Working with Data in ASP.NET 2.0 :: An Overview of Inserting, Updating, and Deleting Data

Introduction

Over the past several tutorials we've examined how to display data in an ASP.NET page using the GridView, DetailsView, and FormView controls. These controls simply work with data supplied to them. Commonly, these controls access data through the use of a data source control, such as the ObjectDataSource. We've seen how the ObjectDataSource acts as a proxy between the ASP.NET page and the underlying data. When a GridView needs to display data, it invokes its ObjectDataSource's `Select()` method, which in turn invokes a method from our Business Logic Layer (BLL), which calls a method in the appropriate Data Access Layer's (DAL) TableAdapter, which in turn sends a `SELECT` query to the Northwind database.

Recall that when we created the TableAdapters in the DAL in our first tutorial, Visual Studio automatically added methods for inserting, updating, and deleting data from the underlying database table. Moreover, in Creating a Business Logic Layer we designed methods in the BLL that called down into these data modification DAL methods.

In addition to its `Select()` method, the ObjectDataSource also has `Insert()`, `Update()`, and `Delete()` methods. Like the `Select()` method, these three methods can be mapped to methods in an underlying object. When configured to insert, update, or delete data, the GridView, DetailsView, and FormView controls provide a user interface for modifying the underlying data. This user interface calls the `Insert()`, `Update()`, and `Delete()` methods of the ObjectDataSource, which then invoke the underlying object's associated methods (see Figure 1).

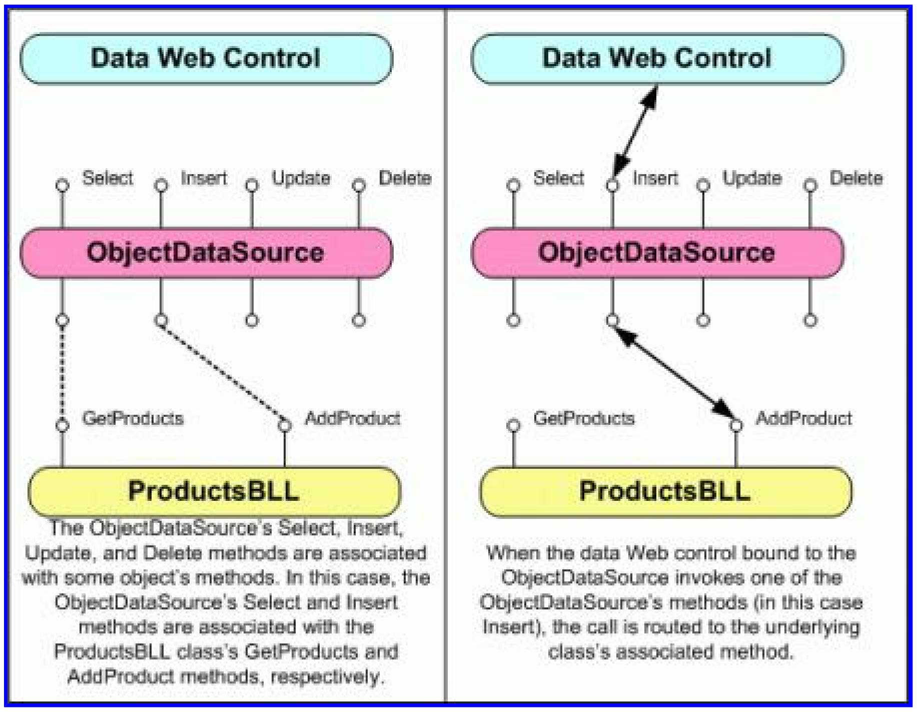

Data Web Control
Select
Insert
Update
Delete
ObjectDataSource
GetProducts
AddProduct
ProductsBLL
The ObjectDataSource's Select, Insert, Update, and Delete methods are associated with some object's methods. In this case, the ObjectDataSource's Select and Insert methods are associated with the ProductsBLL class's GetProducts and AddProduct methods, respectively.
Data Web Control
Select
Insert
Update
Delete
ObjectDataSource
GetProducts
AddProduct
ProductsBLL
When the data Web control bound to the ObjectDataSource invokes one of the ObjectDataSource's methods (in this case Insert), the call is routed to the underlying class's associated method.

Figure 1: The ObjectDataSource's `Insert()`, `Update()`, and `Delete()` Methods Serve as a Proxy into the

BLL

In this tutorial we'll see how to map the ObjectDataSource's `Insert()`, `Update()`, and `Delete()` methods to methods of classes in the BLL, as well as how to configure the GridView, DetailsView, and FormView controls to provide data modification capabilities.

Step 1: Creating the Insert, Update, and Delete Tutorials Web Pages

Before we start exploring how to insert, update, and delete data, let's first take a moment to create the ASP.NET pages in our website project that we'll need for this tutorial and the next several ones. Start by adding a new folder named `EditInsertDelete`. Next, add the following ASP.NET pages to that folder, making sure to associate each page with the `Site.master` master page:

- `Default.aspx`
- `Basics.aspx`
- `DataModificationEvents.aspx`
- `ErrorHandling.aspx`
- `UIValidation.aspx`
- `CustomizedUI.aspx`
- `OptimisticConcurrency.aspx`
- `ConfirmationOnDelete.aspx`
- `UserLevelAccess.aspx`

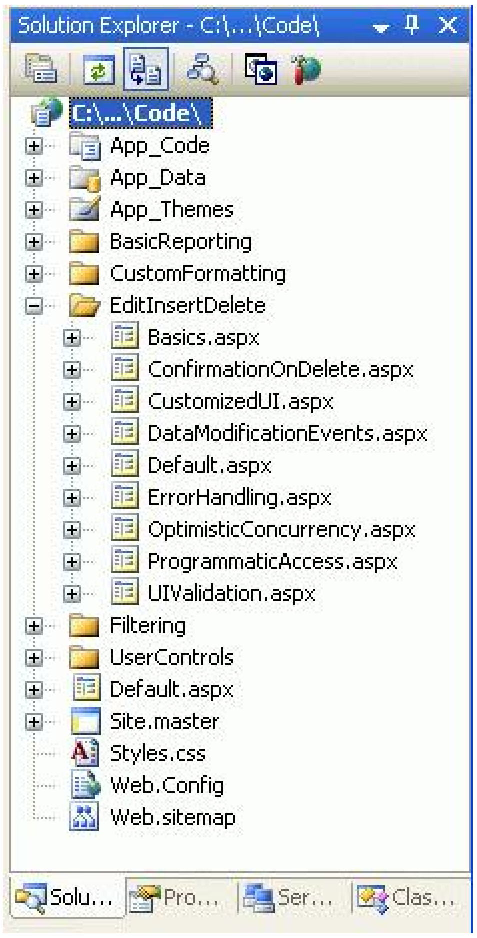


Figure 2: Add the ASP.NET Pages for the Data Modification-Related Tutorials

Like in the other folders, `Default.aspx` in the `EditInsertDelete` folder will list the tutorials in its section. Recall that the `SectionLevelTutorialListing.ascx` User Control provides this functionality. Therefore, add this User Control to `Default.aspx` by dragging it from the Solution Explorer onto the page's Design view.

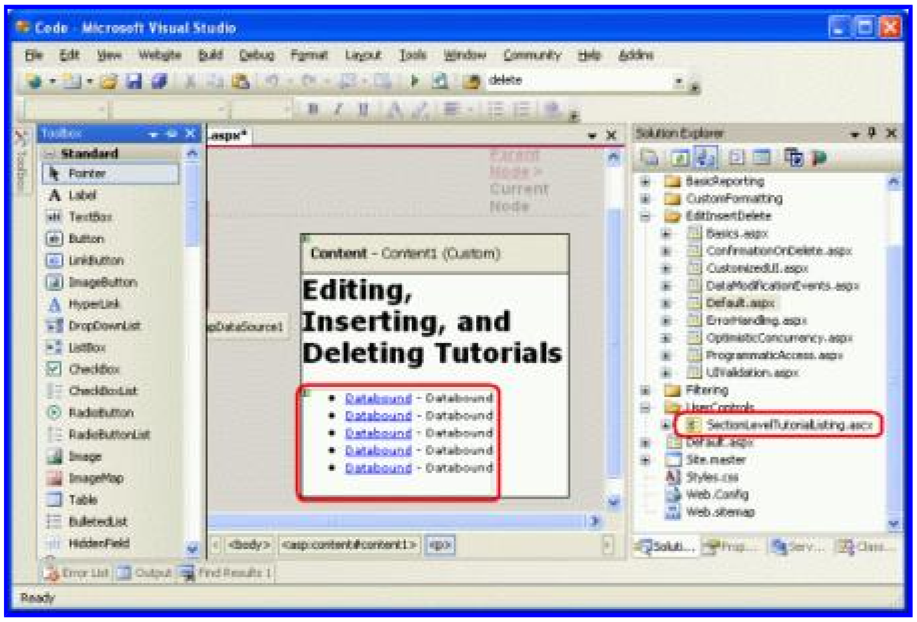


Figure 3: Add the `SectionLevelTutorialListing.ascx` User Control to `Default.aspx`

Lastly, add the pages as entries to the `Web.sitemap` file. Specifically, add the following markup after the Customized Formatting `<siteMapNode>`:

```
<siteMapNode title="Editing, Inserting, and Deleting"
    url="~/EditInsertDelete/Default.aspx"
    description="Samples of Reports that Provide Editing, Inserting,
                  and Deleting Capabilities">

    <siteMapNode url="~/EditInsertDelete/Basics.aspx"
        title="Basics"
        description="Examines the basics of data modification with the
                      GridView, DetailsView, and FormView controls." />
    <siteMapNode url="~/EditInsertDelete/DataModificationEvents.aspx"
        title="Data Modification Events"
        description="Explores the events raised by the ObjectDataSource
                      pertinent to data modification." />
    <siteMapNode url="~/EditInsertDelete/ErrorHandling.aspx"
        title="Error Handling"
        description="Learn how to gracefully handle exceptions raised
                      during the data modification workflow." />
    <siteMapNode url="~/EditInsertDelete/UIValidation.aspx"
        title="Adding Data Entry Validation"
        description="Help prevent data entry errors by providing validation." />
    <siteMapNode url="~/EditInsertDelete/CustomizedUI.aspx"
        title="Customize the User Interface"
        description="Customize the editing and inserting user interfaces." />
    <siteMapNode url="~/EditInsertDelete/OptimisticConcurrency.aspx"
        title="Optimistic Concurrency"
        description="Learn how to help prevent simultaneous users from
                      overwritting one another&apos;s changes." />
    <siteMapNode url="~/EditInsertDelete/ConfirmationOnDelete.aspx"
        title="Confirm On Delete"
        description="Prompt a user for confirmation when deleting a record." />
    <siteMapNode url="~/EditInsertDelete/UserLevelAccess.aspx"
        title="Limit Capabilities Based on User"
        description="Learn how to limit the data modification functionality
                      based on the user role or permissions." />
</siteMapNode>
```

After updating `Web.sitemap`, take a moment to view the tutorials website through a browser. The menu on the left now includes items for the editing, inserting, and deleting tutorials.

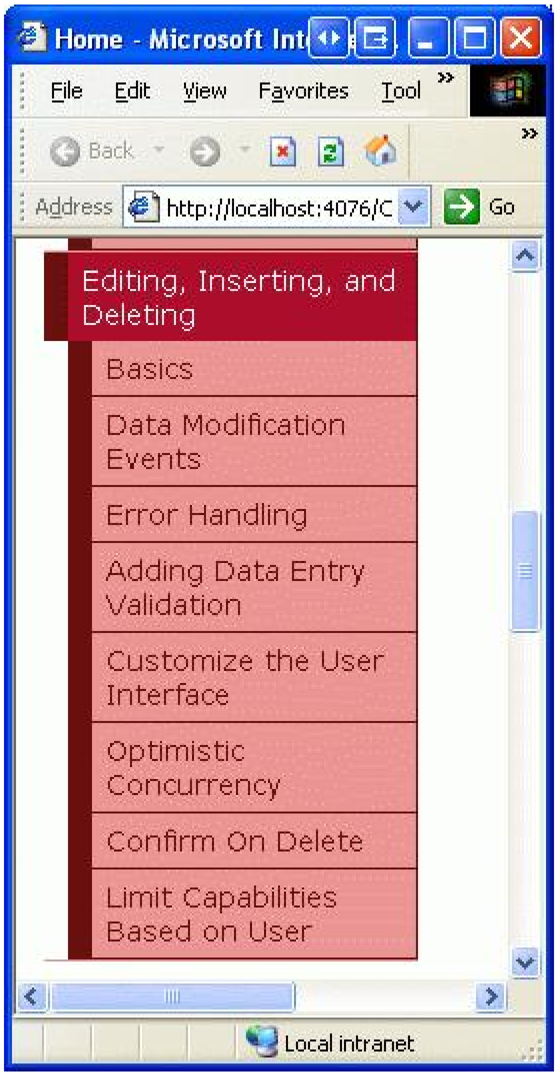
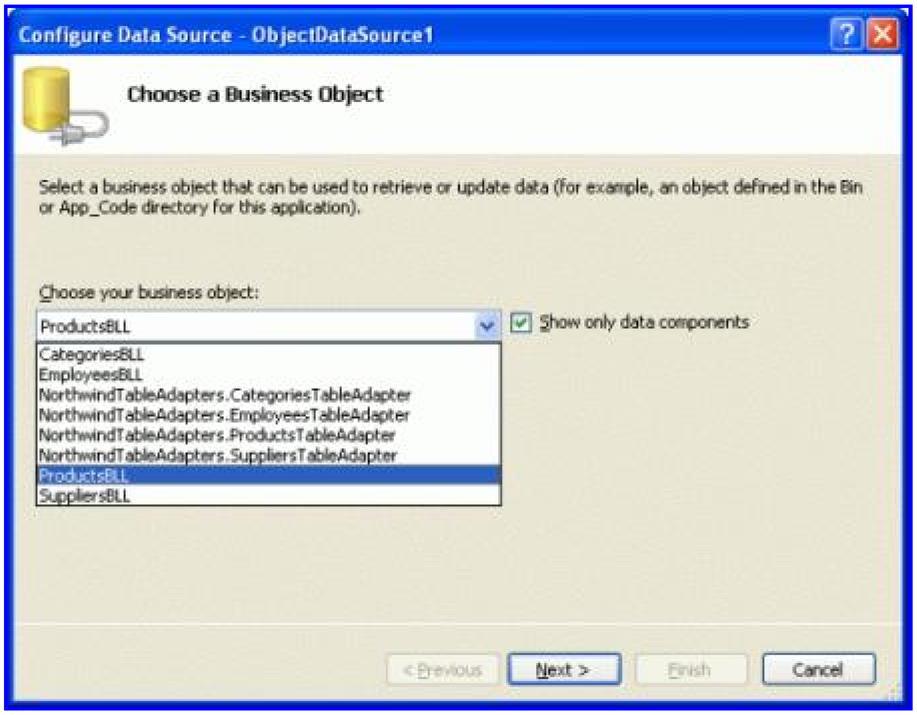


