telerik
deliver more than expected

From Concept to Complete

Turn Requirements into Applications In Record Time with the RadControls for ASP.NET AJAX

Every application development project starts the same way. It doesn't matter whether you're a single developer scribbling notes on a whiteboard or a major enterprise conducting meeting after meeting with your project stakeholders. Every application starts with requirements. Requirements take all shapes and forms, but ultimately they are your customer's description of what they want software to do.

After requirements are crafted, the real development work begins. The time when you, the developer, must convert the requirements from drawings and documents in to real, working software. And in the "real world," you have a deadline and a limited amount of time you can spend making this conversion happen.

Where do you start? How do you take a set of requirements and translate them in to software that not only meets, but also exceeds your customer's expectations- especially in today's economic climate where budgets are tighter than ever?

In the past, you may have managed to find success building software while doing everything yourself, but survival today requires smart decisions that enable you to operate as efficiently as possible. To continue building software that delivers value and makes a profit for your business and your customer's you must adapt. Two things will help you find success building software today:

1. A reliable and diverse set of tools that enable you build software quickly

2. A reliable partner that can help you fix your tools when you hit problems

The Telerik RadControls for ASP.NET AJAX, the leading UI component suite for ASP.NET, and Telerik OpenAccess ORM, an advanced and easy to use object-relational mapper, give you the tools you need to succeed. But *claiming* that the RadControls and Telerik are the tools you need and *proving* that fact are two different things. So let's take a look at the proof.

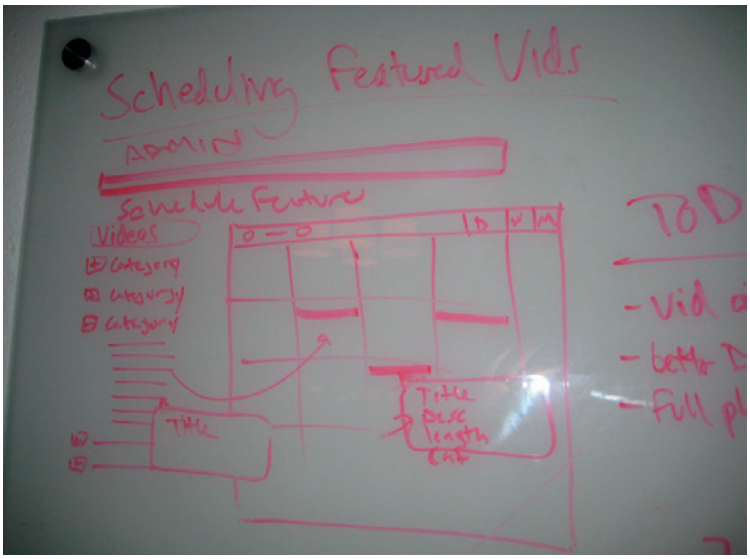
I recently built an entire video portal website using the RadControls for ASP.NET AJAX (Q3 2008), OpenAccess ORM (Q3 2008), and Visual Studio 2008. You can access it now at tv.telerik.com. In this whitepaper, we'll zoom-in on one of the features built for the administration section of the site and see how the RadControls enable developers to go from concept to complete in less than a day.

The Requirements

Telerik TV is a brand new video portal designed to host all videos produced by Telerik. The homepage of the new site features a, well, "featured video" that regularly rotates to highlight new content. The process of scheduling videos for their moment in the spotlight needs a tool so that anyone- even non-technical people- at Telerik can easily select and schedule featured videos in a visual, dynamic manner.

The Concept

With the requirements clear- and deceptively simple-, the next step is to figure out how these requirements should be translated in to a usable tool. Like many developers, I'll turn to my trusty whiteboard and quickly sketch the UX for the new feature:



As you can see, our whiteboard session has produced a very interactive and dynamic user experience for the new admin tool. Here's where a good toolbox like the RadControls for ASP.NET AJAX comes in handy. Even before beginning to write actual code, I know the tools in my toolbox and can use that knowledge to create rich application designs. I can confidently sketch interfaces that include a treeview, a scheduler, and tooltips, and sketch interactions like the ability to drag nodes from my treeview and drop them on my scheduler, all with the confidence things will work cross-browser. Since I know the RadControls all interact very well with each other and they all have very rich client-side APIs, I know I can build this UX with exceptional performance.

