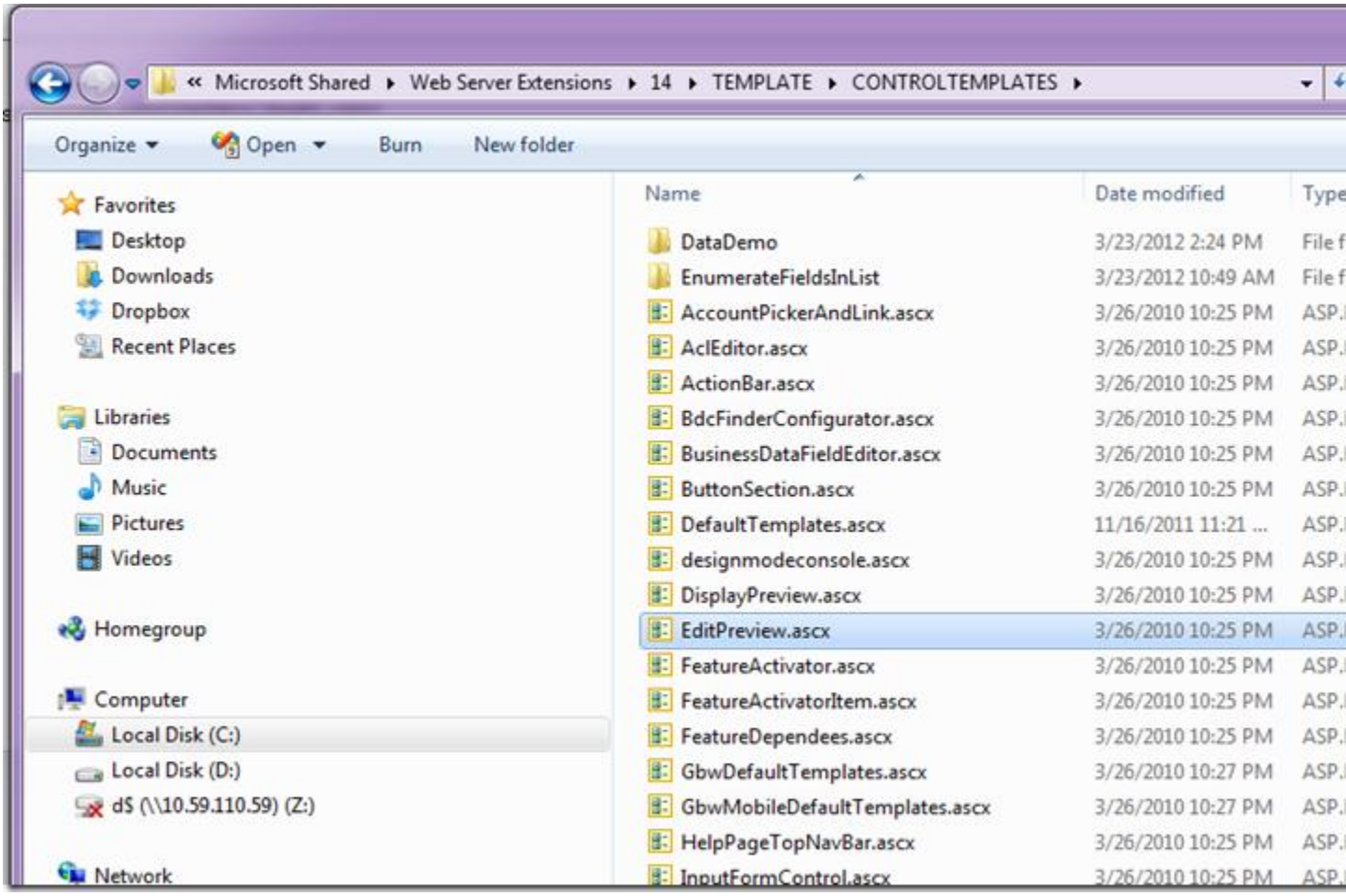
There are many controls (.ascx) files already in the C:\Program Files\Common Files\Microsoft Shared\Web Server Extensions\14\TEMPLATE\CONTROLTEMPLATES folder which we can in our aspx pages. Here's a sample of just a few them