Figure 4: The Site Map Now Includes Entries for the Editing, Inserting, and Deleting Tutorials

Step 2: Adding and Configuring the ObjectDataSource Control

Since the GridView, DetailsView, and FormView each differ in their data modification capabilities and layout, let's examine each one individually. Rather than have each control using its own ObjectDataSource, however, let's just create a single ObjectDataSource that all three control examples can share.

Open the `Basics.aspx` page, drag an ObjectDataSource from the Toolbox onto the Designer, and click the Configure Data Source link from its smart tag. Since the `ProductsBLL` is the only BLL class that provides editing, inserting, and deleting methods, configure the ObjectDataSource to use this class.



Select a business object that can be used to retrieve or update data (for example, an object defined in the Bin or App_Code directory for this application).

Choose your business object:

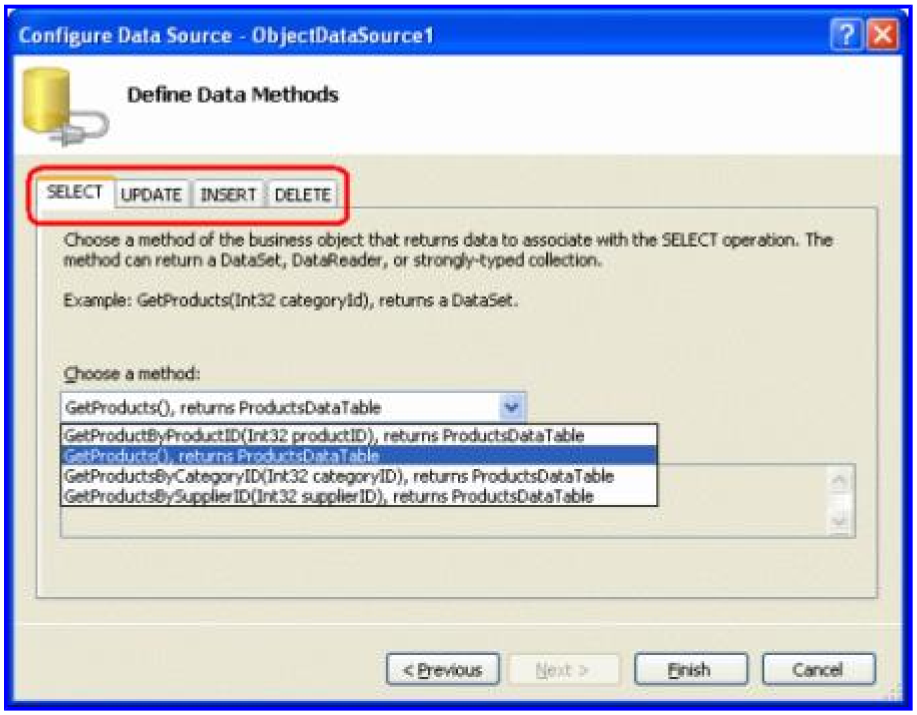
ProductsBLL

☑ Show only data components

CategoriesBLL
EmployeesBLL
NorthwindTableAdapters.CategoriesTableAdapter
NorthwindTableAdapters.EmployeesTableAdapter
NorthwindTableAdapters.ProductsTableAdapter
NorthwindTableAdapters.SuppliersTableAdapter
ProductsBLL
SuppliersBLL

Figure 5: Configure the ObjectDataSource to Use the `ProductsBLL` Class

In the next screen we can specify what methods of the `ProductsBLL` class are mapped to the ObjectDataSource's `Select()`, `Insert()`, `Update()`, and `Delete()` by selecting the appropriate tab and choosing the method from the drop-down list. Figure 6, which should look familiar by now, maps the ObjectDataSource's `Select()` method to the `ProductsBLL` class's `GetProducts()` method. The `Insert()`, `Update()`, and `Delete()` methods can be configured by selecting the appropriate tab from the list along the top.



Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

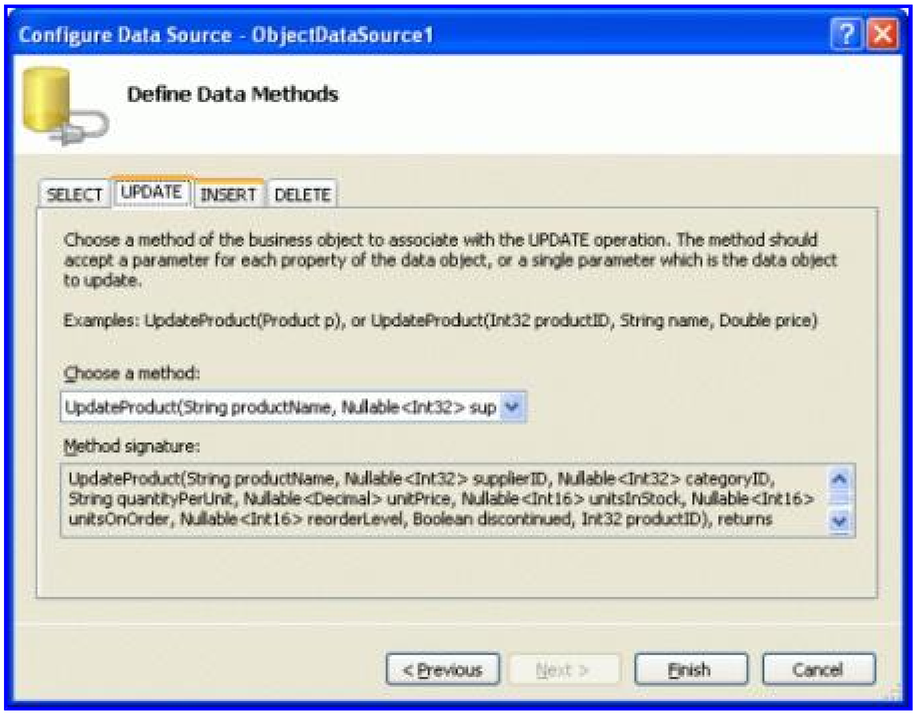
Choose a method:

GetProducts(), returns ProductsDataTable

GetProductByProductID(Int32 productID), returns ProductsDataTable
GetProducts(), returns ProductsDataTable
GetProductsByCategoryID(Int32 categoryID), returns ProductsDataTable
GetProductsBySupplierID(Int32 supplierID), returns ProductsDataTable

Figure 6: Have the ObjectDataSource Return All of the Products

Figures 7, 8, and 9 show the ObjectDataSource's UPDATE, INSERT, and DELETE tabs. Configure these tabs so that the `Insert()`, `Update()`, and `Delete()` methods invoke the `ProductsBLL` class's `UpdateProduct`, `AddProduct`, and `DeleteProduct` methods, respectively.



Choose a method of the business object to associate with the UPDATE operation. The method should accept a parameter for each property of the data object, or a single parameter which is the data object to update.

Examples: UpdateProduct(Product p), or UpdateProduct(Int32 productID, String name, Double price)

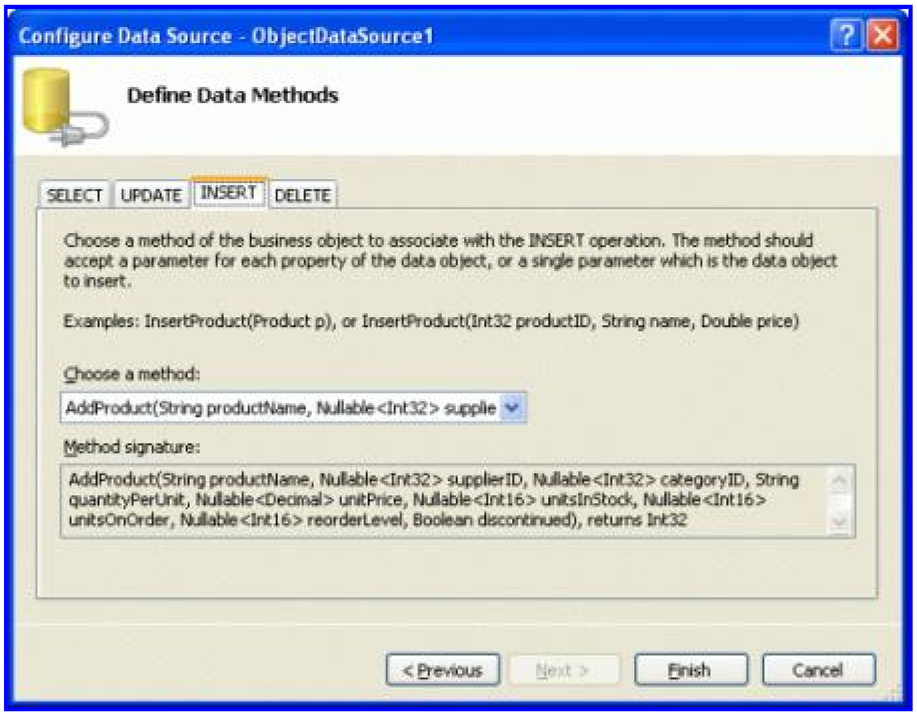
Choose a method:

UpdateProduct(String productName, Nullable<Int32> sup

Method signature:

UpdateProduct(String productName, Nullable<Int32> supplierID, Nullable<Int32> categoryID, String quantityPerUnit, Nullable<Decimal> unitPrice, Nullable<Int16> unitsInStock, Nullable<Int16> unitsOnOrder, Nullable<Int16> reorderLevel, Boolean discontinued, Int32 productID), returns

Figure 7: Map the ObjectDataSource's `Update()` Method to the `ProductBLL` Class's `UpdateProduct` Method



Choose a method of the business object to associate with the INSERT operation. The method should accept a parameter for each property of the data object, or a single parameter which is the data object to insert.

Examples: InsertProduct(Product p), or InsertProduct(Int32 productID, String name, Double price)

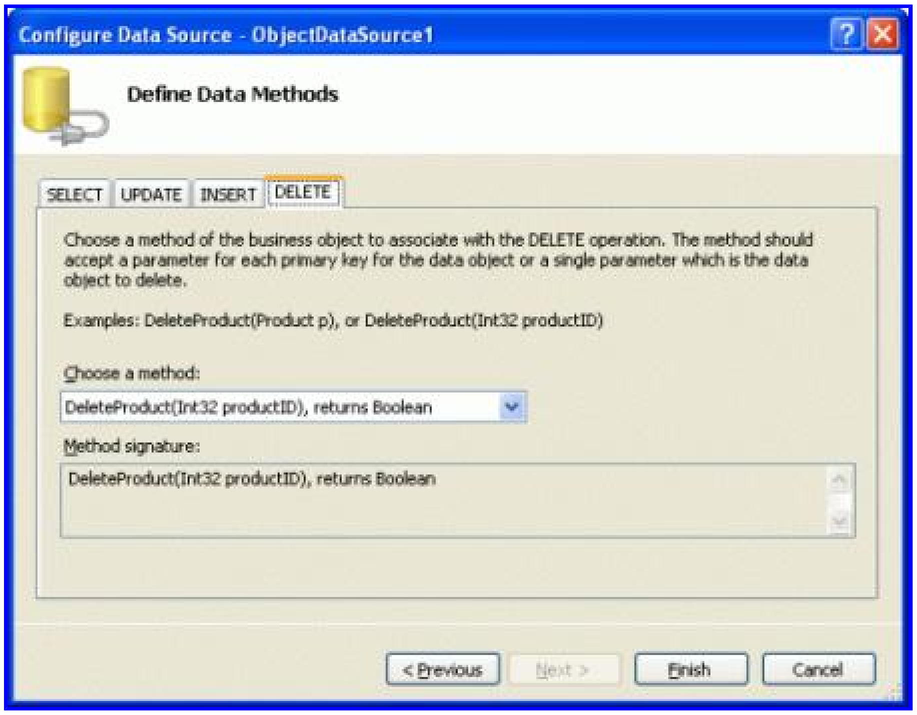
Choose a method:

AddProduct(String productName, Nullable<Int32> supplie

Method signature:

AddProduct(String productName, Nullable<Int32> supplierID, Nullable<Int32> categoryID, String quantityPerUnit, Nullable<Decimal> unitPrice, Nullable<Int16> unitsInStock, Nullable<Int16> unitsOnOrder, Nullable<Int16> reorderLevel, Boolean discontinued), returns Int32

Figure 8: Map the ObjectDataSource's `Insert()` Method to the `ProductBLL` Class's Add`Product` Method



Choose a method of the business object to associate with the DELETE operation. The method should accept a parameter for each primary key for the data object or a single parameter which is the data object to delete.

