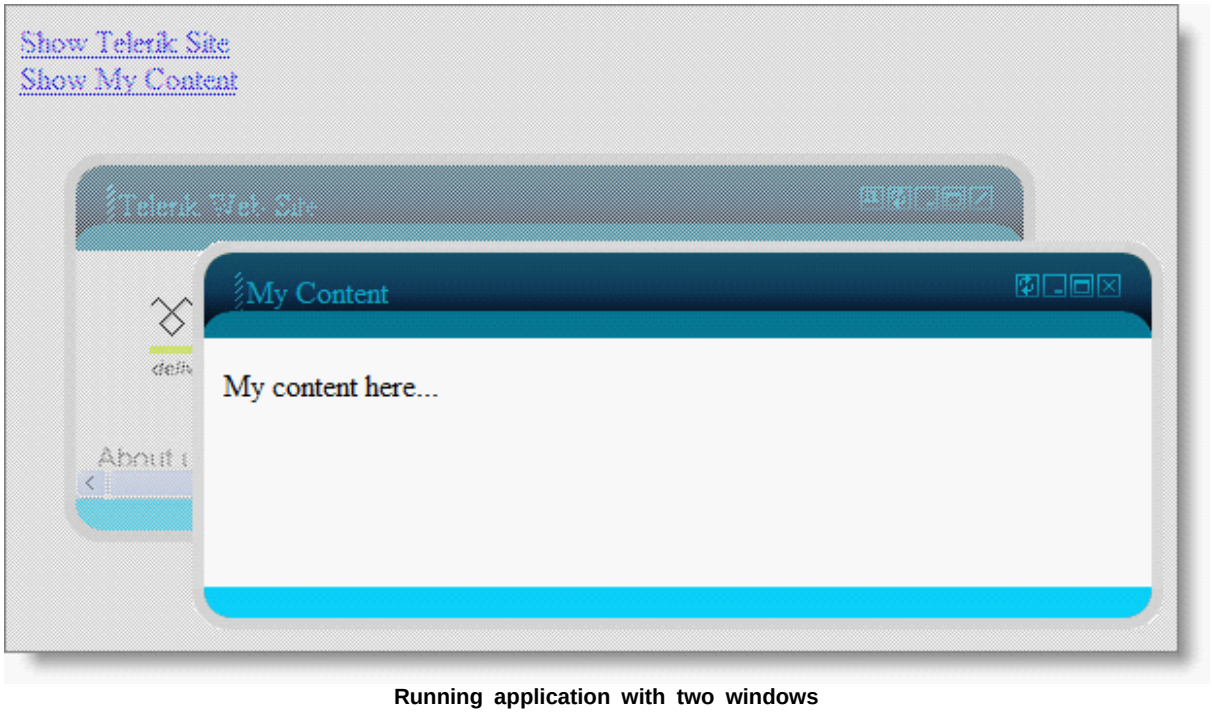
```
<div  style="cursor:  hand"></div>
```

10. Stop the application.
11. Set the RadWindowManager control *Skin* property to "Wizard".

12. In the Solution Explorer create a new web form.
    - Leave the form named as the default "Webform1.aspx".
    - Enter literal text "My content here..." in the form.
    - In the HTML view, set the head/title element value as "My Content". The markup should look similar to this example:

```
<head  runat="server">
    <title>My  Content</title>
</head>
<body>
     <form id="form1" runat="server">
    <div>
      My content here...</div>
    </form>
</body>
```

13. Copy lblTelerik and set the *ID* property of the copy to "lblMyContent".
14. In the Windows property collection editor add a second window:
    - Set the *ID* property to "rwMyContent"
    - Set the *NavigateUrl* property to "WebForm1.aspx".
    - Set the *OpenerElementID* property to "lblMyContent".
    - Set the *Modal* property to "True".
    - Set the *VisibleStatusBar* property to "False".
15. Run the application. Click both labels and notice the difference in execution. The "My Content" window has the title from the markup reflected in the title bar. The new window shows modally.



Running application with two windows

3.4.2 Using RadWindow in the Designer

Unique Properties

You may have noticed properties common between RadWindowManager and the elements of the Windows collection. Both RadWindowManager and RadWindow descend from RadWindowBase so that you can define common properties in RadWindowManager for all windows, but can also override them in RadWindow. Some properties are particular to RadWindowManager though are:

- *Skin*: Use a predefined skin from the RadControls/Windows/Skins folder or create your own using the default as a base.
- *Language*: You automatically have support for English, German and French. Check the Client.NET downloads section at the Telerik site for more translations.
- *UseClassicWindows*: default browser windows used, but these will trigger pop-up blockers.
- *SingleNonMinimizedWindow*: only one active window shown at a time, all others are minimized automatically.

The only unique RadWindow is the *Splash* property that indicates if this window should be used as a place holder for a splash screen.

Opening

There are several ways to open a window:

- Set *VisibleOnPageLoad* to true.
- Set the *OpenerElementId* to some html element that will automatically trigger the open. One way to do this is to bind the control to a server side control:

```
<radW:RadWindow
   OpenerElementId = "<%# MyButton.ClientID %>"
  . . .
</radW:RadWindow>
```

  Note: If you go this route be sure to call Page.DataBind() in the page load.

- Use one of the RadWindowManager or RadWindow client functions to display the window. We will look at these functions in the Client Scripting section coming up.

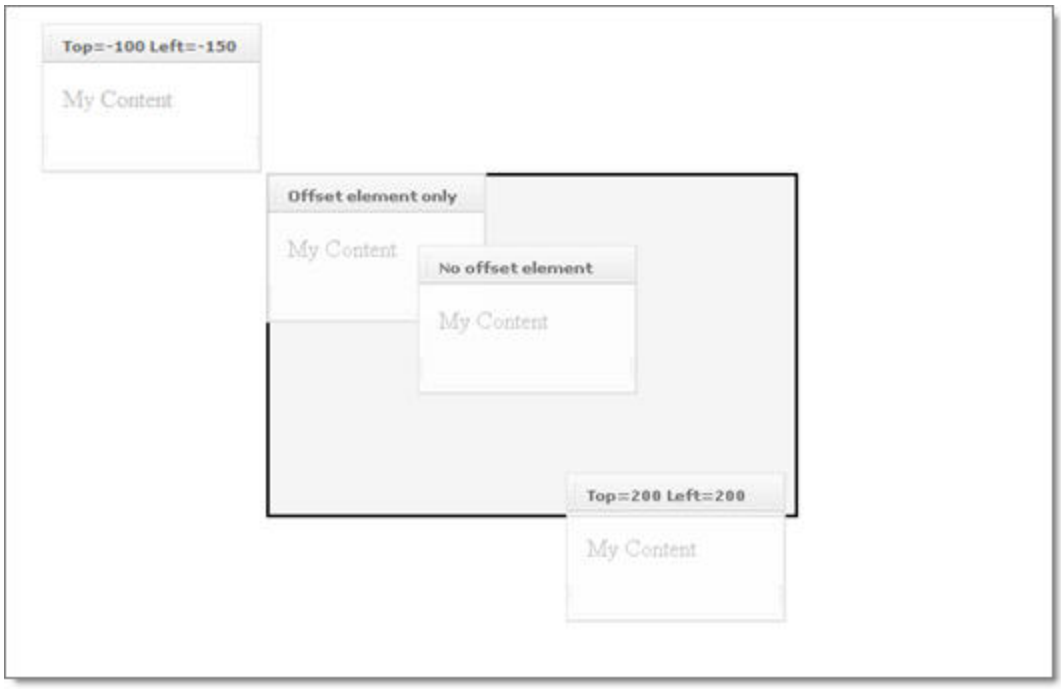
Positioning

To control the initial positioning of the window use *Top* and *Left* properties to place the window relative to:

- The web page.
- *...Or* use the *OffsetElementId* property. *OffsetElementId* indicates an html element, such as a div or button, that *Top* and *Left* will be relative to. You can use negative numbers to place the window above and to the left of an offset element. This can be handy if you want the popup close to the element that triggered it, like a button. You might also want to show windows only within a general area and leave other areas, such as menus, more uncluttered.

For example, in the figure below we have a div in the center of the page with a black border. The div is used by the windows as the *OffsetElementId.* The top left window has Top and Left properties at -100 and -150 respectively, placing it upper left from the div. If you specify the *OffsetElementId* only the window shows inside the upper left of the element. No offset element leaves the window to center in the

page.



Window positioned with and without offset

The markup that achieves the effect shown in the figure above is as follows:

```
<form id="form1" runat="server">
   <radW:RadWindowManager ID="RadWindowManager1" runat="server">
     <Windows>

       <radW:RadWindow
         VisibleOnPageLoad="true"
         VisibleStatusbar="false"
         Behavior="move"
         NavigateUrl="Webform1.aspx"
         Width="150" Height="100"
         Title="No offset element"/>

       <radW:RadWindow
         VisibleOnPageLoad="true"
         VisibleStatusbar="false"
         Behavior="move"
         NavigateUrl="Webform1.aspx"
         Width="150" Height="100"
         OffsetElementId="OffsetElement"
         Title="Offset element only"/>

       <radW:RadWindow
         VisibleOnPageLoad="true"
         VisibleStatusbar="false"
         Behavior="move"
         NavigateUrl="Webform1.aspx"
         Width="150" Height="100"
         OffsetElementId="OffsetElement"
         Top="200" Left="200"
```

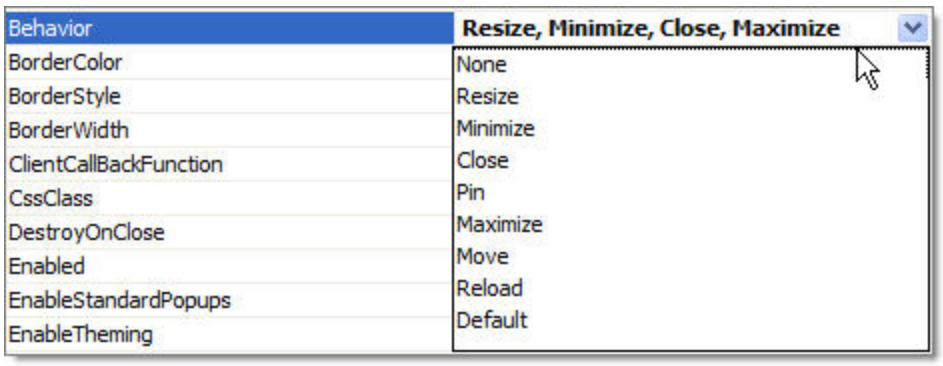
```
        Title="Top=200  Left=200"/>

      <radW:RadWindow
        VisibleOnPageLoad="true"
        VisibleStatusbar="false"
        Behavior="move"
        NavigateUrl="Webform1.aspx"
        Width="150"  Height="100"
        OffsetElementId="OffsetElement"
        Top="-100"  Left="-150"
        Title="Top=-100  Left=-150"/>
    </Windows>
  </radW:RadWindowManager>
  <div id="OffsetElement"  style="background-color:  WhiteSmoke;  width:  50%;  height:  50%;
     border-color:  Black;  border-width:  thin;  border-style:  solid;  left:  25%;
     position:  absolute;  top:  25%;  vertical-align:  middle;  text-align:  center;">
    <br />
    <br />
    <br />
    <br />
    <br />
    <br />
  </div>
</form>
```

Behaviors

The "behaviors" of a window determine the built-in commands that change the window state such as Maximize, Minimize, Pin, Resize, Close, Move and Reload. The default contains all the options.

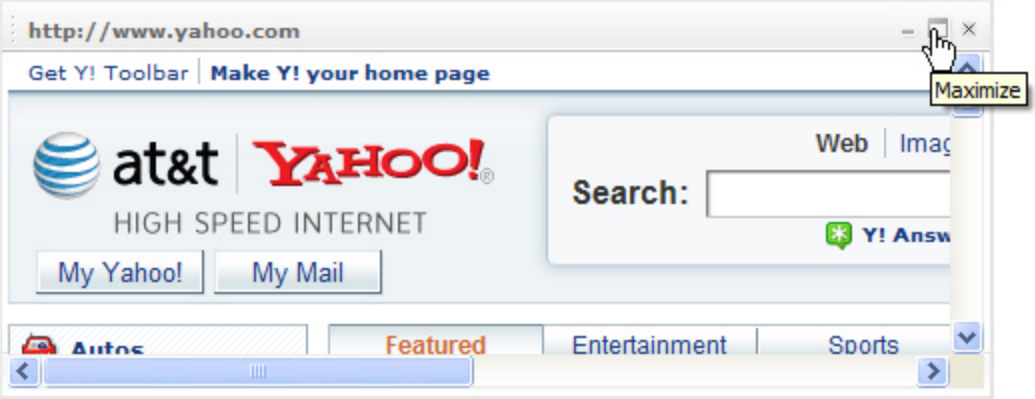
Default behaviors

Here is an example that sets the Behavior property to allow Resize, Minimize, Maximize and Close.



Setting the Behavior property

The resulting window might look like the figure below. Here we see that minimize, maximize, and close are available. You don't see the Pin button available and experimenting with the window would show you that although the window can be resized, it cannot be moved.



Minimize, Maximize, Resize and Close allowed

The Behavior property sets the allowable set of behaviors and the *InitialBehavior* indicates how the page will look when first displayed. Use InitialBehavior when you want the window to start minimized, maximized or pinned.

3.4.3 Using RadWindow at Runtime

Built In Dialogs

In addition to any window you can conceive to build, Telerik provides a number of predefined dialogs that can be used out of the box:

- Splash screens can be displayed on page load or during lengthy operations.
- Alert dialog popups let you communicate with the user when no response is necessary.
- Confirm dialogs get simple yes and no responses from the user.
- Prompt dialogs collect small amounts of information from the user and allow the application to see the response.

Lab: Splash Screen

Telerik provides a built-in splash screen function for use when loading the page or during a long operation. In both cases you fire the radsplash() function client side and you need to define a RadWindow with *Splash* property set to true. Many of the RadWindow properties, such as *Title,* will be ignored because the appearance of the splash screen is defined for each skin in CoreTemplates.xml.

The project for this lab displays the splash during load and by pressing a button. You will also get a chance to alter the CoreTemplates.xml file.

1. Create a new web application "Splash".
2. Create a new RadControls directory in your project. Copy the "Window" directory from the Telerik RadControls directory to the RadControls directory in your project.
3. Drop a RadWindowManager onto the default page.
4. Create the window object. Set the *VisibleOnPageLoad* and *Splash* properties to "True".

Note: These steps alone will provide you with a splash screen during page load. This is very brief for quick test pages but may be helpful if there's more content in the page. You can of course add JavaScript code to slow down the browser loading the page for the sake of simulation, but you should be able to see the splash well enough in the following steps where we display the window for a specific duration.

5. In the html source for the default page add a JavaScript function to display the splash screen. The function calls the built-in radsplash() function and passes "true" to show the splash screen. A timeout is set for "durationSeconds" that fires radsplash() a second time but passing "false" to hide the splash screen. Add the code just inside the form tag:

```
<script>
   function OpenSplash(durationSeconds)
  {
     radsplash(true);
      setTimeout( function() { radsplash(false); }, durationSeconds * 1000);
  }
</script>
```

6. Add an HTML button to trigger the splash. Pass the number of seconds you wish the splash screen to display:

```
<input type="button" onclick="OpenSplash(3);" value="Open Splash" />
```

7. The html for the form should look like this:

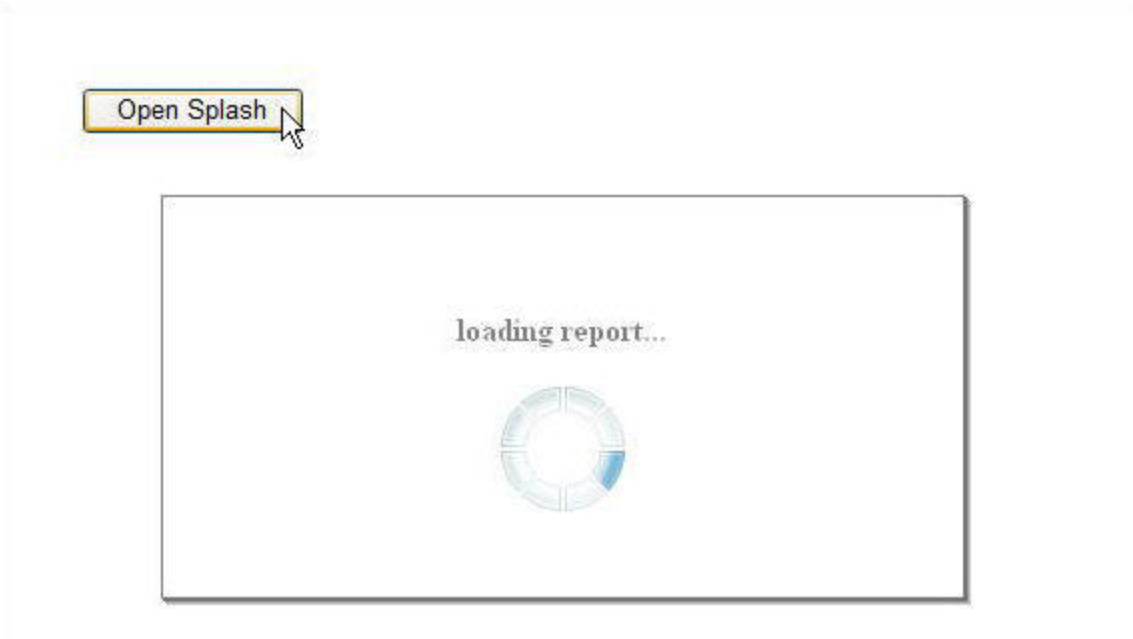
```
<form id="form1" runat="server">

  <script>
       function OpenSplash(durationSeconds)
      {
           radsplash(true);
            setTimeout( function() { radsplash(false); }, durationSeconds * 1000);
      }
   </script>

    <radW:RadWindowManager ID="singleton" runat="server" Skin="Default"
        SkinsPath="~/RadControls/Window/Skins">
      <Windows>
         <radW:RadWindow runat="server"  VisibleOnPageLoad="true" Splash="True">
         </radW:RadWindow>
      </Windows>
    </radW:RadWindowManager>
   <input type="button" onclick="OpenSplash(3);" value="Open Splash" />
</form>
```

8. In the directory \RadControls\Window\Skins\Default find the CoreTemplates.xml file and open it. The first template definition in CoreTemplates is the element "splashtemplate". There's a literal string in the template "loading ...". Locate this string and change it to "loading report...".

```
(4) - title text
(5) - application path
(6) - Text to show (in the alert, confirm, prompt);
<templates>
  <splashtemplate>
  <![CDATA[
    <div style="height: 200px; width: 400px; border: solid 1px #808080; text-align: center; bac
      <br /><br /><br />
      <div style="font-weight: bold; color: #808080; font-size: 16px;">loading report...</div>
      <br />
      <img src="{2}Img/splash.gif" height="62" width="62" alt="" />
    </div>
  ]]>
  </splashtemplate>
  <minimizetemplate>
    <![CDATA[
      <table border='0' style='display:inline' id='RadWMinimized{0}' class='RadWMinimizedActive
      <tr class='RadWTitleRow'>
        <td>
```

Changing the "loading" text

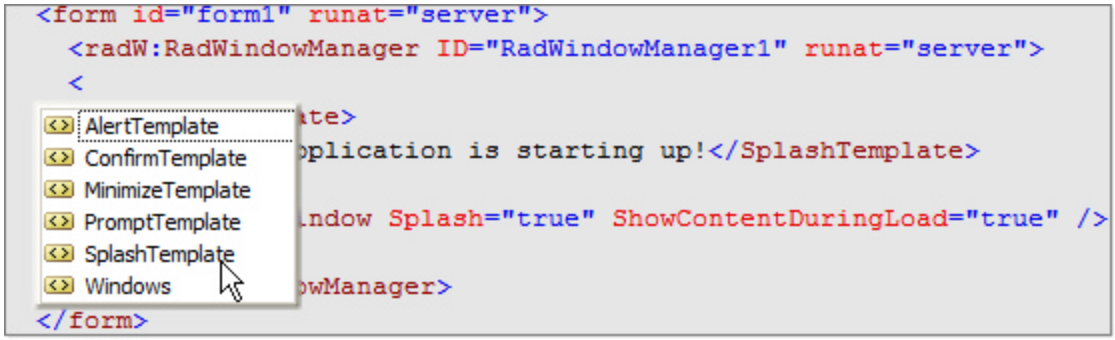
9. Run the application. Notice the changed "loading report..." text.



Running the application

More on templates

Be aware that you can also define templates directly within the markup and also on the server side in code. The figure below shows some of the possibilities inside the RadWindowManager tag in markup.



Template possibilities to choose from

Here's a minimal example that defines an alert template declaratively. It has a substitution parameter "6" that stands in for the message passed in the radalert call (see the comments at the top of CoreTemplates.xml for the full described list) and an ok button that knows how to get the current rad window and close it. You can copy examples from CoreTemplates.xml as starting points for your own templates.

```
<form id="form1" runat="server">
  <radW:RadWindowManager ID="RadWindowManager1" runat="server">
    <AlertTemplate>
      <center>
        My message: {6}
        <button class="Button" onclick="GetRadWindow().Close()">
          Ok</button>
      </center>
    </AlertTemplate>
  </radW:RadWindowManager>
  <button onclick=
   "radalert('Something has happened.', 200, 100, 'My Alert'); return false;">
    Alert</button>
</form>
```

Alert template example

You can also create templates on the server using RadWindowManager properties for AlertTemplate, SplashTemplate and so on. Each template is an ITemplate type so the first thing you need to do is implement an ITemplate in a class. Here's a quick example:

```
C# Example:
public class TimeSplash : ITemplate
{
  public void InstantiateIn(Control container)
  {
    Panel panel = new Panel();
    panel.Width = 250;
    panel.Height = 30;
     panel.BackColor = Color.WhiteSmoke;
     panel.BorderStyle = BorderStyle.Outset;
    panel.BorderWidth = 1;

    LiteralControl timeStamp =
     new LiteralControl("Today is " + DateTime.Today.ToLongDateString());
     panel.Controls.Add(timeStamp);
```

```
      container.Controls.Add(panel);
  }
}

VB Example:
Public Class TimeSplash
Implements ITemplate

 Public Sub InstantiateIn(ByVal container As Control)
   Dim panel As Panel = New Panel
    panel.Width = 250
    panel.Height = 30
     panel.BackColor = Color.WhiteSmoke
     panel.BorderStyle = BorderStyle.Outset
    panel.BorderWidth = 1
   Dim timeStamp As LiteralControl = _
     New LiteralControl("Today is " + DateTime.Today.ToLongDateString)
     panel.Controls.Add(timeStamp)
     container.Controls.Add(panel)
 End Sub
End Class
```

The ITemplate interface has the one InstantiateIn() method. In it we create a panel and populate it with a short message containing today's date. From there we can create the class and assign it to whatever template property is appropriate.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   RadWindowManager1.SplashTemplate = new TimeSplash();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  RadWindowManager1.SplashTemplate = New TimeSplash
End Sub
```

Dynamically created template running as a splash screen

Alert, Confirm, Prompt

Telerik also furnishes replacements for alert, confirm and prompt dialogs. These are called by radalert(), radconfirm() and radprompt() client functions respectively. These functions are a good deal more flexible than their browser counterparts. You can completely control the look and feel of most aspects of these dialogs and it's easier to get information to and from the dialog functions. Unlike the splash screen you do not need to define any windows to make these three predefined dialogs work.

Two notes that appear in the online help are worth repeating here:

- "A limitation of the radconfirm is that unlike the regular browser confirmation dialog (window.confirm), it cannot block the execution thread on the server. This is so because the execution of the thread cannot be blocked by Javascript. A possible workaround for some cases can be found in the following Code Library article: Block the execution thread with radconfirm. Please note that the approach described there work only for elements that have a click method".

"Block the execution thread with radconfirm" can be found at url:

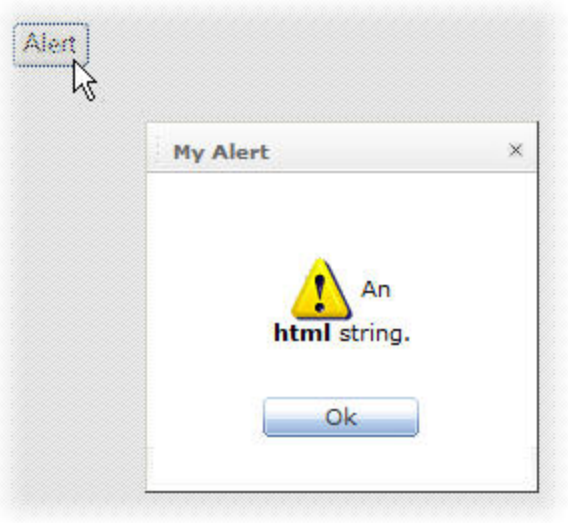
http://www.telerik.com/community/code-library/submission/b311D-gbaga.aspx

- "The RadWindow dialogs are modal, e.g. they block the content underneath and prevent the user from interacting with it, but there are ways to work around this behavior. If it is absolutely necessary that a popup is modal then the default browser popups should be used."

Alert

The easiest of the three is alert which simply takes a string to display, dimensions for the dialog width/height and a title string:

```
<button onclick=
"radalert('An <br /><b>html</b> string.<br />', 200, 100, 'My Alert'); return false;">
Alert</button>
```



Using radalert()

You have a certain amount of latitude customizing the alert because you can pass any html you want to the main content string and you can also edit CoreTemplates.xml to change the icon, button style etc. The RadWindowManager Skin property determines which CoreTemplates.xml copy will be used.

Calling an alert from the server

You can also execute an alert from the server side by a slightly sneaky trick of injecting the javascript into Label text. Here's how it works. Place a label on the form, but remove all text so that it's effectively invisible but still part of the page. In the server side button click event enter the javascript code that will be injected to the page just following the onclick. It's kind of a RegisterStartupScript() lite.

