Master/Detail Using a Selectable Master GridView with a Details DetailView

This tutorial will have a GridView whose rows include the name and price of each product along with a Select button. Clicking the Select button for a particular product will cause its full details to be displayed in a DetailsView control on the same page.

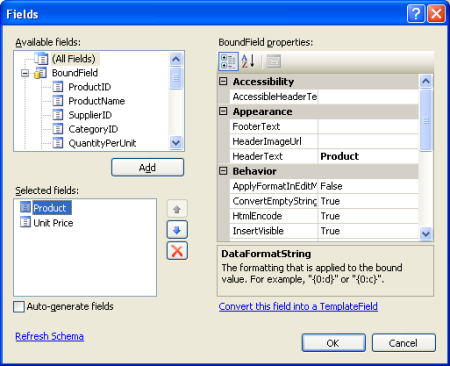


**Figure 1**: Clicking the Select Button Displays the Product's Details

## ****Step 1: Creating a Selectable GridView****

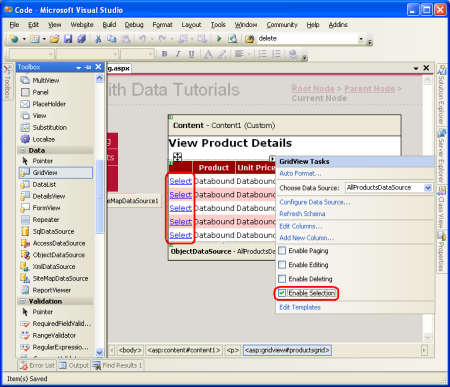
Recall that in the two-page master/detail report that each master record included a hyperlink that, when clicked, sent the user to the details page passing the clicked row's SupplierID value in the querystring. Such a hyperlink was added to each GridView row using a HyperLinkField. For the single page master/details report, we will need a Button for each GridView row that, when clicked, shows the details. The GridView control can be configured to include a Select button for each row that causes a postback and marks that row as the GridView's [SelectedRow](http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.selectedrow.aspx).

Start by adding a GridView control to the DetailsBySelecting.aspx page in the Filtering folder, setting its ID property to ProductsGrid. Next, add a new ObjectDataSource named AllProductsDataSource that invokes the ProductsBLL class's GetProducts() method.

[](http://i1.asp.net/asp.net/images/dataaccess/10fig05cs.png)

**Figure 5**: Remove All But the ProductName and UnitPrice BoundFields

Next, we need to mark the GridView as selectable, which will add a Select button to each row. To accomplish this, simply check the Enable Selection checkbox in the GridView's smart tag.

[](http://i1.asp.net/asp.net/images/dataaccess/10fig06cs.png)

**Figure 6**: Make the GridView's Rows Selectable

Checking the Enabling Selection option adds a CommandField to the ProductsGrid GridView with its ShowSelectButton property set to True. This results in a Select button for each row of the GridView, as Figure 6 illustrates. By default, the Select buttons are rendered as LinkButtons, but you can use Buttons or ImageButtons instead through the CommandField's ButtonType property.

<asp:GridView ID="ProductsGrid" runat="server" AutoGenerateColumns="False"

DataKeyNames="ProductID"

DataSourceID="AllProductsDataSource"

EnableViewState="False">

<Columns>

<asp:CommandField ShowSelectButton="True" />

<asp:BoundField DataField="ProductName" HeaderText="Product"

SortExpression="ProductName" />

<asp:BoundField DataField="UnitPrice" DataFormatString="{0:c}"

HeaderText="Unit Price" HtmlEncode="False"

SortExpression="UnitPrice" />

</Columns>

</asp:GridView>

When a GridView row's Select button is clicked a postback ensues and the GridView's SelectedRow property is updated. In addition to the SelectedRow property, the GridView provides the [SelectedIndex](http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.selectedindex%28VS.80%29.aspx), [SelectedValue](http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.selectedvalue%28VS.80%29.aspx), and [SelectedDataKey](http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.selecteddatakey%28VS.80%29.aspx) properties. The SelectedIndex property returns the index of the selected row, whereas the SelectedValue and SelectedDataKey properties return values based upon the GridView's [DataKeyNames property](http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.gridview.datakeynames%28VS.80%29.aspx).