Examples: DeleteProduct(Product p), or DeleteProduct(Int32 productID)

Choose a method:

DeleteProduct(Int32 productID), returns Boolean

Method signature:

DeleteProduct(Int32 productID), returns Boolean

Figure 9: Map the ObjectDataSource's `Delete()` Method to the `ProductBLL` Class's `DeleteProduct` Method

You may have noticed that the drop-down lists in the UPDATE, INSERT, and DELETE tabs already had these methods selected. This is thanks to our use of the `DataObjectMethodAttribute` that decorates the methods of `ProducstBLL`. For example, the DeleteProduct method has the following signature:

```
[System.ComponentModel.DataObjectMethodAttribute
    (System.ComponentModel.DataObjectMethodType.Delete, true)]
public bool DeleteProduct(int productID)
{
    ...
}
```

The `DataObjectMethodAttribute` attribute indicates the purpose of each method – whether it is for selecting, inserting, updating, or deleting – and whether or not it's the default value. If you omitted these attributes when creating your BLL classes, you'll need to manually select the methods from the UPDATE, INSERT, and DELETE tabs.

After ensuring that the appropriate `ProductsBLL` methods are mapped to the ObjectDataSource's `Insert()`, `Update()`, and `Delete()` methods, click Finish to complete the wizard.

Examining the ObjectDataSource's Markup

After configuring the ObjectDataSource through its wizard, go to the Source view to examine the generated declarative markup. The `<asp:ObjectDataSource>` tag specifies the underlying object and the methods to invoke. Additionally, there are `DeleteParameters`, `UpdateParameters`, and `InsertParameters` that map to the input parameters for the `ProductsBLL` class's `AddProduct`, `UpdateProduct`, and `DeleteProduct` methods:

```
<asp:ObjectDataSource ID="ObjectDataSource1" runat="server"
    DeleteMethod="DeleteProduct" InsertMethod="AddProduct"
    OldValuesParameterFormatString="original_{0}" SelectMethod="GetProducts"
    TypeName="ProductsBLL" UpdateMethod="UpdateProduct">
    <DeleteParameters>
        <asp:Parameter Name="productID" Type="Int32" />
    </DeleteParameters>
    <UpdateParameters>
        <asp:Parameter Name="productName" Type="String" />
        <asp:Parameter Name="supplierID" Type="Int32" />
        <asp:Parameter Name="categoryID" Type="Int32" />
        <asp:Parameter Name="quantityPerUnit" Type="String" />
        <asp:Parameter Name="unitPrice" Type="Decimal" />
        <asp:Parameter Name="unitsInStock" Type="Int16" />
        <asp:Parameter Name="unitsOnOrder" Type="Int16" />
        <asp:Parameter Name="reorderLevel" Type="Int16" />
        <asp:Parameter Name="discontinued" Type="Boolean" />
        <asp:Parameter Name="productID" Type="Int32" />
    </UpdateParameters>
    <InsertParameters>
        <asp:Parameter Name="productName" Type="String" />
        <asp:Parameter Name="supplierID" Type="Int32" />
        <asp:Parameter Name="categoryID" Type="Int32" />
        <asp:Parameter Name="quantityPerUnit" Type="String" />
        <asp:Parameter Name="unitPrice" Type="Decimal" />
        <asp:Parameter Name="unitsInStock" Type="Int16" />
        <asp:Parameter Name="unitsOnOrder" Type="Int16" />
        <asp:Parameter Name="reorderLevel" Type="Int16" />
        <asp:Parameter Name="discontinued" Type="Boolean" />
    </InsertParameters>
</asp:ObjectDataSource>
```

The ObjectDataSource includes a parameter for each of the input parameters for its associated methods, just like a list of `SelectParameter`s is present when the ObjectDataSource is configured to call a select method that expects an input parameter (such as `GetProductsByCategoryID(`*`categoryID`*`)`). As we'll see shortly, values for these `DeleteParameters`, `UpdateParameters`, and `InsertParameters` are set automatically by the GridView, DetailsView, and FormView prior to invoking the ObjectDataSource's `Insert()`, `Update()`, or `Delete()` method. These values can also be set programmatically as needed, as we'll discuss in a future tutorial.

One side effect of using the wizard to configure to ObjectDataSource is that Visual Studio sets the OldValuesParameterFormatString property to `original_{0}`. This property value is used to include the original values of the data being edited and is useful in two scenarios:

- If, when editing a record, users are able to change the primary key value. In this case, both the new primary key value and the original primary key value must be provided so that the record with the original primary key value can be found and have its value updated accordingly.
- When using optimistic concurrency. Optimistic concurrency is a technique to ensure that two simultaneous users don't overwrite one another's changes, and is the topic for a future tutorial.

The `OldValuesParameterFormatString` property indicates the name of the input parameters in the underlying object's update and delete methods for the original values. We'll discuss this property and its purpose in greater detail when we explore optimistic concurrency. I bring it up now, however, because our BLL's methods do not expect the original values and therefore it's important that we remove this property. Leaving the `OldValuesParameterFormatString` property set to anything other than the default (`{0}`) will cause an error when a data Web control attempts to invoke the ObjectDataSource's `Update()` or `Delete()` methods because the ObjectDataSource will attempt to pass in both the `UpdateParameters` or `DeleteParameters` specified as well as original value parameters.

If this isn't terribly clear at this juncture, don't worry, we'll examine this property and its utility in a future tutorial. For now, just be certain to either remove this property declaration entirely from the declarative syntax or set the value to the default value ({0}).

Note: If you simply clear out the `OldValuesParameterFormatString` property value from the Properties window in the Design view, the property will still exist in the declarative syntax, but be set to an empty string. This, unfortunately, will still result in the same problem discussed above. Therefore, either remove the property altogether from the declarative syntax or, from the Properties window, set the value to the default, `{0}`.

Step 3: Adding a Data Web Control and Configuring It for Data Modification

Once the ObjectDataSource has been added to the page and configured, we're ready to add data Web controls to the page to both display the data and provide a means for the end user to modify it. We'll look at the GridView, DetailsView, and FormView separately, as these data Web controls differ in their data modification capabilities and configuration.

As we'll see in the remainder of this article, adding very basic editing, inserting, and deleting support through the GridView, DetailsView, and FormView controls is really as simple as checking a couple of checkboxes. There are many subtleties and edge cases in the real-world that make providing such functionality more involved than just point and click. This tutorial, however, focuses solely on proving simplistic data modification capabilities. Future tutorials will examine concerns that will undoubtedly arise in a real-world setting.

Deleting Data from the GridView

Start by dragging a GridView from the Toolbox onto the Designer. Next, bind the ObjectDataSource to the GridView by selecting it from the drop-down list in the GridView's smart tag. At this point the GridView's declarative markup will be:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="ProductID" DataSourceID="ObjectDataSource1">
    <Columns>
        <asp:BoundField DataField="ProductID" HeaderText="ProductID"
            InsertVisible="False"
            ReadOnly="True" SortExpression="ProductID" />
        <asp:BoundField DataField="ProductName" HeaderText="ProductName"
            SortExpression="ProductName" />
        <asp:BoundField DataField="SupplierID" HeaderText="SupplierID"
           SortExpression="SupplierID" />
        <asp:BoundField DataField="CategoryID" HeaderText="CategoryID"
           SortExpression="CategoryID" />
        <asp:BoundField DataField="QuantityPerUnit"
           HeaderText="QuantityPerUnit"
           SortExpression="QuantityPerUnit" />
        <asp:BoundField DataField="UnitPrice" HeaderText="UnitPrice"
           SortExpression="UnitPrice" />
        <asp:BoundField DataField="UnitsInStock"
           HeaderText="UnitsInStock" SortExpression="UnitsInStock" />
        <asp:BoundField DataField="UnitsOnOrder"
           HeaderText="UnitsOnOrder" SortExpression="UnitsOnOrder" />
        <asp:BoundField DataField="ReorderLevel"
           HeaderText="ReorderLevel" SortExpression="ReorderLevel" />
        <asp:CheckBoxField DataField="Discontinued"
           HeaderText="Discontinued" SortExpression="Discontinued" />
        <asp:BoundField DataField="CategoryName"
           HeaderText="CategoryName" ReadOnly="True"
            SortExpression="CategoryName" />
        <asp:BoundField DataField="SupplierName"
            HeaderText="SupplierName" ReadOnly="True"
            SortExpression="SupplierName" />
    </Columns>
</asp:GridView>
```

Binding the GridView to the ObjectDataSource through its smart tag has two benefits:

- BoundFields and CheckBoxFields are automatically created for each of the fields returned by the ObjectDataSource. Moreover, the BoundField and CheckBoxField's properties are set based on the underlying field's metadata. For example, the `ProductID`, `CategoryName`, and `SupplierName` fields are marked as read-only in the `ProductsDataTable` and therefore shouldn't be updatable when editing. To accommodate this, these BoundFields' ReadOnly properties are set to `true`.
- The DataKeyNames property is assigned to the primary key field(s) of the underlying object. This is essential when using the GridView for editing or deleting data, as this property indicates the field (or set of fields) that unique identifies each record. For more information on the `DataKeyNames` property, refer back to the Master/Detail Using a Selectable Master GridView with a Details DetailView tutorial.

While the GridView can be bound to the ObjectDataSource through the Properties window or declarative syntax, doing so requires you to manually add the appropriate BoundField and `DataKeyNames` markup.

The GridView control provides built-in support for row-level editing and deleting. Configuring a GridView to support deleting adds a column of Delete buttons. When the end user clicks the Delete button for a particular row, a postback ensues and the GridView performs the following steps:

1. The ObjectDataSource's `DeleteParameters` value(s) are assigned
2. The ObjectDataSource's `Delete()` method is invoked, deleting the specified record
3. The GridView rebinds itself to the ObjectDataSource by invoking its `Select()` method

The values assigned to the `DeleteParameters` are the values of the `DataKeyNames` field(s) for the row whose Delete button was clicked. Therefore it's vital that a GridView's `DataKeyNames` property be correctly set. If it's missing, the `DeleteParameters` will be assigned a `null` value in Step 1, which in turn will not result in any deleted records in Step 2.

> **Note:** The `DataKeys` collection is stored in the GridView's control state, meaning that the `DataKeys` values will be remembered across postback even if the GridView's view state has been disabled. However, it is very important that view state remains enabled for GridViews that support editing or deleting (the default behavior). If you set the GridView's `EnableViewState` property to `false`, the editing and deleting behavior will work fine for a single user, but if there are concurrent users deleting data, there exists the possibility that these concurrent users may accidentally delete or edit records that they didn't intend. See my blog entry, WARNING: Concurrency Issue with ASP.NET 2.0 GridViews/DetailsView/FormViews that Support Editing and/or Deleting and Whose View State is Disabled, for more information.
>
> This same warning also applies to DetailsViews and FormViews.

To add deleting capabilities to a GridView, simply go to its smart tag and check the Enable Deleting checkbox.

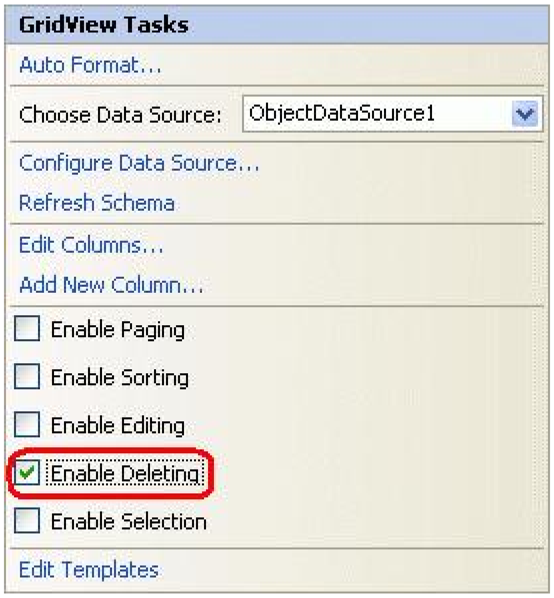


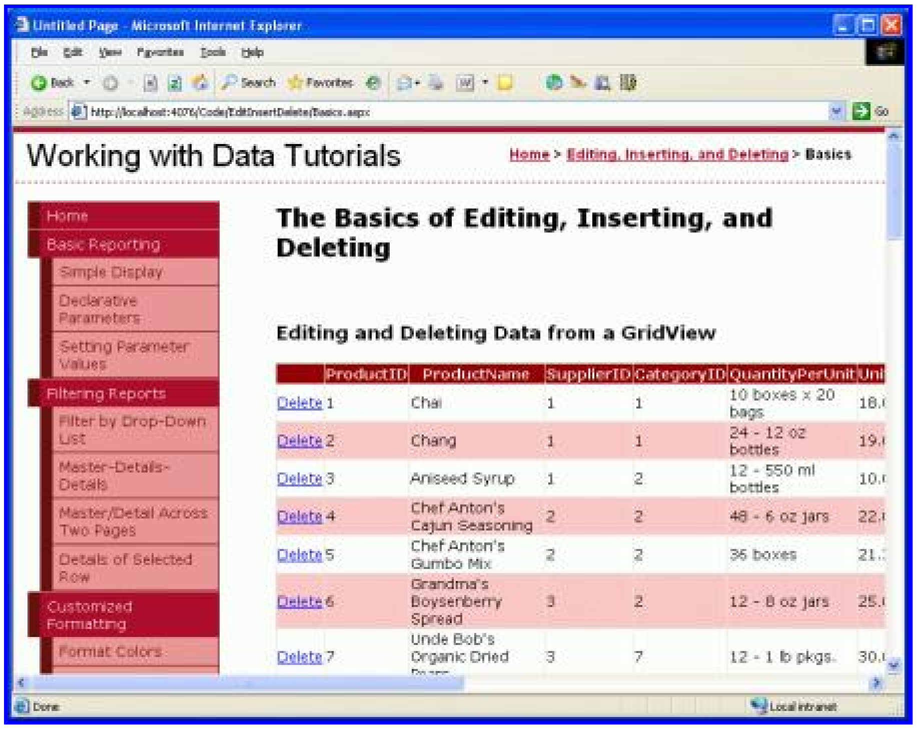
Figure 10: Check the Enable Deleting Checkbox

Checking the Enable Deleting checkbox from the smart tag adds a CommandField to the GridView. The CommandField renders a column in the GridView with buttons for performing one or more of the following tasks: selecting a record, editing a record, and deleting a record. We previously saw the CommandField in action with selecting records in the Master/Detail Using a Selectable Master GridView with a Details DetailView tutorial.

The CommandField contains a number of `ShowXButton` properties that indicate what series of buttons are displayed in the CommandField. By checking the Enable Deleting checkbox a CommandField whose `ShowDeleteButton` property is `true` has been added to the GridView's Columns collection.

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="ProductID" DataSourceID="ObjectDataSource1">
    <Columns>
        <asp:CommandField ShowDeleteButton="True" />
        ... BoundFields removed for brevity ...
    </Columns>
</asp:GridView>
```