```
C# Example:
protected void Button1_Click(object sender, EventArgs e)
{
  const string scriptFormat = "<script language='javascript'>{0}</script>";
   Label1.Text = string.Format(scriptFormat,
     "window.onload = function(){radalert('An alert!', 300, 200);}");
}
```

```
VB Example:
Protected Sub Button1_Click(ByVal sender As Object, ByVal e As EventArgs)
 Const scriptFormat As String = "<script language='javascript'>{0}</script>"
  Label1.Text = String.Format(scriptFormat, _
   "window.onload = function(){radalert('An alert!', 300, 200);}")
End Sub
```

Confirm

Confirm also presents a message but also displays Yes/No buttons for the user. Unlike its regular browser counterpart this version is non-blocking so the rest of the page continues processing even before the dialog closes. Because the dialog is non-blocking it's designed to respond via a callback. The radconfirm() function call takes message, dimensions and title parameters, but it also takes a callback function name *and* an object you can use to stuff whatever information you think will be helpful in the callback. This object can be programmer defined or you can send a reference to some HTML object on the page.

Here's an example that is again triggered by a standard HTML button. Most of the action takes place in a JavaScript function that triggers the radconfirm() call and the callback function that responds to the users actions. The button onclick calls a programmer defined confirm() JavaScript function:

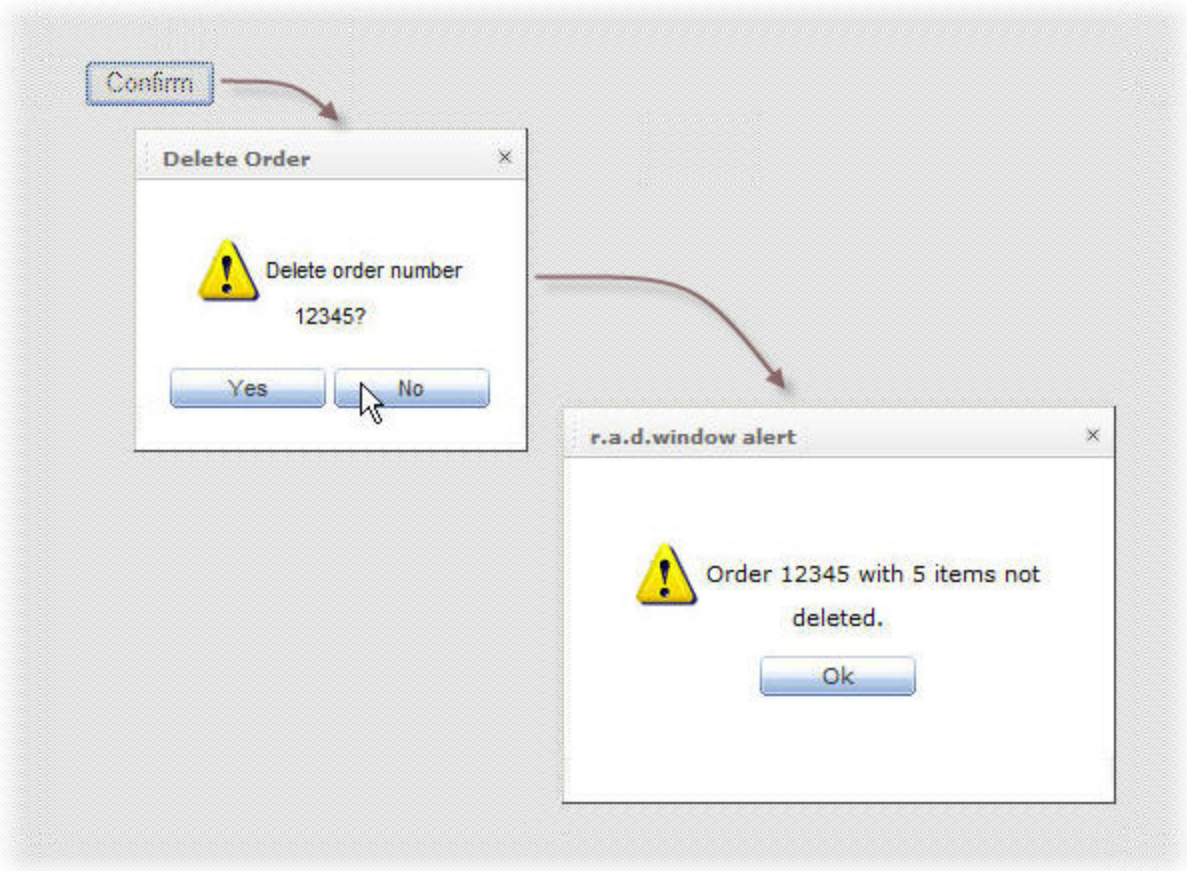
```
<button onclick="confirm();">Confirm</button>
```

The confirm() JavaScript function creates a new custom Object and assigns new properties Number and ItemCount. This is just to demonstrate that any amount of information could be passed along using the object as a container. In radconfirm() we pass the text that will be shown in the confirm dialog, the callback function name that will process the users response, width/height dimensions, our custom order object and finally the title for the dialog. The custom object and the dialog title parameters are optional.

```
function confirm()
{
  var order = new Object();
  order.Number = 12345;
  order.ItemCount = 5;
  var message = 'Delete order number ' + order.Number + '?';
   radconfirm(message, confirmCallBack, 200, 100, order, 'Delete Order');
}
```

The callback function has two parameters. The first resolves to a Boolean that is true if the user clicked the "yes" button. The second, optional parameter, contains the object passed in the radconfirm call. In this example we use another alert to display the contents of the object and report on what the user did.

```
function confirmCallBack(clickedYes, userObject)
{
  var returnValue = clickedYes ? "deleted." : "not deleted.";
  var message = 'Order ' + userObject.Number +
    ' with ' + userObject.ItemCount + ' items ' + returnValue;
   radalert(message);
}
```



radconfirm() called and responded to

Prompt

The mechanics of radprompt() are almost identical to confirm except that the callback gets an argument containing something the user entered instead of a true/false. Here is another short example triggered by a button:

```
<button onclick="prompt();">Prompt</button>
```

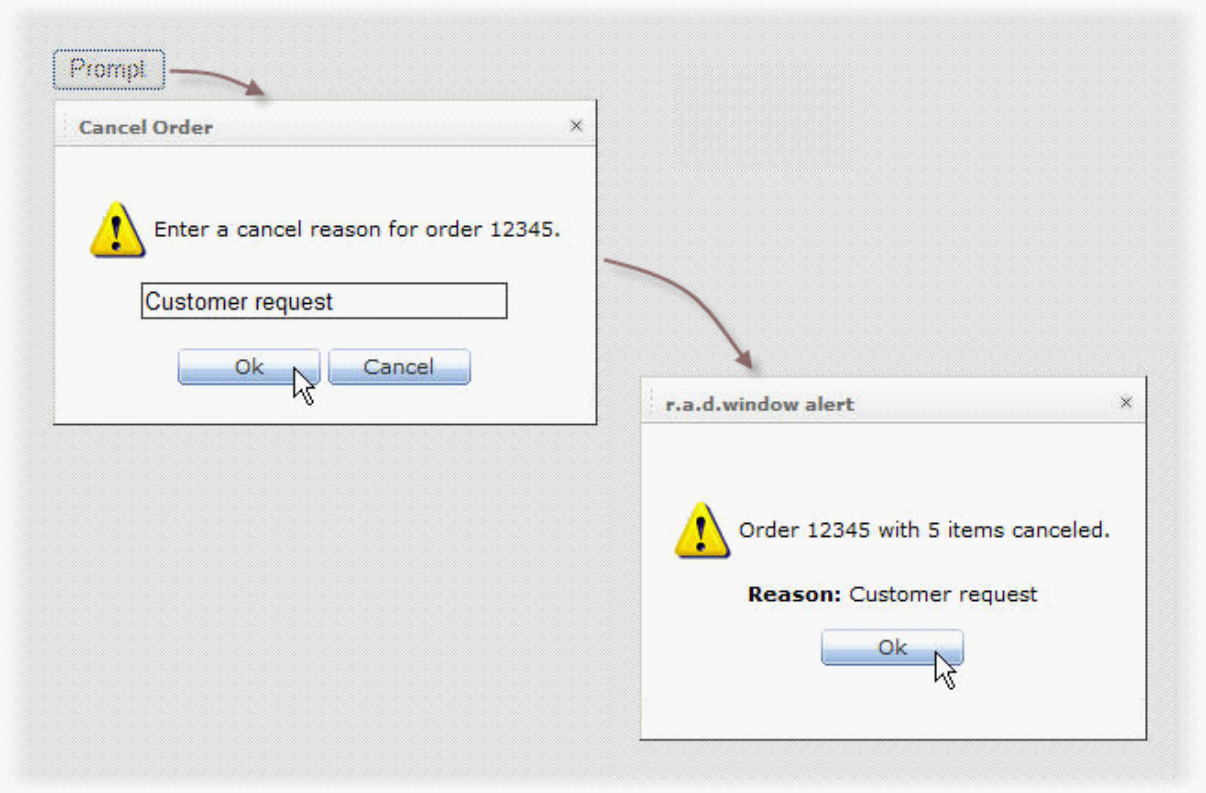
The only functional difference from the confirm example is the call to radprompt():

```
function prompt()
{
  var order = new Object();
  order.Number = 12345;
  order.ItemCount = 5;
  var message = 'Enter a cancel reason for order ' + order.Number + '.';
   radprompt(message, promptCallBack, 300, 100, order, 'Cancel Order');
}
```

The real change is in the first argument to the callback. Here we get whatever value was entered by the user. This function gets called *both when the user clicks Ok or Cancel!*. The difference is that the userInput argument will be blank if the user clicks cancel.

```
function promptCallBack(userInput, obj)
{
  var message = 'Order ' + obj.Number +
    ' with ' + obj.ItemCount +
    ' items canceled.<br /><br /><b>Reason:</b> ' + userInput;
   radalert(message);
```

```
}
```



radprompt() called and responded to

Creating Windows Server Side

Not surprisingly you can create RadWindow instances in code behind and add them to the RadWindowManager Windows collection. Here is a short example that creates a window that is visible immediately.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   RadWindow myWindow = new RadWindow();
    myWindow.VisibleOnPageLoad = true;
    myWindow.NavigateUrl = "http://www.google.com";
      RadWindowManager1.Windows.Add(myWindow);
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 Dim myWindow As RadWindow = New RadWindow
  myWindow.VisibleOnPageLoad = True
  myWindow.NavigateUrl = "http://www.google.com"
    RadWindowManager1.Windows.Add(myWindow)
End Sub
```

Minimize Zones

Popups in most applications really can't be treated like regular windows. They may or may not allow resizing, and certainly not minimizing or maximizing. The RadWindow not only can be minimized, it can minimize to an area functionally equivalent to a task bar. This area is called a minimize zone. The minimize zone can be any HTML element such as a div or Panel control. You can tailor the element to look like a task bar, or to run vertically along the left side or any other configuration that suits your purpose.

Point the *MinimizeZoneId* property of the window at the HTML element you want to use as the area that will hold a placeholder icon for minimized windows. You also need to set the window *MinimizeMode* property to "MinimizeZone". You can also set the *MinimizeIconUrl* to replace the stock image.

Note: The look of the minimized "icon" can be changed in CoreTemplates.xml. You may want the graphic wider to allow more text for example. Look for the <minimizetemplate> tag.

The figure below shows an example of a panel used as a minimize zone and several windows

Minimize zone in use

Here's an HTML layout with a panel in the bottom portion. Note the panel ID is "pnlMinimize".

```
<body>
   <radW:RadWindowManager ID="RadWindowManager1" runat="server"></radW:RadWindowManager>
  <table border="0" cellpadding="0" cellspacing="0" style="width: 100%; height: 100%">
    <tr>
      <td colspan="2" style="height: 200px"></td>
    </tr>
    <tr>
      <td style="width: 200px"></td>
      <td></td>
    </tr>
    <tr>
      <td colspan="2" style="height: 50px">
        <asp:Panel ID="pnlMinimize" runat="server" BackColor="LightGray" BorderStyle="Inset"
          BorderWidth="1px" Height="100%" Width="100%">
        </asp:Panel>
      </td>
    </tr>
  </table>
</body>
```

Now in the code behind for a form we can create a few windows a one time. For each window we set the MinimizeMode to "MinimizeZone" and MinimizeZoneId to "pnlMinimize".

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   RadWindowManager1.Skin = "Mac";
  for (int i = 0; i < 4; i++)
  {
     RadWindow window = new RadWindow();
     window.NavigateUrl = "WebForm1.aspx";
     window.Title = "Window" + i.ToString();
     window.Top = 10;
     window.Left = 10 + (i * 180);
     window.Width = 180;
     window.Height = 100;
      window.VisibleOnPageLoad = true;
      window.VisibleStatusbar = false;
       window.MinimizeMode = RadWindowMinimizeModes.MinimizeZone;
      window.MinimizeZoneId = "pnlMinimize";

       RadWindowManager1.Windows.Add(window);
  }
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 RadWindowManager1.Skin = "Mac"
 Dim i As Integer = 0
 While i < 4
   Dim window As RadWindow = New RadWindow
    window.NavigateUrl = "WebForm1.aspx"
    window.Title = "Window" + i.ToString
    window.Top = 10
    window.Left = 10 + (i * 180)
    window.Width = 180
    window.Height = 100
     window.VisibleOnPageLoad = True
     window.VisibleStatusbar = False
      window.MinimizeMode = RadWindowMinimizeModes.MinimizeZone
     window.MinimizeZoneId = "pnlMinimize"
      RadWindowManager1.Windows.Add(window)
       System.Math.Min(System.Threading.Interlocked.Increment(i),i-1)
 End While
End Sub
```

Lab: Creating a Vertical Task Bar

This lab will be similar to the last example except the "task bar" area will be located on the left side. The lab will show how to designate the minimize zone and make changes to the minimize template appearance.

1. Create a new web application "MinimizeZones".
2. Create a new RadControls directory in your project. Copy the "Window" directory from the Telerik RadControls directory to the RadControls directory in your project.
3. Create a second web form "WebForm1.aspx" (the first being default.aspx). Enter some text on the

page to stand as content, e.g. "My content goes here".

4. The rest of the work is back on default.aspx. Drop a RadWindowManager onto the default page.

5. In source for the default page add the following HTML markup after the RadWindowManager tag:

```
<table border="0" cellpadding="0" cellspacing="0" style="width: 100%; height: 100%">
  <tr>
    <td colspan="2"></td>
  </tr>
  <tr>
    <td style="height: 300px; width: 100px"></td>
    <td></td>
  </tr>
  <tr>
    <td colspan="2"></td>
  </tr>
</table>
```

6. Add a Panel control from the Standard section of the Toolbox.

   - The html table has a top, bottom, left and right sections. Add the panel to the left section (second row, first cell).
   - Set the *ID* property to "pnlMinimize".
   - *Set BackColor* to "LightGray", *BorderStyle to "*Inset*", BorderWidth* to "1px". Both *Width* and *Height* properties should be "100%".

   Now the table markup should look like:

```
<table border="0" cellpadding="0" cellspacing="0" style="width: 100%; height: 100%">
  <tr>
    <td colspan="2"></td>
  </tr>
  <tr>
    <td style="height: 300px; width: 100px">
      <asp:Panel ID="pnlMinimize" runat="server" BackColor="LightGray"
        BorderStyle="Inset" BorderWidth="1px" Height="100%" Width="100%">
      </asp:Panel>
    </td>
    <td></td>
  </tr>
  <tr>
    <td colspan="2"></td>
  </tr>
</table>
```

7. Create a series of windows that will all use the minimize zone. In the code behind for the page load event handler create a Page_Load event handler and add the sample code shown below. The code for Width, Height, Top and Left is intended to cascade the windows and make them available for the purposes of demonstrating the minimizing zone. Note the Skin property is set in code to be "Mac".

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  const int WinHeight = 100;
  const int WinWidth = 180;
```

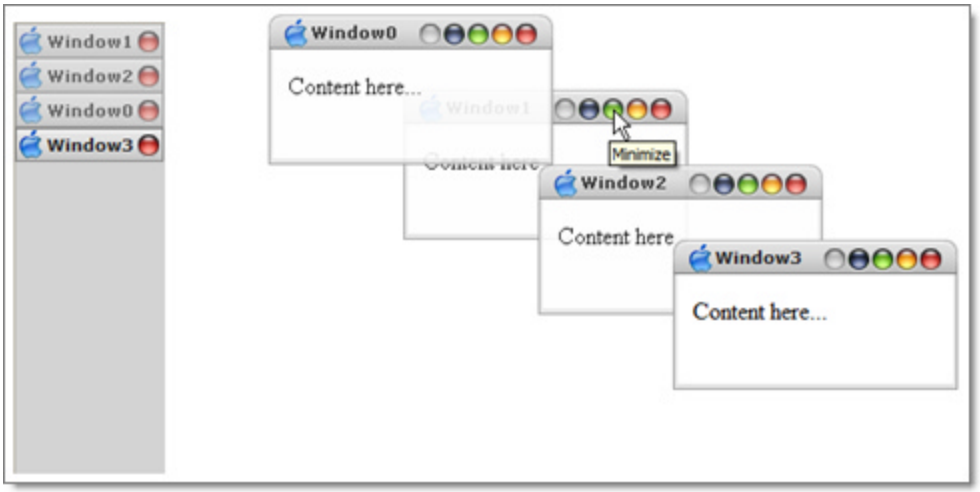
```
   RadWindowManager1.Skin = "Mac";
  for (int i = 0; i < 4; i++)
  {
     RadWindow window = new  RadWindow();
     window.NavigateUrl = "WebForm1.aspx";
     window.Title = "Window" + i.ToString();
     window.Width = WinWidth + 10;
     window.Height = WinHeight;
     window.Top = 10 + (i * (WinHeight / 2));
     window.Left = WinWidth + (i * (WinWidth / 2));
     window.VisibleOnPageLoad = true;
     window.VisibleStatusbar = false;
      window.MinimizeMode = RadWindowMinimizeModes.MinimizeZone;
     window.MinimizeZoneId = "pnlMinimize";

      RadWindowManager1.Windows.Add(window);
  }
}
```

VB Example:

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
 Const WinHeight As Integer = 100
 Const WinWidth As Integer = 180
 RadWindowManager1.Skin = "Mac"
 Dim i As Integer = 0
 While i < 4
   Dim window As RadWindow = New RadWindow
    window.NavigateUrl = "WebForm1.aspx"
    window.Title = "Window" + i.ToString
    window.Width = WinWidth + 10
    window.Height = WinHeight
    window.Top = 10 + (i * (WinHeight / 2))
    window.Left = WinWidth + (i * (WinWidth / 2))
    window.VisibleOnPageLoad = True
    window.VisibleStatusbar = False
     window.MinimizeMode = RadWindowMinimizeModes.MinimizeZone
    window.MinimizeZoneId = "pnlMinimize"
     RadWindowManager1.Windows.Add(window)
      System.Math.Min(System.Threading.Interlocked.Increment(i),i-1)
 End While
End Sub
```

8. In the RadControls\Window\Skins directory find the "Mac" skin directory and open it. In the Mac directory locate "CoreTemplates.xml and open. In CoreTemplates.xml find the string in the minimizetemplate tag "padding:1px;width:40px" and change the width to be 60px. If left at 40px the title of your window in the minimized "icon" would be truncated.

9. Run the application. Notice how the minimized icons highlight when working with a given window. Try minimizing, clicking the minimized icon and closing windows.



The running application

3.4.4 Client Scripting with RadWindow

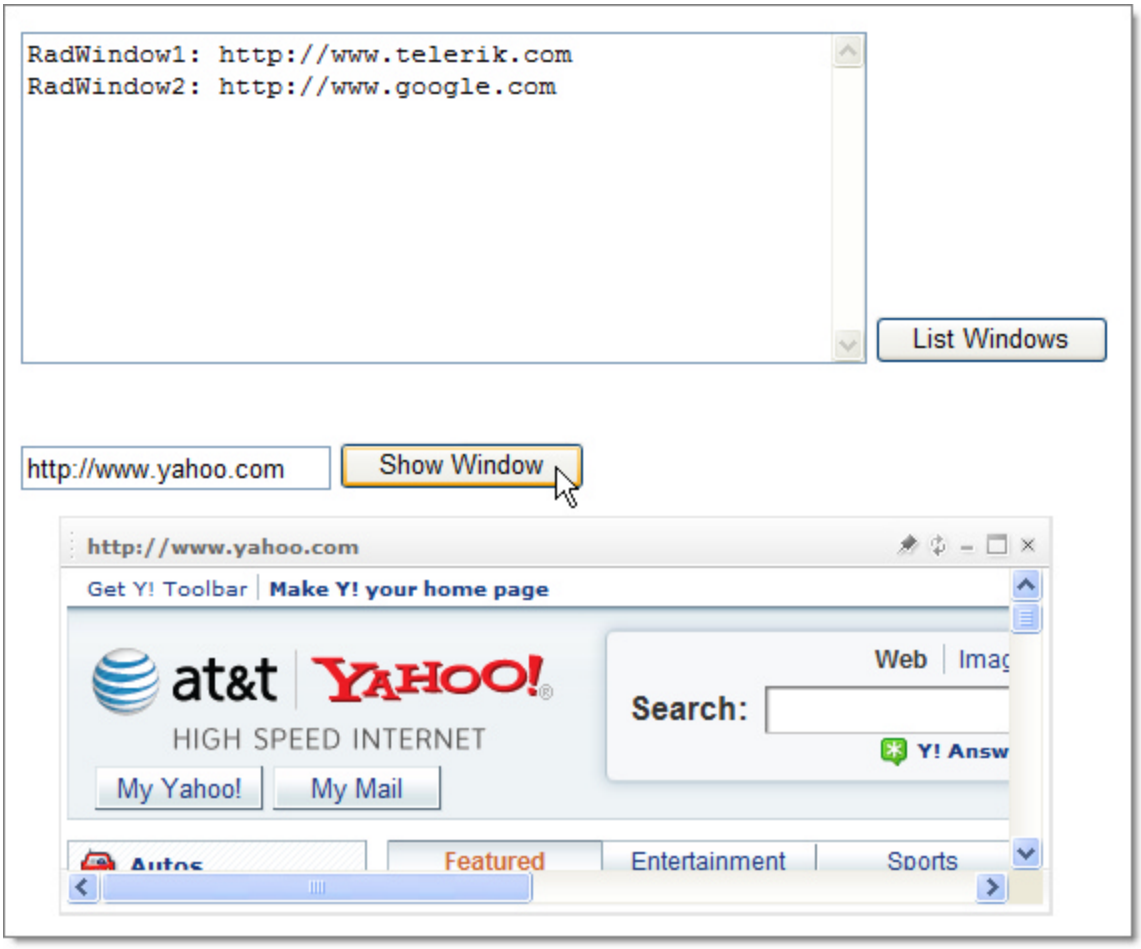
RadWindow

RadWindow has a voluminous number of properties and methods. Here are groupings of some properties of interest (check the online help for the complete listing):

- One set of properties indicates the current state of the window. These are: IsModal, IsClosed, IsPinned, IsMaximized, IsVisible, IsActive.
- Another set of methods causes changes in window behavior or state: MoveTo(left, top), Center(), Show(), Hide(), Minimize(), Maximize(), Restore(), Close().
- Look for Setxxx() methods to modify window properties like SetTitle(), SetUrl(), SetOpenerElementId() for example.
- BrowserWindow and GetContentFrame() allow you to get the information out of the window to somewhere useful. Use BrowserWindow to call functions back on the parent page document. GetContentFrame() gets a reference to the window's content area (below the title bar and above the status bar) IFrame. Use the content frame to call methods on the window page from the parent page or modify the window page DOM.

Here is an example that gets a reference to a particular window, changes the title and url for the window and displays the window:

```
function handleShowWindowClick()
{
   var manager = GetRadWindowManager();
   var txtShowWindow = document.getElementById("txtShowWindow");
   var window1 = manager.GetWindowByName("RadWindow1");
    window1.SetUrl(txtShowWindow.value);
   window1.SetTitle(txtShowWindow.value);
   window1.Show();
}
```



RadWindowManager and RadWindow examples running in the browser

Client Methods

The RadWindowManager has a few handy methods for dealing with all windows as a group:

Property	Description
GetRadWindowManager	Returns a reference to the window manager.
GetWindows	Returns a reference to the array of rad window objects managed by the RadWindowManager.
GetWindowByName	Returns a reference to a rad window. The Name is the window's server side *Id* property.
GetActiveWindow	Returns a reference to the current active window.
Open	Opens a new or existing window. Can be used to change the url of existing window as well.

Property	Description
Tile	Tiles the window objects.
Cascade	Cascades the window objects.
CloseAll	Closes all windows.
MinimizeAll	Minimizes all windows.
MinimizeActiveWindow	Minimizes the current active windows.
RestoreActiveWindow	Restores the active window (if it was minimized or maximized).

The basic pattern for using these methods is to call GetRadWindowManager(), then use the window manager reference to call any of the other methods. For example, this next code snippet is triggered from a html button client onclick event. It gets the window manager reference, calls GetWindows() to return an array of all windows, then cycles through them printing the window name and Url to a text area:

```
<script>
   function handleClick()
  {
     var manager = GetRadWindowManager();
     var windows = manager.GetWindows();
     var TextArea1 = document.getElementById("TextArea1");
    for(var i = 0;i<windows.length;i++)
    {
       TextArea1.value += windows[i].Name + ": " + windows[i].Url + "\n";
    }
  }
</script>
```

Client Events

Client events are fired for RadWindowManager every time a window is shown, loaded, closed or when a callback is defined for the window. Each event handler gets a reference to the window involved. The callback function also gets the return value for the window.

