Turbo Charge your Search Experience with dtSearch and Telerik UI for ASP.NET

Jeffrey T. Fritz, 8 May 2014

In this article, I'm going to add the Telerik UI for ASP.NET to my previous Faceted Search with dtSearch article and do some refactoring to make my search page look better and easier to use.

Editorial Note

This article is in the Product Showcase section for our sponsors at CodeProject. These articles are intended to provide you with information on products and services that we consider useful and of value to developers.

Part 1: Faceted Search with dtSearch (using SQL and .NET)
Part 2: Turbo Charge your Search Experience with dtSearch and Telerik UI for ASP.NET

Related Article: A Search Engine in Your Pocket -- Introducing dtSearch on Android

Introduction

I previously wrote an article about how to get started using Faceted Search functionality with the dtSearch library. The code was a bit complex, but did achieve my goal of not querying my SQL database for every search request. I want to spruce things up a bit, and make the code a bit simpler. In this article, I'm going to add the Telerik UI for ASP.NET and do some refactoring to make my search page look better and easier to use.

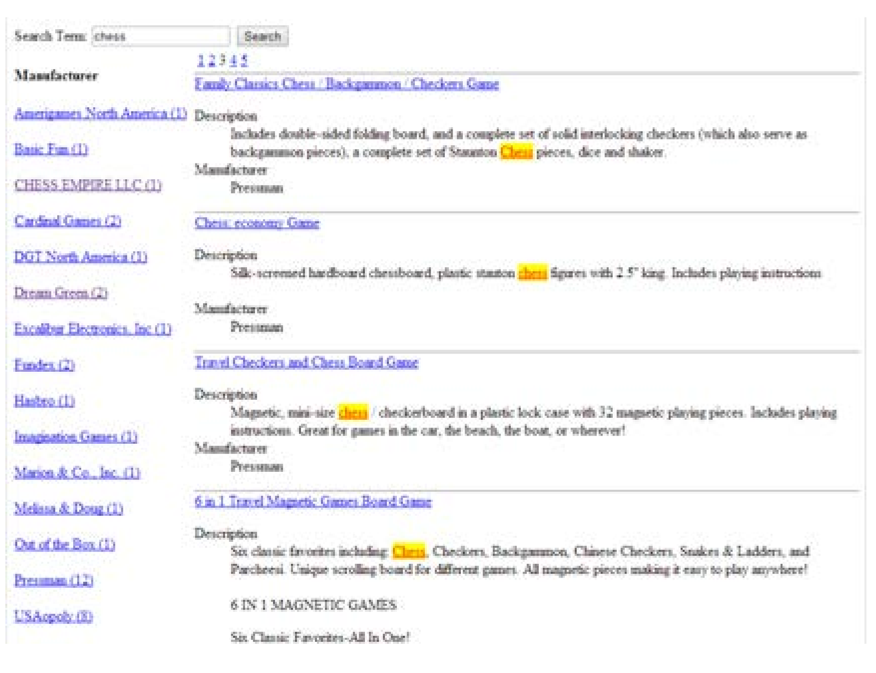

Figure 1 - Existing state of our search page

Layout – Let's make this easier to see

The first problem that I want to address, is making the left column of facets in my search results less fixed to a certain width. That width is assuming a size of the browser requesting the content. To make this a bit more flexible, I'm going to wrap the entire page in a Telerik RadSplitter with RadPanes. With two horizontal panes, this will give a bit of a framed effect, with the content in the left column being allowed to grow or hide behind scrollbars based on when the split bar resides. The markup to create this layout starts with the following:

```
<telerik:RadSplitter runat="server" Orientation="Vertical" Width="100%" Height="600">
  <telerik:RadPane runat="server" Width="150" MinWidth="150" MaxWidth="300">
    <!-- FACET SEARCH BAR GOES HERE -->
  </telerik:RadPane>
  <telerik:RadSplitBar runat="server"></telerik:RadSplitBar>
  <telerik:RadPane runat="server">
    <!-- SEARCH RESULTS GO HERE -->
  </telerik:RadPane>
</telerik:RadSplitter>
```

Listing 1 - Layout of the RadSplitter and Panes

The RadSplitter will contain two panes, split vertically courtesy of the Orientation attribute. The panes are presented left to right in the order they appear in the markup. The left pane has a minimum width of 150 pixels, and a maximum width of 300 pixels. A RadSplitBar control is next; this is the bar that the user can drag to resize the two panes. The final pane will fill the remaining space. No additional C# code is required to make this layout function in the browser.