So now we have a UX concept that meets the requirements. We even uncovered some functional requirements during our whiteboard session that are going to require some modifications to the database (we need some way to save these scheduled video features). OpenAccess and Forward Mapping are going to make that change really easy, though, and we'll see at the end of this process that we never have to leave Visual Studio or put on our "database" hat to update our schema.

Let's see how we can convert this simple concept to a fully functional and high-performance feature for our application.

The Implementation

The core of this feature's UX is made of three RadControls for ASP.NET AJAX: RadScheduler, RadTreeview, and RadToolTip. I know before I begin coding that performance is very important to this application, so we are going to want to use web services to move data between the AJAX tier and server. Many of the RadControls, including RadTreeview and RadToolTip (and very soon RadScheduler) make it easy to bind directly to web services. For example, to load the list of videos in the RadTreeview from a web service, we can simply write code like this (important properties in bold):

ASPX

```
<telerik:RadTreeView runat="server" ID="radTreeVideos"
                DataValueField="Id"
                DataSourceId="oadsCategories"
                EnableDragAndDrop="true"
                DataTextField="Name"
                PersistLoadOnDemandNodes="false"
                OnClientMouseOver="onClientMouseOver"
                OnClientNodeDropping="nodeDropping"
                OnClientNodeDragStart="nodeDragStart"
                OnClientNodeDropped="nodeDropped"
                OnNodeDataBound="radTreeVideos_NodeDataBound"
```

```
            EnableViewState="false">
        <WebServiceSettings Method="GetTreeVideosByCateogry"
                            Path="~/Services/SampleVideoService.asmx" />
</telerik:RadTreeView>
<telerik:OpenAccessDataSource ID="oadsCategories" runat="server"
            ObjectContextProvider="TelerikTv.ObjectScopeProvider1, TelerikTv"
            TypeName="TelerikTv.Data.Category">
</telerik:OpenAccessDataSource>
```

C# (Code Behind)

```
protected void radTreeVideos_NodeDataBound(object sender,
                                           Telerik.Web.UI.RadTreeNodeEventArgs e){
        if(e.Node.Level == 0)
        {
        e.Node.ExpandMode = TreeNodeExpandMode.WebService;
    }
  }
```

C# (Web Service)

```
[WebMethod]
public IEnumerable GetTreeVideosByCateogry(RadTreeNodeData node,
                                                    IDictionary context){
  var rtrnVids = new List<TreeNodeData>();
  //Get videos from DAL based on category ID
  var videos = new VideosRepository().GetVideosByCategory(node.Value.ToInt());
  if (videos.Count > 0){
    //Create TreeNodes for each video found and add to return collection
    foreach (var video in videos){
      rtrnVids.Add(new TreeNodeData(video.Id, video.Name));
    }
  }

  return rtrnVids;
}
```

That's easy! Granted, there's a little more going on here than "basic" web service binding (this is based on a "real world" app after all). Essentially, we're binding the "top level" categories to our DAL on the initial page load via the OpenAccessDataSource (reducing the requests required to initialize the page) and then using a web service to load all data below the top level nodes. A little optimization trick, but in the end we have a treeview containing all of the videos bound directly to a web service. If we wanted to achieve this same behavior in our RadTreeView, we could optimize our code even further, eliminating the C# "NodeDataBound" code and instead use RadTreeView DataBindings, like this:

ASPX

```
<telerik:RadTreeView runat="server" ID="radTreeVideos"
        DataValueField="Id"
        ...
        EnableViewState="false">
        <DataBindings>
            <telerik:RadTreeNodeBinding Depth="0" ExpandMode="WebService" />
        </DataBindings>
    <WebServiceSettings Method="GetTreeVideosByCateogry"
                Path="~/Services/SampleVideoService.asmx" />
</telerik:RadTreeView>
```

And did you notice the EnableViewState property on the RadTreeView? It's set to *false*. Since all operations on this page will occur asynchronously via web services after the initial load, we have no need for ViewState on the page. That's another hidden performance benefit of using web services to build your applications.