```
<radW:RadWindowManager  ID="RadWindowManager1" runat="server"
   ClientCallBackFunction="callBack"
   OnClientClose="clientClose"
   OnClientPageLoad="clientPageLoad"
   OnClientShow="clientShow"
  >
```

```
<script>
   function clientPageLoad(window)
```

```
  {
    LogIt("OnClientPageLoad:" + window.Name);
  }
   function clientShow(window)
  {
    LogIt("OnClientShow:" + window.Name);
  }
   function clientClose(window)
  {
    LogIt("OnClientClose:" + window.Name);
  }
   function callBack(window, returnValue)
  {
    LogIt("OnClientClose:" + window.Name + " value: " + returnValue);
  }

   function LogIt(text)
  {
     var TextArea1 = document.getElementById("TextArea1");
    TextArea1.value += text + "\n";
  }
</script>
```

The window also has a few events for onminimize, onmaximize, and onrestore events. Use AttachClientEvent to set these up. For example:

```
function Load()
{
   var rwEntry = GetRadWindow();

    rwEntry.AttachClientEvent("onminimize", min);
    rwEntry.AttachClientEvent("onmaximize", max);
    rwEntry.AttachClientEvent("onrestore", restore);
}

function min(window)
{
   alert("onminimize: " + window.Name);
}

function max(window)
{
   alert("onmaximize: " + window.Name);
}

function restore(window)
{
   alert("onrestore: " + window.Name);
}
```

Passing data to and from a window

A common scenario is moving information from parent window to popup window, get user edits, then update the parent window. The first hurdle is getting information to the popup window if it needs initialization. There are at least two ways to handle this.

- On the OnClientPageLoad event handler use the GetContentFrame() function to access code in the popup page. Notice the syntax below is in the form of myWindow.GetContentFrame(). contentWindow.<my popup page defined function>. The OnClientPageLoad handler itself is

defined in the parent page and "SetName()" is the function defined on the popup page.

```
function   clientPageLoad(window)
{
  if (window.name = "rwEntry")
  {
      var contentWindow = window.GetContentFrame().contentWindow;
      contentWindow.SetName('Bob', 'Smith');
  }
}
```

- The second way is to populate the "argument" property of the window. You can code this in JavaScript of the parent page. The example below creates a new JavaScript object, provides properties for first and last name, then assigns the object to the RadWindow "argument":

```
function handleEntryClick()
{
   var manager = GetRadWindowManager();
   var rwEntry = manager.GetWindowByName("rwEntry");
   var userObj = new Object();
   userObj.First = "Mr";
   userObj.Last = "Telerik";
   rwEntry.argument = userObj;
   rwEntry.Show();
}
```

To access the RadWindow in the popup itself you'll need to arm yourself with a few bits of code. A standard chunk you should have available gets the RadWindow for the popup we're in at that time:

```
function  GetRadWindow()
{
  var oWindow = null;
  if  (window.radWindow)
    oWindow  =  window.radWindow;
  else  if  (window.frameElement.radWindow)
    oWindow  =  window.frameElement.radWindow;
  return  oWindow;
}
```

This example defines an onload() event handler for the body of the popup and uses your GetRadWindow () function to retrieve the RadWindow argument. Once you have the object back you manipulate the current DOM as appropriate:

```
function  Load()
{
   var currentWindow = GetRadWindow();
   userObj = currentWindow.argument;
    document.getElementById('tbFirst').value  =  userObj.First;
    document.getElementById('tbLast').value = userObj.Last;
}
```

To get user changed values *back* to the parent window, use the RadWindow CallBack() function and pass a return value. In this example the user may have changed the first or last name inputs so we scrape those values back into a JavaScript object and return the result. This example is triggered from the onclick of a button on the popup page:

```
function  Done()
```

```
{
   var currentWindow = GetRadWindow();
  var userObj = new Object();
   userObj.First = document.getElementById('tbFirst').value;
   userObj.Last = document.getElementById('tbLast').value;
    currentWindow.CallBack(userObj);
   currentWindow.Close();
}
```

Back on the parent page we have the ClientCallBackFunction defined and simply retrieve the values there:

```
function callBack(window, returnValue)
{
  LogIt("ClientCallBackFunction: " + window.Name);
   ListProperties(returnValue);
}
```

Keyboard support

You can supply keyboard shortcuts to any client side RadWindowManager public method that doesn't require parameters. Hooking them up actually takes place in the server side code behind. Use the RadWindowManager Shortcuts collection and add new WindowShortcut objects, passing the name of the command and the keyboard combination in the constructor. For example:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    RadWindowManager1.Shortcuts.Add(new WindowShortcut("Tile", "ALT+T"));
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
   RadWindowManager1.Shortcuts.Add(New WindowShortcut("Tile", "ALT+T"))
End Sub
```

Note: the WindowShortcut object is in the namespace Telerik.WebControls.

Default shortcuts are already built in:

- FocusNextWindow: CTRL+TAB
- MinimizeAll: CTRL+F2
- EscapeActiveWindow: ESC

Use the Shortcuts.Clear() method if you don't want the defaults.

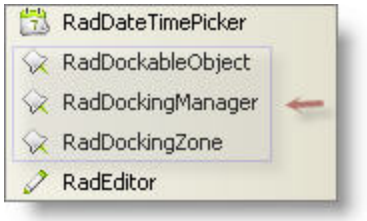
3.4.5 Summary

This section explored some of the many capabilities of RadWindowManager and RadWindow objects from basic opening, positioning and behavior to transferring data to and from parent and popup windows. We looked at various flavors of built-in windows like the splash screen as well as alert, confirm and prompt dialogs. We briefly explored how to manipulate templates, both directly in CoreTemplates.xml and in code. The client section overviewed the methods and properties available to RadWindowManager and RadWindow.

3.5 RadDock

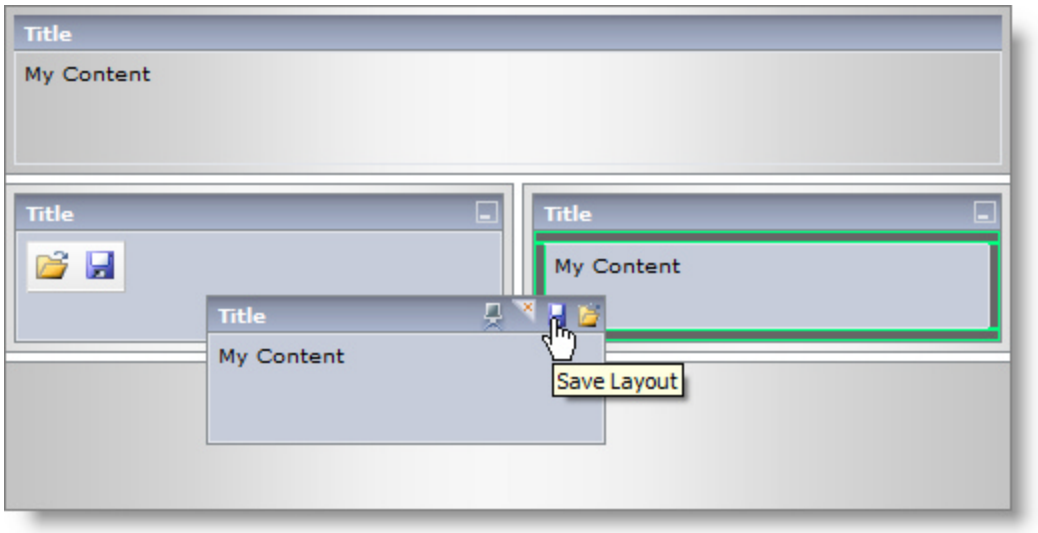
3.5.1 Getting Started

RadDock lets you drag objects and groups of objects on a web page without writing a line of JavaScript. The suite has a RadDockingManager that controls docking for the entire page, a RadDockingZone that defines places where items can be docked, and RadDockableObject that acts as a container for items you want to drag within the page.



RadDock controls in the toolbox

Docking applications can be visually as simple as a few areas on the screen used for docking and a group of components in a dockable object container. You can use docking to create personalized "portal" style pages that remember the user layout, dockable toolbars, shopping carts where the items can actually be dragged to the cart, "post-it" notes and just about anything that requires dynamic custom, user-driven layout at runtime.



An skinned example with a custom command, floating toolbar and styled grips

The RadDock feature set is loaded. Here is a sampling:

- Modes: Dockable objects can be various flavors of disabled, docked and floating.
- Behaviors: Dockable objects can be resized, closed, collapsed and pinned to a location on the page.
- Commands: Use pre-defined commands to expand/collapse, pin/unpin and close the dockable object container. You can also define your own custom commands and display them with an image in the title bar of the dockable object.

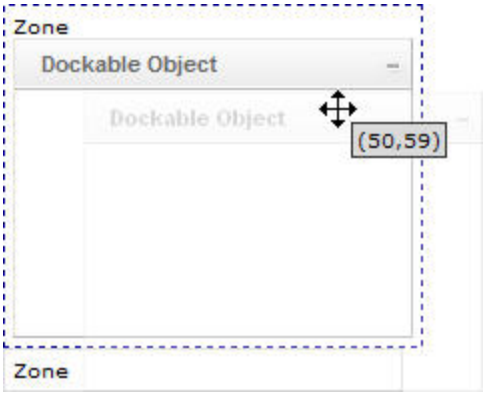
- Page Layout: Dynamic page layout provided by RadDock can be edited by the user or set to allow view-only. The entire layout can be persisted using the built-in save/load state functionality. The state can be saved to any medium, e.g. database, ViewState, text file.
- Visual appearance: Dockable object and zone appearance can be customized with skins and styles. Movement aspects of dockable objects are animated with developer control of duration and refresh rate.
- Docking zone control: You have multiple levels of control over where objects may be dragged

The basic steps to start using RadDock:

1. Add a RadDockingManager to your page.
2. Add one or more RadDockingZones and place them where you want them on the page.
3. Add one or more RadDockableObjects
4. Edit the template for the RadDockableObject. This can be done easily in the designer by dragging controls from the toolbox.

Lab: A minimal docking example

1. Create a new web application "GettingStarted"
2. From the toolbox drop a RadDockingManager on the default page.
3. Drop a RadDockingZone on the page.
4. Drop a second RadDockingZone on the page.
5. Drop a RadDockableObject on the page. Set the *Text* property to "RadDockableObject". The text will show up in the title bar of the dockable object.. **Note**: Without the *Text* property defined it may be difficult finding the "grips" for the docking object to be dragged. Without text the effect is that you are unable to drag using the title bar, even though the title bar is clearly visible.
6. Run the application. You should be able to drag the dockable object to the two zones and have them dock.



The running application

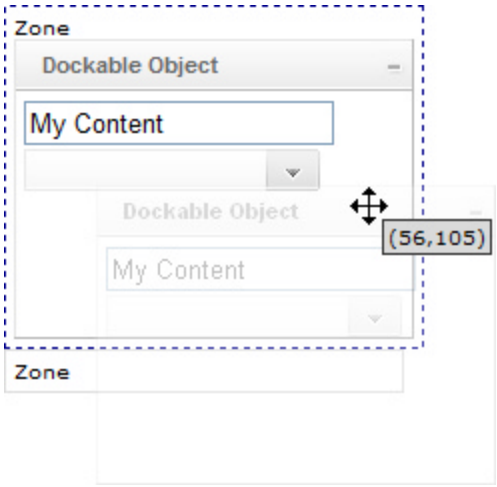
7. Stop the application.

8. Click the Smart Tag "Edit Templates" option.

9. Drag a standard text box to the template and set the *Text* property to "My Content"

10. Drag a RadComboBox to the template area.

11. Re-run the application.



Application running with content

3.5.2 Using RadDock in the Designer

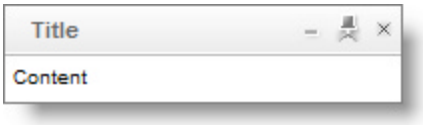
Dockable Objects

The RadDockableObject is a container for controls you want to move independently on the page. Each dockable object can be customized separately. Some of the key aspects of the dockable object that can be controlled both at design time and programmatically:

- **Docking modes**: Controls what circumstances the object can be docked. The *DockingMode* property can be Disabled (acts as a simple control placeholder), AlwaysDock (always in a docking zone), NeverDock (always floating) or Dockable.

```
RadDockableObject1.DockingMode   =   RadDockingModeFlags.AlwaysDock;
```

- **Behaviors**: Each dockable object has built-in capabilities that allow it to collapse/expand, close, resize and be pinned to a location on the page. The *Behavior* property can be one or more flags: Resize, Collapse, Close, Pin, Resizable:



Behaviors collapse, pin and close

In the designer set the flags separated by commas:

```
<cc1:RadDockableObject . . .Behavior="Collapse, Close, Pin">. . .
```

In code use the bitwise "or" operator to fuse the flags together:

```
RadDockableObject1.Behavior = RadDockableObjectBehaviorFlags.Collapse |
    RadDockableObjectBehaviorFlags.Close |
     RadDockableObjectBehaviorFlags.Pin;
```

Note: The Resizable flag is a combination of Collapse and Resize.

Note: The Close flag allows the user to hide the dockable object. Call the dockable object client function Show() to make it visible again.

- **Grips**: Grips are the areas on the dockable object that the user clicks on to drag the object. You can customize the location and appearance of the grips for each dockable object. RadDockableObject has grips located on the title bar, top, bottom, left and right. Use the *FloatingObjectEnabledGrips* and *DockedObjectEnabledGrips* to mandate which grips are enabled. In the designer set the flags separated by commas:

```
<cc1:RadDockableObject . . . DockedObjectEnabledGrips="Top, Bottom">. . .
```

In code use the bitwise "or" operator to combine flags:

```
RadDockableObject1.FloatingObjectEnabledGrips =
    RadDockableObjectGripFlags.Top | RadDockableObjectGripFlags.Bottom;
```

The default appearance of the grips looks like the figure below when Top and Bottom flags are specified.

Top and Bottom grips enabled

Note: Use *HorizontalGripStyle* and *VeriticalGripStyle* to tailor the exact appearance of the grip area.

Lab: Configuring Dockable Objects

This lab explores the capabilities of RadDockableObject. In the lab you will learn how to customize the docking behavior and appearance of the dockable object.

1. Create a new application "DockableObject".

2. Between the form tags, add the following HTML table to the default page. The cells define the basic

layout of top, bottom, left and right areas.

```
<table  width="500px">
  <tr>
    <td colspan="2"  style="width:  100%;  height:  75px"></td>
  </tr>
  <tr>
    <td style="width:  50%;  height:  75px"></td>
    <td style="width:  50%;  height:  75px"></td>
  </tr>
  <tr>
    <td colspan="2"  style="width:  100%;  height:  75px"></td>
  </tr>
</table>
```

3. In the designer add a RadDockingZone to the top table cell. Set the *ID* property to "zoneTop". Set the Height and Width to "100%" and set the BackColor property to "lightblue".

4. Copy zoneTop to the left, right and bottom table cells. Name the zones "zoneLeft", "zoneRight" and "zoneBottom" respectively.

5. Add a RadDockableObject to zoneTop.

   - Select the Smart Tag "Edit Templates" option. The only available template to edit here is the "Content Template".
   - In the template area add the literal text "My Content".
   - From the Smart Tag select "End Template Editing".
   - Set the *Text* property to "Title".
   - Set the *ID* property to "rdoTop".
   - Set the *Width* property to "100%" and *Height* to "75px".

6. Copy rdoTop to the left, right and bottom zones. Set the *ID* properties to "rdoLeft", "rdoRight" and "rdoBottom" respectively.

7. Run the application with just the settings we have so far. Experiment briefly with dockable object behavior. Can you drag the object out of a zone? Can you collapse the object? Can you resize the object?

8. Stop the application.

9. Set the *DockingMode* property of rdoTop to "Disabled". Also set the *ShowTitleBarWhenDisabled* property to "True" so we it will be easier to see the effect.

10. Set the *DockingMode* property of rdoLeft to "AlwaysDock".

11. Set the *DockingMode* property of rdoRight to "Dockable".

12. Set the *DockingMode* property of rdoBottom to "NeverDock". Move rdoBottom to just outside the end of the table tag. Set the *Width* property to "75px". **Note**: If we were to leave the dockable object within a zone with this setting and try to run the application, we would see an error "This object must be placed outside any docking zone."

13. Re-run the application and observe the dragging behavior changes. Try to move rdoLeft and rdoRight outside of any docking and note the differences. Try moving rdoBottom, the "floating" object, inside of a docking area.

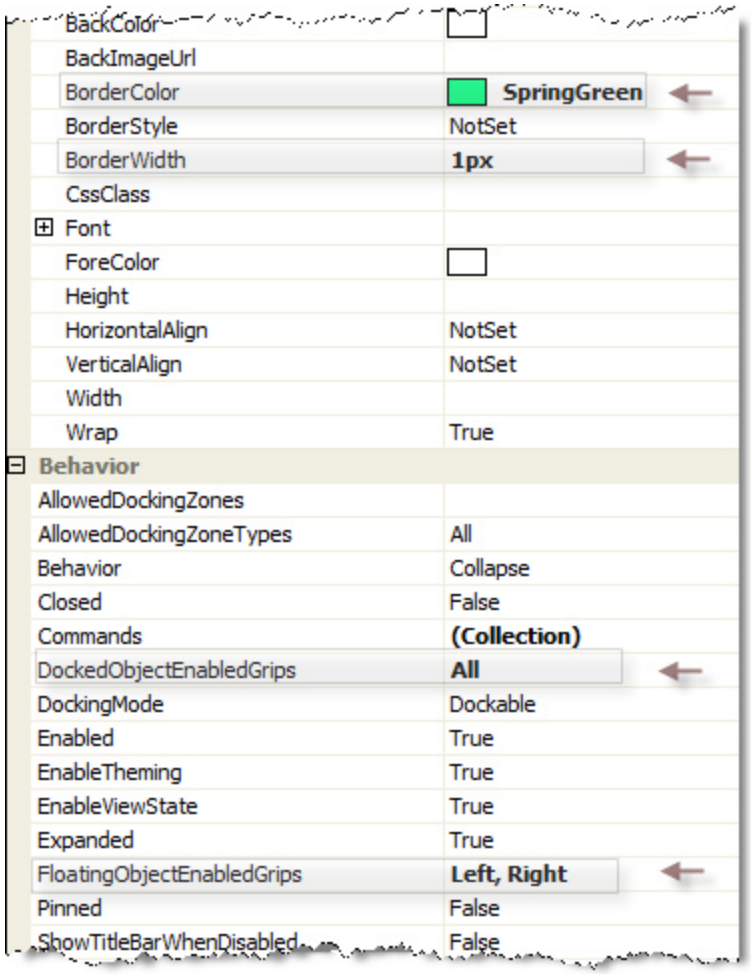
14. Stop the application.

15. Set the *Behavior* property of rdoBottom to "Resize,Pin,Close".

16. Set grip related properties on rdoRight. Note that rdoRight DockingMode property is set to "Dockable", i.e. it can both dock and "float" on the page. Here we set which grips are available when floating or docked and what the grips look like.

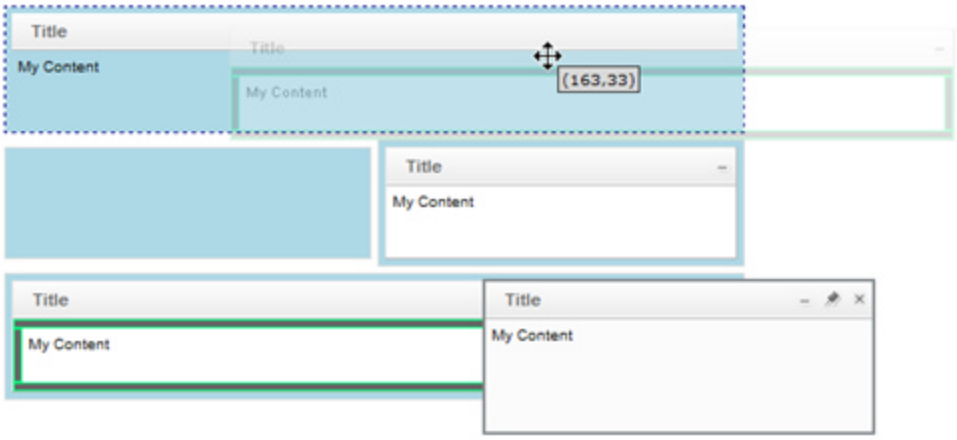
    - In the Properties Window click the sort by "Categorized" button to see the properties grouped.
    - Set the *DockedObjectEnabledGrips* property to "All". This will allow gripping on the title bar, or top/bottom/left/right grips.
    - Set the *FloatingObjectEnabledGrips* property to "Left,Right". This will enable just the left and right grips when the object is not docked.

17. In the Appearance property category of the Properties Window find the *VerticalGripStyle* property and click the plus sign (+) to view the sub properties. Set the *BorderColor* to "SpringGreen" and the *BorderWidth* to "1px". Find the *HorizontalGripStyle* property and set the *BorderColor* to "SpringGreen" and the *BorderWidth* to "1px".



Setting grip properties

18. Run the application. You should be able to resize rdoBottom by selecting one of its edges with the mouse. Try dragging the object from the title bar, then click the pin button to fix it in place. rdoRight should allow gripping the title bar or any edge while its docked. After you drag it outside of a dockable zone then only the left and right grips should be visible.



Testing the application

Lab: Controlling where objects may be dragged

This lab will show you how to control where dockable objects can be dragged to.

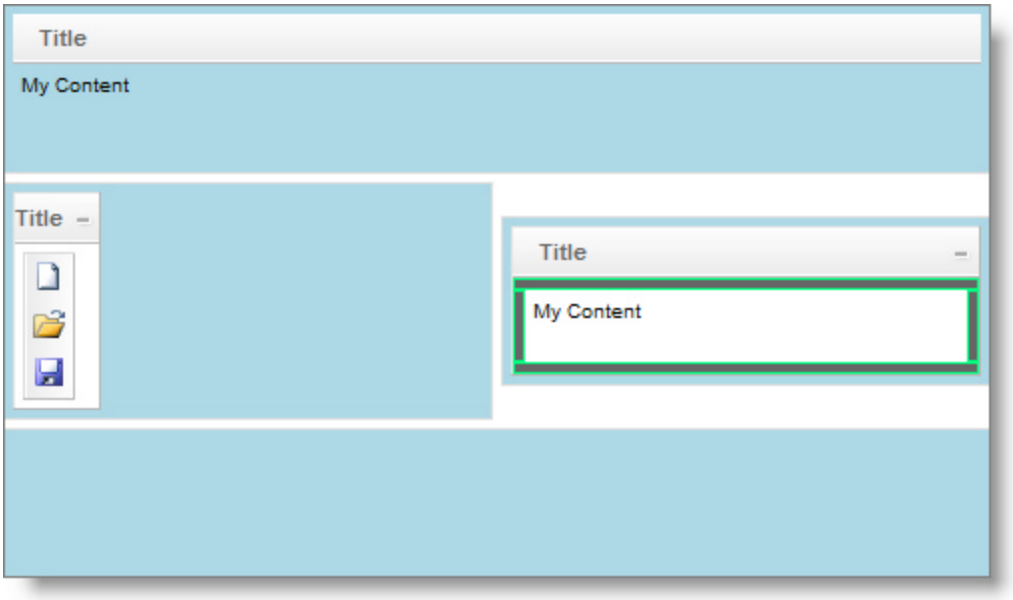
RadDockingZone has a *Type* property that indicates a logic area and can be Top, Bottom, Left, Right, Vertical, Horizontal or Custom. We will talk about the use for Custom shortly.

RadDockableObject has two properties, *AllowDockingZoneTypes* and *AllowDockingZones*. *AllowDockingZoneTypes* matches with the *Type* property of the RadDockingZone. If a zone type is "Left" for example, then a objects with *AllowDockingZoneTypes* of "Left" or "Vertical" can dock there. What if you want an object to only dock at left and bottom zones? Here we use a RadDockingZone type of "Custom". On the docking object *AllowDockingZones* takes a list of zone ID's.

Be aware that these areas, i.e. top, bottom, etc, are *logical* areas only, even though they sound like they might have something to do with layout. That said, if you place a tool bar in a dockable object, the toolbar will be arranged vertically if placed in a logical vertical zone i.e. of *Type* Vertical, Left or Right.

Note: As of this writing the actual layout has to be done with HTML tables. Future versions are likely to be CSS style driven.

1. Use the previous project "DockableObject" as a starting point.
2. Set the *Type* property for zoneLeft, zoneRight, zoneTop and zoneBottom to "Left", "Right", "Top" and "Bottom" respectively.
3. In the rdoLeft object set the *AllowedDockingZoneTypes* property to "Custom" and the *AllowedDockingZones* property to "zoneLeft,zoneBottom".
4. Use the Smart Tag to begin editing the content template. Replace the literal text "My Content" with a RadToolBar from the Tool box. In the *Items* property add three RadToolbarButton objects to the collection and set the *CommandName* properties to "New", "Open" and "Save" respectively. Set the *Skin* property to "Default". Close the template editor.
5. Create a RadControls directory in the project and copy the Telerik RadControls "Toolbar" directory there.
6. Run the application and notice that you can drag only to the left and bottom zones and nowhere else. Also notice the layout changes in the toolbar.



The running application with vertical toolbar

3.5.3 Using RadDock at Runtime

Lab: Persisting Page Layout

The RadDockingManager has a method SaveState() that retrieves a string containing the current layout and settings of the docking controls. A second method LoadState() restores the docking layout using the string returned by SaveState().

1. Use the project from the previous lab "DockableObject" as a starting point.
2. Use the Smart Tag to edit the rdoLeft object content template. In the *Items* property of the toolbar remove the "New" command button. Set the RadToolbar *AutoPostback* property to "True'. Create an event handler for the RadToolBar OnClick event.
3. In the code first create a helper property to persist the state information:

```
C# Example:
private string DockingState
{
  get
  {
    return ViewState["DockingState"] == null ?
      "" : ViewState["DockingState"].ToString();
  }
  set
  {
     ViewState["DockingState"] = value;
  }
}

VB Example:
Private Property DockingState() As String
 Get
   Return  Microsoft.VisualBasic.IIf(ViewState("DockingState")  Is Nothing,_
      "",ViewState("DockingState").ToString)
 End Get
```

```
 Set
    ViewState("DockingState") = value
 End Set
End Property
```

4. Add the code listed below to the toolbar OnClick event handler. Here we're simply calling SaveState if the user presses the "Save" button and storing it in the ViewState. If the user presses the "Load" button we check if there is any state information to restore and call LoadState if there is.

```
C# Example:
protected void RadToolbar1_OnClick(object sender,
   RadToolbarClickEventArgs e)
{
  if (e.Button.CommandName == "Save")
  {
    this.DockingState = RadDockingManager1.SaveState();
  }
  else
  {
    if (this.DockingState.Length > 0)
    {
        RadDockingManager1.LoadState(this.DockingState);
    }
  }
}

VB Example:
Protected Sub RadToolbar1_OnClick(ByVal sender As Object, _
  ByVal e As RadToolbarClickEventArgs)
 If e.Button.CommandName = "Save" Then
    Me.DockingState = RadDockingManager1.SaveState
 Else
   If Me.DockingState.Length > 0 Then
       RadDockingManager1.LoadState(Me.DockingState)
   End If
 End If
End Sub
```

5. Run the application. Note the location and settings of the windows and click the Save button. Change the arrangement and settings, then click the Load button. The previous layout will be restored.