At this point, believe it or not, we're done with adding deleting support to the GridView! As Figure 11 shows, when visiting this page through a browser a column of Delete buttons is present.



The Basics of Editing, Inserting, and Deleting

Editing and Deleting Data from a GridView

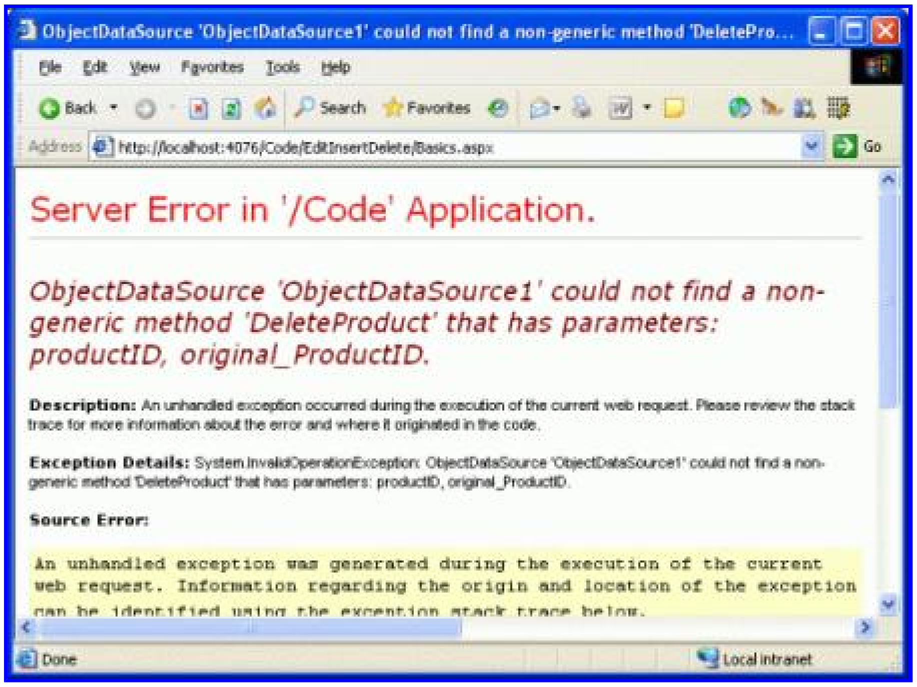
	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	Uni
Delete	1	Chai	1	1	10 boxes x 20 bags	18.
Delete	2	Chang	1	1	24 - 12 oz bottles	19.
Delete	3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.
Delete	4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	22.
Delete	5	Chef Anton's Gumbo Mix	2	2	36 boxes	21.
Delete	6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	25.
Delete	7	Uncle Bob's Organic Dried	3	7	12 - 1 lb pkgs.	30.

Figure 11: The CommandField Adds a Column of Delete Buttons

If you've been building this tutorial from the ground up on your own, when testing this page clicking the Delete button will raise an exception. Continue reading to learn as to why these exceptions were raised and how to fix them.

Note: If you're following along using the download accompanying this tutorial, these problems have already been accounted for. However, I encourage you to read through the details listed below to help identify problems that may arise and suitable workarounds.

If, when attempting to delete a product, you get an exception whose message is similar to "*ObjectDataSource 'ObjectDataSource1' could not find a non-generic method 'DeleteProduct' that has parameters: productID, original_ProductID*," you likely forgot to remove the `OldValuesParameterFormatString` property from the ObjectDataSource. With the `OldValuesParameterFormatString` property specified, the ObjectDataSource attempts to pass in both `productID` and `original_ProductID` input parameters to the `DeleteProduct` method. `DeleteProduct`, however, only accepts a single input parameter, hence the exception. Removing the `OldValuesParameterFormatString` property (or setting it to `{0}`) instructs the ObjectDataSource to not attempt to pass in the original input parameter.


ObjectDataSource 'ObjectDataSource1' could not find a non-generic method 'DeletePro...

http://localhost:4076/Code/EditInsertDelete/Basics.aspx

Server Error in '/Code' Application.

ObjectDataSource 'ObjectDataSource1' could not find a non-generic method 'DeleteProduct' that has parameters: productID, original_ProductID.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

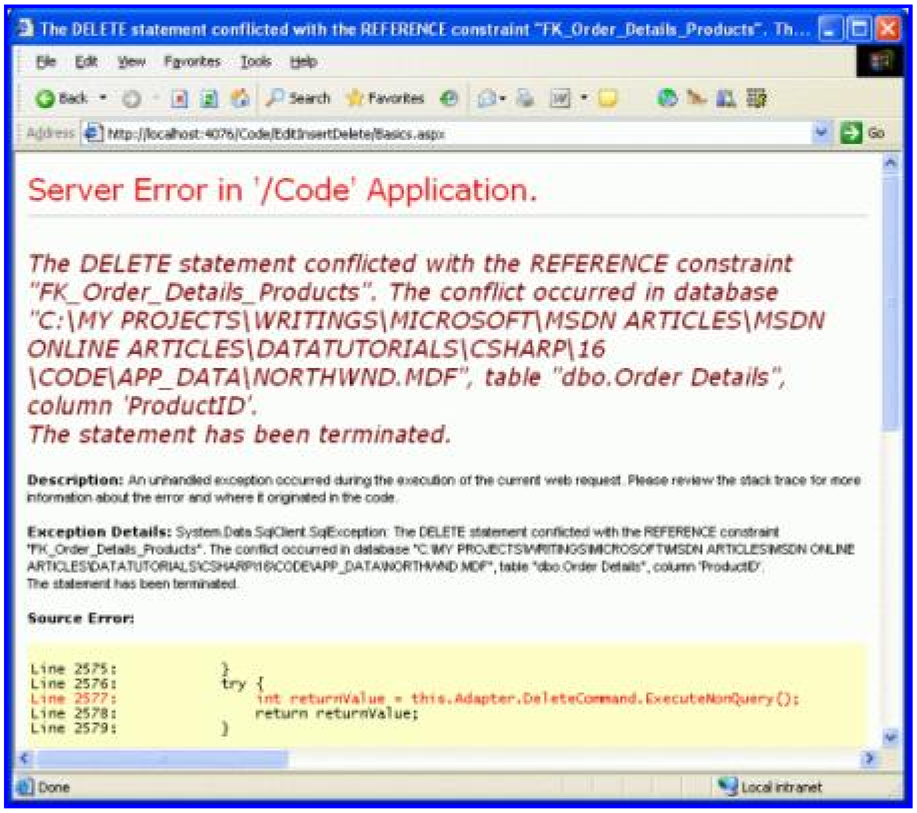
Exception Details: System.InvalidOperationException: ObjectDataSource 'ObjectDataSource1' could not find a non-generic method 'DeleteProduct' that has parameters: productID, original_ProductID.

Source Error:

```
An unhandled exception was generated during the execution of the current
web request. Information regarding the origin and location of the exception
can be identified using the exception stack trace below.
```

Figure 12: Ensure That the `OldValuesParameterFormatString` Property Has Been Cleared Out

Even if you had removed the `OldValuesParameterFormatString` property, you still will get an exception when trying to delete a product with the message: "*The DELETE statement conflicted with the REFERENCE constraint 'FK_Order_Details_Products'*." The Northwind database contains a foreign key constraint between the `Order Details` and `Products` table, meaning that a product cannot be deleted from the system if there are one or more records for it in the `Order Details` table. Since every product in the Northwind database has at least one record in `Order Details`, we cannot delete any products until we first delete the product's associated order details records.


The DELETE statement conflicted with the REFERENCE constraint "FK_Order_Details_Products". Th...
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:4076/Code/EditInsertDelete/Basics.aspx
Go
Done
Local intranet

Server Error in '/Code' Application.

The DELETE statement conflicted with the REFERENCE constraint "FK_Order_Details_Products". The conflict occurred in database "C:\MY PROJECTS\WRITINGS\MICROSOFT\MSDN ARTICLES\MSDN ONLINE ARTICLES\DATATUTORIALS\CSHARP\16 \CODE\APP_DATA\NORTHWND.MDF", table "dbo.Order Details", column 'ProductID'.
The statement has been terminated.

Description: An unhandled exception occurred during the execution of the current web request. Please review the stack trace for more information about the error and where it originated in the code.

Exception Details: System.Data.SqlClient.SqlException: The DELETE statement conflicted with the REFERENCE constraint "FK_Order_Details_Products". The conflict occurred in database "C:\MY PROJECTS\WRITINGS\MICROSOFT\MSDN ARTICLES\MSDN ONLINE ARTICLES\DATATUTORIALS\CSHARP\16\CODE\APP_DATA\NORTHWND.MDF", table "dbo.Order Details", column 'ProductID'.
The statement has been terminated.

Source Error:

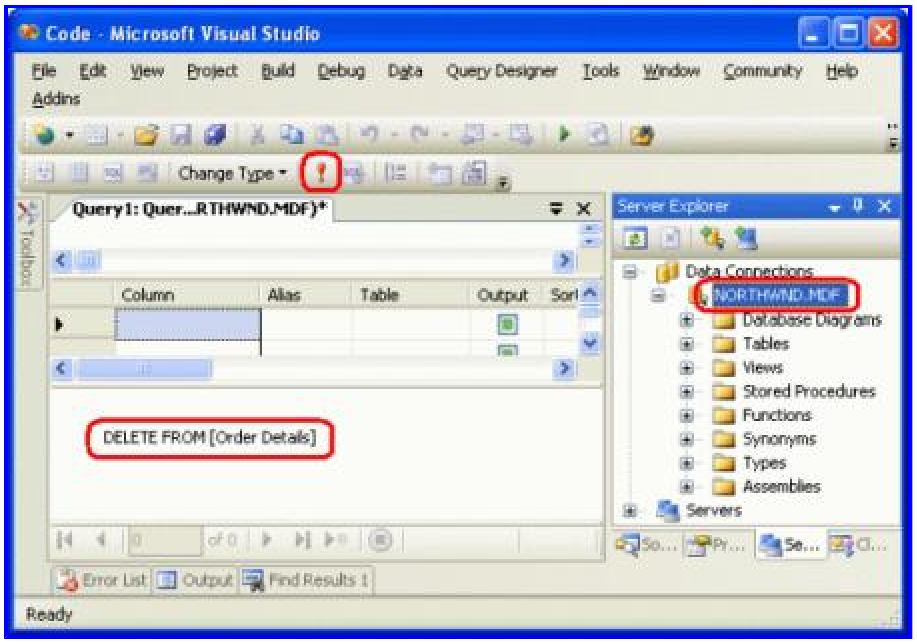
```
Line 2575:            }
Line 2576:            try {
Line 2577:                int returnValue = this.Adapter.DeleteCommand.ExecuteNonQuery();
Line 2578:                return returnValue;
Line 2579:            }
```

Figure 13: A Foreign Key Constraint Prohibits the Deletion of Products

For our tutorial, let's just delete all of the records from the `Order Details` table. In a real-world application we'd need to either:

- Have another screen to manage order details information
- Augment the `DeleteProduct` method to include logic to delete the specified product's order details
- Modify the SQL query used by the TableAdapter to include deletion of the specified product's order details

Let's just delete all of the records from the `Order Details` table to circumvent the foreign key constraint. Go to the Server Explorer in Visual Studio, right-click on the `NORTHWND.MDF` node, and choose New Query. Then, in the query window, run the following SQL statement: `DELETE FROM [Order Details]`


Code - Microsoft Visual Studio
File Edit View Project Build Debug Data Query Designer Tools Window Community Help Addins
Change Type
Query1: Quer...RTHWND.MDF)*
Column Alias Table Output Sort
Server Explorer
Data Connections
NORTHWND.MDF
Database Diagrams
Tables
Views
Stored Procedures
Functions
Synonyms
Types
Assemblies
Servers
Error List Output Find Results 1
Ready

DELETE FROM [Order Details]

Figure 14: Delete All Records from the `Order Details` Table

After clearing out the `Order Details` table clicking on the Delete button will delete the product without error. If clicking on the Delete button does not delete the product, check to ensure that the GridView's `DataKeyNames` property is set to the primary key field (`ProductID`).