First, we need to register the controls that we are going to use at the top of the ASPX pages. This is not a comprehensive list, but should give you the idea:

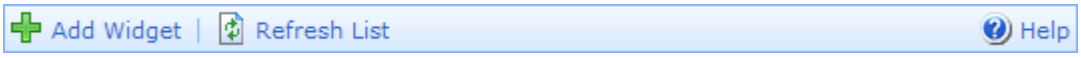
```
<%@ Register TagPrefix="wssuc" TagName="InputFormSection"
Src="/_controltemplates/InputFormSection.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="InputFormControl"
Src="/_controltemplates/InputFormControl.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ButtonSection"
Src="/_controltemplates/ButtonSection.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ToolBar"
Src="/_controltemplates/ToolBar.ascx" %>
<%@ Register TagPrefix="wssuc" TagName="ToolBarButton"
Src="/_controltemplates/ToolBarButton.ascx" %>
<%@ Register TagPrefix="SharePoint"
Namespace="Microsoft.SharePoint.WebControls"
    Assembly="Microsoft.SharePoint, Version=12.0.0.0, Culture=neutral,
PublicKeyToken=71e9bce111e9429c" %>
```

Toolbar/SPToolBarButton

Many applications have a need for a toolbar, and even SharePoint is littered with them. You're able to build one of your own by using the Toolbar.ascx user control (inside the CONTROLTEMPLATES folder), and the SPToolBarButton controls (found within the Microsoft.SharePoint.WebControls namespace):

```
<wssuc:Toolbar id="tb" runat="server">
    <Template_Buttons>
        <SharePoint:SPToolBarButton
            ID="btnAdd" runat="server" Text="Add Widget"
            ImageUrl="Images/add.gif" NavigateUrl="AddWidget.aspx" />
        <SharePoint:SPToolBarButton
            ID="btnRefresh" runat="server" Text="Refresh List"
            ImageUrl="Images/refresh1.ico" OnClick="btnRefresh_Click" />
    </Template_Buttons>
    <Template_RightButtons>
        <SharePoint:SPToolBarButton
            ID="btnHelp" runat="server" Text="Help"
            ImageUrl="Images/help.gif" NavigateUrl="Help.aspx"  />
    </Template_RightButtons>
</wssuc:Toolbar>
```

The above markup will render:



SPGridView

The next control worth mentioning is the all-powerful SPGridView control. This control is inherited from the ASP.NET GridView control which is already a great control, but the SPGridView provides a ton more functionality. In supports grouping, it automatically inherits the styles of SharePoint, and you are able to add drop-down menus to your items, like users are already used to in lists and libraries. The markup syntax is pretty straightforward:

```
<SharePoint:SPGridView
    ID="grid" runat="server" AutoGenerateColumns="false" AllowSorting="true"
    AllowGrouping="true" GroupField="Category" AllowGroupCollapse="true">

    <Columns>
        <asp:BoundField HeaderText="ID" DataField="ID" SortExpression="ID" />
         <asp:BoundField
            HeaderText="Name" DataField="Name"
            SortExpression="Name" />
        <asp:BoundField
            HeaderText="Price" DataField="Price"
            SortExpression="Price" />
        <asp:BoundField
            HeaderText="Quantity on Hand" DataField="QuantityOnHand"
            SortExpression="QuantityOnHand" />
        <asp:BoundField
            HeaderText="Date Added" DataField="DateAdded"
            SortExpression="DateAdded" />
    </Columns>
</SharePoint:SPGridView>
```