Lab: Commands

This lab will demonstrate adding custom commands to the RadDockableObject title bar. The lab also uses the skinning features of RadDock.

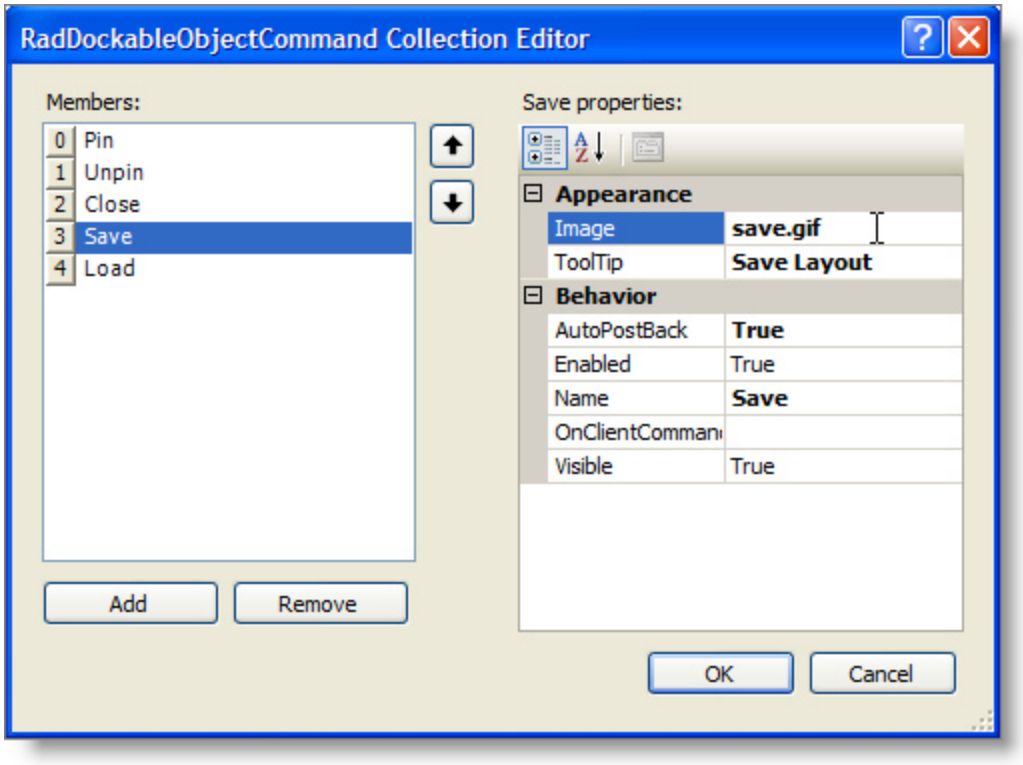
Dockable objects come with their own predefined commands for resize, pin/unpin, expand/collapse and close. As a convenience these are defined by the *Behavior* property. For greater control of the built in commands and to define your own custom commands use the Commands collection.

1. Use the project from the previous lab "DockableObject" as a starting point.
2. Create a new RadControls directory in your project. Copy the "Dock" directory from the Telerik RadControls directory to the RadControls directory in your project.
3. Copy the files "save.gif" and "open.gif" to the directory holding the images for the "Inox" skin. These are 15 x 15 pixel icons that will fit in the limited space of the title bar.

4. Set the *Skin* property of the RadDockingManager to "Inox".

5. Edit the commands for the floating dockable object rdoBottom.

- Add a new RadDockableObjectCommand element to the Commands collection. Set the *Name* property to "Save", *AutoPostBack* to "True", *ToolTip* to "Save Layout" and *Image* to "save.gif".
- Add a second command. Set the *Name* property to "Load", *AutoPostBack* to "True", *ToolTip* to "Restore Layout" and *Image* to "open.gif".



Setting custom command properties

6. At the time of this writing there is no "Command" event property. So just create one in the code editor from the Page_Load.

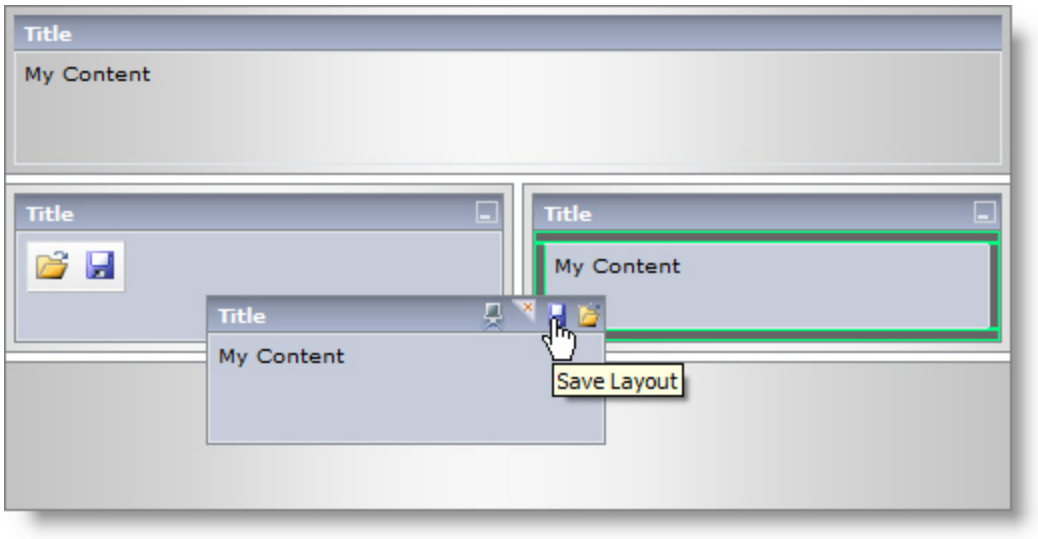
```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   rdoBottom.Command += new   EventHandler(rdoBottom_Command);
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  rdoBottom.Command += New   EventHandler(rdoBottom_Command)
End Sub
```

7. The handler will be very similar to the toolbar OnClick event handler. Here there is one additional step to take. You first must cast the "e" parameter from being a generic EventArgs type to a RadDockableObjectCommandEventArgs. That provides access to the Command object and the Name property of the command.

```
C# Example:
void  rdoBottom_Command(object sender, EventArgs e)
{
```

```
    RadDockableObjectCommandEventArgs args = (RadDockableObjectCommandEventArgs)e;
  if (args.Command.Name == "Save")
  {
    this.DockingState = RadDockingManager1.SaveState();
  }
  else
  {
    if (this.DockingState.Length > 0)
    {
        RadDockingManager1.LoadState(this.DockingState);
    }
  }
}

VB Example:
Sub rdoBottom_Command(ByVal sender As Object, ByVal e As EventArgs)
 Dim args As RadDockableObjectCommandEventArgs = _
   CType(e, RadDockableObjectCommandEventArgs)
 If args.Command.Name = "Save" Then
    Me.DockingState = RadDockingManager1.SaveState
 Else
   If Me.DockingState.Length > 0 Then
       RadDockingManager1.LoadState(Me.DockingState)
   End If
 End If
End Sub
```

8. Run the application. You should be able to save the docking state on the page using the Save button located on the floating dockable object btnBottom.



Using the custom command button

3.5.4 Client Scripting with RadDock

Client Events

Client events for RadDock are fired by way of the RadDockableObject control from one of the following properties:

OnClientDock: A dockable object is docked in a docking zone.

OnClientDragStart: A drag action is started.

OnClientDragEnd: The drag ends, before the OnClientDrop.

OnClientDrop: The dockable object is dropped, just after OnClientDragEnd.

OnClientUnDock: A dockable object is undocked from a zone.

OnClientDockStateChange: A dockable object is being docked or undocked.

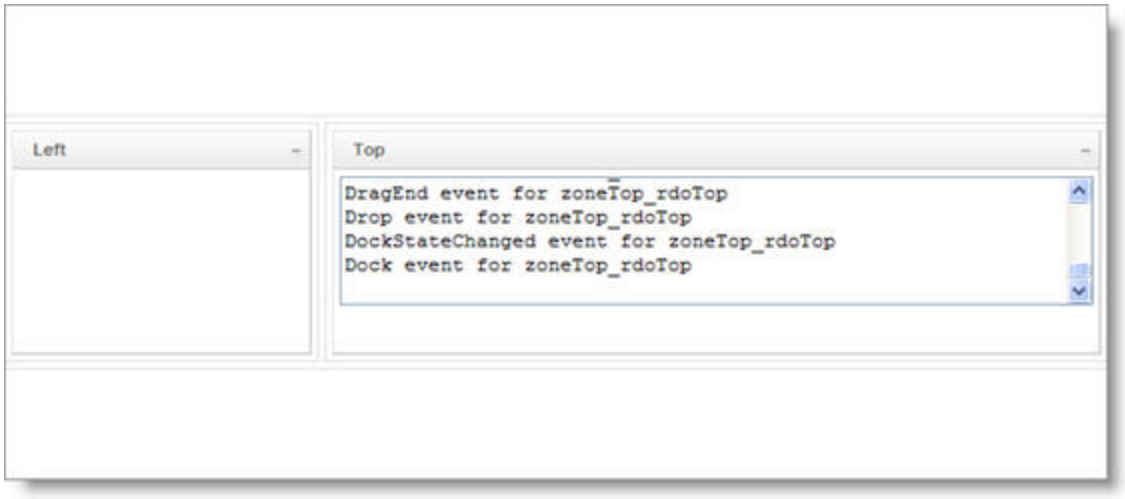
Each event takes two arguments:

- The dockable object where you can access the objects id for use in other client API calls.
- An "eventArgs" parameter that has an "eventName" property.

Here is an example that logs events to a text area housed in a dockable object. The ClientEvent() function below is hooked up to every client event on two RadDockableObjects.

```
<script type="text/javascript">
  //<!--
   function ClientEvent (dockObj, eventArgs)
  {
       Log(eventArgs.eventName + " event for " + dockObj.id);
  }

   function Log(message)
  {
     var taEvents = document.getElementById("taEvents");

     taEvents.value += message + "\n";
      taEvents.scrollTop = taEvents.scrollHeight;
  }

  //-->
</script>
```

The figure below shows the output and gives you some idea of the event ordering on the client.



Logging RadDock client events

Client API

The client API parallels much of the server functionality including access to the RadDockingManager, zones and dockable objects. Some methods and properties of interest follow for each. See the Telerik online help for a complete list of methods, properties and flags.

RadDockManager

To get a reference to the RadDockingManager use the client ID:

```
manager = <%= RadDockingManager1.ClientID %>;
```

The RadDockingManager has collections of all zones and dockable objects in the application. You can iterate each zone and object as in the following example:

```
function GetInfo(dockObj, command)
{
     manager = <%= RadDockingManager1.ClientID %>;
    Log("Zones:");
     var zones = manager.DockingZones;
    for(index in zones)
    {
       Log(zones[index].id);
    }

    Log("Dockable Objects:");
     var objects = manager.DockableObjects;
    for(index in objects)
    {
       Log(objects[index].id);
    }
}
```

Once you have a reference to a zone or dockable object you can use any of the client API methods available.

RadDockingZone

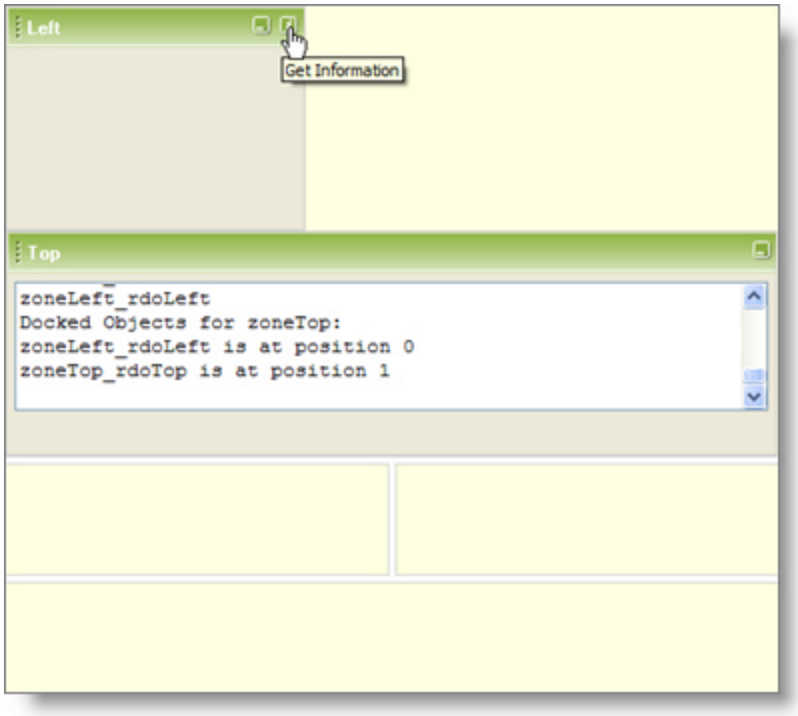
As with the RadDockingManager, use the ClientID to get a reference to a zone object:

```
var topZone = <%= zoneTop.ClientID %>;
```

Each zone also has a method to get its docked objects and their ordinal positions within the zone.

```
Log("Docked Objects for zoneTop:");
var topZone = <%= zoneTop.ClientID %>;
var topObjects = topZone.GetDockedObjects();

for(index in topObjects)
{
   var topObject = topObjects[index];
   var position = topZone.GetPosition(topObject);
   Log(topObject.id + " is at position " + position);
}
```

The output for that would look like this:

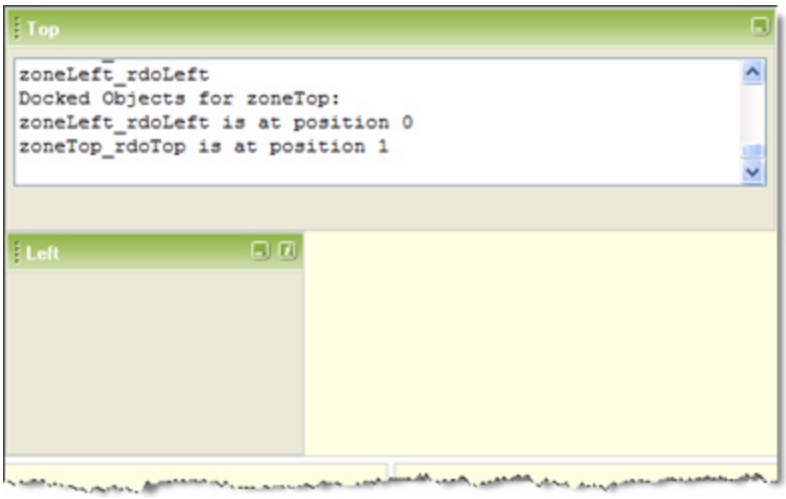


Zone client methods output

Use SetAtPosition() to rearrange the objects. For example the output shown in the figure above has rdoLeft at position 0 and rdoTop at position 1. This code snippet reverses their positions:

```
var rdoTop = <%= rdoTop.ClientID %>;
topZone.SetAtPosition(rdoTop, 0);
```

...and now would look like the figure below.



Dockable objects repositioned in the zone

RadDockableObject

Once you're down in the dockable object level of the client API the field really opens up for numerous properties and methods. You can get a reference to a particular dockable object the same as with the

RadDockingManager and zones, using the ClientID:

```
var rdoTop = <%= rdoTop.ClientID %>;
```

A series of boolean properties and methods give you an idea of the characteristics of the object:

- IsDockableObject
- IsDocked()
- IsVisible()

Another series of method pairs handle the visibility, pinning and expansion:

- Show()/Hide()
- Pin()/Unpin()
- Expand()/Collapse(). Expand actually takes a boolean parameter that if true expands, otherwise collapses.

All the grip objects are accessible through properties TopGrip, BottomGrip, LeftGrip and RightGrip. Find and change their visibility using methods IsGripVisible(grip) and SetGripVisible(grip, visible).

Lab: Confirming a dockable object close

1. Create a new web application "ClientAPI".
2. In the default web page add the following under the page tag to register the controls. Normally this will happen automatically, but we will be working directly in the HTML for the immediate future.

```
<%@ Register Assembly="RadDock.Net2" Namespace="Telerik.WebControls" TagPrefix="cc1" %>
```

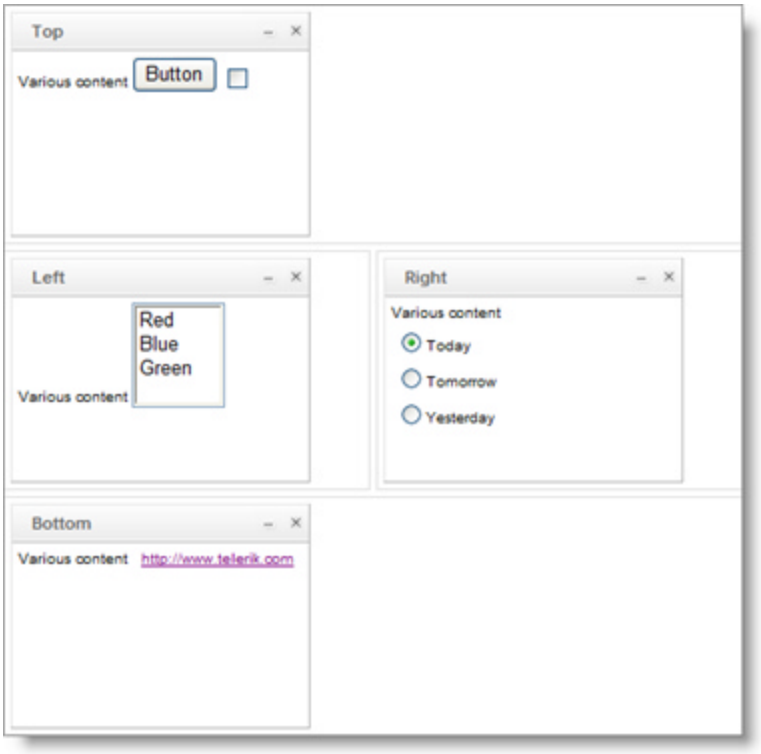
3. Between the form tags add the html markup snippet below. This sets up the basic table layout, adds the RadDockingManager, docking zones and a dockable object in each zone. Each dockable object is populated with one or more controls. Also notice that each dockable object has a "Close" command. Its the Close command that we will be working with.

```
<cc1:RadDockingManager ID="RadDockingManager1" runat="server" />
<table width="500px">
  <tr>
    <td colspan="2" style="width: 100%; height: 75px">
      <cc1:RadDockingZone ID="zoneTop" runat="server"
       Width="100%" Height="100%">
        <cc1:RadDockableObject ID="rdoTop" runat="server" Text="Top">
          <ContentTemplate>
            Various content
            <asp:Button ID="Button1" runat="server" Text="Button" />
            <asp:CheckBox ID="CheckBox1" runat="server" />
          </ContentTemplate>
          <Commands>
             <cc1:RadDockableObjectCommand Enabled="False" Name="Expand"
              ToolTip="Expand" />
             <cc1:RadDockableObjectCommand Name="Collapse"
              ToolTip="Collapse" />
             <cc1:RadDockableObjectCommand Name="Close"
              ToolTip="Close" />
          </Commands>
        </cc1:RadDockableObject>
```

```
      </cc1:RadDockingZone>
    </td>
  </tr>
  <tr>
    <td style="width: 50%; height: 75px">
      <cc1:RadDockingZone ID="zoneLeft" runat="server"
       Width="100%" Height="100%">
        <cc1:RadDockableObject ID="rdoLeft" runat="server" Text="Left">
          <ContentTemplate>
            Various content
            <asp:ListBox ID="ListBox1" runat="server">
               <asp:ListItem>Red</asp:ListItem>
               <asp:ListItem>Blue</asp:ListItem>
               <asp:ListItem>Green</asp:ListItem>
            </asp:ListBox>
          </ContentTemplate>
          <Commands>
             <cc1:RadDockableObjectCommand Enabled="False" Name="Expand"
              ToolTip="Expand" />
             <cc1:RadDockableObjectCommand Name="Collapse"
              ToolTip="Collapse" />
             <cc1:RadDockableObjectCommand Name="Close"
              ToolTip="Close" />
          </Commands>
        </cc1:RadDockableObject>
      </cc1:RadDockingZone>
    </td>
    <td style="width: 50%; height: 75px">
      <cc1:RadDockingZone ID="zoneRight" runat="server"
       Width="100%" Height="100%">
        <cc1:RadDockableObject ID="rdoRight" runat="server" Text="Right">
          <ContentTemplate>
            Various content  
            <asp:RadioButtonList ID="RadioButtonList1" runat="server">
              <asp:ListItem Selected="True">Today</asp:ListItem>
               <asp:ListItem>Tomorrow</asp:ListItem>
               <asp:ListItem>Yesterday</asp:ListItem>
            </asp:RadioButtonList>
          </ContentTemplate>
          <Commands>
             <cc1:RadDockableObjectCommand Enabled="False" Name="Expand"
              ToolTip="Expand" />
             <cc1:RadDockableObjectCommand Name="Collapse"
              ToolTip="Collapse" />
             <cc1:RadDockableObjectCommand Name="Close"
              ToolTip="Close" />
          </Commands>
        </cc1:RadDockableObject>
      </cc1:RadDockingZone>
    </td>
  </tr>
  <tr>
    <td colspan="2" style="width: 100%; height: 75px">
      <cc1:RadDockingZone ID="zoneBottom" runat="server"
       Width="100%" Height="100%">
        <cc1:RadDockableObject ID="rdoBottom" runat="server" Text="Bottom">
          <ContentTemplate>
            Various content  
            <asp:HyperLink ID="HyperLink1" runat="server"
              NavigateUrl="http://www.telerik.com">http://www.telerik.com
```

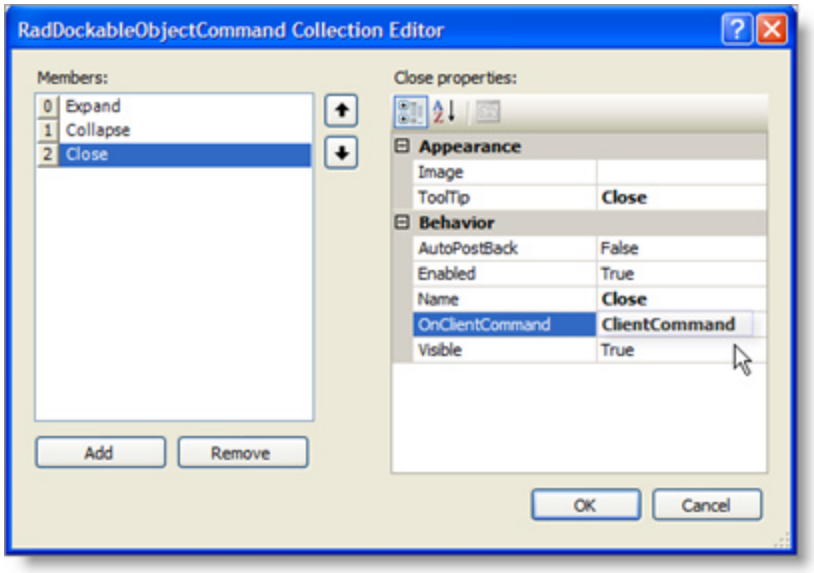
```
                </asp:HyperLink>
            </ContentTemplate>
            <Commands>
                <cc1:RadDockableObjectCommand Enabled="False" Name="Expand"
                 ToolTip="Expand" />
                <cc1:RadDockableObjectCommand Name="Collapse"
                 ToolTip="Collapse" />
                <cc1:RadDockableObjectCommand Name="Close"
                 ToolTip="Close" />
            </Commands>
          </cc1:RadDockableObject>
        </cc1:RadDockingZone>
    </td>
  </tr>
</table>
```

The project so far if you were to run it would look like the figure below.



Running application appearance so far

4. In the designer use the Property Window to access the Commands collection for each of the dockable objects on the form. Set the *OnClientCommand* property to be "ClientCommand".



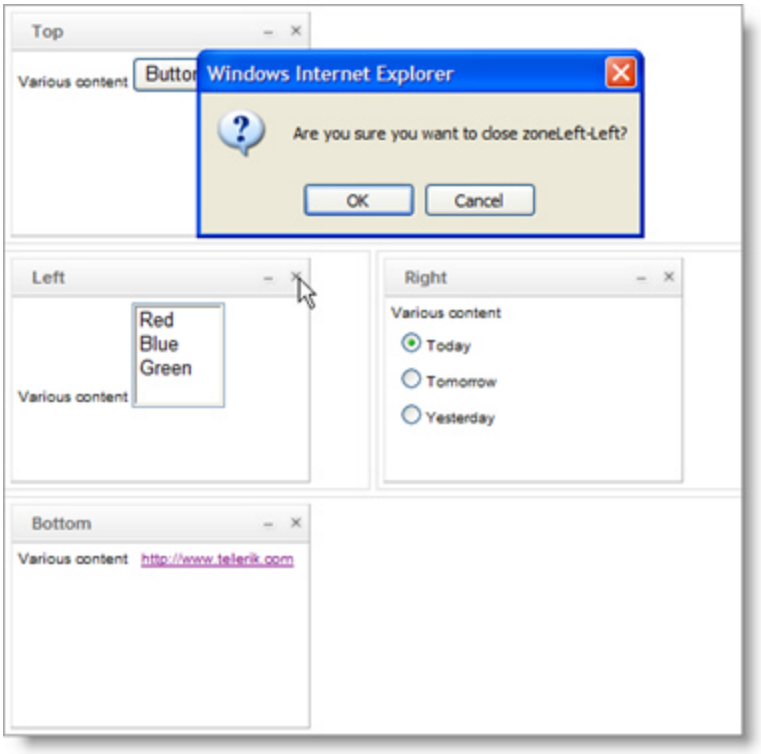
Setting the OnClientCommand property

5. Just inside the form tag place the following JavaScript to respond to the close command:

```
<script type="text/javascript">
//<!--
function ClientCommand (dockObj, command)
{
  if (command.Name == "Close")
  {
     var zone = dockObj.ParentDockingZone;
    if (zone != null)
    {
         var fullName = zone.id + "-" + dockObj.TitleBar.innerText;
        if (confirm("Are you sure you want to close " + fullName + "?"))
        {
           dockObj.Hide();
        }
    }
  }
}
//-->
</script>
```

6. This client event handler checks that the command object Name property is "Close". This is really unnecessary here because we only have the one command hooked up to the handler, but its included for completeness. Then the handler demonstrates that we can get the parent zone from the dockable object using the ParentDockingZone property. We then access the dockable objects TitleBar property and retrieve the innerText. If the user confirms the information and clicks OK the dockable object is hidden.