Search Results Grid – Now with Sorting!

While my previous presentation used a grid and presented the information about my product search to the user in a clean format, it would be nice to show some other features of the products and allow sorting of the results. Fortunately, dtSearch and the Telerik RadGrid both have sorting functionality. Let's display our search results in the RadGrid and connect its sort function to our dtSearch sort handler.

I've completely 'tricked out' my RadGrid with the following markup:

```
<telerik:RadGrid runat="server" ID="grid" PageSize="10" Height="100%"
  OnPageIndexChanged="grid_PageIndexChanged"
  OnPageSizeChanged="grid_PageSizeChanged"
  OnSortCommand="grid_SortCommand"
  OnItemDataBound="grid_ItemDataBound"
  AllowPaging="true" AllowCustomPaging="true"
  AllowSorting="true"
  EnableHeaderContextFilterMenu="false">
  <MasterTableView AutoGenerateColumns="false"
    PagerStyle-AlwaysVisible="true"
    AlternatingItemStyle-BackColor="LightBlue">
    <Columns>
      <telerik:GridBoundColumn DataField="ProductNum" HeaderText="Product ID" />
      <telerik:GridBoundColumn DataField="Name" HeaderText="Name" />
      <telerik:GridBoundColumn DataField="Manufacturer" HeaderText="Manufacturer" />
      <telerik:GridBoundColumn DataField="Age" HeaderText="Recommended Age" />
      <telerik:GridBoundColumn DataField="NumPlayers" HeaderText="# Players" />
      <telerik:GridBoundColumn DataField="Price" HeaderText="Price"
        DataFormatString="{0:$0.00}" ItemStyle-HorizontalAlign="Right" />
    </Columns>
  </MasterTableView>
  <ClientSettings Resizing-AllowColumnResize="true">
  </ClientSettings>
</telerik:RadGrid>
```

Listing 2 - RadGrid Markup, Fully 'Tricked Out'

Let's review this markup; there's a lot going on:

- Event handlers are defined for `PageIndexChanged`, `PageSizeChanged`, `SortCommand`, and `ItemDataBound`.
- Sorting, Paging, and CustomPaging are activated. CustomPaging is particularly necessary in order to drive the custom data paging operation that is required by the dtSearch SearchJob object. Normally, the Telerik RadGrid will handle sort and page operations automatically when an `IQueryable` collection is passed to it. We'll look at how to implement those handlers next.
- The columns are defined with appropriate formatting, and `AutoGenerateColumns` disabled. This prevents some extra properties of our `ProductSearchResult` from being converted into columns. I could have defined a `SortExpression` value on these columns to define how the server-side should sort each column. Without this attribute, the sort field defaults to the name of the field bound to the column.
- The `ClientSettings` – Resizing - `AllowColumnResize` is set to true to allow the end-user to be able to resize columns as they desire.

I have refactored the `DoSearch` method from my previous article to have the following signature, where it will return a collection of `ProductSearchResult` objects:

```
public IEnumerable<ProductSearchResult> DoSearch(
  string searchTerm,
  int pageNum = 0,
  int pageSize = 10,
  string sortFieldName = "")
{
```

Listing 3 - New DoSearch method signatiure

This allows me to connect my `grid_SortCommand` method to pass along to the `DoSearch` method the sort information the grid is submitting to the server. The syntax for the paging operations are very similar, delegating paging operations back to the `DoSearch` method:

```
protected void grid_SortCommand(object sender, GridSortCommandEventArgs e)
{
  var sortExpression = string.Concat(e.SortExpression, " ", e.NewSortOrder == GridSortOrder.Ascending ? "ASC"
: "DESC");

  grid.DataSource = DoSearch(searchBox.Text.Trim(), 0,
    grid.PageSize, sortExpression);
  grid.CurrentPageIndex = 0;
  grid.DataBind();
}
  protected void grid_PageIndexChanged(object sender, GridPageChangedEventArgs e)
  {
    grid.DataSource = DoSearch(searchBox.Text.Trim(), e.NewPageIndex,
      grid.PageSize, grid.MasterTableView.SortExpressions.GetSortString());
    grid.CurrentPageIndex = e.NewPageIndex;
    grid.DataBind();
  }
```