Note: When clicking on the Delete button a postback ensues and the record is deleted. This can be dangerous since it is easy to accidentally click on the wrong row's Delete button. In a future tutorial we'll see how to add a client-side confirmation when deleting a record.

Editing Data with the GridView

Along with deleting, the GridView control also provides built-in row-level editing support. Configuring a GridView to support editing adds a column of Edit buttons. From the end user's perspective, clicking a row's Edit button causes that row to become editable, turning the cells into textboxes containing the existing values and replacing the Edit button with Update and Cancel buttons. After making their desired changes, the end user can click the Update button to commit the changes or the Cancel button to discard them. In either case, after clicking Update or Cancel the GridView returns to its pre-editing state.

From our perspective as the page developer, when the end user clicks the Edit button for a particular row, a postback ensues and the GridView performs the following steps:

1. The GridView's `EditItemIndex` property is assigned to the index of the row whose Edit button was clicked
2. The GridView rebinds itself to the ObjectDataSource by invoking its `Select()` method
3. The row index that matches the `EditItemIndex` is rendered in "edit mode." In this mode, the Edit button is replaced by Update and Cancel buttons and BoundFields whose `ReadOnly` properties are False (the default) are rendered as TextBox Web controls whose `Text` properties are assigned to the data fields' values.

At this point the markup is returned to the browser, allowing the end user to make any changes to the row's data. When the user clicks the Update button, a postback occurs and the GridView performs the following steps:

1. The ObjectDataSource's `UpdateParameters` value(s) are assigned the values entered by the end user into the GridView's editing interface
2. The ObjectDataSource's `Update()` method is invoked, updating the specified record
3. The GridView rebinds itself to the ObjectDataSource by invoking its `Select()` method

The primary key values assigned to the `UpdateParameters` in Step 1 come from the values specified in the `DataKeyNames` property, whereas the non-primary key values come from the text in the TextBox Web controls for the edited row. As with deleting, it is vital that a GridView's `DataKeyNames` property be correctly set. If it's missing, the `UpdateParameters` primary key value will be assigned a `null` value in Step 1, which in turn will not result in any updated records in Step 2.

Editing functionality can be activated by simply checking the Enable Editing checkbox in the GridView's smart tag.

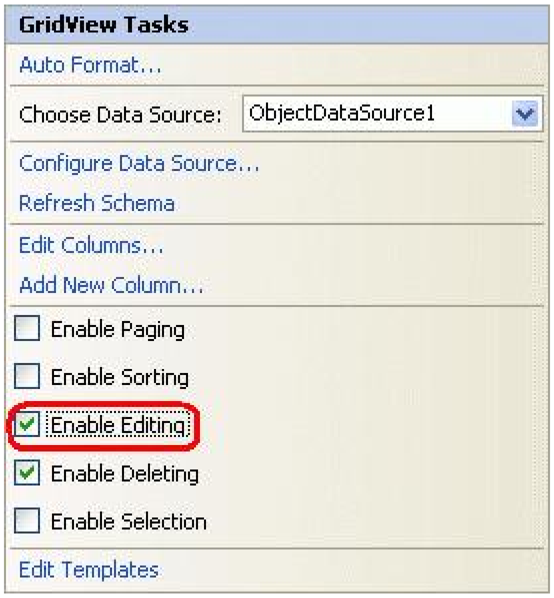
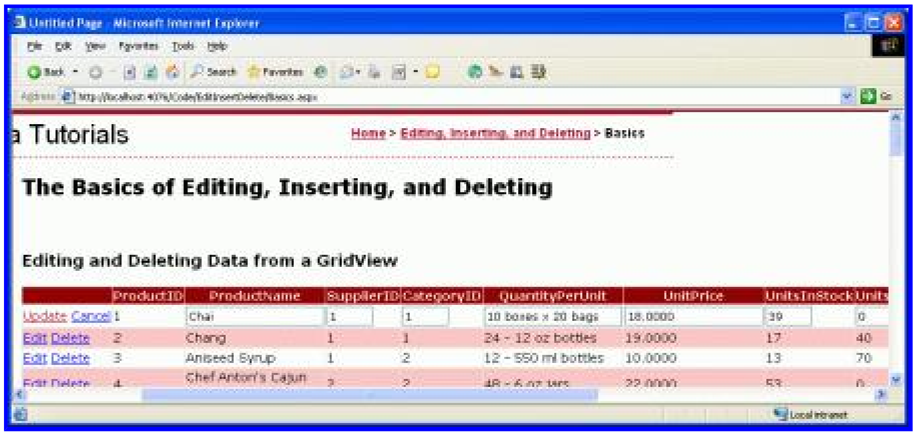


Figure 15: Check the Enable Editing Checkbox

Checking the Enable Editing checkbox will add a CommandField (if needed) and set its `ShowEditButton` property to `true`. As we saw earlier, the CommandField contains a number of `Show`*`X`*`Button` properties that indicate what series of buttons are displayed in the CommandField. Checking the Enable Editing checkbox adds the `ShowEditButton` property to the existing CommandField:

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
    DataKeyNames="ProductID" DataSourceID="ObjectDataSource1">
    <Columns>
        <asp:CommandField ShowDeleteButton="True"
            ShowEditButton="True" />
        ... BoundFields removed for brevity ...
    </Columns>
</asp:GridView>
```

That's all there is to adding rudimentary editing support. As Figure16 shows, the editing interface is rather crude – each BoundField whose `ReadOnly` property is set to `false` (the default) is rendered as a TextBox. This includes fields like `CategoryID` and `SupplierID`, which are keys to other tables.


Untitled Page - Microsoft Internet Explorer
a Tutorials
Home > Editing, Inserting, and Deleting > Basics
Local intranet

The Basics of Editing, Inserting, and Deleting

Editing and Deleting Data from a GridView

	ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit	UnitPrice	UnitsInStock	Units
Update Cancel	1	Chai	1	1	10 boxes x 20 bags	18.0000	39	0
Edit Delete	2	Chang	1	1	24 - 12 oz bottles	19.0000	17	40
Edit Delete	3	Aniseed Syrup	1	2	12 - 550 ml bottles	10.0000	13	70
Edit Delete	4	Chef Anton's Cajun	2	2	48 - 6 oz jars	22.0000	53	0

Figure 16: Clicking Chai's Edit Button Displays the Row in "Edit Mode"

In addition to asking users to edit foreign key values directly, the editing interface's interface is lacking in the following ways:

- If the user enters a `CategoryID` or `SupplierID` that does not exist in the database, the `UPDATE` will violate a foreign key constraint, causing an exception to be raised.
- The editing interface doesn't include any validation. If you don't provide a required value (such as `ProductName`), or enter a string value where a numeric value is expected (such as entering "Too much!" into the `UnitPrice` textbox), an exception will be thrown. A future tutorial will examine how to add validation controls to the editing user interface.
- Currently, *all* product fields that are not read-only must be included in the GridView. If we were to remove a field from the GridView, say `UnitPrice`, when updating the data the GridView would not set the `UnitPrice` `UpdateParameters` value, which would change the database record's `UnitPrice` to a `NULL` value. Similarly, if a required field, such as `ProductName`, is removed from the GridView, the update will fail with the same "*Column 'ProductName' does not allow nulls*" exception mentioned above.
- The editing interface formatting leaves a lot to be desired. The `UnitPrice` is shown with four decimal points. Ideally the `CategoryID` and `SupplierID` values would contain DropDownLists that list the categories and suppliers in the system.

These are all shortcomings that we'll have to live with for now, but will be addressed in future tutorials.

Inserting, Editing, and Deleting Data with the DetailsView

As we've seen in earlier tutorials, the DetailsView control displays one record at a time and, like the GridView, allows for editing and deleting of the currently displayed record. Both the end user's experience with editing and deleting items from a DetailsView and the workflow from the ASP.NET side is identical to that of the GridView. Where the DetailsView differs from the GridView is that it also provides built-in inserting support.

To demonstrate the data modification capabilities of the GridView, start by adding a DetailsView to the `Basics.aspx` page above the existing GridView and bind it to the existing ObjectDataSource through the DetailsView's smart tag. Next, clear out the DetailsView's `Height` and `Width` properties, and check the Enable Paging option from the smart tag. To enable editing, inserting, and deleting support, simply check the Enable Editing, Enable Inserting, and Enable Deleting checkboxes in the smart tag.

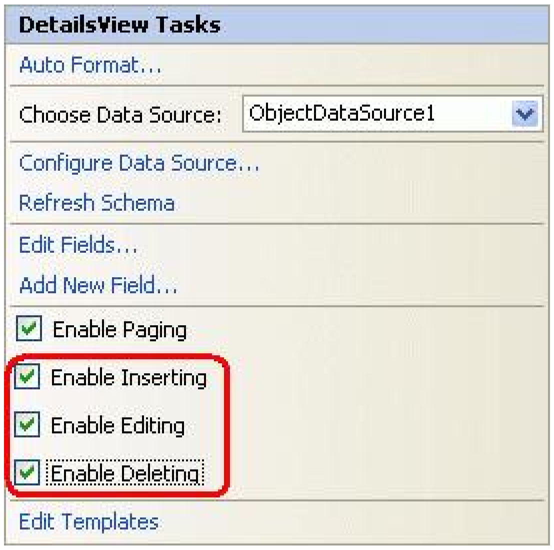
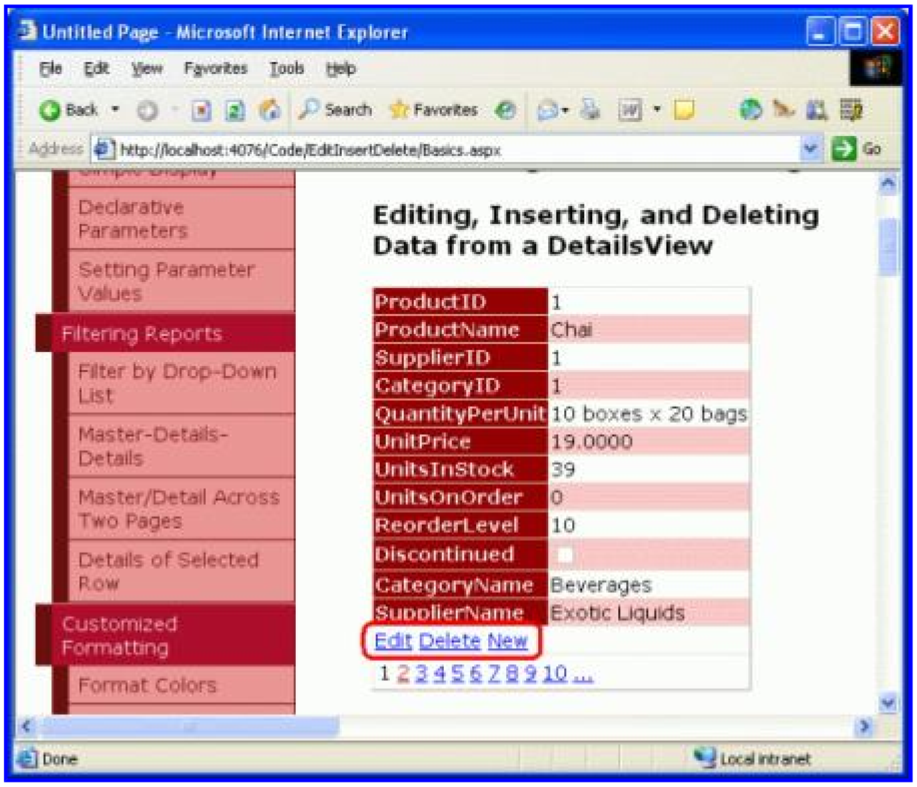


Figure 17: Configure the DetailsView to Support Editing, Inserting, and Deleting

As with the GridView, adding editing, inserting, or deleting support adds a CommandField to the DetailsView, as the following declarative syntax shows:

```
<asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"
    DataKeyNames="ProductID" DataSourceID="ObjectDataSource1" AllowPaging="True">
    <Fields>
        <asp:BoundField DataField="ProductID"
            HeaderText="ProductID" InsertVisible="False"
            ReadOnly="True" SortExpression="ProductID" />
        <asp:BoundField DataField="ProductName"
            HeaderText="ProductName" SortExpression="ProductName" />
        <asp:BoundField DataField="SupplierID" HeaderText="SupplierID"
            SortExpression="SupplierID" />
        <asp:BoundField DataField="CategoryID" HeaderText="CategoryID"
            SortExpression="CategoryID" />
        <asp:BoundField DataField="QuantityPerUnit"
            HeaderText="QuantityPerUnit"
            SortExpression="QuantityPerUnit" />
        <asp:BoundField DataField="UnitPrice"
            HeaderText="UnitPrice" SortExpression="UnitPrice" />
        <asp:BoundField DataField="UnitsInStock"
            HeaderText="UnitsInStock" SortExpression="UnitsInStock" />
        <asp:BoundField DataField="UnitsOnOrder"
            HeaderText="UnitsOnOrder" SortExpression="UnitsOnOrder" />
        <asp:BoundField DataField="ReorderLevel"
            HeaderText="ReorderLevel" SortExpression="ReorderLevel" />
        <asp:CheckBoxField DataField="Discontinued"
            HeaderText="Discontinued" SortExpression="Discontinued" />
        <asp:BoundField DataField="CategoryName"
            HeaderText="CategoryName" ReadOnly="True"
            SortExpression="CategoryName" />
        <asp:BoundField DataField="SupplierName"
            HeaderText="SupplierName" ReadOnly="True"
            SortExpression="SupplierName" />
        <asp:CommandField ShowDeleteButton="True"
            ShowEditButton="True" ShowInsertButton="True" />
    </Fields>
</asp:DetailsView>
```

Note that for the DetailsView the CommandField appears at the end of the Columns collection by default. Since the DetailsView's fields are rendered as rows, the CommandField appears as a row with Insert, Edit, and Delete buttons at the bottom of the DetailsView.



Editing, Inserting, and Deleting Data from a DetailsView

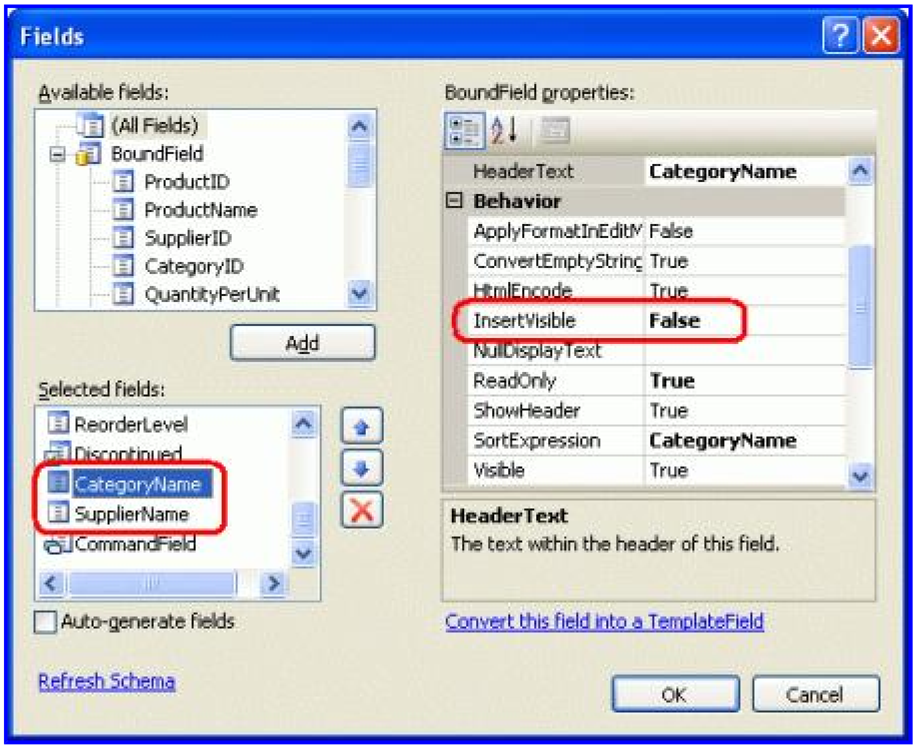
ProductID	1
ProductName	Chai
SupplierID	1
CategoryID	1
QuantityPerUnit	10 boxes x 20 bags
UnitPrice	19.0000
UnitsInStock	39
UnitsOnOrder	0
ReorderLevel	10
Discontinued	☐
CategoryName	Beverages
SupplierName	Exotic Liquids
Edit Delete New	
1 2 3 4 5 6 7 8 9 10 ...	

Figure 18: Configure the DetailsView to Support Editing, Inserting, and Deleting

Clicking on the Delete button starts the same sequence of events as with the GridView: a postback; followed by the DetailsView populating its ObjectDataSource's `DeleteParameters` based on the `DataKeyNames` values; and completed with a call its ObjectDataSource's `Delete()` method, which actually removes the product from the database. Editing in the DetailsView also works in a fashion identical to that of the GridView.

For inserting, the end user is presented with a New button that, when clicked, renders the DetailsView in "insert mode." With "insert mode" the New button is replaced by Insert and Cancel buttons and only those BoundFields whose `InsertVisible` property is set to `true` (the default) are displayed. Those data fields identified as auto-increment fields, such as `ProductID`, have their InsertVisible property set to `false` when binding the DetailsView to the data source through the smart tag.

When binding a data source to a DetailsView through the smart tag, Visual Studio sets the `InsertVisible` property to `false` only for auto-increment fields. Read-only fields, like `CategoryName` and `SupplierName`, will be displayed in the "insert mode" user interface unless their `InsertVisible` property is explicitly set to `false`. Take a moment to set these two fields' `InsertVisible` properties to `false`, either through the DetailsView's declarative syntax or through the Edit Fields link in the smart tag. Figure 19 shows setting the `InsertVisible` properties to `false` by clicking on the Edit Fields link.

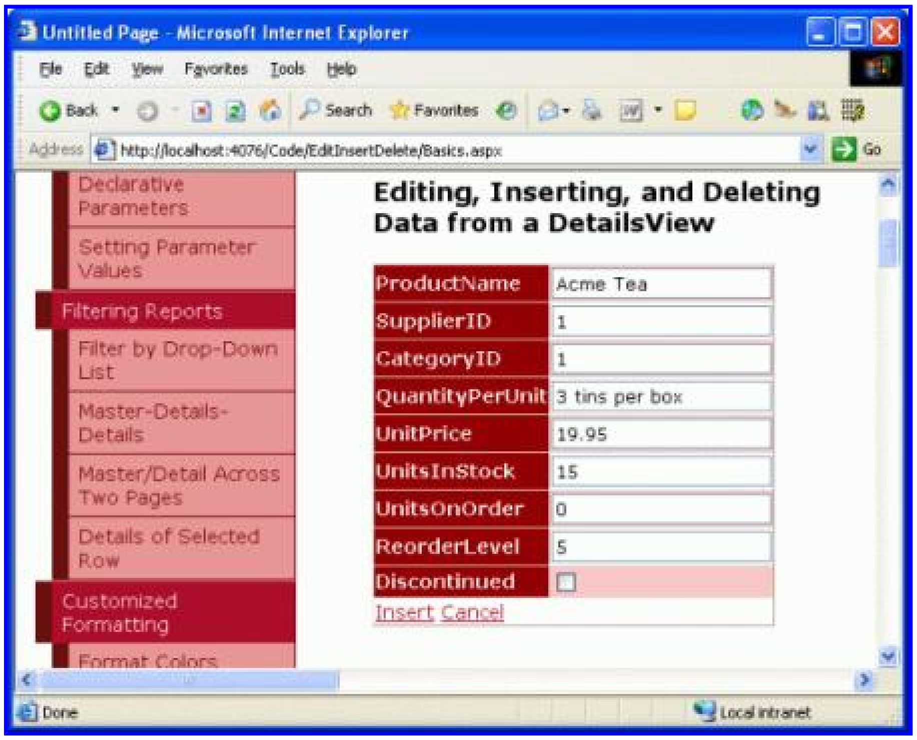


HeaderText	CategoryName
Behavior	
ApplyFormatInEditM	False
ConvertEmptyStrinc	True
HtmlEncode	True
InsertVisible	False
NullDisplayText	
ReadOnly	True
ShowHeader	True
SortExpression	CategoryName
Visible	True

HeaderText
The text within the header of this field.

Figure 19: Northwind Traders Now Offers Acme Tea

After setting the `InsertVisible` properties, view the `Basics.aspx` page in a browser and click the New button. Figure 20 shows the DetailsView when adding a new beverage, Acme Tea, to our product line.



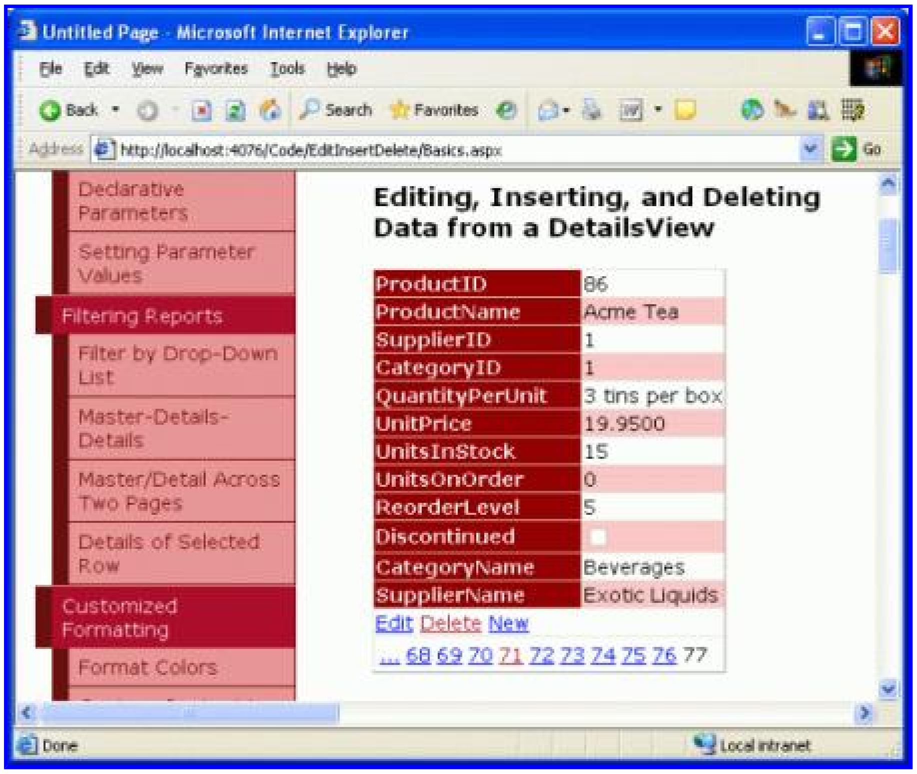
Editing, Inserting, and Deleting Data from a DetailsView

ProductName	Acme Tea
SupplierID	1
CategoryID	1
QuantityPerUnit	3 tins per box
UnitPrice	19.95
UnitsInStock	15
UnitsOnOrder	0
ReorderLevel	5
Discontinued	☐

Insert Cancel

Figure 20: Northwind Traders Now Offers Acme Tea

After entering the details for Acme Tea and clicking the Insert button, a postback ensues and the new record is added to the `Products` database table. Since this DetailsView lists the products in order with which they exist in the database table, we must page to the last product in order to see the new product.



Editing, Inserting, and Deleting Data from a DetailsView

ProductID	86
ProductName	Acme Tea
SupplierID	1
CategoryID	1
QuantityPerUnit	3 tins per box
UnitPrice	19.9500
UnitsInStock	15
UnitsOnOrder	0
ReorderLevel	5
Discontinued	☐
CategoryName	Beverages
SupplierName	Exotic Liquids

Edit Delete New

... 68 69 70 71 72 73 74 75 76 77

Figure 21: Details for Acme Tea

Note: The DetailsView's CurrentMode property indicates the interface being displayed and can be one of the following values: `Edit`, `Insert`, or `ReadOnly`. The DefaultMode property indicates the mode the DetailsView returns to after an edit or insert has been completed and is useful for displaying a DetailsView that is permanently in edit or insert mode.

The point and click inserting and editing capabilities of the DetailsView suffer from the same limitations as the GridView: the user must enter existing `CategoryID` and `SupplierID` values through a textbox; the interface lacks any validation logic; all product fields that do not allow `NULL` values or don't have a default value specified at the database level must be included in the inserting interface, and so on.

The techniques we will examine for extending and enhancing the GridView's editing interface in future articles can be applied to the DetailsView control's editing and inserting interfaces as well.

Using the FormView for a More Flexible Data Modification User Interface

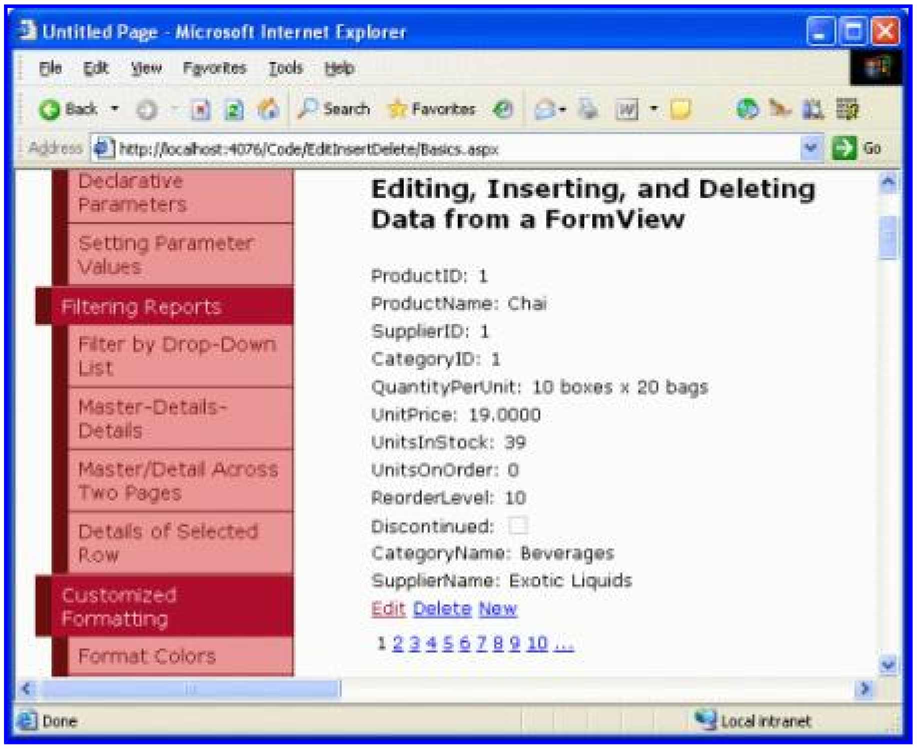
The FormView offers built-in support for inserting, editing, and deleting data, but because it uses templates instead of fields there's no place to add the BoundFields or the CommandField used by the GridView and DetailsView controls to provide the data modification interface. Instead, this interface – the Web controls for collecting user input when adding a new item or editing an existing one along with the New, Edit, Delete, Insert, Update, and Cancel buttons – must be added manually to the appropriate templates. Fortunately, Visual Studio will automatically create the needed interface when binding the FormView to a data source through the drop-down list in its smart tag.

To illustrate these techniques, start by adding a FormView to the `Basics.aspx` page and, from the FormView's smart tag, bind it to the ObjectDataSource already created. This will generate an `EditItemTemplate`, `InsertItemTemplate`, and `ItemTemplate` for the FormView with TextBox Web controls for collecting the user's input and Button Web controls for the New, Edit, Delete, Insert, Update, and Cancel buttons. Additionally, the FormView's `DataKeyNames` property is set to the primary key field (`ProductID`) of the object returned by the ObjectDataSource. Lastly, check the Enable Paging option in the FormView's smart tag.

The following shows the declarative markup for the FormView's `ItemTemplate` after the FormView has been bound to the ObjectDataSource. By default, each non-Boolean value product field is bound to the `Text` property of a Label Web control while each Boolean value field (`Discontinued`) is bound to the `Checked` property of a disabled CheckBox Web control. In order for the New, Edit, and Delete buttons to trigger certain FormView behavior when clicked, it is imperative that their `CommandName` values be set to `New`, `Edit`, and `Delete`, respectively.

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="ProductID"
    DataSourceID="ObjectDataSource1" AllowPaging="True">
    <EditItemTemplate>
        ...
    </EditItemTemplate>
    <InsertItemTemplate>
        ...
    </InsertItemTemplate>
    <ItemTemplate>
        ProductID:
        <asp:Label ID="ProductIDLabel" runat="server"
            Text='<%# Eval("ProductID") %>'></asp:Label><br />
        ProductName:
        <asp:Label ID="ProductNameLabel" runat="server"
            Text='<%# Bind("ProductName") %>'>
        </asp:Label><br />
        SupplierID:
        <asp:Label ID="SupplierIDLabel" runat="server"
            Text='<%# Bind("SupplierID") %>'>
        </asp:Label><br />
        CategoryID:
        <asp:Label ID="CategoryIDLabel" runat="server"
            Text='<%# Bind("CategoryID") %>'>
        </asp:Label><br />
        QuantityPerUnit:
        <asp:Label ID="QuantityPerUnitLabel" runat="server"
            Text='<%# Bind("QuantityPerUnit") %>'>
        </asp:Label><br />
        UnitPrice:
        <asp:Label ID="UnitPriceLabel" runat="server"
            Text='<%# Bind("UnitPrice") %>'></asp:Label><br />
        UnitsInStock:
        <asp:Label ID="UnitsInStockLabel" runat="server"
            Text='<%# Bind("UnitsInStock") %>'>
        </asp:Label><br />
        UnitsOnOrder:
        <asp:Label ID="UnitsOnOrderLabel" runat="server"
            Text='<%# Bind("UnitsOnOrder") %>'>
        </asp:Label><br />
        ReorderLevel:
        <asp:Label ID="ReorderLevelLabel" runat="server"
            Text='<%# Bind("ReorderLevel") %>'>
        </asp:Label><br />
        Discontinued:
        <asp:CheckBox ID="DiscontinuedCheckBox" runat="server"
            Checked='<%# Bind("Discontinued") %>'
            Enabled="false" /><br />
        CategoryName:
        <asp:Label ID="CategoryNameLabel" runat="server"
            Text='<%# Bind("CategoryName") %>'>
        </asp:Label><br />
        SupplierName:
        <asp:Label ID="SupplierNameLabel" runat="server"
            Text='<%# Bind("SupplierName") %>'>
        </asp:Label><br />
        <asp:LinkButton ID="EditButton" runat="server"
            CausesValidation="False" CommandName="Edit"
            Text="Edit">
        </asp:LinkButton>
        <asp:LinkButton ID="DeleteButton" runat="server"
            CausesValidation="False" CommandName="Delete"
            Text="Delete">
        </asp:LinkButton>
        <asp:LinkButton ID="NewButton" runat="server"
            CausesValidation="False" CommandName="New"
            Text="New">
        </asp:LinkButton>
    </ItemTemplate>
</asp:FormView>
```