7. Run the application and test both OK and Cancel functionality.



The running application

3.5.5 Summary

This chapter started with basic functional docking scenarios developed in the designer which included working with content templates to populate dockable objects. We also discussed the relationship between the dock manager, docking zones and the dockable objects within those zones. We focused particularly on dockable objects and how to control their appearance and behavior.

The runtime labs demonstrated custom commands and persisting the users page layout. Along the way the example was given a Skin and had various CSS styles applied.

The client section covered how to get references to RadDockingManager, zones and dockable objects, how to iterate various collections and some of the functions and properties of interest for individual objects within those collections.

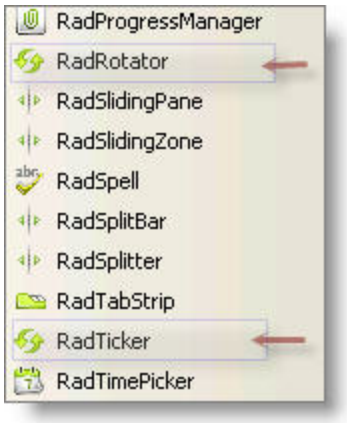
3.6 RadRotator

3.6.1 Getting Started

The Telerik RadRotator for ASP.Net (which also includes the RadRotator and RadTicker controls) is designed to keep a website looking fresh in a number of ways. Whether you want to supply real-time information as it changes, or merely vary the content of what would otherwise be a static page, the rotator control gives you the ability to easily organize and present dynamic content, all without the need for using Flash or other plugins.

In this section of the manual, we will discuss the various ways you can bind the control to a data source, and the use of templates to maintain a consistent and visually appealing content stream. We'll cover transition effects and how to include ASP.Net controls in your rotator. We'll also show how to next nest tickers and ticklines.

Now that you know where we're headed, let's rotate!



RadRotator controls in the toolbox

3.6.2 Using RadRotator

In this section, we will use the Developer Studio interface to add a RadRotator control to a form, and explore several different methods of binding it to various data sources, including a live XML data source. We'll then build a little more complicated template, with a graphic, a label, and an embedded ticker. We'll explore how to control the rotator display using controls on the user interface, and how to respond to client-side events. We'll finish by adding a pair of rotator controls to a form, with a brief look at how to implement the ITemplate interface specifically for RadRotator.

1. Create a new ASP.NET project named "Rotation".

2. Create a RadControls directory within the project.

3. Copy the \Rotator directory to your project's RadControls directory. (This step makes the javascripts available which are used at runtime by both the rotator and ticker controls). The \Rotator directory is located under the directory where you installed your rad controls. In a typical installation, this will be

something like "*C:\Program Files\telerik\r.a.d.controlsQ4 2006\NET2\RadControls*".

RadTicker control

The RadTicker control provides a means of displaying text messages in a screen area as if they were being typed out onto the screen by the little guy inside the computer.

Using the basic ticker control is pretty straightforward, though there are a couple of tricks which you can see displayed in the code below - specifically, if you want the ticker to start displaying its messages without any interaction from the user, you need to set the AutoStart property to true. Also, you have a choice of whether or not to enable looping. If looping is not enabled, the ticker will stop after the last item is displayed, with the last item displayed on the screen (which you can avoid, of course, by adding an empty string as the last element in your display array). Alternatively, you can enable looping as we have done below, and the message content will be displayed repeatedly until terminated through some other user or programmatic action. **Note: the default for both of these settings is "false".** If you're not seeing any display from your ticker, this is probably why.

Lab: Displaying data with the ticker control

1. Drop a RadTicker control onto the design surface of your "Rotation" project's default.aspx page.

2. Double-click on the page. Add the following code to the page load event:

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  string[] myTickerLines =  new string[4];
   myTickerLines[0] = "A stitch in time saves nine";
   myTickerLines[1] = "A rolling stone gathers no moss";
   myTickerLines[2] = "The early bird catches the worm";
   myTickerLines[3] = "A penny saved is virtually worthless";
    RadTicker1.UserTickerLines = myTickerLines;
   RadTicker1.AutoStart = true;
   RadTicker1.Loop = true;
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
     Handles MyBase.Load
  Dim  myTickerLines(4) As String
   myTickerLines(0) = "A stitch in time saves nine"
   myTickerLines(1) = "A rolling stone gathers no moss"
   myTickerLines(2) = "The early bird catches the worm"
   myTickerLines(3) = "A penny saved is virtually worthless"
    RadTicker1.UserTickerLines = myTickerLines
   RadTicker1.AutoStart = True
   RadTicker1.Loop = True
End Sub
```

RadRotator control

Lab: Binding RadRotator to a Data Source

A major consideration when using many controls is determining where you're going to get the data, and that's certainly true for a control like the rotator where the entire point of the control is to keep the content constantly changing. Telerik'srotator control has a very flexible approach to databinding, permitting you to bind to four major types of data sources. Let's take a look at each.

Binding to an IList data source

1. Start by opening VS2005, and create a new file-system based website.

2. Drop a RadRotator control onto the default.aspx page.

3. Double-click the default.aspx page to access the Page_Load method in the code behind.

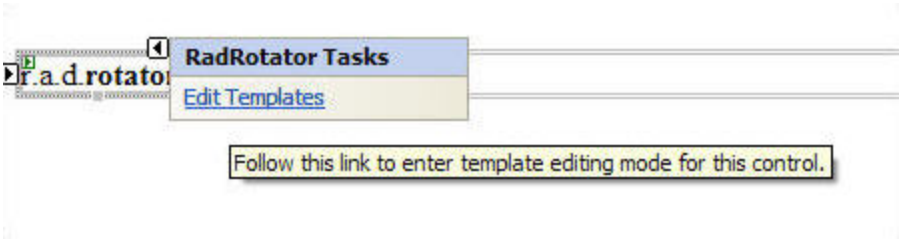
The IListSource interface includes objects like Arrays, DataSets, DataTables, and DataViews. You will notice that some objects (like Arrays) implement more than one interface. Generally speaking, any object that looks like it's a container for a collection of other objects is going to have at least one method available to do databinding with.

4. Start by adding the following to your Page_Load event. Feel free to substitute your own content, since we'll be discussing principles here, not content.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
   DataTable table = new DataTable();
    table.Columns.Add("proverb");

    table.Rows.Add("A stitch in time saves nine" );
    table.Rows.Add("A rolling stone gathers no moss" );
    table.Rows.Add("The early bird catches the worm" );
    table.Rows.Add("A penny saved is virtually worthless" );

    RadRotator1.DataSource = table;
     RadRotator1.DataBind();
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
     Handles MyBase.Load
  Dim table As New DataTable()
    table.Columns.Add("proverb")

   table.Rows.Add("A stitch in time saves nine")
   table.Rows.Add("A rolling stone gathers no moss")
   table.Rows.Add("The early bird catches the worm")
   table.Rows.Add("A penny saved is virtually worthless")

   RadRotator1.DataSource = table
    RadRotator1.DataBind()
End Sub
```

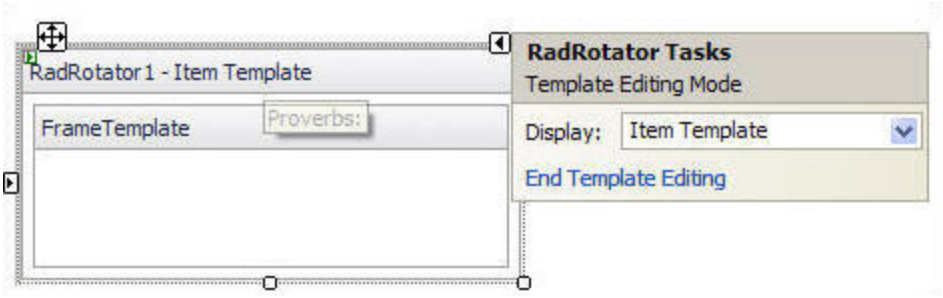
Creating the display template

5. Now that you've established and hooked up the data, you still have a couple more things to do before you can see it displayed. The most important of these is to create the template for displaying the data. Continuing with the example above, switch over to your default.aspx page. Open up the smart tag on the rotator control, and select "Edit Template"



Opening the RadRotator smart tag

The template editor will open up, defaulting to the ItemTemplate.

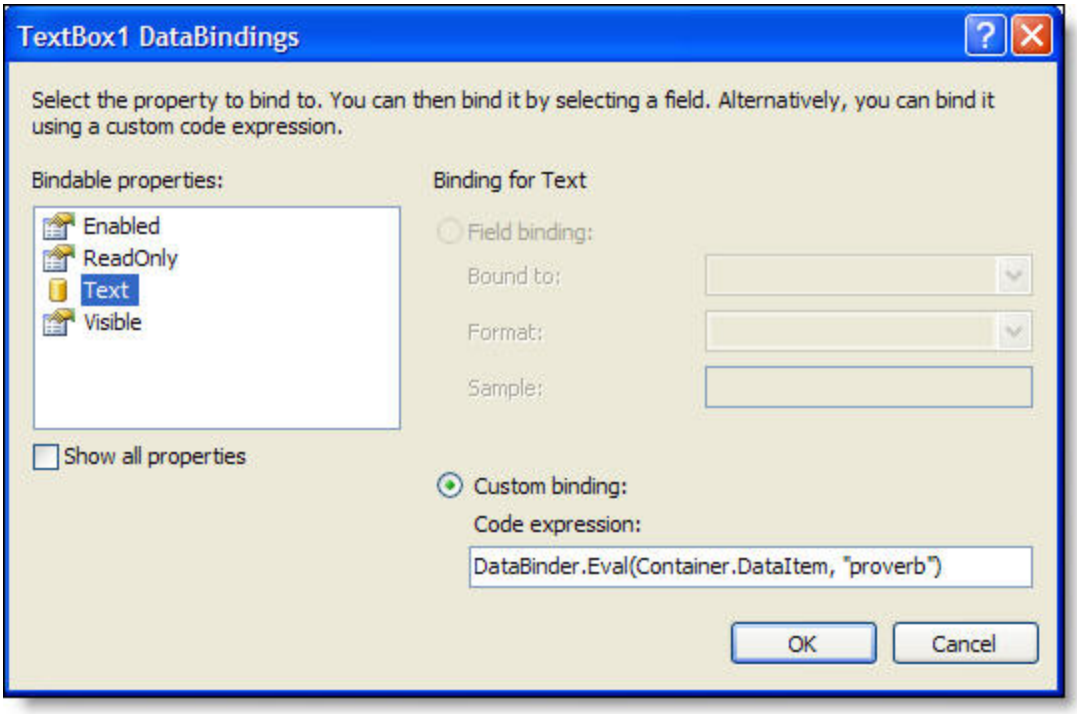


The RadRotator template editor

Drop an ASP.Net textbox into the frame template area, then open up the textbox's smart tag.

Accessing Data Bindings within the template editor

When you select "Edit Data Bindings", you will be taken to a further wizard which will help you select the data you'd like to bind to. For now, let's simply bind the text property of the textbox to the "proverb" column in the table.



Attaching a DataBinder

After you click the "OK" button, open up the smart tag on the rotator control once again, and select "End Template Editing".

6. If you run the rotator project at this point, the textbox control is displayed at a default width. Even if you change the width property of the rotator, it doesn't help - some of the text in the example above will still get truncated. You need to pay attention to the width of BOTH the textbox control and the rotator control, and finding the correct balance to produce an acceptable visual appearance can be a matter of trial, error and patience. Here's one example of some markup that works:

```
<radR:RadRotator  ID="RadRotator1"  runat="server"  Width="50%">
   <FrameTemplate>
      <asp:TextBox  ID="TextBox1"  runat="server"   width="98%"
        Text='<%#  DataBinder.Eval(Container.DataItem,  "proverb")  %>'>
       </asp:TextBox>
   </FrameTemplate>
</radR:RadRotator>
```

Now if you run the project, you will get a textbox with complete (not partial) outlining, where the message changes every few seconds.

Using IEnumerable

Binding to an IEnumerable object is nearly identical to using an IListSource. The primary difference is that the enumerable object can be a simple type which is returned directly to the rotator. Let's implement a simple enumerable collection

7. Add the following to your Page_Load event. Remember to include the System.Collections

namespace.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
  ArrayList al = new ArrayList();

  al.Add("A stitch in time saves nine" );
  al.Add("A rolling stone gathers no moss" );
  al.Add("The early bird catches the worm" );
  al.Add("A penny saved is virtually worthless" );

  RadRotator1.DataSource = al;
   RadRotator1.DataBind();
}

VB Example:
protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
  Dim al As New ArrayList

  al.Add("A stitch in time saves nine")
  al.Add("A rolling stone gathers no moss")
  al.Add("The early bird catches the worm")
  al.Add("A penny saved is virtually worthless")

  RadRotator1.DataSource = al
   RadRotator1.DataBind()
End Sub
```

8. You'll also need to modify your page markup very slightly. One possible way to present the data looks like this:

```
<radR:RadRotator ID="RadRotator1" runat="server"
    Title="Proverbs to Live By" Width="50%">
  <FrameTemplate>
    <div>
      <%# Container.DataItem %>
    </div>
  </FrameTemplate>
</radR:RadRotator>
```

Notice that we no longer need to use the "DataBinder.Eval" method.

Reading data from a static XML file

In addition to databinding, the rotator control has the ability to take its data from an XML content file.

9. In your project, create a new XML file named "Proverbs.xml". If you're creating your own data independent of the courseware, be sure to substitute your own designations as appropriate.

```
Proverbs.XML:
<?xml version="1.0" encoding="utf-8" ?>
<Proverbs>
  <Proverb>
    <Saying>A stitch in time saves nine</Saying>
  </Proverb>
  <Proverb>
    <Saying>A rolling stone gathers no moss</Saying>
  </Proverb>
  <Proverb>
```

```
    <Saying>The early bird catches the work</Saying>
  </Proverb>
  <Proverb>
    <Saying>A penny saved is virtually worthless</Saying>
  </Proverb>
</Proverbs>
```

10. Because there is no databinding *per se*, you can eliminate anything in the Page_Load routine (or the entire routine, for that matter).

11. You still need to hook up the data. In the case of a static XML file, you'll hook the data up via the ContentFile property of the rotator. A completed implementation might look like this:

```
<radR:RadRotator ID="RadRotator1" runat="server"
     ContentFile="Proverbs.xml" Title="Proverbs to Live By" Width="50%">
    <FrameTemplate>
       <div>
             <%# DataBinder.Eval(Container.DataItem, "Saying") %>
       </div>
    </FrameTemplate>
</radR:RadRotator>
```

12. Run the project. As with the two previous databinding methods, you should see one of the proverbs displayed, and the displayed proverb should change every few seconds.

Reading data from a live XML data source

The final means of attaching data to your rotator control is by using a live XML data source. There is no substantial difference between this method and the previous (static) XML method, except that your content this time is specified as an rss feed. Obviously, you need to determine the structure of the records provided by the feed in order to select and display the desired data fields, but once you've determined what you want to display, the rotator control handles the rest. We'll take the rest of this example straight from the sample code distributed by Telerik with the rotator component.

13. In the markup, change the ContentFile property to point at the Telerik website, specify the DataMember (="Item") and hook up your item template like this:

```
<radR:RadRotator ID="RadRotator2" runat="server" DataMember="Item"
      ContentFile="http://www.telerik.com/support.rss" Width="500" Height="400" >
    <FrameTemplate>
         <div style="height:360px; overflow-y:auto">
            <strong>
                  <a href="<%# DataBinder.Eval(Container.DataItem, "link") %>"
             target="_blank">
                       <%# DataBinder.Eval(Container.DataItem, "title") %></a>
            </strong>
            <br />
              <%# DataBinder.Eval(Container.DataItem, "pubDate") %>
            <p>
                  <%#DataBinder.Eval(Container.DataItem, "description") %>
            </p>
        </div>
    </FrameTemplate>
</radR:RadRotator>
```

14. Run the project. The rotator control will connect to the Telerik rss feed, and start paging through the news items from the feed. Note that the "Live" XML connection is "live" only in the sense that the data is under the control of an external data source. All available items will be loaded at the time your page is first loaded, and will be displayed in turn. If new items are added to the content of the remote connection, they will not be available for display until the page is refreshed.

Transition Effects and other property settings

There are a number of property settings which affect the way your rotator is displayed. They're easily locatable in the property setter at design time, and are also well documented in the help manual. They're just what you'd expect to find in a well-designed rotator control. We point them out here primarily for completeness.

AutoAdvance (true/false) determines whether the rotator will advance frames automatically, or wait for a command like ShowNextFrame (covered in the section Taking control of the rotator using the Client-Side API).

FramesToShow allows you to specify how many content frames to show at one time on your page. Specifying two frames, for instance, means that when the frameTimeout expires, the first and second frames will be replaced by the second and third frames - in other words, only one frame at a time rolls off the page.

FrameTimeout is the length of time, in milliseconds, that the frame set will be displayed

PauseOnMouseOver (true/false) determines if frame rotation is suspended when the mouse is over the rotator control.

ScrollDirection (up/downleft/right) dictates the direction in which the disappearing frames will leave the visible area.

Transition Effect is one of seventeen different effects used to move between successive frames in the rotation. **Important:** transition effects only work in Internet Explorer, and only when the TransitionType property is set to SlideShow.

TransitionType is used in conjunction with other properties. When set to SlideShow, the TransitionEffect setting determines how the view will change from one frame to the next; in Scroll mode, the ScrollDirection setting takes effect.

UseRandomSlide (true/false), when in SlideShow mode, allows the rotator to display slides randomly from the frameset..

Lab: Using more complex templates

Binding and displaying simple text is pretty straightforward, as you have seen. The good news is, incorporating images and even other controls like the RadTicker control is also easily accomplished inside the rotator control.

1. Start this exercise by clearing all controls from your "Rotation" form.

2. Drop a fresh RadRotator control onto your form

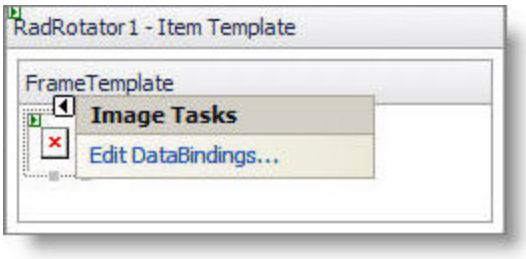
3. Add an XML file named "RotatorDemo.xml" to your Rotation project to supply the data for the rotator. If you have a set of jpg or other graphic images you'd like to use, go ahead and use them; otherwise, there are four standard images distributed with Windows XP which we'll use. When you're done, the XML file should look something like this:

```
<?xml version="1.0" encoding="utf-8" ?>
<Items>
  <Item>
     <Image>images\Blue hills.jpg</Image>
    <Name>Blue hills</Name>
    <Description>
```

```
      This is the birthplace of Paul Bunyan's big blue ox named Babe.
    </Description>
  </Item>
  <Item>
     <Image>images\Sunset.jpg</Image>
    <Name>Sunset</Name>
    <Description>
       If you want to see another lousy sunset in paradise like this,
        visit Puerta Vallarta.
    </Description>
  </Item>
  <Item>
     <Image>images\Water  lilies.jpg</Image>
    <Name>Water  lilies</Name>
    <Description>
      This particular pond was the inspiration for Claude Monet's famous painting
     of the same name.
    </Description>
  </Item>
  <Item>
     <Image>images\Winter.jpg</Image>
    <Name>Winter</Name>
    <Description>
      Mark Twain is famous for (among other things) his quote that the coldest
      winter he ever spent was a summer in San Francisco.
    </Description>
  </Item>
</Items>
```

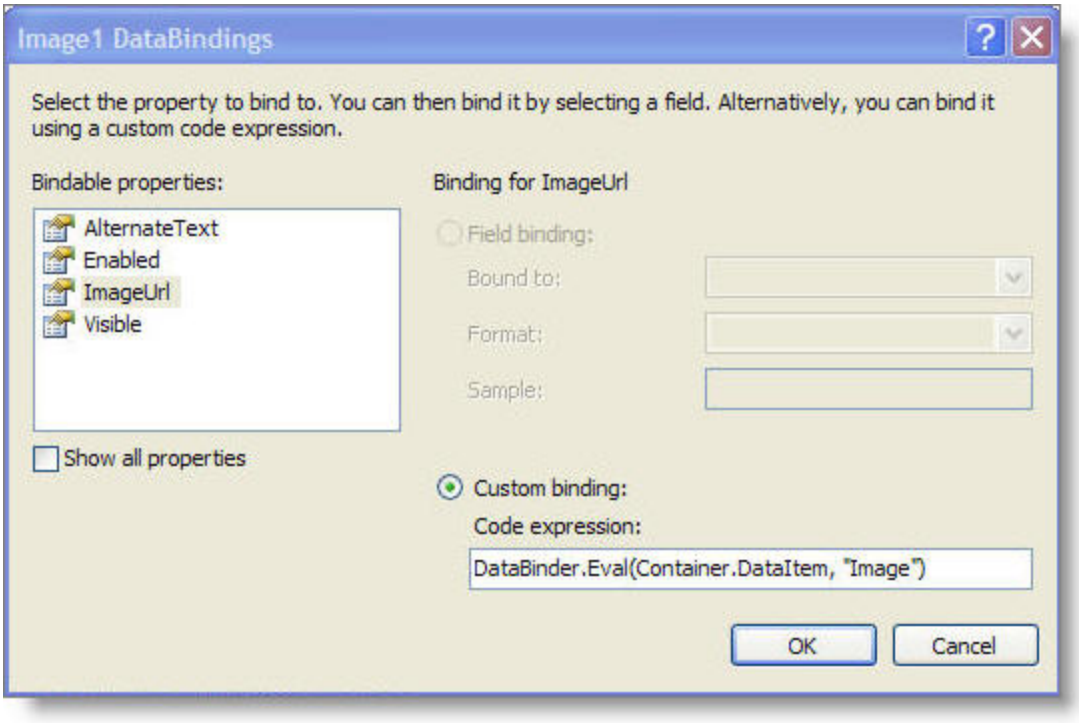
Using an image control in a template

4. Just as we did in Creating the display template, we'll add a control to the rotator's template. This time, however, we'll add an Image control.



Editing the template

5. Edit the databindings, and bind the Image control to the "Image" field of the data source.



Editing the databindings

6. Before seeing the rotator display our images, we also need to hook up the XML data source as we did in Reading data from a static XML file. Make sure you add the ContentFile setting to your rotator control's definition. Another of the issues you need to consider when displaying images in the RadRotator is the size of the graphic. In the case of the images we'll be using, they're all 800 x 600 pixel images - but the default size for the rotator control is smaller than that. You'll either need to set the rotator control size to accommodate the graphics you're using, or set the size of the image control. In order to display the pictures we've suggested, your control declaration will need to look something like this:

```
<radR:RadRotator  ID="RadRotator2"  runat="server"  ContentFile="RotatorDemo.xml"  >
     Width="800px"  Height="600px">
     <FrameTemplate>
        <div>
              <asp:Image ID="Image1" runat="server"
                 ImageUrl='<%# DataBinder.Eval(Container.DataItem, "Image") %>' />
        </div>
     </FrameTemplate>
</radR:RadRotator>
```

7. The last thing you need to do before running your project is to place the images where they can be accessed by the controls. The ASP.Net image control accesses graphics in either a relative or an absolute directory structure. However, when the image control is contained within a rotator template, absolute paths don't work, so we need to move the images where the control will be able to access them. A simple way to do this is to create an "images" folder in the website directory structure, and copy into it the four files (listed in the "RotatorDemo.xml" content file above) from their location in the "Sample Pictures" folder of C:\Documents and Settings\Default User\My Documents\My Pictures.

8. Press <F5> to run the lab. You should see the four pictures from the content file display continuously on the rotator control.

Adding more to the template: Labels and a ticker

One of the nice features about using templates for your display is that you can take full advantage of HTML and css formatting capabilities, using it to organize multiple display elements.

9. Using the smart tag on the RadRotator control, open the item template for editing.

10. Add some structure and controls to the template so that you can display the name of the picture as well as the description. Place the description display in a ticker control. You should end up with markup looking something like this. **Notice the data binding for the ticker text:** it's placed between the beginning and ending ticker tags.