Listing 4 - Sort and Paging Handlers

The sort operation in the dtSearch `SearchJob` could not be any more straightforward. `SearchJob.Results` has a Sort method that will accept the same name of the field that was stored to sort on. I added an extra bit to handle the `NewSortOrder` property being passed in to the method so that it can be passed along like a normal sort expression to the `DoSearch` method. The snippet added to my `DoSearch` method looks like:

```
// Handle sort requests from the grid
if (sortFieldName == string.Empty)
{
  searchJob.Results.Sort(SortFlags.dtsSortByRelevanceScore, "Name");
}
else
{
  var sortDirection = sortFieldName.ToUpperInvariant().Contains("DESC") ? SortFlags.dtsSortDescending :
SortFlags.dtsSortAscending;
  sortFieldName = sortFieldName.Split(' ')[0];
  sj.Results.Sort(SortFlags.dtsSortByField | sortDirection, sortFieldName);
}
```

Listing 5 - Sorting mechanism with SearchJob

The initial sort mechanism is to sort based on the relevance of the search result. For a field to be sorted against, that field's name is passed in to the `Sort` method and the `SortFlags` are set appropriately for ascending or descending order. The only catch in this sort operation with dtSearch is that the search results are fetched first and then sorted. Consequently, I need to raise the limit on the `SearchJob.MaxFilesToRetrieve` to something astronomical so that I am sure all of my results are fetched in one batch before the sort operation is applied.

To complete the grid of product data, I modified my paging operation in the `DoSearch` method to now fetch each of the fields in the product from the search results. The full syntax of `DoSearch` is below:

```
public IEnumerable<ProductSearchResult> DoSearch(
  string searchTerm,
  int pageNum = 0,
  int pageSize = 10,
  string sortFieldName = "")
{

  var sj = new SearchJob();
  sj.IndexesToSearch.Add(SearchIndexer._SearchIndex);
  sj.MaxFilesToRetrieve = 2000;
  sj.WantResultsAsFilter = true;
  sj.Request = searchTerm;

  // Add filter condition if necessary
  if (!string.IsNullOrEmpty(FacetFilter) && FacetFilter.Contains('='))
  {
    var filter = FacetFilter.Split('=');
    sj.BooleanConditions = string.Format("{0} contains {1}", filter[0], filter[1]);
  }

  // Prevent the code from running endlessly
  sj.AutoStopLimit = 1000;
  sj.TimeoutSeconds = 10;
  sj.Execute();

  this.TotalHitCount = sj.FileCount;
  ExtractFacets(sj);

  // Handle sort requests from the grid
  if (sortFieldName == string.Empty)
  {
    sj.Results.Sort(SortFlags.dtsSortByRelevanceScore, "Name");
  }
  else
  {
    var sortDirection = sortFieldName.ToUpperInvariant().Contains("DESC") ? SortFlags.dtsSortDescending :
SortFlags.dtsSortAscending;
    sortFieldName = sortFieldName.Split(' ')[0];
    sj.Results.Sort(SortFlags.dtsSortByField | sortDirection, sortFieldName);
  }

  this.SearchResults = sj.Results;

  // Manual Paging
  var firstItem = pageSize * pageNum;
  var lastItem = firstItem + pageSize;
  lastItem = (lastItem > this.SearchResults.Count) ? this.SearchResults.Count : lastItem;
  var outList = new List<ProductSearchResult>();
  for (int i = firstItem; i < lastItem; i++)
  {
    this.SearchResults.GetNthDoc(i);
    outList.Add(new ProductSearchResult
    {
      DocPosition = i,
      ProductNum = this.SearchResults.DocName,
      Name = this.SearchResults.get_DocDetailItem("Name"),
      Manufacturer = this.SearchResults.get_DocDetailItem("Manufacturer"),
      Age = this.SearchResults.get_DocDetailItem("Age"),
      NumPlayers = this.SearchResults.get_DocDetailItem("NumPlayers"),
      Price = decimal.Parse(this.SearchResults.get_DocDetailItem("Price"))
    });
  }

  return outList;

}
```

Listing 6 - Full listing of DoSearch method

First, you'll see that I am filtering using a property called `FacetFilter`. This is a string value that is being passed back into my page from my facet list on the left side. I have updated this column to use a Telerik RadPanelBar. The markup for this is easy, and defines the three facets I want to be able to filter on:

```
<telerik:RadPanelBar runat="server" ID="facets"
  Width="100%" OnItemClick="facets_ItemClick">
  <Items>
    <telerik:RadPanelItem runat="server" Text="Manufacturer"
       Value="Manufacturer"></telerik:RadPanelItem>
    <telerik:RadPanelItem runat="server" Text="Age"
       Value="Age"></telerik:RadPanelItem>
    <telerik:RadPanelItem runat="server" Text="# Players"
       Value="NumPlayers"></telerik:RadPanelItem>
  </Items>
</telerik:RadPanelBar>
```