Figure 22 shows the FormView's `ItemTemplate` when viewed through a browser. Each product field is listed with the New, Edit, and Delete buttons at the bottom.



Editing, Inserting, and Deleting Data from a FormView

ProductID: 1
ProductName: Chai
SupplierID: 1
CategoryID: 1
QuantityPerUnit: 10 boxes x 20 bags
UnitPrice: 19.0000
UnitsInStock: 39
UnitsOnOrder: 0
ReorderLevel: 10
Discontinued: ☐
CategoryName: Beverages
SupplierName: Exotic Liquids
Edit Delete New
1 2 3 4 5 6 7 8 9 10 ...

Figure 22: The Defaut FormView `ItemTemplate` Lists Each Product Field Along with New, Edit, and Delete Buttons

Like with the GridView and DetailsView, clicking the Delete button – or any Button, LinkButton, or ImageButton whose `CommandName` property is set to Delete – causes a postback, populates the ObjectDataSource's `DeleteParameters` based on the FormView's `DataKeyNames` value, and invokes the ObjectDataSource's `Delete()` method.

When the Edit button is clicked a postback ensues and the data is rebound to the `EditItemTemplate`, which is responsible for rendering the editing interface. This interface includes the Web controls for editing data along with the Update and Cancel buttons. The default `EditItemTemplate` generated by Visual Studio contains a Label for any auto-increment fields (`ProductID`), a TextBox for each non-Boolean value field, and a CheckBox for each Boolean value field. This behavior is very similar to the auto-generated BoundFields in the GridView and DetailsView controls.

Note: One small issue with the FormView's auto-generation of the `EditItemTemplate` is that it renders TextBox Web controls for those fields that are read-only, such as `CategoryName` and `SupplierName`. We'll see how to account for this shortly.

The TextBox controls in the `EditItemTemplate` have their `Text` property bound to the value of their corresponding data field using *two-way databinding*. Two-way databinding, denoted by `<%# Bind ("dataField") %>`, performs databinding both when binding data to the template and when populating the ObjectDataSource's parameters for inserting or editing records. That is, when the user clicks the Edit button from the `ItemTemplate`, the `Bind()` method returns the specified data field value. After the user makes their changes and clicks Update, the values posted back that correspond to the data fields specified using `Bind()` are applied to the ObjectDataSource's `UpdateParameters`. Alternatively, one-way databinding, denoted by `<%# Eval("dataField") %>`, only retrieves the data field values when binding data to the template and does *not* return the user-entered values to the data source's parameters on postback.

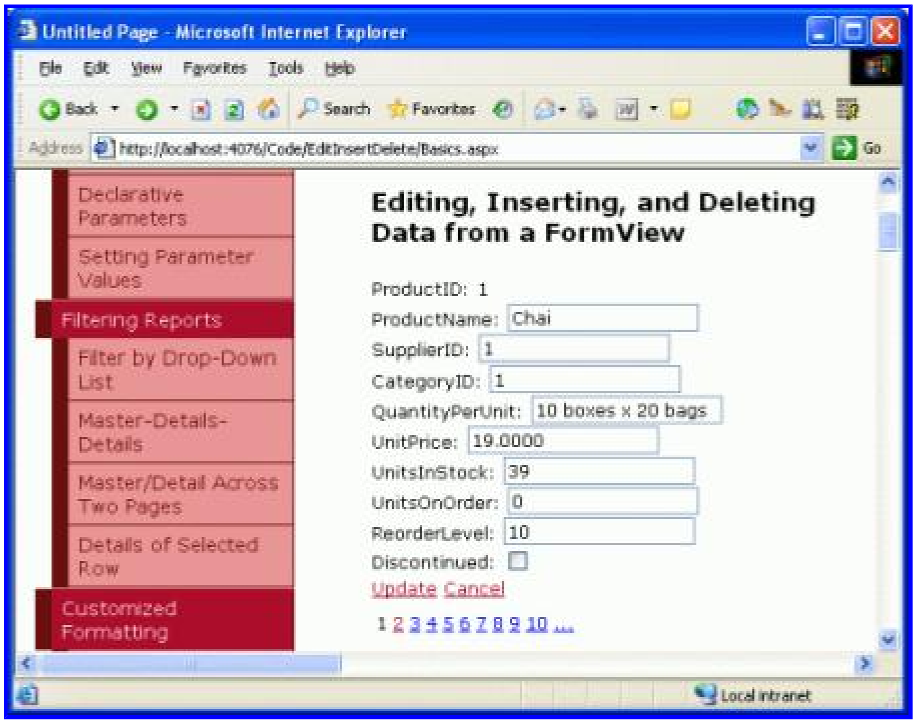
The following declarative markup shows the FormView's `EditItemTemplate`. Note that the `Bind()` method is used in the databinding syntax here and that the Update and Cancel Button Web controls have their `CommandName` properties set accordingly.

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="ProductID"
    DataSourceID="ObjectDataSource1" AllowPaging="True">
    <EditItemTemplate>
        ProductID:
        <asp:Label ID="ProductIDLabel1" runat="server"
          Text="<%# Eval("ProductID") %>"></asp:Label><br />
        ProductName:
        <asp:TextBox ID="ProductNameTextBox" runat="server"
          Text="<%# Bind("ProductName") %>">
        </asp:TextBox><br />
        SupplierID:
        <asp:TextBox ID="SupplierIDTextBox" runat="server"
          Text="<%# Bind("SupplierID") %>">
        </asp:TextBox><br />
        CategoryID:
        <asp:TextBox ID="CategoryIDTextBox" runat="server"
          Text="<%# Bind("CategoryID") %>">
        </asp:TextBox><br />
        QuantityPerUnit:
        <asp:TextBox ID="QuantityPerUnitTextBox" runat="server"
           Text="<%# Bind("QuantityPerUnit") %>">
        </asp:TextBox><br />
        UnitPrice:
        <asp:TextBox ID="UnitPriceTextBox" runat="server"
           Text="<%# Bind("UnitPrice") %>">
        </asp:TextBox><br />
        UnitsInStock:
        <asp:TextBox ID="UnitsInStockTextBox" runat="server"
           Text="<%# Bind("UnitsInStock") %>">
        </asp:TextBox><br />
        UnitsOnOrder:
        <asp:TextBox ID="UnitsOnOrderTextBox" runat="server"
           Text="<%# Bind("UnitsOnOrder") %>">
        </asp:TextBox><br />
        ReorderLevel:
```