```
<radR:RadRotator   ID="RadRotator1"   runat="server"   ContentFile="RotatorDemo.xml"
   Width="800px"  Height="600px"  >
     <FrameTemplate>
         <table width="100%">
             <tr  width="100%">
                 <td width="60%" ></td>
                 <td width="40%" ></td>
             </tr>
             <tr>
                <td align="center">
                   <asp:TextBox id="txtTitle" runat="server"
                     Text='<%#  DataBinder.Eval(Container.DataItem,  "Name")  %>'  />
               </td>
               <td></td>
             </tr>
             <tr>
               <td>
                   <asp:Image  ID="Image1" runat="server" Width="480px"
                     ImageUrl='<%#  DataBinder.Eval(Container.DataItem,  "Image")  %>'  />
               </td>
               <td>
                   <radR:RadTicker  ID="RadTicker1" runat=server>
                     <%# DataBinder.Eval(Container.DataItem, "Description") %>
                  </radR:RadTicker>
               </td>
             </tr>
         </table>
     </FrameTemplate>
</radR:RadRotator>
```

11. Now run your project. If you did everything correctly, you should now have a rotator which displays the graphic image, the title of the image, and the description we entered in the XML file earlier in the lab.

Taking control of the rotator using the Client-Side API

Sometimes you'll want to leave control of the content displayed in a rotator in the hands of the user. The RadRotator control provides client-side event handling to take care of this for you. With the events provided, you can start and stop the frames from advancing, as well as scroll the images left and right, up and down. In this next lab section, we'll add some scroll control to our interface. When we're finished, you'll be able to start, stop and scroll the images in your rotator

12. You may wish to start by creating the icons you'll use for your directional controls. You can draw them yourself, if you'd like, using a utility like Microsoft Paint; or you can download them from some other source. Making some quick-and-dirty icons for the lab exercise will probably suffice. Here's what I came up with. They're names lt, stop, start, and rt, respectively.

Some rotator control images

13. Because we don't need (or even want) postbacks while we're controlling the rotator display, we are going to use simple HTML button controls instead of ASP.Net buttons on the form. In order to make effective use of the images with our icons, we'll set their properties using a stylesheet. Add a new stylesheet to your project named "rotator.css", and set the properties for your stytles like this:

```
.stop
{
       width:  22px;
       height:  22px;
       background-image:    url(images/stop.gif);
       background-color:   transparent;
       border-style:  solid;
       border-width:  thin;
       background-repeat:   no-repeat;
}
.start
{
       width:  22px;
       height:  22px;
       background-image:    url(images/start.gif);
       background-color:   transparent;
       border-style:  solid;
       border-width:  thin;
       background-repeat:   no-repeat;
}
.left
{
       width:  22px;
       height:  22px;
       background-image:   url(images/lt.gif);
       background-color:   transparent;
       border-style:  solid;
       border-width:  thin;
       background-repeat:   no-repeat;
}
.right
{
       width:  22px;
       height:  22px;
       background-image:   url(images/rt.gif);
       background-color:   transparent;
       border-style:  solid;
       border-width:  thin;
       background-repeat:   no-repeat;
}
```

14. Next, add a table to your page immediately below the rotator. Populate it with four HTML buttons (remember: **not** asp:buttons), and set the styles and click events like this:

```
<table  width="480px">
    <tr>
         <td width="130px"></td>
        <td width="40px">
              <button onclick="RadRotator1.ShowPrevFrame();" type="button"
```

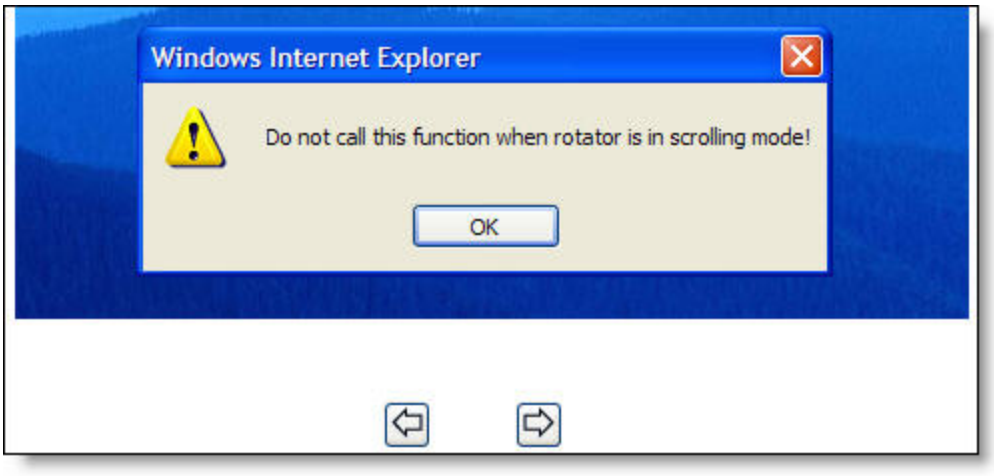
```
                class="left" />
            <td width="20px"></td>
            <td width="40px">
                <button onclick="RadRotator1.StopRotator();" type="button"
                class="stop" />
            <td width="20px"></td>
            <td width="40px">
                <button onclick="RadRotator1.StartRotator();" type="button"
                class="start" />
            <td width="20px"></td>
            <td width="40px">
                <button onclick="RadRotator1.ShowNextFrame();" type="button"
                class="right" />
            <td width="130px"></td>
        </tr>
</table>
```

15. Also, don't forget to add the style sheet reference in the head section of your page:

```
<head  runat="server">
     <title>Rotator Demo</title>
      <link rel="stylesheet" type="text/css" href="rotator.css" />
</head>
```

Now run your project. If you left the rotator in AutoAdvance mode, you should see the procession of images across the browser page as before. Clicking on the control buttons should change the displayed picture, along with the title and the description displayed in the RadTicker. You may notice that even though the rotator stops advancing, the ticker continues to re-display the description associated with the picture according to whatever timing parameters you've set for the ticker. This might be desirable behavior; or it might be very annoying. If it's annoying, you may not want to use the ticker control.

There's one other issue you should be aware of when controlling the rotator, and that is that there are two different transition types available for the rotator (Scroll and SlideShow), and eight different motion controls. Start and Stop apply to both transition types; however, ShowNextFrame and ShowPrevFrame only apply when the control is in SlideShow mode; and the Scroll-Up/Down/Left/Right-NextFrame apply only to the scrolling mode. If you apply an out-of-context command to the current mode, you'll get scolded by the component:



Error generated from the rotator

Responding to Client-Side Events

The RadRotator provides two client-side events, OnClientFrameChanging and OnClientFrameChanged.

The OnClientFrameChanging event fires just as the rotator frame is about to change. Handling this event can be used to halt execution of the change by returning a value of false. In this section of the lab, we'll use the OnClientFrameChanging event to show an alternate means of stopping the movement of the rotator.

16. Add a checkbox named "cbSuspend" and the following code to your aspx page. Although the function definition portion of the script can appear just about anywhere within the page, the "attachEvent" portion **must** appear after the Rotator control has been declared. If you're going to keep the entire script together in one place as we do here, make sure that it appears after the Rotator declaration, or you'll get an error telling you that the Rotator is undefined. Also, notice the logic on the checkbox - if we check the checkbox to suspend rotation, but also need to return "false" to stop rotation, we need to invert the checkbox state to accomplish our goal.

```
<script   type="text/javascript">
       <!--
           RadRotator1.attachEvent("OnClientFrameChanging","bchange_handler");

         function bchange_handler(sender, args)
       {
             var target = document.getElementById("cbSuspend");
            return !target.checked;
       }
       -->
</script>
```

17. Run your project. Verify that rotation stops when you check the "Suspend" checkbox.

It is also important to know that once you have stopped the rotator frame from changing, it stays stopped until the state of the control is reset in some fashion. One way to do this is to move the rotator image manually as we did in the previous section "Taking Control of the Rotator".

18. Try unchecking the checkbox, and notice that the image displayed remains the same. Next, click on one of the frame forward/back buttons. After the normal delay, rotation will resume.

The other client-side event available on the Rotator is OnClientFrameChanged, which fires immediately after the frame has changed.

19. Add a label named "lblCount" to the design surface. Add the following script, again paying attention to the rules about script placement within the page:

```
<script   type="text/javascript">
       <!--
       RadRotator1.attachEvent("OnClientFrameChanged","achange_handler");
       var clickCount = 1;
       function  achange_handler(sender,  args)
       {
              clickCount += 1;
              document.getElementById("lblCount").innerText    =
                "Total frames viewed : " + clickCount;
       }
       -->
</script>
```

20. Run the project. Note You should see the view counter increment each time a different frame is displayed.

3.6.3 Using RadRotator at Runtime

At first glance, using a RadRotator control might seem like a fairly static exercise; but there are cases where creating the control programmatically might also be useful. Consider, for instance, page customization: one user might be interested in current stock market results from various stock exchanges, while another user wants to see weather reports and commodities prices. One way to accomplish this, of course, would be to put several controls on the page and set their properties and display them as needed. That seems like a fair amount of brute force, and has obvious limitations.

It is possible to achieve the same goal dynamically - even though the intellisense help provided by Telerik is pretty straightforward in its warning that "You should not use this", the following lab uses an approach from the Telerik code library.

Lab: Programming the RadRotator dynamically

1. Remember back at the beginning of the rotator material, where we placed a rotator on the page and then used the smart tags to edit the display template? Well, dynamically created rotators need templates, too, only now it's up to you to create them programatically. The first thing you need is a class to help with the ITemplate interface. Define the class like this:

```
C# example:
class RotatorTemplate : ITemplate
{
    string columnName;
    public RotatorTemplate(string column)
    {
        columnName = column;
    }

    public void InstantiateIn(Control c)
    {
        Label lbl = new Label();
        lbl.DataBinding += new EventHandler(OnDisplayDataBinding);
        c.Controls.Add(lbl);
    }

    void OnDisplayDataBinding(object sender, EventArgs e)
    {
        Label lbl = (Label)sender;
          RadRotatorFrame iContainer = (RadRotatorFrame)lbl.NamingContainer;
         DataRowView drv = ((DataRowView) iContainer.DataItem);
         lbl.Text = drv[columnName].ToString();
          lbl.BackColor = System.Drawing.Color.FromName(lbl.Text);
    }
}

VB example:
Class RotatorTemplate
    Implements ITemplate
   Private columnName As String

   Public Sub New(ByVal column As String)
       columnName = column
   End Sub

   Public Sub InstantiateIn(ByVal c As Control)
      Dim lbl As Label = New Label
```

```
        AddHandler lbl.DataBinding, AddressOf  OnDisplayDataBinding
        c.Controls.Add(lbl)
    End Sub

    Sub  OnDisplayDataBinding(ByVal sender As Object, ByVal e As  EventArgs)
       Dim lbl As Label = CType(sender, Label)
       Dim iContainer As  RadRotatorFrame  =  CType(lbl.NamingContainer,  RadRotatorFrame)
       Dim drv As  DataRowView  =  CType(iContainer.DataItem,  DataRowView)
         lbl.Text = drv(columnName).ToString
          lbl.BackColor  =  System.Drawing.Color.FromName(lbl.Text)
    End Sub
End Class
```

A brief digression to look at what's going on in the ITemplate interface implementation is appropriate. Most of the references found on the internet about implementing ITemplate are very specifc, and this one is frankly no better; but it should help you understand what's going on here.

The class name: Naming the class RotatorTemplate is a convenience for keeping things straight. The use of a "CustomerGridEditTemplate" class should be obvious.

The declared private variables: Use what you need. In this case, we're going to use the same RotatorTemplate to bind controls to two different tables, one with a column named "proverb" and one named "color", so when binding the template's textbox control, we have to know which column to bind to. Obviously, a more complex template (one binding several controls to several columns) would require the use of more private variables and more parameters in the constructor.

The constructor: Here, it's used only as a vehicle for persisting the data columns we're going to bind to, though you could pass other meaningful or useful parameters as well.

InstantiateIn: This is where the template's controls get defined. The InstantiateIn method gets called when the template gets created as the control is being prepared for rendering to the page, and you'll see where this fits in a little later in the lab, This example is a very simple case, since we're using only a single label to bind to one display-only piece of data; but we could as easily add multiple controls in editable textboxes, or images, or RadTicker controls, or - well, you get the idea. Also, when you're constructing the template, don't forget the formatting (such as line breaks) . This is also where you add the DataBinding event handler for the control. It's not necessary to add a DataBinding handler for more than one control - we'll discuss that a little further in a moment.

OnDisplayDataBinding: Again, this is a name of convenience with no special naming requirement. The typecast of the sender to be of type Label in the first line should be of the same type as the control actually triggering the OnDisplayDataBinding event that you defined in the InstantiateIn routine. The next line is the trickiest part when first trying to figure out this implementation, though in hindsight it's perfectly clear, and that is that the iContainer declaration is of the type of the object holding the template (in this case, a RadRotatorFrame), and is also the object to which the data is getting bound. This last point is crucial, because it's what enables us to make the typecast in the following line of the DataItem into a DataRowView. By the way, DataItem is a property of the IDataItemContainer interface.

When there are multiple controls to be bound in the databinding event, you only really need to add a handler for just one of the databound controls. Then, inside the handler, loop through the controls collection of the naming container, locate the controls to be databound, and set the values. You don't need to write multiple event handlers or complex assignment logic based on which control was the sender. Limit the sender to one control, and bind everything on the one event.

Finally, just for fun, we've added the line to set the BackColor property of the label. When it tries to convert one of our proverbs to a color it doesn't have any success, and so leaves the background white; however, if you slip in a color "proverb" (like "Turquiose"), you'll get a colored background in the first rotator, which changes back to white as soon as it tries to convert a real proverb again.

2. After defining the RotatorTemplate class, the rest of the process is trivial - all you need to do is construct the logic to determine how many rotators you want to display to the user, create them, and hook up the appropriate rotator content. We'll leave the construction of the user preferences page as an independent exercise; for now, let's assume that the user has selected proverbs and colors as the items to be viewed in the rotator. Let's create and test the proverbs display first.

```
C# Example:
protected void Page_Load(object sender, EventArgs e)
{
    DataTable proverbsTable = new DataTable();
     proverbsTable.Columns.Add("proverb");

     proverbsTable.Rows.Add("A stitch in time saves nine");
     proverbsTable.Rows.Add("A rolling stone gathers no moss");
     proverbsTable.Rows.Add("The early bird catches the worm");
     proverbsTable.Rows.Add("A penny saved is virtually worthless");

    RadRotator rotator1 = new RadRotator();
    rotator1.ID = "rotator1";
     rotator1.DataSource = proverbsTable;

     rotator1.FrameTemplate = new RotatorTemplate("proverb");
      rotator1.TransitionType = RadRotator.RotatorTransitionType.Slideshow;
      rotator1.TransitionEffect = RadRotator.RotatorTransitionEffect.GradientWipe;
    rotator1.Width = 500;
    rotator1.Height = 50;
    rotator1.AutoPostBack = true;
     PlaceHolder1.Controls.Add(rotator1);

     rotator1.DataBind();
}

VB Example:
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim proverbsTable As DataTable = New DataTable
      proverbsTable.Columns.Add("proverb")
     proverbsTable.Rows.Add("A stitch in time saves nine")
     proverbsTable.Rows.Add("A rolling stone gathers no moss")
     proverbsTable.Rows.Add("The early bird catches the worm")
     proverbsTable.Rows.Add("A penny saved is virtually worthless")
    Dim rotator1 As RadRotator = New RadRotator
     rotator1.ID = "rotator1"
     rotator1.DataSource = proverbsTable
     rotator1.FrameTemplate = New RotatorTemplate("proverb")
       rotator1.TransitionType = RadRotator.RotatorTransitionType.Slideshow
       rotator1.TransitionEffect = RadRotator.RotatorTransitionEffect.GradientWipe
     rotator1.Width = 500
     rotator1.Height = 50
     rotator1.AutoPostBack = True
      PlaceHolder1.Controls.Add(rotator1)
     rotator1.DataBind
End Sub
```

You should be starting this as a new project - if you're reusing a previous project, make sure that you delete any rotators you might already have placed on the form to avoid naming conflicts with the rotator you're creating here.The only object you should have on your form is the placeholder named PlaceHolder1 (the reason you can't just add the rotator directly to the form's controls collection is because the rotator must be placed inside a form tag with runat=server, which the placeholder does).

The only interesting thing about the code above is the line:

```
C#: rotator1.FrameTemplate   = new  RotatorTemplate("proverb");

VB: rotator1.FrameTemplate   = New   RotatorTemplate("proverb")
```

because this is where you reference the RotatorTemplate class you created, and where you pass the column to be bound to the label displayed in the rotator template.

3. Run your project. Verify that your page displays our series of proverbs.

4. We'll now add a second rotator control to the project, but you can take the example to demonstrate that you could add as many as you could want. Add the following to your Page_Load:

```
C#  Example:
     DataTable colorsTable = new  DataTable();
      colorsTable.Columns.Add("color");
      colorsTable.Rows.Add("Red");
      colorsTable.Rows.Add("Orange");
      colorsTable.Rows.Add("Yellow");
      colorsTable.Rows.Add("Green");
      colorsTable.Rows.Add("Blue");
      colorsTable.Rows.Add("Indigo");
      colorsTable.Rows.Add("Violet");

     RadRotator rotator2 = new  RadRotator();
     rotator2.ID = "rotator2";
      rotator2.DataSource = colorsTable;
      rotator2.FrameTemplate = new  RotatorTemplate("color");
        rotator2.TransitionType  =  RadRotator.RotatorTransitionType.Slideshow;
        rotator2.TransitionEffect  =  RadRotator.RotatorTransitionEffect.Pixelate;
     rotator2.Width = 500;
     rotator2.Height = 50;
    this.Controls.Add(rotator2);
     rotator2.DataBind();

VB  Example:
    Dim  colorsTable As DataTable = New  DataTable
      colorsTable.Columns.Add("color")
     colorsTable.Rows.Add("Red")
      colorsTable.Rows.Add("Orange")
      colorsTable.Rows.Add("Yellow")
      colorsTable.Rows.Add("Green")
      colorsTable.Rows.Add("Blue")
      colorsTable.Rows.Add("Indigo")
      colorsTable.Rows.Add("Violet")

    Dim  rotator2 As RadRotator = New  RadRotator
     rotator2.ID = "rotator2"
      rotator2.DataSource = colorsTable
      rotator2.FrameTemplate = New   RotatorTemplate("color")
        rotator2.TransitionType  =  RadRotator.RotatorTransitionType.Slideshow
        rotator2.TransitionEffect  =  RadRotator.RotatorTransitionEffect.Pixelate
     rotator2.Width = 500
     rotator2.Height = 50
      PlaceHolder1.Controls.Add(rotator2)
     rotator2.DataBind
```

5. Run your project. If you followed the code example exactly, you should now see two rotators in your browser, each displaying its own associated (databound) elements. Even though they each have different numbers of elements to display, they keep rotating through their respective elements to the end of the list, then start over. And, each has a different transition effect. You get the idea - these rotators act just like you'd expect them to!

3.6.4 Summary

In this chapter we discussed the utility of RadTicker and RadRotater and how they can be used together in concert or separately. We populated the RadTicker in code and worked through several methods of binding data to the RadRotator. We worked with progressively more complex templates for RadRotator. We used the client side api and events to manipulate RadRotator.

For more information going forward, the Telerik website has a wealth of information and examples, as well as downloads of the help manual and trial versions of other controls, white papers, and sample projects. Visit the Telerik website at http://www.telerik.com/demos/aspnet/Rotator/Examples/Overview/DefaultCS.aspx

If you're interested in learning more about the ITemplate interface, there are lots of materials to be found on the internet. Dino Esposito provides an introduction to templates at http://msdn.microsoft.com/msdnmag/issues/02/01/cutting/, and there's a microsoft support article at http://support.microsoft.com/default.aspx/kb/310898.

3.7 Appendix 1: MSDE

3.7.1 Installing MSDE

1. Download "Microsoft SQL Server 2000 Desktop Engine (MSDE 2000) Release A" at:

   http://www.microsoft.com/downloads/details.aspx?familyid=413744d1-a0bc-479f-bafa-e4b278eb9147&displaylang=en

2. The downloaded file will be MSDE2000A.exe. Run it to unpack the installation. When the installation wizard prompts for "Installation Folder", leave the default path "C:\MSDERelA".

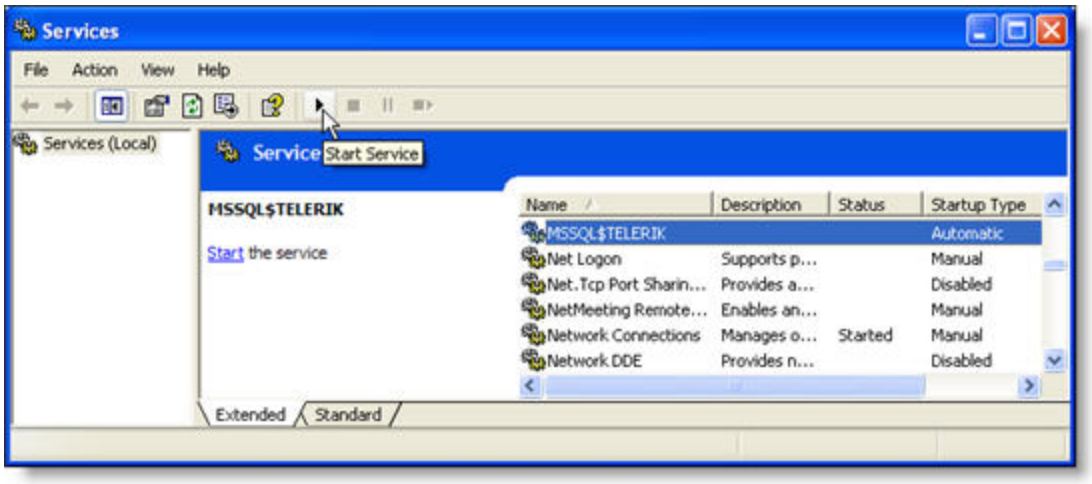
3. From the Start menu run "cmd".

4. On the command line navigate to the installation folder. For example: cd \MSDERelA

5. On the command line run:

   C:\MSDERelA>**setup.exe INSTANCENAME=Telerik SAPWD=sa SECURITYMODE=sql**

6. From the Start menu run Control Panel | Administrative Tools | Services. Find MSSQL$TELERIK and start the service.



Starting the Telerik MSDE instance

3.7.2 Connecting to MSDE on the command line

To connect with the command line directly, start a command prompt and run the following:

osql -U sa -PSA -S localhost\Telerik

This command runs the MS SQL command interpreter, passing it a user name of "sa" (system administrator) and password of "SA" where the MSDE instance is "Telerik" on the local machine.

Note: If the command times out citing connection issues, make sure the MSSQL$TELERIK service is started (see the Installing MSDE section).

3.7.3 Connecting to MSDE in Visual Studio

1. In a new web application drop a SqlDataSource on the default page.
2. Click the Smart Tag "Configure Data Source" option.
3. In the "Configure Data Source" wizard click the New Connection button.
4. In the "Add Connection" dialog set the following:

- Server Name: localhost\telerik.
- Log on to server: Use Sql Server Authentication, User Name = "sa", Password "SA", Save My Password = checked.
- Connect to database: Select or enter a database name = "master". Note: the drop down list should show you "master" as one of the items. If the list doesn't drop down there is likely a problem with one of the preceding settings.

5. Click the "Test Connection" button.

Testing the connection settings

3.7.4 Installing Northwind database

The steps described here will install the Northwind database to your MSDE instance.

1. Download the Northwind install scripts install from:

http://www.microsoft.com/downloads/details.aspx?FamilyID=06616212-0356-46a0-8da2-eebc53a68034&DisplayLang=en

2. Run the install (SQL2000SampleDb.msi). Leave the defaults and finish the install wizard.

3. The scripts by default are installed to:

    c:\SQL Server 2000 Sample Databases

4. From the Start | Run menu run "cmd".

5. From the command line run:

   osql -U sa -PSA -S localhost\Telerik -iinstnwnd.sql

6. To test the Northwind installation run from the command line:

   osql -U sa -PSA -S localhost\Telerik

7. On the osql command line run:

   use Northwind

   select CategoryName from Categories

   go

A list of category names should display. Type "exit" to leave osql.