The DataKeyNames property is used to associate one or more data field values with each row and is commonly used to attribute uniquely identifying information from the underlying data with each GridView row. The SelectedValue property returns the value of the first DataKeyNames data field for the selected row where as the SelectedDataKey property returns the selected row's DataKey object, which contains all of the values for the specified data key fields for that row.

**Clicking the Select button causes a postback.** In Step 2 we'll see how to have a DetailsView respond to this postback by displaying the details for the selected product.



**Figure 7**: Each Product Row Contains a Select LinkButton

**Highlighting the Selected Row**

The **ProductsGrid** GridView has a **SelectedRowStyle** property that can be used to dictate the visual style for the selected row. Used properly, this can improve the user's experience by more clearly showing which row of the GridView is currently selected. For this tutorial, let's have the selected row be highlighted with a yellow background.

As with our earlier tutorials, let's strive to keep the aesthetic-related settings defined as CSS classes. Therefore, create a new CSS class in **Styles.css** named **SelectedRowStyl**e.

.SelectedRowStyle { background-color: Yellow; }

To apply this CSS class to the **SelectedRowStyle** property of all GridViews in our tutorial series, edit the **GridView.skin** Skin in the **DataWebControls** Theme to include the **SelectedRowStyle** settings as shown below:

<asp:GridView runat="server" **CssClass="DataWebControlStyle"**>

<AlternatingRowStyle CssClass="AlternatingRowStyle" />

<RowStyle CssClass="RowStyle" />

<HeaderStyle CssClass="HeaderStyle" />

<SelectedRowStyle CssClass="SelectedRowStyle" />

</asp:GridView>

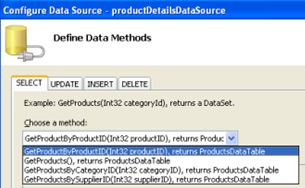
With this addition, the selected GridView row is now highlighted with a yellow background color.



**Figure 8**: Customize the Selected Row's Appearance Using the GridView's SelectedRowStyle Property

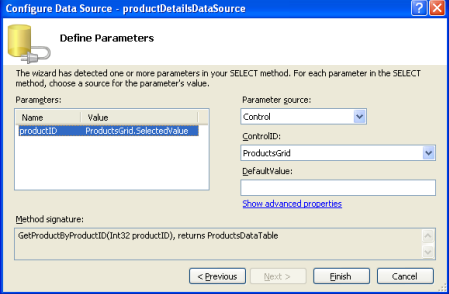
## ****Step 2: Displaying the Selected Product's Details in a DetailsView****

All that remains is to add a DetailsView that displays information about the particular product selected. Add a DetailsView control above the GridView and create a new ObjectDataSource named **ProductDetailsDataSource**. Since we want this DetailsView to display particular information about the selected product, configure the **ProductDetailsDataSource** to use the **ProductsBLL** class's **GetProductByProductID(productID)** method.



**Figure 9**: Invoke the ProductsBLL Class's GetProductByProductID(productID) Method

Have the *productID* parameter's value obtained from the GridView control's SelectedValue property. As we discussed earlier, the GridView's SelectedValue property returns the first data key value for the selected row. Therefore, it's imperative that the GridView's DataKeyNames property is set to ProductID, so that the selected row's ProductID value is returned by SelectedValue.



**Figure 10**: Set the *productID* Parameter to the GridView's SelectedValue Property

Once the productDetailsDataSource ObjectDataSource has been configured correctly and bound to the DetailsView, this tutorial is complete! When the page is first visited no row is selected, so the GridView's SelectedValue property returns null. Since there are no products with a NULL ProductID value, no records are returned by the GetProductByProductID(productID) method, meaning that the DetailsView isn't displayed. Upon clicking a GridView row's Select button, a postback ensues and the DetailsView is refreshed. This time the GridView's SelectedValue property returns the ProductID of the selected row, the GetProductByProductID(productID) method returns a ProductsDataTable with information about that particular product, and the DetailsView shows these details (see Figure 12).



**Figure 12**: Upon Selecting a Row, the Product's Details are Displayed