Listing 7 - Markup to support the facet PanelBar

This control will give an accordion look and feel to the facets displayed on the page. The data for each of the facets being presented is updated in the `ExtractFacets` method:

```
private void ExtractFacets(SearchJob sj)
{

  var filter = sj.ResultsAsFilter;

  var facetsToSearch = new[] { "Manufacturer", "Age", "NumPlayers" };

  // Configure the WordListBuilder to identify our facets
  var wlb = new WordListBuilder();
  wlb.OpenIndex(Server.MapPath("~/SearchIndex"));
  wlb.SetFilter(filter);

  // For each facet or field
  for (var facetCounter = 0; facetCounter < facetsToSearch.Length; facetCounter++)
  {

    // Identify the header for the facet
    var fieldValueCount = wlb.ListFieldValues(facetsToSearch[facetCounter], "", int.MaxValue);
    var thisPanelItem = facets.Items.FindItemByValue(facetsToSearch[facetCounter]);
    thisPanelItem.Items.Clear();

    // For each matching value in the field
    for (var fieldValueCounter = 0; fieldValueCounter < fieldValueCount; fieldValueCounter++)
    {

      string thisWord = wlb.GetNthWord(fieldValueCounter);

      if (string.IsNullOrEmpty(thisWord) || thisWord == "-") continue;

      var label = string.Format("{0}: ({1})", thisWord, wlb.GetNthWordCount(fieldValueCounter));

      var filterValue = string.Format("{0}={1}", facetsToSearch[facetCounter], thisWord);
      thisPanelItem.Items.Add(new RadPanelItem(label) { Value = filterValue });

    }

  }
```

Listing 8 - Updated ExtracFacets method

You'll see this time, instead of writing a header and line items into the panel, this code is finding the facet header and adding panel items inside of that header's content area. The value of the panel item is defined as a name-value pair so that I can construct the appropriate filter criteria in the search method. The onclick handler for these items just triggers another search operation with the FacetFilter set to the value of the Panel Item submitted.

```
protected void facets_ItemClick(object sender, RadPanelBarEventArgs e)
{
  SearchResults = null;
  FacetFilter = e.Item.Value;
  grid.DataSource = DoSearch(searchBox.Text.Trim());
  grid.CurrentPageIndex = 0;
  grid.MasterTableView.VirtualItemCount = TotalHitCount;
  grid.DataBind();
}
```

Listing 9 - Facet Item Click operation

Highlighting Results, with Style!

You will notice this time I'm not returning the highlighted results with the product data in the grid. I've updated my page to present that code in a handy hover tooltip using the Telerik TooltipManager control. This control allows me to define a server-side method that will fetch and return the appropriate HTML to display in the tooltip.

The markup for the tooltip is represented by this syntax for the ToolTipManager:

```
<telerik:RadToolTipManager runat="server" ID="tipMgr" OnAjaxUpdate="tipMgr_AjaxUpdate" RelativeTo="Element"
  width="400px" Height="150px" RenderInPageRoot="true">
</telerik:RadToolTipManager>
```

Listing 10 - Markup for the ToolTipManager

The `RenderInPageRoot` value means that the panel that is used to house the tooltip will be rendered on the root of the HTML on the page, and not inside of another element. This is useful for this sample, as we will be showing the tooltip relative to a grid row.

I define the relationship between the ToolTipManager and the grid through the `ItemDataBound` event handler on the grid. I add the grid row to the tip manager's collection of target controls with this code:

```
protected void grid_ItemDataBound(object sender, GridItemEventArgs e)
{
  var thisRow = e.Item;
  if (thisRow.ItemType == GridItemType.Item || thisRow.ItemType == GridItemType.AlternatingItem)
  {
    var dataItem = thisRow.DataItem as ProductSearchResult;
    tipMgr.TargetControls.Add(thisRow.ClientID, dataItem.DocPosition.ToString(), true);
  }
}
```

Listing 11 - Adding GridRows to the collection of controls managed by the ToolTipManager