The above markup will render

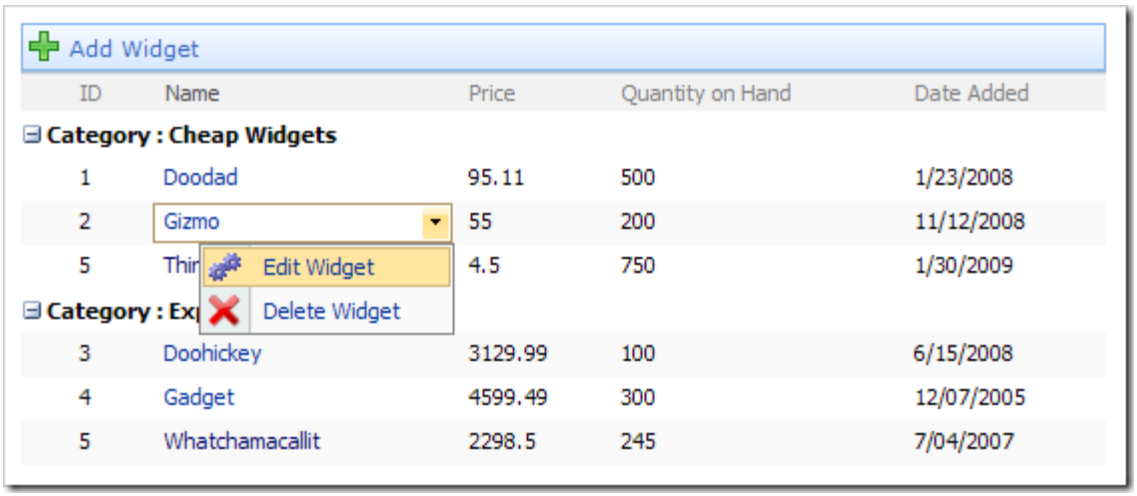
Let's take this a step further and add the familiar drop-down list to the Name column. There are a ton of examples on how to do this in code-behind, most notably Powlo's posts here and here, but here's a little preview on how to do this in the markup. First, we need to create our MenuTemplate, which defines the items in the drop-down list:

```
<SharePoint:MenuTemplate ID="menuTemplate" runat="server">
    <SharePoint:MenuItemTemplate ID="menuEdit" runat="server"
        Text="Edit Widget" ImageUrl="Images/gear.gif"
        ClientOnClickScript="javascript:editSomething('%ID%');" />
    <SharePoint:MenuItemTemplate ID="menuDelete" runat="server"
        Text="Delete Widget" ImageUrl="Images/delete.gif"
        ClientOnClickScript="javascript:deleteSomething('%ID%');" />
</SharePoint:MenuTemplate>
```

Next, replace the BoundField with an SPMenuField, and specify the menu this is bound to by assigning the MenuTemplateID property to the ID of the menu template we just created. The TokenNameAndValueFields property assigns a token to a field in the data source. In this example, I'm declaring two tokens, one for ID and another for Name, which can then be used elsewhere. The NavigateUrlFields specifies the fields that can be used in the URL set in the NavigateUrlFormat property, and in the same order (it works like the String.Format() method):

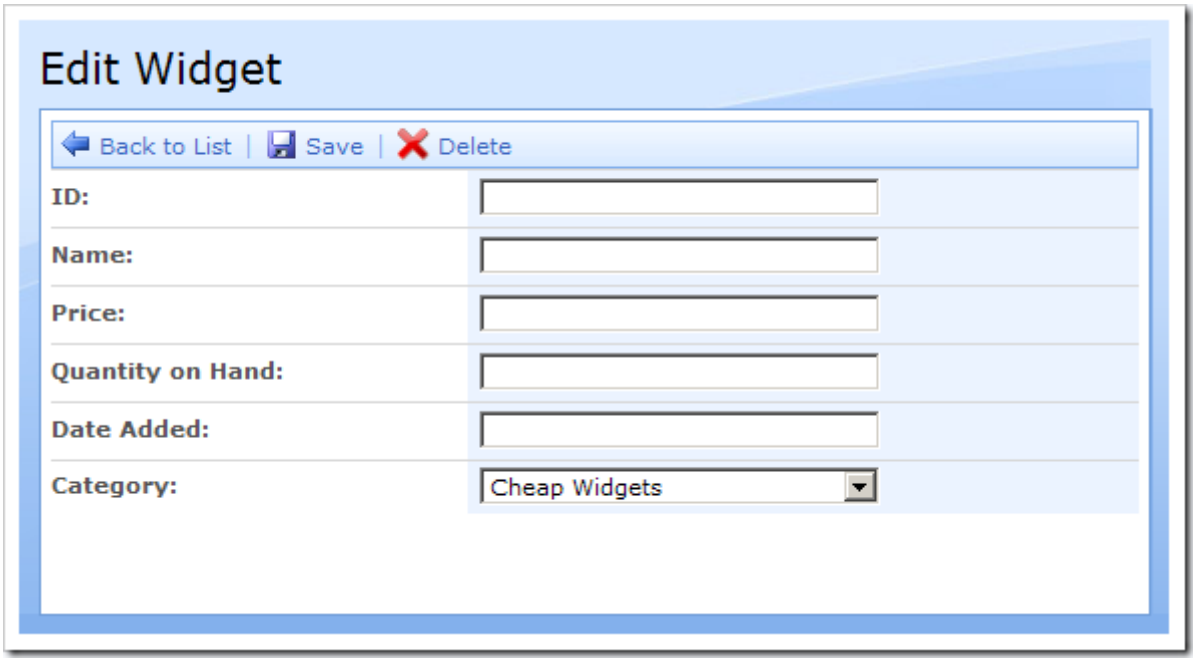
```
<SharePoint:SPMenuField
    HeaderText="Name" TextFields="Name" MenuTemplateId="menuTemplate"
    TokenNameAndValueFields="ID=ID,NAME=Name" NavigateUrlFields="ID"
    NavigateUrlFormat="EditWidget.aspx?id={0}" />
```