```
C:\SQL Server 2000 Sample Databases>osql -U sa -PSA -S localhost\Telerik
1> use Northwind
2> select CategoryName from Categories
3> go
 CategoryName
 ---------------
 Beverages
 Condiments
 Confections
 Dairy Products
 Grains/Cereals
 Meat/Poultry
 Produce
 Seafood

(8 rows affected)
1> exit

C:\SQL Server 2000 Sample Databases>
```

Testing the Northwind database

3.8 Appendix 2: Firebug

3.8.1 Getting Started

Firebug is an exciting new tool that lets you monitor the entire stack of technologies used in AJAX. This outstanding debugger has features previously available only in full featured IDEs and some features that are completely fresh.

Telerik recommends Firebug for debugging client scripts, tweaking CSS, monitoring AJAX network traffic and quite a bit more. You will understand why Firebug is so powerful after using it for thirty seconds than any amount of description can provide, but here is a quick teaser before we take the tour:

- The Firebug script debugger has capabilities for inspecting, logging and profiling JavaScript. The Firebug API provides powerful logging features for assertions, enhanced logging for errors, stack traces, warnings, object and xml dumps, the ability to group logging statements together. Firebug has a complete array of debugging navigation capabilities for stepping through code. The Firebug profiler pinpoints frequently called or long running JavaScript functions.
- Use Firebug to inspect *and* edit HTML on the fly. The Inspect option lets you easily locate tags deep in the page and edit the html to see instant results. For example you could add items to an ordered list "<ol>" tag. Hours of fun.
- Inspect and edit style sheets on the fly. This feature lets you visualize how style rules are working on your page, what elements are being inherited or overridden. Edit or disable style elements to see the effect immediately.
- The Document Object Model contains a hierarchy of objects and functions. Firebug lets you find DOM objects easily and edit them.
- Monitor network traffic, particularly XMLHttpRequests used to drive AJAX functionality. See the incoming HTTP requests and the entire returned response content.
- You can even use Firebug for security reconnaissance to detect security holes and sql injection vulnerabilities. The difference between Firebug and the current crop of web security software is that Firebug detects AJAX requests that might be exploited like any other web page access. See "Hacking Web 2.0 Applications with Firefox" at http://www.securityfocus.com/infocus/1879 for that discussion.

Firebug running in Firefox

Lab: Installing and Running Firebug

Firebug is a free, open-source add-in to Firefox written by Joe Hewitt. As a Firefox extension it requires Firefox to be installed first.

1. Find the Firefox install at http://www.mozilla.com/en-US/firefox and install it on your machine.
2. Once you have Firefox on your system, get Firebug at http://www.getfirebug.com/ and install it. **Note**: you can only run the Firebug install from the Firefox browser.
3. Display any page in Firefox. Click the green arrow in the lower right hand of the browser to invoke Firebug.
4. Click the bug icon to see Firebug basic configuration, options and information links. Click through the top level menu items "Console", "HTML", "CSS", "Script", "DOM" and "Net" to get a feel for how the navigation works in Firebug.

Invoking Firebug

3.8.2 Tour of Firebug

The Firebug toolbar has a main menu under the bug icon. Here you can set up either a black list to disallow sites from Firebug debugging or a white to allow only certain sites.

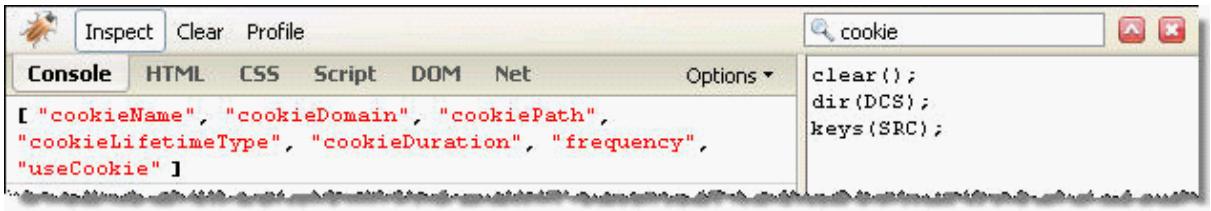
Next to the bug icon, a set of context sensitive commands change as the menu items below are clicked. This area may also display breadcrumbs when inspecting HTML to help locate you in DOM hierarchy.

To the right of the menu items an Options drop down has configuration specific to the currently selected menu item. In the figure below the Console menu item is selected so the options list the kinds of information the console should display.

Much of the content in Firebug cross links between menu items. For example if you find a JavaScript function using the DOM tool, clicking on the function will automatically navigate to the Script section where you will see the code for the function.

Menus on the Firebug toolbar

To the right of the menu and options is a search tool. The behavior is dependant on the currently selected menu. If the Console menu item is selected the search acts as a filter and only shows log items that match part of the string. If the HTML item is selected the search is incremental as you type, moving you forward through the HTML to the closest matching item. In either case the usability is very smooth and responsive. The red upward arrow opens Firebug in a new window and the "X" button closes Firebug.

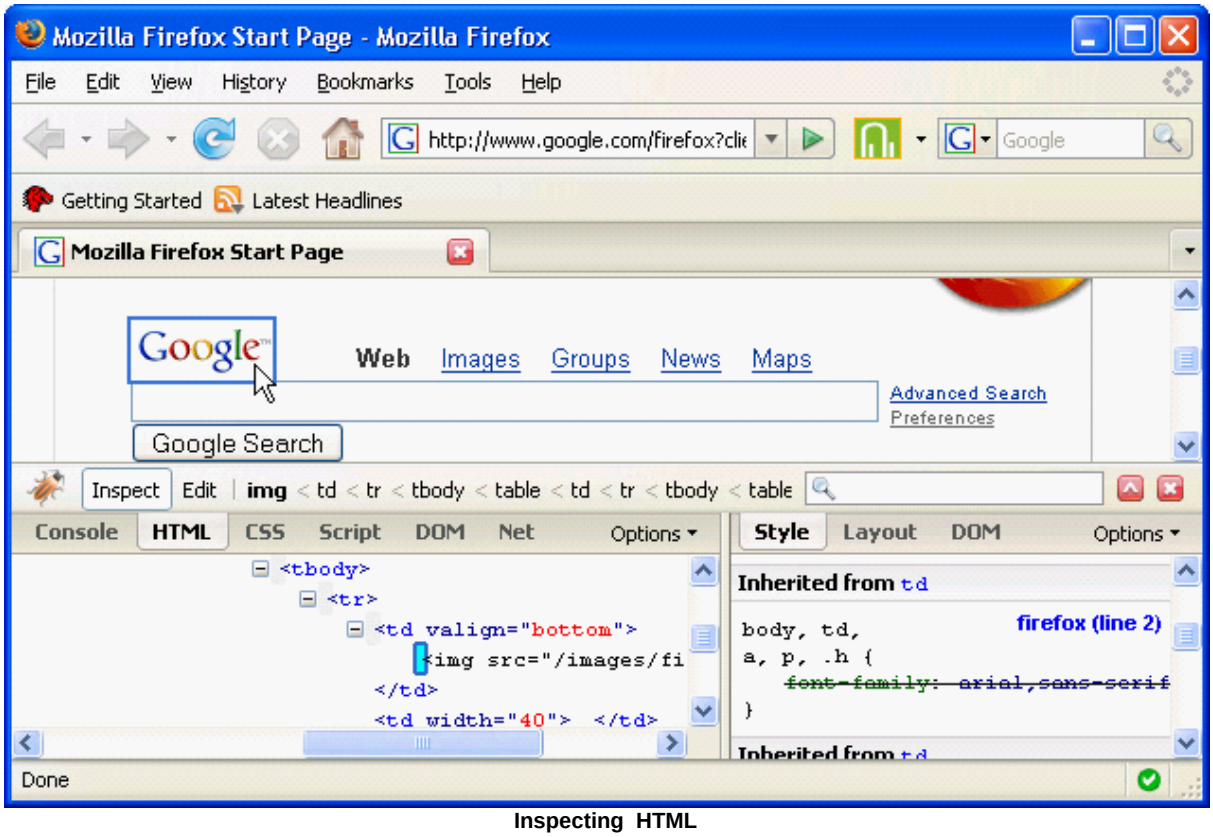


Filtering log items containing the fragment "cookie"

Lab: HTML

We start our tour on the HTML tab because its so darn easy to use, but very powerful just the same.

1. Run the Firefox browser and navigate to their default home page http://www.google.com/firefox. Click the Firebug button (green button, lower right) to start the debugger.
2. Click the Inspect button. That will automatically put you on the HTML tab. Move the mouse over web page and notice how the HTML tracks in the lower left window and the current styles in play show in the lower right hand window. Also try moving the mouse in the HTML to see elements under the mouse highlighted in the browser.
3. Move the mouse over the Google image and left click. Clicking on the HTML page has the effect of toggling off the Inspect button so you can move down to the HTML and Style windows to take a closer look. The HTML window has the "<img>" tag selected.



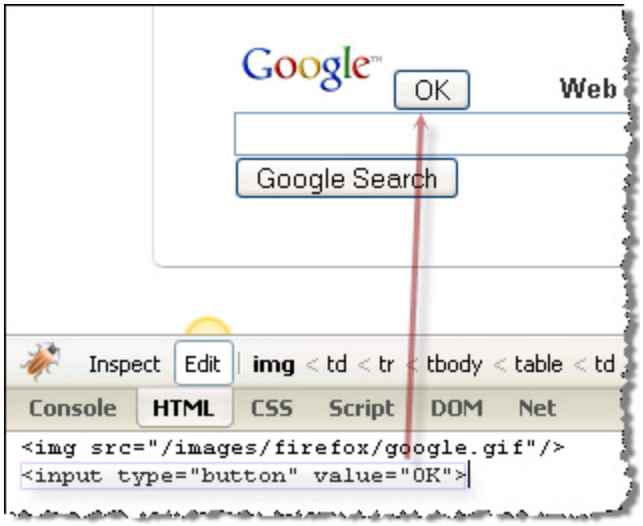
Inspecting HTML

4. The content menu in the HTML window has commands for adding, editing and deleting HTML elements and attributes. A shortcut to editing the an attribute is simply to click it. Try clicking the width attribute in the "<td>" element just below the image. Edit the width to a new value. Use the arrow keys, up to raise the value, down to lower the value. The changes are reflected in the browser.

Editing an HTML attribute

Note: Changes you make will not be persisted. The next time you hit the reload button all changes are erased. This is by design. The intent for this version of Firebug is that you audition changes to HTML and style here then copy them back to your usual source editor.

5. Click the Edit button (just to the right of the Inspect button). This provides an editable area for free entry for the element you were just in. As in the example below, add a new HTML input element tag and observe the result in the browser.



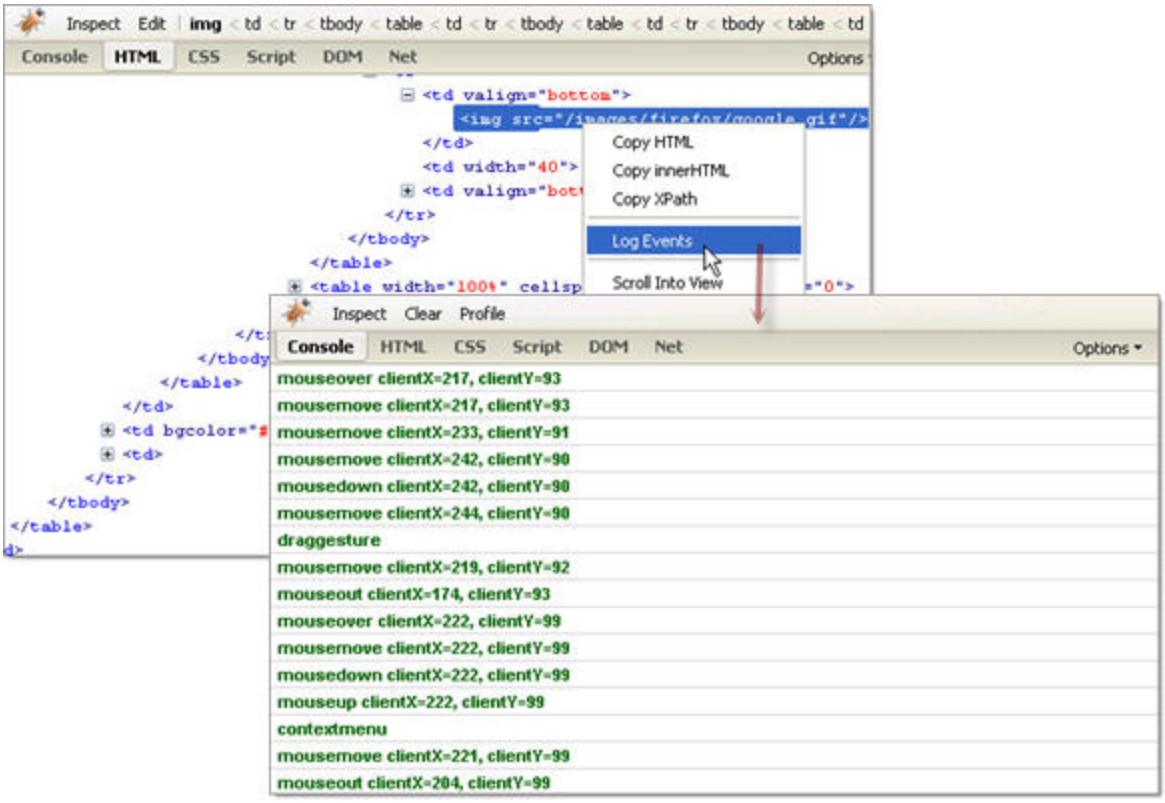
Adding an HTML element

6. Next to the Inspect and Edit buttons is the current HTML tag. A bread crumb trail to the right shows the containing HTML all the way up to the "<body>" and "<HTML>" tags. Click on these and notice how the html and styles respond. When you're done, toggle the Edit button off.

7. Firebug is able to display images and colors right in the HTML area. Move the mouse over the Google image tag "src" attribute. The image itself will display along with the image dimensions.

Firebug HTML displaying image and dimensions

8. Using the Inspect button, make sure the HTML is focused on the Google icon. Right click in the HTML and select Log Events. In the Browser generate HTML events by moving the mouse over the Google icon, right clicking or attempting to drag the icon. Now navigate to the Console tab and you will see a list of HTML events logged that include mouse actions, drag gestures and context menu events.



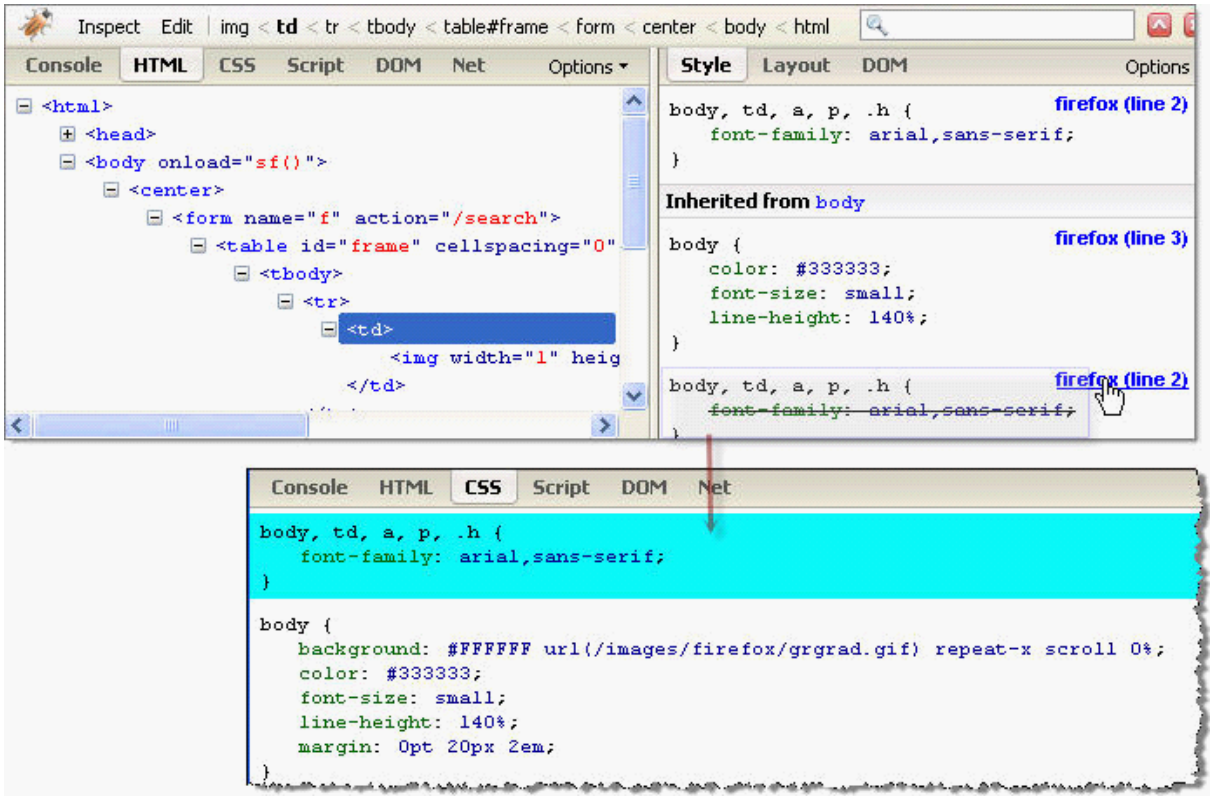
Logging HTML Events

Lab: CSS

The Firebug style window can be effective when making changes to RadControl skins. Here you can easily audition changes and visualize the impact without cycling the application.

1. Run the Firefox browser and navigate to their default home page http://www.google.com/firefox. Click the Firebug button (green button, lower right) to start the debugger.

2. In the HTML navigate to the first cell "<td>" of the first table. The Style window reflects the styles for the current element, the inheritance for these styles, and shows styles that have been superseded as crossed out. For example, look at the font-family. We can see that the style cascades from <body>. Click on the hyperlink to see the CSS where the style is established.



Navigating to style in use

3. Navigate back to the HTML tab. On the Style tab locate the first instance of the font-family (without the line through it). As you pass the mouse over the left side of the style declaration you will see a "Disable" icon. Toggle this icon and notice the difference in the browser. Because it defines the style for the body the font should be reflected everywhere on the page.

4. Right clicking in the CSS class definition will display a context menu allowing properties to be added, edited, deleted or disabled. Next to the body font-family entry, right click and select "New Property". Enter "background-color" then tab to the value portion and enter "blue". Notice the intellisense in the editor that knows the possible property names and provides type-ahead for you. After entering the property try running the mouse over it -- notice that Firebug displays a swatch representation of the color.

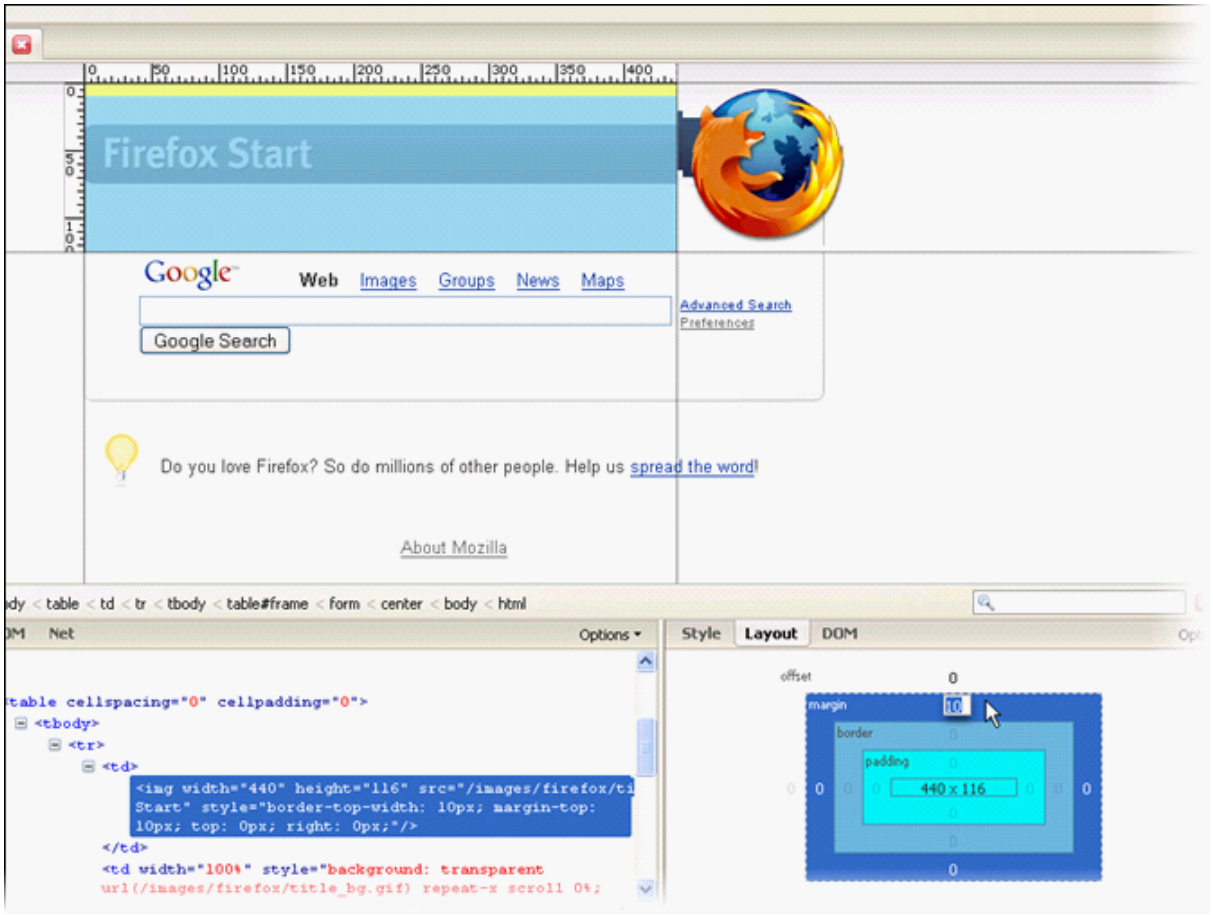
Entering a new property

5. Blue is a reprehensible background color in this context so right click the property and select "Delete Background-Color".

6. Click the Layout tab. Use the Layout window to assist with those tricky problems that come up when you're trying to line up blocks of content on the page and adjust spatial relationships.

7. Click the Inspect button, then click the "Firefox Start" banner on the page. Now run your mouse over the top of the layout area that controls offset, margin, border and padding. Notice how the ruled areas

appear in the browser? Click on the top margin number. Use the arrow button to increase this to 10. Using the arrow buttons instead of entering the numbers (which you can also do) makes it easy to watch what's happening on the screen while you make changes. All the numbers including the dimensions of the Firefox banner itself can be altered this way.



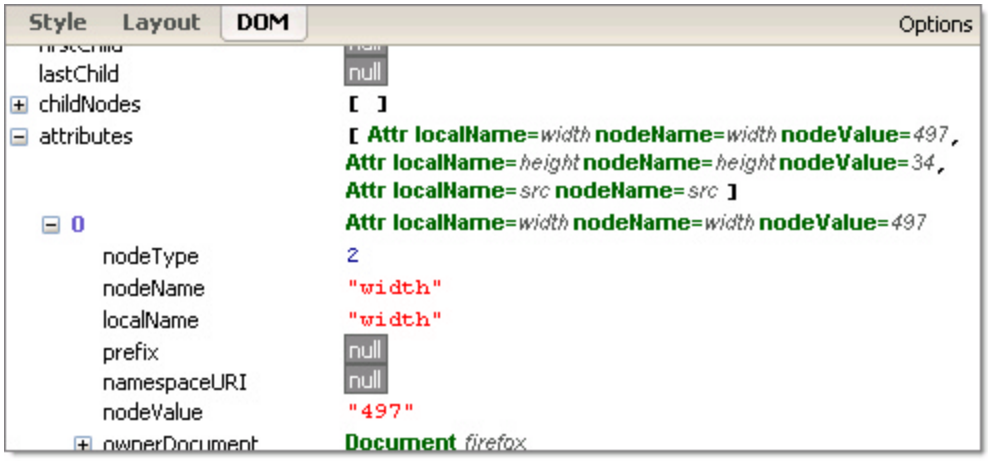
Making changes in the Layout window

Lab: DOM

The DOM tab displays all the objects and functions within the scope of the selected element in the HTML tag.

1. Run the Firefox browser and navigate to their default home page http://www.google.com/firefox. Click the Firebug button (green button, lower right) to start the debugger.

2. Click the DOM tab. The Document Object Model (DOM) contains a collection of objects and methods that can be operated on by JavaScript. This tab displays DOM objects and their values. The values can be edited on the fly. Move the cursor within the HTML window and observe the changes to the DOM.

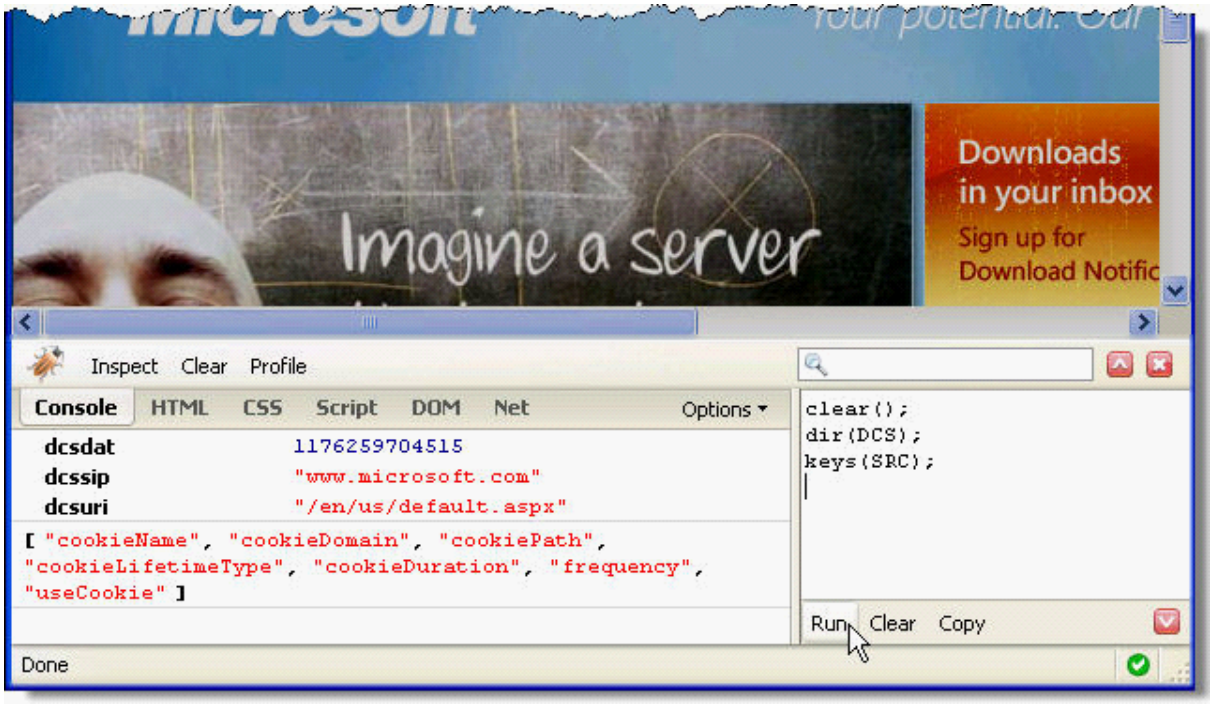


Using the DOM tab

3. The options menu filters to selectively show user defined properties, user defined functions, DOM defined properties, DOM defined functions and DOM constants. Select DOM defined properties and observe the result.

Console

When you select the Console button you have commands above it to *Inspect* the html, *Clear* the console logging area or start a *Profile* of browser activity. To the right of the logging is a command line to run JavaScript interactively that also has a command line API. **Note**: Find the most up-to-date list of available commands at http://www.getfirebug.com/commandline.html. We'll delve into inspecting and profiling later. Our first stop is the powerful logging functionality in Firebug.



Running commands output to the console

Logging

With Firebug you don't need "alert()" as a debugging mechanism. The Firebug Console API has a much richer set of functions. For example, instead of a statement like this:

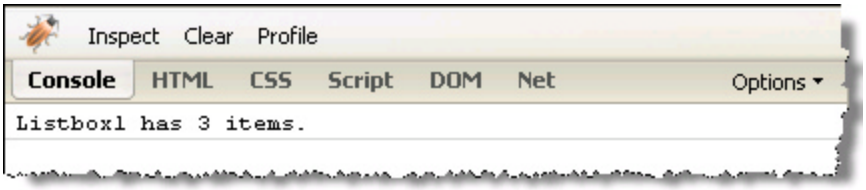
```
alert("Listbox1 has " + Listbox1.length + " items.");
```

...you would use the Firebug console log function:

```
console.log("Listbox1 has " + Listbox1.length + " items.");
```

...or pass any number of arguments:

```
console.log("Listbox1 has ", Listbox1.length, " items.");
```

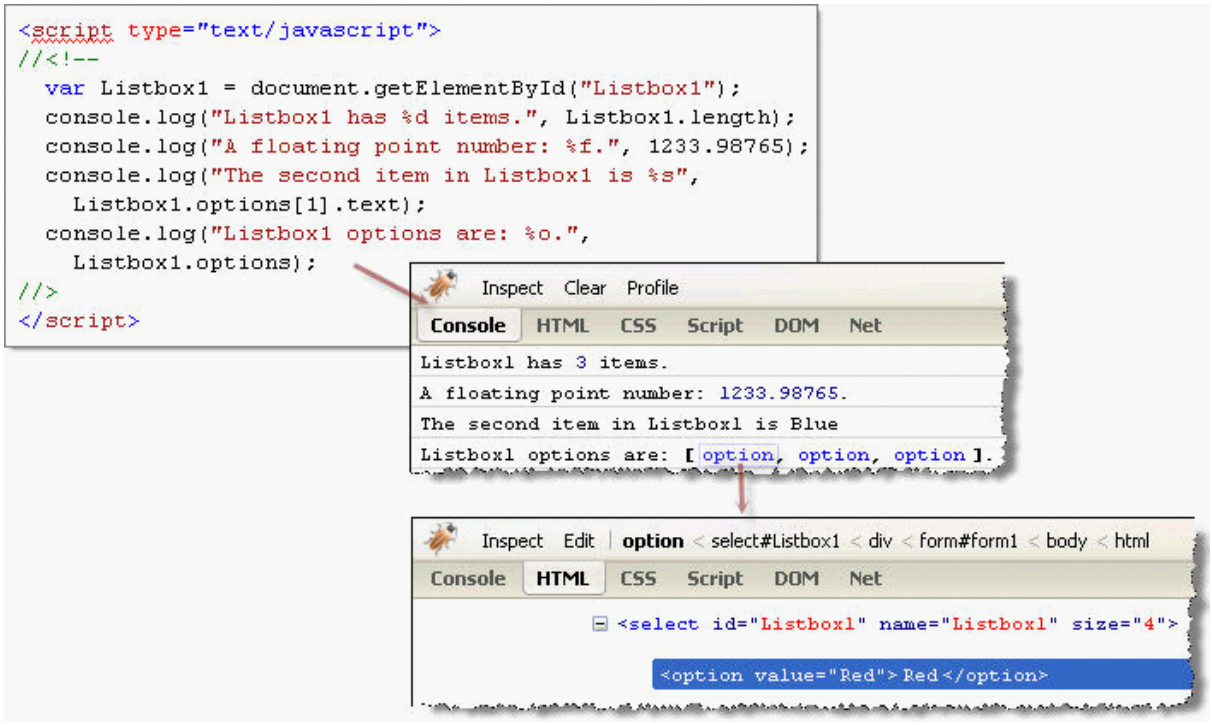


Logging output to the Console

Note: See the current complete list of API possibilities at http://www.getfirebug.com/console.html.

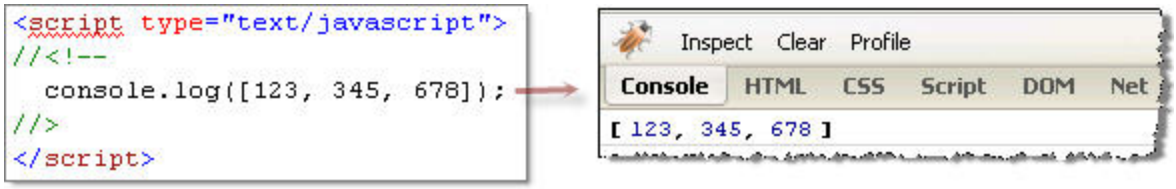
Formatting

Better yet, use the "String.Format" like capabilities of the log function to get the same effect. The log function supports string substitutions for string "%s", integer "%d" or "%i", floating point number "%f" and an object hyperlink "%o". Notice how the numbers and objects are highlighted in the example below. Clicking any one of the object hyperlinks navigates to the HTML for that specific option. You sure can't get that kind of flexibility out of alert()!



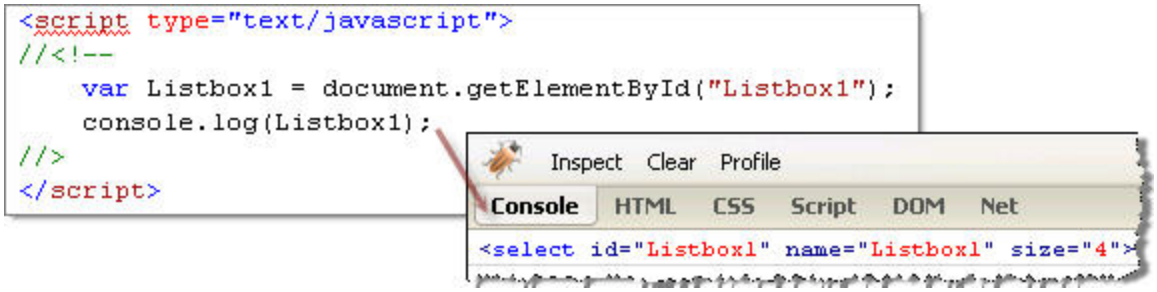
Console log string formatting

Arrays are formatted automatically. The example below declares the array within the logging function, but any array reference will do as well:



Array logged to the console

Likewise you can log an entire HTML element with Firebug doing the work of parsing and formatting for easy readability:

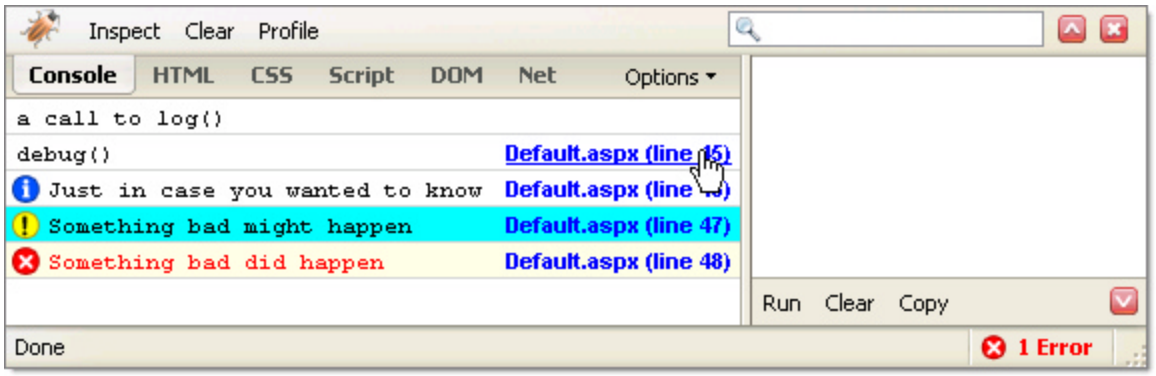


Logging an HTML element to the console

Grouping

Logging levels are established visually through console functions debug(), info(), warn(), and error(). Icons to the left of the make it easy to find important information and hyperlinks to the right left you jump to the Script tab for further analysis. Also note that double clicking the "1 Error" message in the

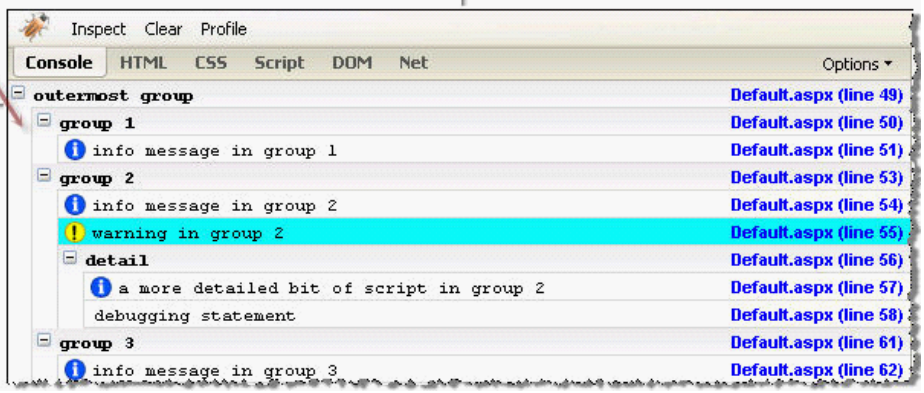
lower right hand corner will navigate to the console.error() call that triggered it.



Formatted logging in the console

Another way that Firebug makes logging more readable is through grouping functions:

```
<script type="text/javascript">
//<!--
  console.group("outermost group");
  console.group("group 1");
  console.info("info message in group 1");
  console.groupEnd();
  console.group("group 2");
  console.info("info message in group 2");
  console.warn("warning in group 2");
  console.group("detail");
  console.info("a more detailed bit of script in group 2");
  console.debug("debugging statement");
  console.groupEnd();
  console.groupEnd();
  console.group("group 3");
  console.info("info message in group 3");
  console.groupEnd();
  console.groupEnd();
//>
</script>
```



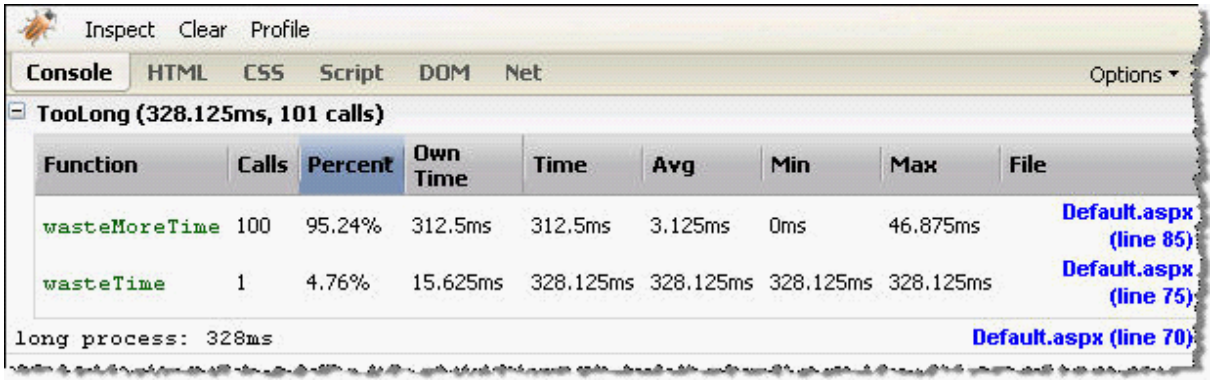
group() and groupEnd() functions in action

Timing and Profiling

What functions are taking the most time? How many times is a particular function called? Firebug provides two functions that pinpoint the bottlenecks: time() and profile(). The script below has a function wasteTime() that calls function wasteMoreTime(). We start profiling using the title "TooLong", start a timer "long process" and then executing the time wasting functions. Be aware that you can nest the timers if you need more detail.

```
<script  type="text/javascript">
//<!--
   console.profile("TooLong");
   console.time("long  process");
   wasteTime();
   console.timeEnd("long  process");
    console.profileEnd();

   function  wasteTime()
  {
      for(var  i=0; i <100; i++)
      {
          var  Listbox1 = document.getElementById("Listbox1");
         Listbox1.options[0].text = i;
         wasteMoreTime();
      }
  }

   function  wasteMoreTime()
  {
      for(var  i=0; i <100; i++)
      {
          var  Listbox1 = document.getElementById("Listbox1");
         Listbox1.options[1].text = i;
      }
  }

//>
</script>
```

This of course is very simplistic, but the output of the profile() function does a very nice job of breaking down the information at a high level and then allowing you to drill into the details of the script to craft a solution. In the example output the number of Calls is listed for each function and the highest percentage time consumers are on top. Notice the "Time" and "Own Time" columns; "Time" is the total time to execute the function while "Own Time" is just the time executing that specific function minus the time used in calling down to other functions.

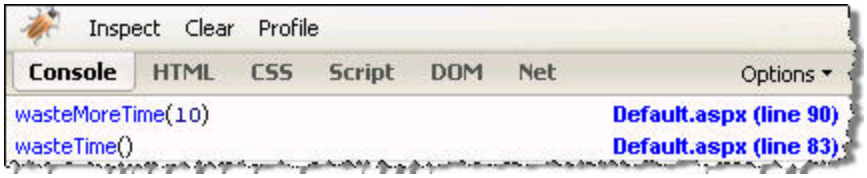


Profiling and timing in the console

The profiler also runs conveniently from the Console menu item. Toggle the Profile button down, use the web page, then toggle the Profile button back up.

Call Stack Tracing

Firebug will print an interactive stack trace of all functions and their parameters at the point you call console.trace(). A hyperlink on the function name will navigate to the Script for that function.



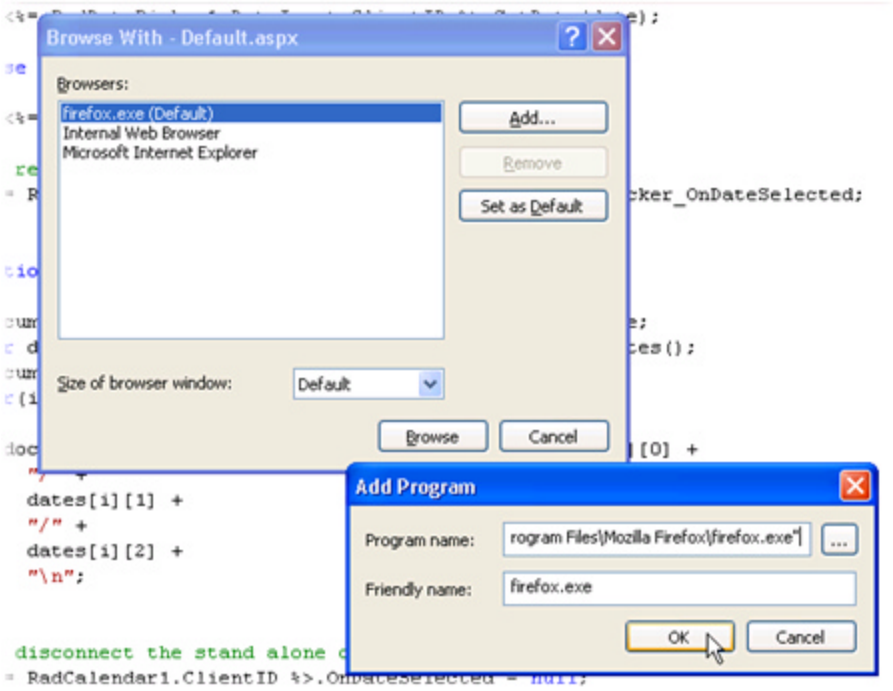
Stack trace output to the console

Lab: Script

The Script tab is the heart of the debugging in Firebug. Here you can set breakpoints, step through JavaScript code and set watch expressions to monitor specific objects.

1. We will use the RadCalendar ClientAPI project as a starting point. Load the project into Visual Studio.
2. In the Solution Explorer right click default.aspx and select the "Browse With" option.
3. This next step assumes you haven't been using Firefox as a browser for Visual Studio, so you will need to add it here.
   - Click the Add button.
   - Use the ellipses button to navigate to the Firefox executable (typically found in the path \Program Files\Mozilla Firefox).
   - Click the Ok button to add the Program.

     Note: You may also want to set Firefox as your default browser so you can click F5 and always have it run.



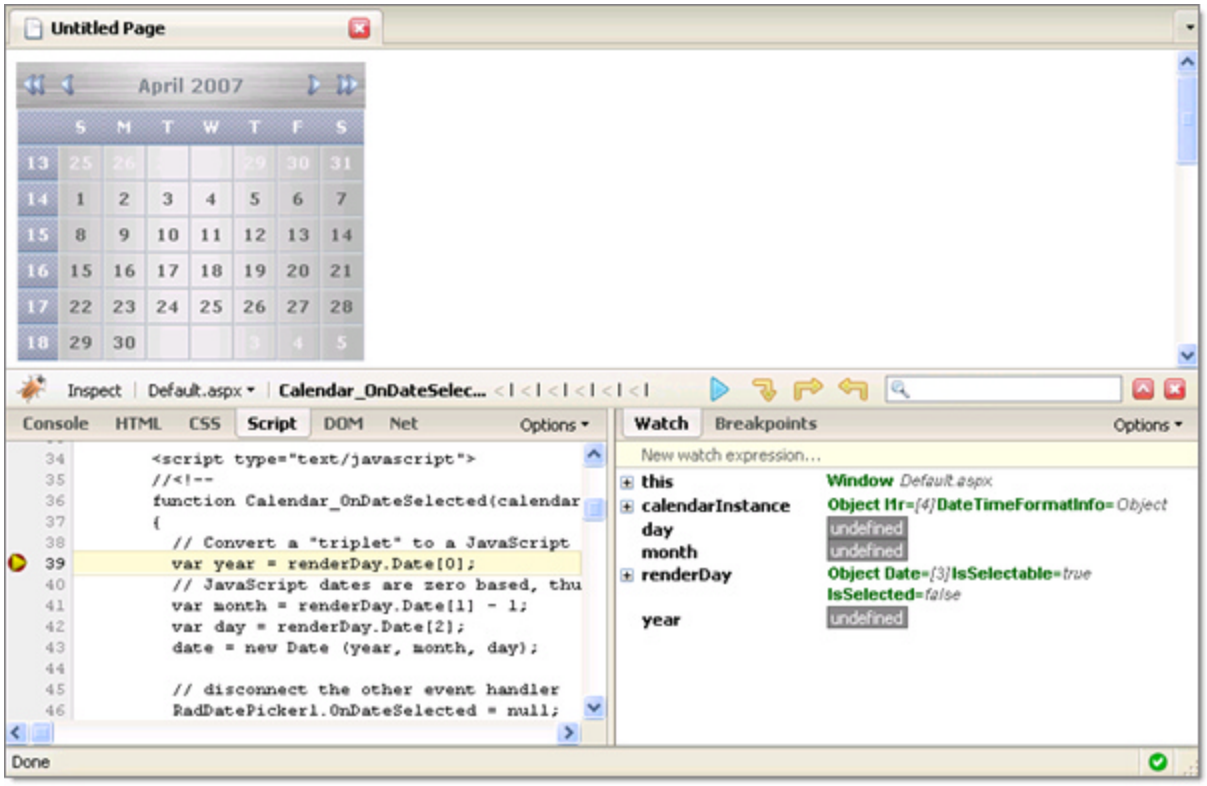
Adding Firefox as a browser for Visual Studio

- Click the Browse button to run Firefox.

4. Open the Firebug debugger (green arrow, lower right of the browser). Click the Script tab on the debugger.
5. There are two OnDateSelected client event handlers in this project, one for the calendar, one for the date picker. We're going to debug into Calendar_OnDateSelected. Set a breakpoint by clicking the gutter area to the left of the line that reads:

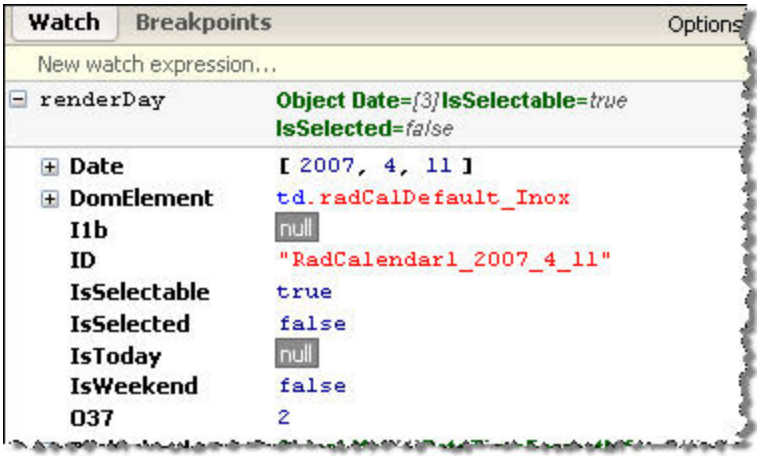
```
var year = renderDay.Date[0];
```

6. Click one of the dates in the calendar to trigger the event. The execution stops at the line with the breakpoint and local variables are already available in the Watch window. Pass the mouse over objects on the Script tab to see their values pop up as hints.



Breakpoint and watches in the Script tab

7. Click "New watch expression" at the top of the Watch window. Enter "renderDay" and hit enter. The entire object is dumped to the Watch window in treeview form with all values formatted for readability.



New expression "renderDay" in the Watch window

8. In the Script window inside the Calendar_OnDateSelected event handler, right click and select "Log Calls to Calendar_OnDateSelected".

9. From here you have the options to step through the code using Continue (F8), Step Over (F10), Step Into (F11), or step out. You can do this from the keyboard, right click for the context menu or using the buttons above the Watch window. Click the F8 key to continue.

10. Navigate to the Console tab and you should see the Calendar_OnDateSelected event has been logged. By clicking on the blue Calendar_OnDateSelected you can navigate back to the Script tab.

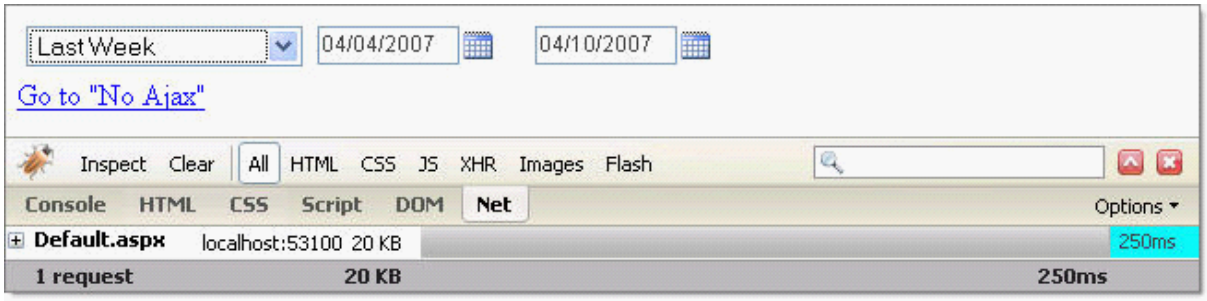
Clicking any of the green parameters navigates to the DOM for the clicked on object.

Calendar_OnDateSelected event logged to the console

Lab: Net

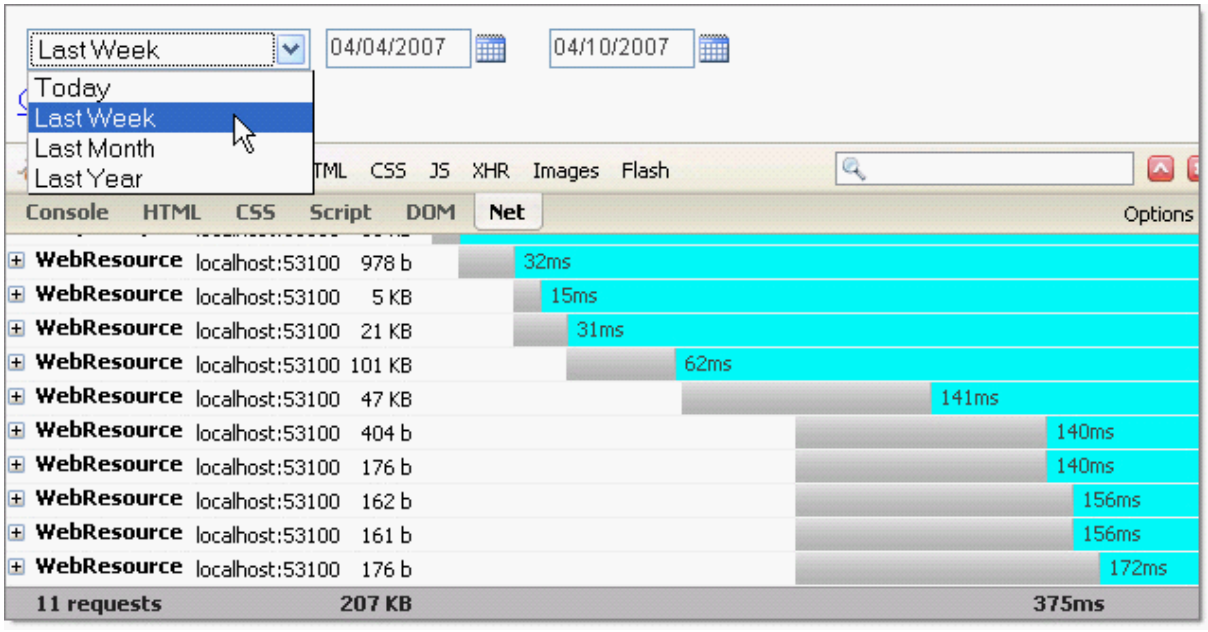
One of the most powerful features of Firebug is the ability to track XmlHttpRequest (XHR) requests and responses. XHRs are fundamental to AJAX and so the ability to dig into XHR requests and responses is so valuable. This lab will use a modified version of the RadAjax chapter AjaxDateRange project. The project simply has a second page without the RadAjaxManager and links have been added to navigate between the two pages for comparison.

1. Load the RadDateRange project from the \Firebug\projects folder.
2. Run the project using the Firefox browser as a vehicle (see the previous Script lab for directions on running Firefox from Visual Studio).
3. Navigate to the Net tab. Above the Net tab are the kind of network traffic you want to view. Click the All button. The component parts of the web page are shown in timeline form where each item loaded relative to the other items on the page, how long each item took to load and the size of the item.
4. Click to expand one of the items. The expanded detail will sometimes show a Params tab if appropriate and will also display tabs for HTML request and response headers. The last tab will show the entire response content.
5. Click the Clear button to remove the existing net logging.
6. Drop down the combo box list and select "Last Week". In the figure below this brings back 34 KB of data.



An AJAX request triggered by the combo click

7. Click the "No Ajax" link. Click the Clear button. Again select "Last Week" from the drop down combo. The difference between 20 KB and 207 KB is startling.



A page refresh triggered by the combo click

3.8.3 Summary

In this chapter we took a tour of Firebug functionality paying particular attention to its CSS capabilities for use in modifying RadControl skins, for help identifying both performance and functionality problems in script debugging, and we saw how the Net tab lets us peak into the world of AJAX requests and responses. Although simply having AJAX present won't automatically make all applications faster, Firebug can help get the best performance from RadControls.

Index

- . -

- < -

- A -

- B -

- C -

- D -

- E -

- F -

- G -

- H -

- I -

- J -

- K -

- L -

- M -

- N -

- O -

- P -

- R -

- S -

- T -

- U -

- V -

- W -

- X -

- Z -

www.telerik.com