Finally, to make our highlighted search results appear in the tooltip, I implemented the `tipMgr_AjaxUpdate` event handler to add the highlighted results to the requesting HTML panel hosted by the tooltip. Since I wrote my HighlightResult method statically in the previous article, I can reuse that method to fetch and add the HTML to the div:

```
protected void tipMgr_AjaxUpdate(object sender, ToolTipUpdateEventArgs e)
{

  var newDiv = new HtmlGenericControl("div");
  newDiv.InnerHtml = FacetedSearch.HighlightResult(this.SearchResults, int.Parse(e.Value));
  e.UpdatePanel.ContentTemplateContainer.Controls.Add(newDiv);
}
```

Listing 12 - Syntax to add highlighted results to the Tooltip

Results

The tooltip and sorting options added make the results grid a great deal easier and friendlier to use. As a consumer, I want to be able to search for products and then sort by price or some other field that is important to me. The Telerik controls make the screen much easier to read, and the results stand out with no additional coding or design work.

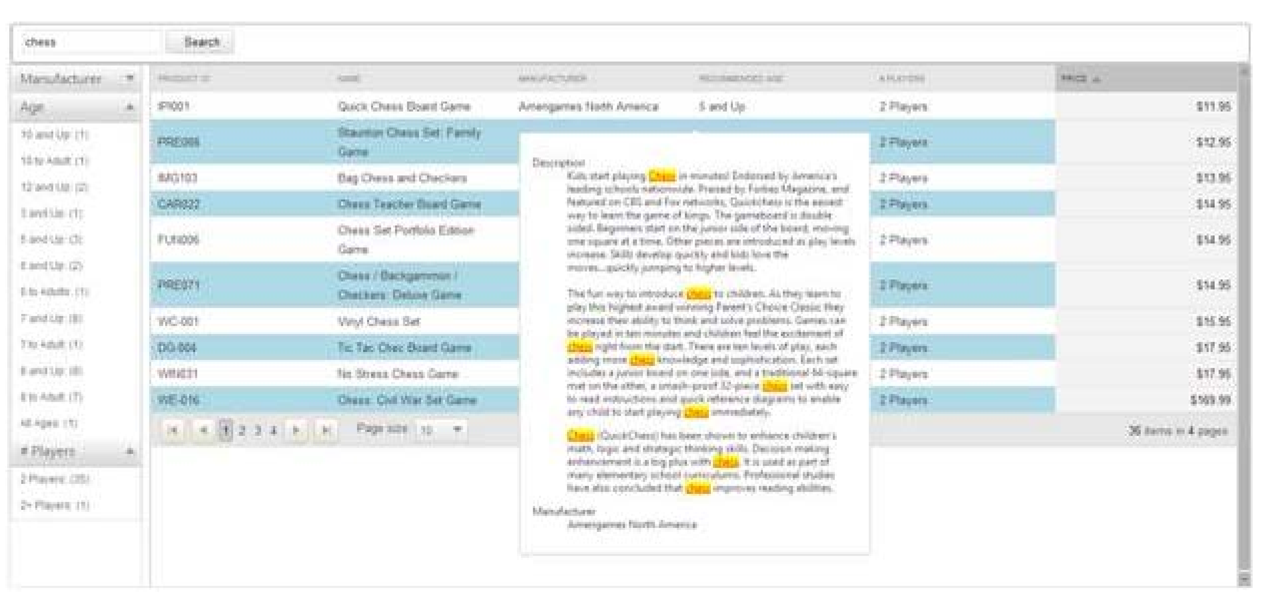

Figure 2 - Search Results using Telerik UI for ASP.NET

Summary

dtSearch provides a wealth of capabilities for an Enterprise grade search library. Coupled with a first class user-interface tool like the Telerik UI for ASP.NET, amazing results can be delivered with a little bit of integration work. Offload the work to search and present your enterprise data to dtSearch. Download a developer trial copy at www.dtsearch.com and get your trial of the Telerik UI for ASP.NET at Telerik UI for ASP.NET

License

Share

About the Author

Jeffrey T. Fritz
Program Manager
United States

Jeffrey is a software developer coach, architect, and speaker in the Microsoft.Net community. He currently works as a program manager for the Microsoft .NET Developer Outreach group. He has delivered training videos on Pluralsight, WintellectNow, and on YouTube. Jeffrey makes regular appearances delivering keynotes, workshops, and breakout sessions at conferences such as TechEd, Ignite, DevIntersection, CodeStock, FalafelCon, VSLive as well as user group meetings in an effort to grow the next generation of software developers.

Comments and Discussions