Next-up, we need to enable dynamic tool tips that display a video's details for each video in our treeview. We want to display these tool tips after the user hovers over the title in the treeview for a few seconds. Here, we can use the rich client-side API of the RadToolTip to quickly deliver a solution:

ASPX

```
<telerik:RadToolTipManager Width="250px" Height="75px" ID="radTipMngrTree"
        runat="server"
        RelativeTo="Element"
        AutoTooltipify="false"
        ShowDelay="6000"
        ShowEvent="FromCode"
        Position="MiddleRight">
    <WebServiceSettings Method="GetVideoTipDetailById"
                Path="~/Services/SampleVideoService.asmx" />
</telerik:RadToolTipManager>
```

JAVASCRIPT

```
//Fires when mouse is over a RadTreeView node
function onClientMouseOver(sender, args) {
    //Init tooltipmanager reference (if necessary)
    if (tooltipManager == null)
        tooltipManager = $find("<%= radTipMngrTree.ClientID %>");

    //Hide any open tips
    HideTooltip();
```

```
            //Global variable used to share args with tooltip
        tipArgs = args;

            //Show the tooltip after 3 seconds (if not canceled by OnMouseOut)
        timer = setTimeout("showVideoDetails()", 3000);
}
function showVideoDetails() {
    if (allowTip) {
            //Get the TreeView node from the shared global variable
        var nodeElem = tipArgs.get_node();
            //Don't apply tips to top level nodes
        if (nodeElem.get_level() != 0) {
          var node = nodeElem.get_textElement();

          //If the user hovers the image before the page has
          //loaded, there is no manager created
          if (!tooltipManager) return;

          //Find the tooltip for this element if it has been created
          var tooltip = tooltipManager.getToolTipByElement(node);

          //Create a tooltip if no tooltip exists for such element
          if (!tooltip) {
            tooltip = tooltipManager.createToolTip(node);
                    //Add the video ID to the tooltip context
            tooltip.set_value(nodeElem.get_value());
            tooltip.set_manualClose(true);
            tooltip.show(); //Display the new tip
          }
        }
    }
}
```

C# (Web Service)

```
[WebMethod]
```

```
public string GetVideoTipDetailById(IDictionary context)
{
  //Get video details from DAL based on video Id
  //(supplied in the tooltip context)
  var vid = new VideosRepository().GetVideoById(context["Value"].ToInt());

  if (vid == null)
    return null;

  //Build the tip HTML and return
  var result = new StringBuilder();
  result.Append("<div class='vidTipContent'>");
  ...

  return result.ToString();
}
```

Notice that we're actually creating the tooltips completely client-side! At design-time, we add a RadToolTipManager to our page, and that gives us easy access to a rich JavaScript API that enables us to quickly create and destroy tooltips that get their data automatically from the web service defined in the Manager. Combined with the rich RadTreeView client-side API, we're able to quickly add high performance tool tips that load after a few seconds of mouse hovering.

Finally, we need our big scheduler "calendar" interface. For this feature, we only want to enable people to schedule video dates, not times, so we need to disable some of the default RadScheduler views (specifically, we only want the Month view to be displayed and disable the user's ability to use Week and Day views). That's super simple with the properties exposed by RadScheduler:

ASPX

```
<telerik:RadScheduler runat="server" ID="radScheduleFeatures"
        Width="100%" Height="100%"
        EnableAdvancedForm="false"
        DataSourceID="odsScheduler"
        FirstDayOfWeek="Monday" LastDayOfWeek="Friday"
        ShowAllDayRow="false"
        SelectedView="MonthView"
        ShowViewTabs="false"
        AllowInsert="false"
        DataStartField="StartDateTime" DataEndField="EndDateTime"
        DataSubjectField="Subject" DataKeyField="Id"
        CustomAttributeNames="VideoId"
```

```
        ...>
</telerik:RadScheduler>
```

