

The Boca Raton DotNetNuke Meetup Group

Meet other local users of DotNetNuke open source portal applications. We'll focus on module development as well as skinning.

Discuss DNN, learn new tricks and tips along with being in a social networking environment and have a place to share your ideas as well as your challenges.



February 22 7:00 PM

Section 1.01 Using Object DataSource with DNN in a Simple Application

In our next meeting, we will discuss a recent project that we did for a client of ours that gave me the opportunity to use the new Object DataSource from Microsoft in an application.

I found there were some useful tips and some issues that could drive you crazy with both. In this session I'll provide the issues that I found were helpful with building this application and some tips to help you avoid the same issues.

The format for this meeting will be very open and anyone can attend whether you are a developer or would like to see what is involved with writing a module for DotNetNuke. Bring all your questions and we'll see if we can have some fun with them.

If you have been using DNN for a while, or are just getting started with the Portal below are some resources that I've found helpful.

What is the Object DataSource and why do I care about it?

ObjectDataSource belongs to the family of data source controls in ASP.NET, which enables a declarative databinding model against a variety of underlying data stores, such as SQL databases or XML. Most data source controls encourage a two-tiered application architecture, where the presentation layer (the page) interacts directly with the backend data provider. However, it is also common for page developers to encapsulate data retrieval (and optionally business logic) into a component object, introducing an additional layer between the presentation page and data provider. The ObjectDataSource control allows developers to structure their applications using this traditional three-tiered architecture and still take advantage of the ease-of-use benefits of the declarative databinding model in ASP.NET.

The ObjectDataSource control object model is similar to the SqlDataSource control. Instead of a ConnectionString property, ObjectDataSource exposes a **TypeName** property that specifies an object type (class name) to instantiate for performing data operations. Similar to the command properties of SqlDataSource, the ObjectDataSource control supports properties such as **SelectMethod**, **UpdateMethod**, **InsertMethod**, and **DeleteMethod** for specifying methods of the associated type to call to perform these data operations. This section describes techniques for building data access layer and business logic layer components and exposing them through an ObjectDataSource control.

An example of what the Code might look like in our library:

```
public class MyDataLayer {  
  
    public DataView GetRecords();  
    public DataView GetRecordsByCategory(String categoryName);  
    public DataView GetRecordByID(int recordID);  
  
    public int UpdateRecord(int recordID, String recordData);  
    public int DeleteRecord(int recordID);  
}
```

```
public int InsertRecord(int recordID, String recordData);  
}
```

The **ObjectDataSource** can be used in our pages in the following manner:

```
<asp:ObjectDataSource TypeName="MyDataLayer" SelectMethod="GetRecords" UpdateMethod="UpdateRecord"  
DeleteMethod="DeleteRecord" InsertMethod="InsertRecord" runat="server"/>
```

The **ObjectDataSource** can be used in our pages in the following manner if we are using parameters such as **QueryString**:

```
<asp:ObjectDataSource TypeName="MyDataLayer" SelectMethod="GetRecords" UpdateMethod="UpdateRecord"  
DeleteMethod="DeleteRecord" InsertMethod="InsertRecord" runat="server">  
<SelectParameters>  
  <asp:QueryStringParameter Name="albumID" QueryStringField="id" Type="Int32"/>  
</SelectParameters>  
</asp:ObjectDataSource>
```

On a really high level, you create your code which will get the records or do some processing out of your database or XML files or wherever you need to go.

Our page contains many controls that bind directly to the **ObjectDataSource**. Once you place an **ObjectDataSource** control on your page, you just need to tell it what methods in **YOUR** library to use for the Inserts, Updates, Deletes and Selects. You are not required to use all these. I find that sometimes I'm just using the Select or the Select and the Delete.

Your page can have a **GridView** on it, one of the new 2.0 controls built into the .net framework and it comes with sorting and paging out of the box when the **SelectMethod** returns a **DataSet**, **DataView**, or **DataTable** object. Custom sorting is also supported. You can just point this **ObjectDataSource** at your code and it's wired up for you. This is a real big time savings form doing all the work yourself. This doesn't mean you can use it for every situation.

- ✓ **Allows flexible n-tier architecture**
- ✓ **Easy to use & a Time Saver**
- ✓ **Adding complex business logic is easy, protect the data easily by forcing it to go through your business layer**
- ✓ **Works with your custom code very well, big plus with DNN**

What's the difference between the SqlDataSource and the Object DataSource ... and again, why do I care about it?

SqlDataSource is an object that works like **ObjectDataSource**, but is linked directly to a MS SQL Server database or a provider that you have setup in your machine.config file. You can set the provider up to any valid provider factory as long as you change this in your connection string and the properties of your **SelectCommand**. We do not need to write any code like the ObjectDataSource to use this object. We can simply put this on our page, use the property wizard to connect this to our database and set the sql up right there. Sounds much simpler and better right? For some projects this might be the ideal approach. It's faster and easier to setup for sure.

Our ObjectDataSource we needed to create our business layer that would connect to our data layer that is actually doing the work. In this model, just drop it on your page, use the properties wizard and connect wire it up to a GridView and your done!

I've found using this object very limited and your are stuck with basically have no business rules in your code and having to stick them into your stored procedures if possible or worse yet going directly to the database from your GUI with very little to no business rules.