The above markup will render:



Form Fields

For creating form fields, there are a ton of ways to skin this cat, but I typically choose one of the following two approaches. Typically your data-entry forms should either look like the forms used for new list items, or the forms for new lists, sites, or pages. The simplest and arguably cleanest approach is to just use the ASP.NET controls you're used to, such as a TextBox, DropDownList, etc., and place them in a table that have the SharePoint styles applied to them. The end result will look something like this:

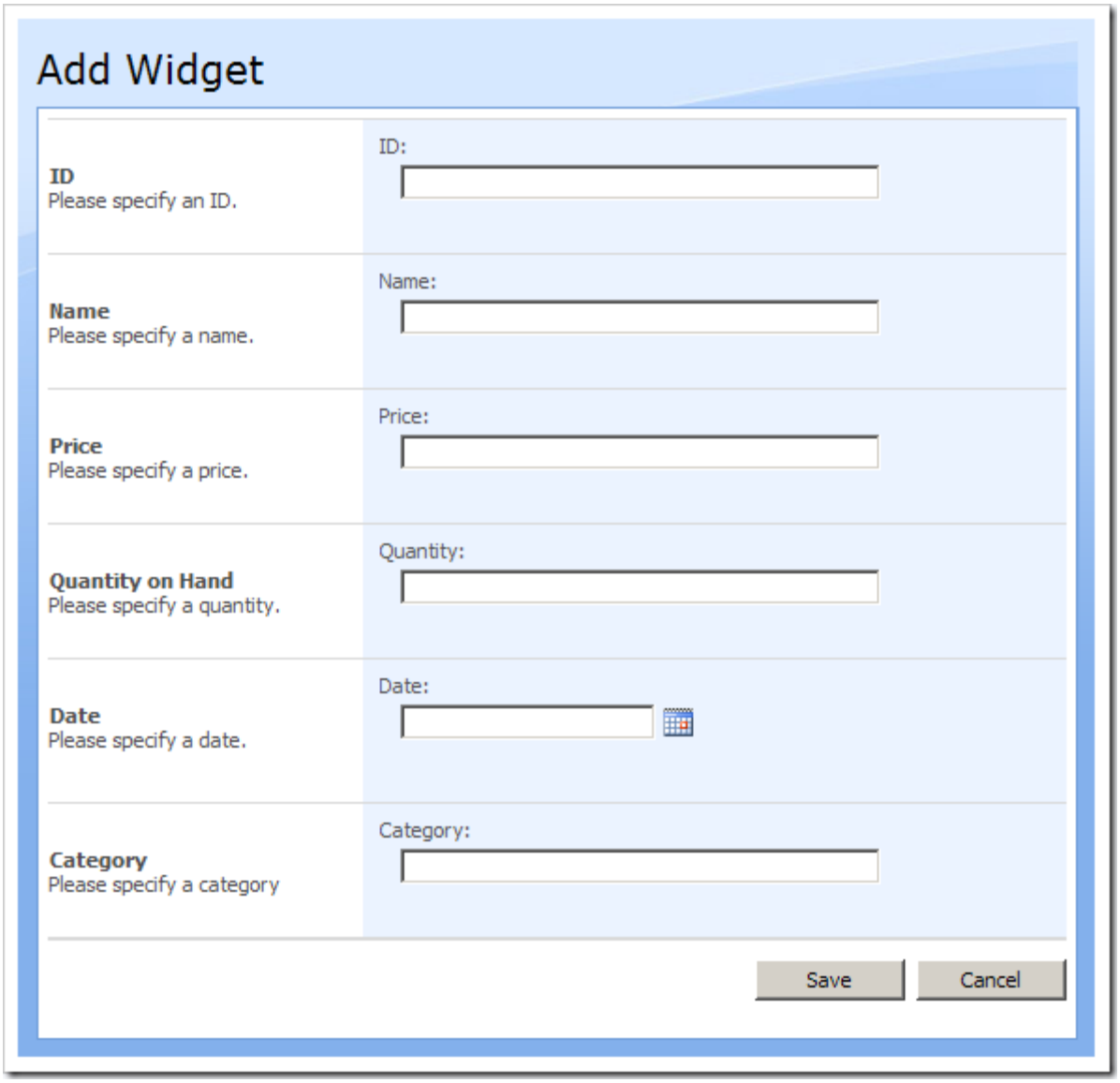


The markup is pretty simple; the important part is is the style classes applied to the elements, specifically ms-formlabel, ms-formbody, and ms-input. For the sake of brevity, here is a portion of the markup for the above form:

```
<tr>
  <td class="ms-formlabel">
      Date Added:
  </td>
  <td class="ms-formbody">
```

```
      <asp:TextBox id="txtDateAdded" runat="server" CssClass="ms-input"
width="200px" />
  </td>
</tr>
<tr>
  <td class="ms-formlabel">
      Category:
  </td>
  <td class="ms-formbody">
      <asp:DropDownList id="txtCategory" runat="server" CssClass="ms-input"
width="200px">
          <asp:ListItem>Cheap Widgets</asp:ListItem>
          <asp:ListItem>Expensive Widgets</asp:ListItem>
      </asp:DropDownList>
  </td>
</tr>
```

The second approach is to make the form look like the new list/site pages in SharePoint. This is perfectly fine too, but takes up a little more space:



The markup for this is a little more complex, and involves the use of InputFormSection and InputFormControl sections. A sample of the markup used for the above form is as follows:

```
<wssuc:InputFormSection runat="server" Title="" id="dateSection">
    <template_description>
       <b>Date</b><br />Please specify a date.
    </template_description>
    <template_inputformcontrols>
        <wssuc:InputFormControl runat="server" LabelText="Date:">
            <Template_Control>
                <wssawc:DateTimeControl  ID="txtDate" runat="server"
                    DateOnly="true" /><BR />
            </Template_Control>
        </wssuc:InputFormControl>
    </template_inputformcontrols>
</wssuc:InputFormSection>
```

```
<wssuc:InputFormSection runat="server" Title="" id="categorySection">
    <template_description>
       <b>Category</b><br />Please specify a category
    </template_description>
    <template_inputformcontrols>
        <wssuc:InputFormControl runat="server" LabelText="Category:">
            <Template_Control>
                <wssawc:InputFormTextBox ID="txtCategory" runat="server"
                    Columns="40" class="ms-input"  /><BR />
            </Template_Control>
        </wssuc:InputFormControl>
    </template_inputformcontrols>
</wssuc:InputFormSection>

<wssuc:ButtonSection runat="server" ShowStandardCancelButton="false">
    <Template_Buttons>
        <asp:Button runat="server" class="ms-ButtonHeightWidth"
            Text="Save" id="btnSave" />
        <asp:Button runat="server" class="ms-ButtonHeightWidth"
            Text="Cancel" id="btnCancel" CausesValidation="false" />
    </Template_Buttons>
</wssuc:ButtonSection>
```