We also want to provide some "advanced" client-side functionality that will automatically prevent people from scheduling two videos on the same day. And while we're at it, let's make all data operations (such as adding a new video, moving it, or deleting it on the RadScheduler) work with web services. If we had to do this by hand, it could take days of work (at least) to implement all of these client-side events. RadScheduler, of course, makes it a straightforward task with its JavaScript API. For example, here's how we quickly and easily add real-time validation and disable the drag and drop of videos on to the same day (no trips to the server required):

JAVASCRIPT

```
function onClientAptMoveEnd(sender, args) {

  //Determine if target calendar slot is free

  var free = isSlotFree(args.get_targetSlot(), args.get_appointment());


  if (!free) {

    //If not free, cancel the move

    args.get_appointment().get_element().style.border = "0px none black";

    args.set_cancel(true);

  }

  else {

    //If free, update the video feature date

    var apt = args.get_appointment();

    var newTime = args.get_newStartTime();

    var featureId = apt.get_id();


    //Call webservice to update apt

    WebServiceReference.UpdateVideoFeature(featureId, newTime,

                                                    onServiceComplete);

  }

}


function isSlotFree(targetSlot, appointment) {

  //Get start time of target calendar slot

  var startTime = targetSlot.get_startTime();


  //Ensure reference to RadScheduler client object

  if(scheduler == null)

    scheduler = $find('<%= radScheduleFeatures.ClientID %>');
```

```
    //Calculate end time (to create span that cannot be overlapped)
    var endTime = new Date(startTime.getTime() + 86400000); //24hr period
    //Determine if appt being moved overlaps with another appointment
    return overlapsWithAnotherAppointment(appointment, startTime, endTime);
}
function overlapsWithAnotherAppointment(appointment, startTime, endTime) {
    //Ensure client ref to RadScheduler object
    if (scheduler == null)
      scheduler = $find('<%= radScheduleFeatures.ClientID %>');
    //Get all appts in range of start and end time
    var appointments =
            scheduler.get_appointments().getAppointmentsInRange(startTime, endTime);
    //If no appts found, slot is free
    if (appointments.get_count() == 0)
      return true;
   //If 1 appt found, make sure it's not the current appt being moved
    if (appointment && appointments.get_count() == 1 &&
                        appointments.getAppointment(0) == appointment)
      return true;
    //Otherwise, slot is not free
    return false;
}
```

And with that, our UX is done! Now the database. We said we needed to update our database so that we can save these scheduled features in our persistence layer. Since we're using OpenAccess ORM to power this site's data operations that means a couple of things:

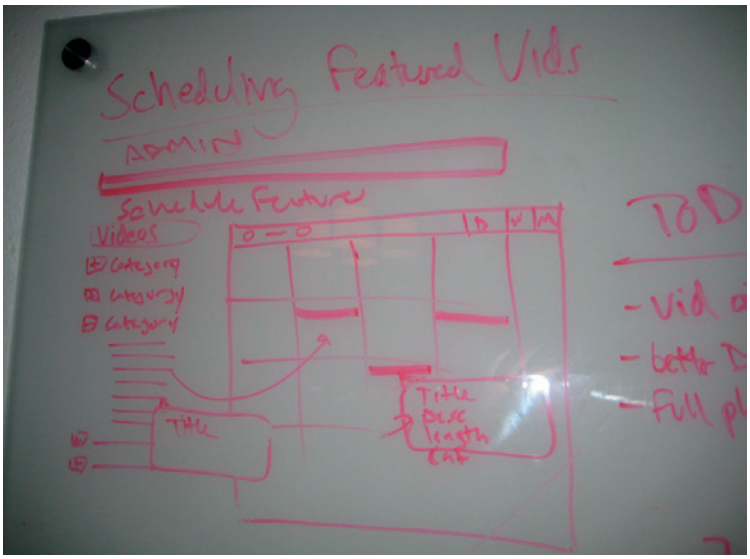
1. We can simply create a new C# class that defines the new object needed to schedule video features and OpenAccess will *automatically* update the database schema. That's definitely a huge time saver since we don't have to switch "developer modes" and start thinking like a database guy.