The SqlDataSource can be used in our pages in the following manner:

```
<asp:SqlDataSource ID="SqlDataSource1" runat="server"
  ConnectionString="<%%$ ConnectionStrings:Pubs %>"
  SelectCommand="SELECT [au_id], [au_lname], [au_fname], [state] FROM [authors]"
  UpdateCommand="UPDATE [authors] SET [au_lname] = @au_lname, [au_fname] = @au_fname, [state] = @state WHERE [au_id] = @au_id"
  DeleteCommand="DELETE FROM [authors] WHERE [au_id] = @au_id"/>
```

- ✓ Very easy to use
- ✓ Great for smaller projects without any complex business logic
- ✓ Get data on your pages quickly without writing a lot of code

Ok, you've convinced me this is a cool thing, now what do I need to do and why am I not using the SqlDataSource again?

As many projects are unique, most or all have a lot of the same issues and needs. You are going to have to have data stored somewhere. Someone will inevitably wish to view this data. They are not only going to want to view this, but might even want to add or update it!

Wow, quit a lot of stuff these pesky clients need isn't it. Using common tools and code reuse can save you a lot of time. If you able to spend a majority of your time working on how the program should work and design you should have better output. Working with an object like ObjectDataSource should allow you to spend more time on the areas of the project you might need to without sacrificing flexibility.

How does this compare to how we did the before we started using asp.net 2.0 objects?

Before we started using objects like the **GridView** and **ObjectDataSource** we would often have a code base or library using an n-tier architecture with BLL/DAL layers. The DAL or Data Access Layer would be the code that is actually written against our data source. This layer is normally called by our BLL or Business Logic Layer and can have complicated business logic contained within it.

Our GUI in many cases web pages would then display data that is sent back from our BLL.

Using 2.0 Objects like **ObjectDataSource** we are able to create our DAL objects and have our BLL objects and point this to our BLL and many times without very little to no coding have it display for us with features such as paging or sorting.

- ✓ Simplify our GUI and presentation layer
- ✓ Add in bonus features like sorting and paging with no code
- ✓ Drag and Drop implementation

What tricks, tips or traps have you found so far?

Good question. A project that we had to create involved using **Microsoft ASP.NET 2.0 AJAX Extensions** and using the **ObjectDataSource** with **GridView** objects. Initially we were using Update Panels with Update Progress Panels. Very easy simple approach, we did however find there were some quirks and we worked around them as we went along. Some of these approaches might not be best practice, but they worked.

One issue that we noticed was if we used an **ObjectDataSource** within an Update Panel and we used Control Parameters we had timing issues. We'd set the default value within our page load to ensure we had the right values.

Learn about the objects assigned to the **SelectParameters**, **FilterParameters**, **UpdateParameters**, **DeleteParameters**, or **InsertParameters** collections.

You will have to learn how to set defaults and programmatically set values to your **ObjectDataSource** if you wish to use them extensively.

When using this object for Inserts, Updates and Deletes contained within an Update Panel you'll usually have to refresh your clients screen or they'll be looking at old data.

I'll often have something like this with a GridView that contains a list of people and a box above it that allows the user to enter new ones:

```
/// <summary>
/// Add new person
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
protected void ImageButton1_Click(object sender, ImageClickEventArgs e)
{
    try
    {
        PeopleTypeID = Convert.ToInt32(ddlPersonType.SelectedValue);

        dbs_LeagleEagleController controller = new dbs_LeagleEagleController();
        controller.InsertPerson(txtPersonName.Text, PeopleTypeID, txtPersonNotes.Text,
            UserInfo.Username, PortalId, ModuleId);

        lblAddPeopleMessages.Text = "Person added.";

        RefreshPeopleOnScreen();
    }
    catch (Exception ex)
    {
        Debug.WriteLine(ex);
        Exceptions.ProcessModuleLoadException(this, ex);
    }
}

/// <summary>
/// Keep our screen up to date
/// </summary>
private void RefreshPeopleOnScreen()
{
    txtPersonNotes.Text = "";
    txtPersonName.Text = "";
    GridView2.DataBind();
}
}
```

What an example of a Control Parameter might look like when it's bound to a text box field:

```
<asp:ControlParameter ControlID="txtSearchText" DefaultValue="" Name="SearchText"
PropertyName="Text" Type="String" />
```

How you could programmatically set the values of your params before they are selected by your ObjectDataSource.Select():

```
ObjectDataSourceDocuments.SelectParameters["SearchText"].DefaultValue = searchText;
```

Overwhelmingly our issues came from either adding to the page trigger collection. Timing issues would

ensue.

This is what it would look like to set a post back trigger on a control that's on your page. In this case it's a submit button. Use the PostBackTrigger control to enable controls inside an UpdatePanel to cause a postback instead of performing an asynchronous postback. You might find this useful when you are using the FileUpload control.

```
<Triggers>
  <asp:PostBackTrigger ControlID="btnSubmit" />
</Triggers>
```

Wrap it up would you.

As many projects are unique, most or all have a lot of the same issues and needs. You are going to have to have data stored somewhere. Someone will inevitably wish to view this data.

We've been using this with DotNetNuke version 3.1.1 and up to 4.4.1 and seen a lot of time shaved off by using these objects.

- ✓ Buy some source for a Module on Snowcovered of a project that you like. Look at how the developer coded it if you wish to see a good implementation we could point you in the right direction.
- ✓ Get involved. Find a sponsor project that you can get started with and learn from someone that's done it before you. Visit <http://www.nonprofitways.com/> or somewhere like that.
- ✓ Attend Groups like this one. Invaluable way to share ideas.