```
        <asp:TextBox ID="ReorderLevelTextBox" runat="server"
           Text="<%# Bind("ReorderLevel") %>">
        </asp:TextBox><br />
        Discontinued:
        <asp:CheckBox ID="DiscontinuedCheckBox" runat="server"
            Checked="<%# Bind("Discontinued") %>" /><br />
        CategoryName:
        <asp:TextBox ID="CategoryNameTextBox" runat="server"
             Text="<%# Bind("CategoryName") %>">
        </asp:TextBox><br />
        SupplierName:
        <asp:TextBox ID="SupplierNameTextBox" runat="server"
             Text="<%# Bind("SupplierName") %>">
        </asp:TextBox><br />
        <asp:LinkButton ID="UpdateButton" runat="server"
            CausesValidation="True" CommandName="Update"
            Text="Update">
        </asp:LinkButton>
        <asp:LinkButton ID="UpdateCancelButton" runat="server"
            CausesValidation="False" CommandName="Cancel"
            Text="Cancel">
        </asp:LinkButton>
    </EditItemTemplate>
    <InsertItemTemplate>
        ...
    </InsertItemTemplate>
    <ItemTemplate>
        ...
    </ItemTemplate>
</asp:FormView>
```

Our `EditItemTemplate`, at this point, will cause an exception to be thrown if we attempt to use it. The problem is that the `CategoryName` and `SupplierName` fields are rendered as TextBox Web controls in the `EditItemTemplate`. We either need to change these TextBoxes to Labels or remove them altogether. Let's simply delete them entirely from the `EditItemTemplate`.

Figure 23 shows the FormView in a browser after the Edit button has been clicked for Chai. Note that the `SupplierName` and `CategoryName` fields shown in the `ItemTemplate` are no longer present, as we just removed them from the `EditItemTemplate`. When the Update button is clicked the FormView proceeds through the same sequence of steps as the GridView and DetailsView controls.



Editing, Inserting, and Deleting Data from a FormView

ProductID: 1
ProductName: Chai
SupplierID: 1
CategoryID: 1
QuantityPerUnit: 10 boxes x 20 bags
UnitPrice: 19.0000
UnitsInStock: 39
UnitsOnOrder: 0
ReorderLevel: 10
Discontinued: ☐
Update Cancel

1 2 3 4 5 6 7 8 9 10 ...

Figure 23: By Default the `EditItemTemplate` Shows Each Editable Product Field as a TextBox or CheckBox

When the Insert button is clicked the FormView's `ItemTemplate` a postback ensues. However, no data is bound to the FormView because a new record is being added. The `InsertItemTemplate` interface includes the Web controls for adding a new record along with the Insert and Cancel buttons. The default `InsertItemTemplate` generated by Visual Studio contains a TextBox for each non-Boolean value field and a CheckBox for each Boolean value field, similar to the auto-generated `EditItemTemplate`'s interface. The TextBox controls have their `Text` property bound to the value of their corresponding data field using two-way databinding.

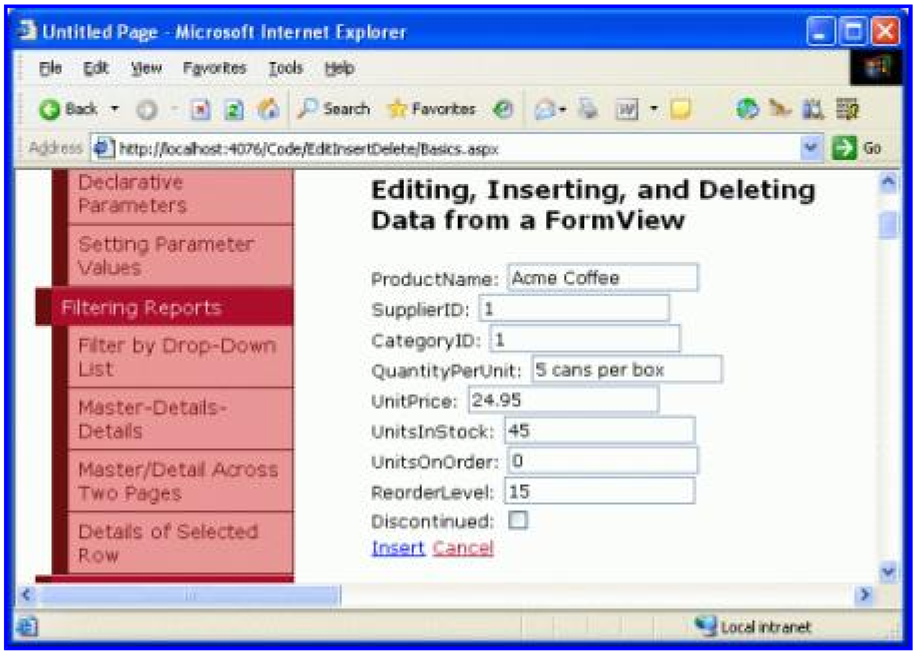
The following declarative markup shows the FormView's `InsertItemTemplate`. Note that the `Bind()` method is used in the databinding syntax here and that the Insert and Cancel Button Web controls have their `CommandName` properties set accordingly.

```
<asp:FormView ID="FormView1" runat="server" DataKeyNames="ProductID"
    DataSourceID="ObjectDataSource1" AllowPaging="True">
    <EditItemTemplate>
        ...
    </EditItemTemplate>
    <InsertItemTemplate>
        ProductName:
        <asp:TextBox ID="ProductNameTextBox" runat="server"
           Text="<%# Bind("ProductName") %>">
        </asp:TextBox><br />
        SupplierID:
        <asp:TextBox ID="SupplierIDTextBox" runat="server"
           Text="<%# Bind("SupplierID") %>">
        </asp:TextBox><br />
        CategoryID:
        <asp:TextBox ID="CategoryIDTextBox" runat="server"
           Text="<%# Bind("CategoryID") %>">
        </asp:TextBox><br />
        QuantityPerUnit:
        <asp:TextBox ID="QuantityPerUnitTextBox" runat="server"
           Text="<%# Bind("QuantityPerUnit") %>">
        </asp:TextBox><br />
        UnitPrice:
        <asp:TextBox ID="UnitPriceTextBox" runat="server"
           Text="<%# Bind("UnitPrice") %>">
        </asp:TextBox><br />
        UnitsInStock:
        <asp:TextBox ID="UnitsInStockTextBox" runat="server"
           Text="<%# Bind("UnitsInStock") %>">
        </asp:TextBox><br />
        UnitsOnOrder:
        <asp:TextBox ID="UnitsOnOrderTextBox" runat="server"
           Text="<%# Bind("UnitsOnOrder") %>">
        </asp:TextBox><br />
        ReorderLevel:
        <asp:TextBox ID="ReorderLevelTextBox" runat="server"
           Text="<%# Bind("ReorderLevel") %>">
        </asp:TextBox><br />
        Discontinued:
        <asp:CheckBox ID="DiscontinuedCheckBox" runat="server"
           Checked="<%# Bind("Discontinued") %>" /><br />
        CategoryName:
        <asp:TextBox ID="CategoryNameTextBox" runat="server"
            Text="<%# Bind("CategoryName") %>">
        </asp:TextBox><br />
        SupplierName:
        <asp:TextBox ID="SupplierNameTextBox" runat="server"
           Text="<%# Bind("SupplierName") %>">
        </asp:TextBox><br />
        <asp:LinkButton ID="InsertButton" runat="server"
            CausesValidation="True" CommandName="Insert"
            Text="Insert">
        </asp:LinkButton>
        <asp:LinkButton ID="InsertCancelButton" runat="server"
            CausesValidation="False" CommandName="Cancel"
            Text="Cancel">
        </asp:LinkButton>
    </InsertItemTemplate>
    <ItemTemplate>
        ...
    </ItemTemplate>

</asp:FormView>
```

There's a subtlety with the FormView's auto-generation of the `InsertItemTemplate`. Specifically, the TextBox Web controls are created even for those fields that are read-only, such as `CategoryName` and `SupplierName`. Like with the `EditItemTemplate`, we need to remove these TextBoxes from the `InsertItemTemplate`.

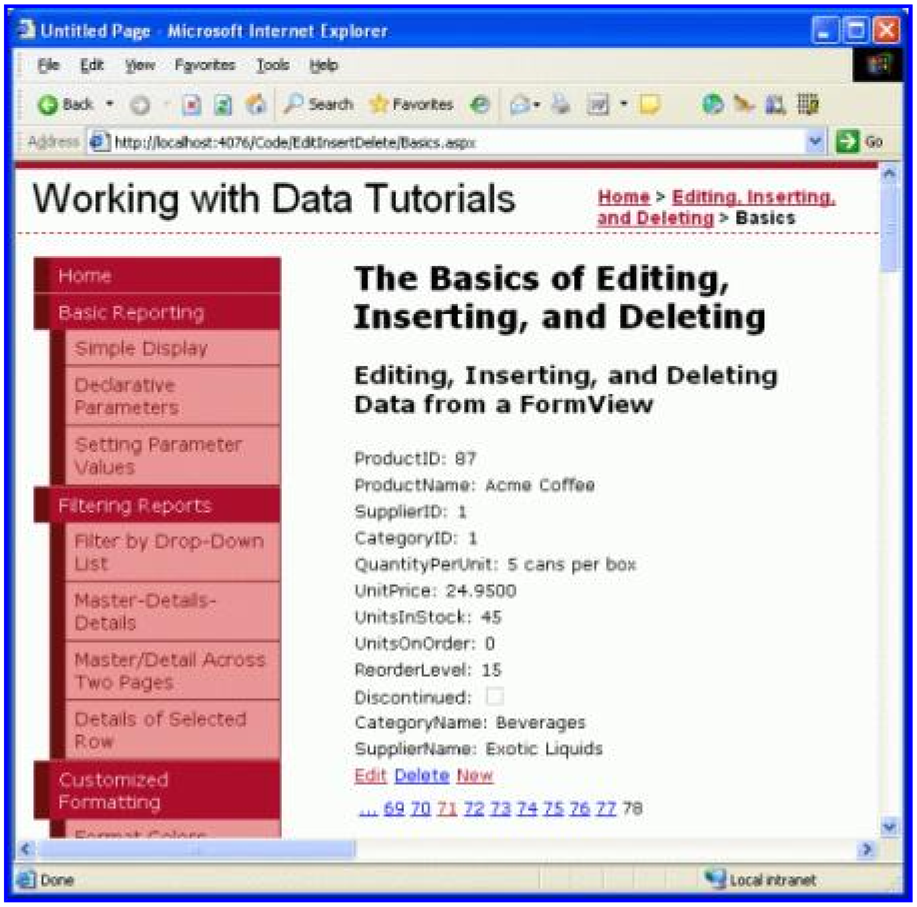
Figure 24 shows the FormView in a browser when adding a new product, Acme Coffee. Note that the `SupplierName` and `CategoryName` fields shown in the `ItemTemplate` are no longer present, as we just removed them. When the Insert button is clicked the FormView proceeds through the same sequence of steps as the DetailsView control, adding a new record to the `Products` table. Figure 25 shows Acme Coffee product's details in the FormView after it has been inserted.


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:4076/Code/EditInsertDelete/Basics.aspx
Go
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Local intranet

Editing, Inserting, and Deleting Data from a FormView

ProductName: Acme Coffee
SupplierID: 1
CategoryID: 1
QuantityPerUnit: 5 cans per box
UnitPrice: 24.95
UnitsInStock: 45
UnitsOnOrder: 0
ReorderLevel: 15
Discontinued: ☐
Insert Cancel

Figure 24: The `InsertItemTemplate` Dictates the FormView's Inserting Interface


Untitled Page - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites
Address http://localhost:4076/Code/EditInsertDelete/Basics.aspx
Go
Working with Data Tutorials
Home > Editing, Inserting, and Deleting > Basics
Home
Basic Reporting
Simple Display
Declarative Parameters
Setting Parameter Values
Filtering Reports
Filter by Drop-Down List
Master-Details-Details
Master/Detail Across Two Pages
Details of Selected Row
Customized Formatting
Done
Local intranet

The Basics of Editing, Inserting, and Deleting

Editing, Inserting, and Deleting Data from a FormView

ProductID: 87
ProductName: Acme Coffee
SupplierID: 1
CategoryID: 1
QuantityPerUnit: 5 cans per box
UnitPrice: 24.9500
UnitsInStock: 45
UnitsOnOrder: 0
ReorderLevel: 15
Discontinued: ☐
CategoryName: Beverages
SupplierName: Exotic Liquids
Edit Delete New
... 69 70 71 72 73 74 75 76 77 78

Figure 25: The Details for New Product, Acme Coffee, are Displayed in the FormView

By separating out the read-only, editing, and inserting interfaces into three separate templates, the FormView allows for a finer degree of control over these interfaces than the DetailsView and GridView.

Note: Like the DetailsView, the FormView's `CurrentMode` property indicates the interface being displayed and its `DefaultMode` property indicates the mode the FormView returns to after an edit or insert has been completed.

Summary

In this tutorial we examined the basics of inserting, editing, and deleting data using the GridView, DetailsView, and FormView. All three of these controls provide some level of built-in data modification capabilities that can be utilized without writing a single line of code in the ASP.NET page thanks to the data Web controls and the ObjectDataSource. However, the simple point and click techniques render a fairly frail and naïve data modification user interface. To provide validation, inject programmatic values, gracefully handle exceptions, customize the user interface, and so on, we'll need to rely on a bevy of techniques that will be discussed over the next several tutorials.

Happy Programming!

About the Author

Scott Mitchell, author of six ASP/ASP.NET books and founder of 4GuysFromRolla.com, has been working with Microsoft Web technologies since 1998. Scott works as an independent consultant, trainer, and writer, recently completing his latest book, Sams Teach Yourself ASP.NET 2.0 in 24 Hours. He can be reached at mitchell@4guysfromrolla.com or via his blog, which can be found at http://ScottOnWriting.NET.