2. We don't have to write any data "plumbing code." OpenAccess ORM handles all of that. We just define the object, and then changes to instances of the object in the application will be persisted to the database automatically. It's not "magic;" it's just a super-smart ORM doing the ADO.NET work for us so we can focus on our application's functionality.

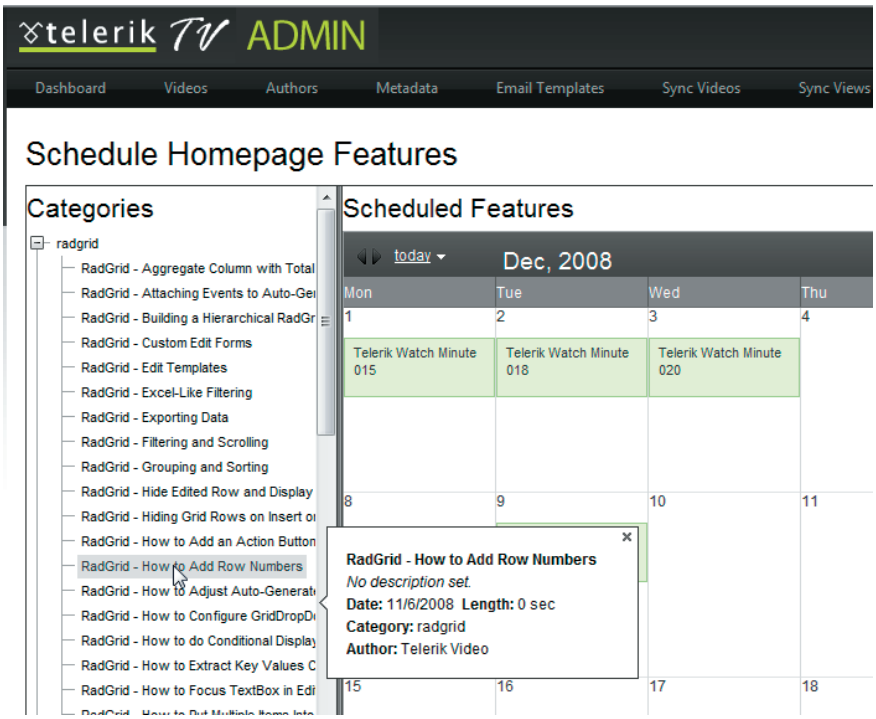
So with the quick creation of a C# class and a rebuild of the DAL (that contains our C# objects, or "model"), I'm ready to persist video features in the database. We never have to open SQL Management Studio or write a line of SQL. Think about how long would traditionally take you to switch gears, create a new table, then build your business object, then write plumbing code for your object, and *then* get back to using your object for business logic? OpenAccess ORM enables you to do what you do well- develop code- and in the process saves you lots of time (and code).

The Result

In less than a day, we were able to go from this whiteboard sketch:



To this final product:



And that's one day, by one person, from start to finish, without ever leaving Visual Studio. Thanks to the skins provided by the RadControls, and the database "wizardry" provided by OpenAccess ORM, you never need to open a graphics program or a database management program to build great features like this.

With the RadControls for ASP.NET AJAX and Telerik OpenAccess ORM in your toolbox, your creativity as a developer is unlocked. Suddenly, you don't have to worry about satisfying your users' desires for rich application interfaces; instead, you'll probably find yourself *suggesting* to users ways to make applications *more* dynamic. And you'll do it while being more productive than ever before. Your customers will think you're a hero for delivering great software on a tight budget, and that's a serious competitive advantage in these economic times.

Ultimately, of course, the best proof is experience. In the words of the great Levar Burton, "you don't have to take my word for it." Hopefully the journey documented here will open your mind to a new a way of innovatively crafting great software that you can apply to your own projects. I encourage you to download the free trials of the RadControls and try this experiment for yourself. Then you will truly see how Telerik enables you to go from concept to complete in record time.

Download your free trial of the RadControls for ASP.NET AJAX today