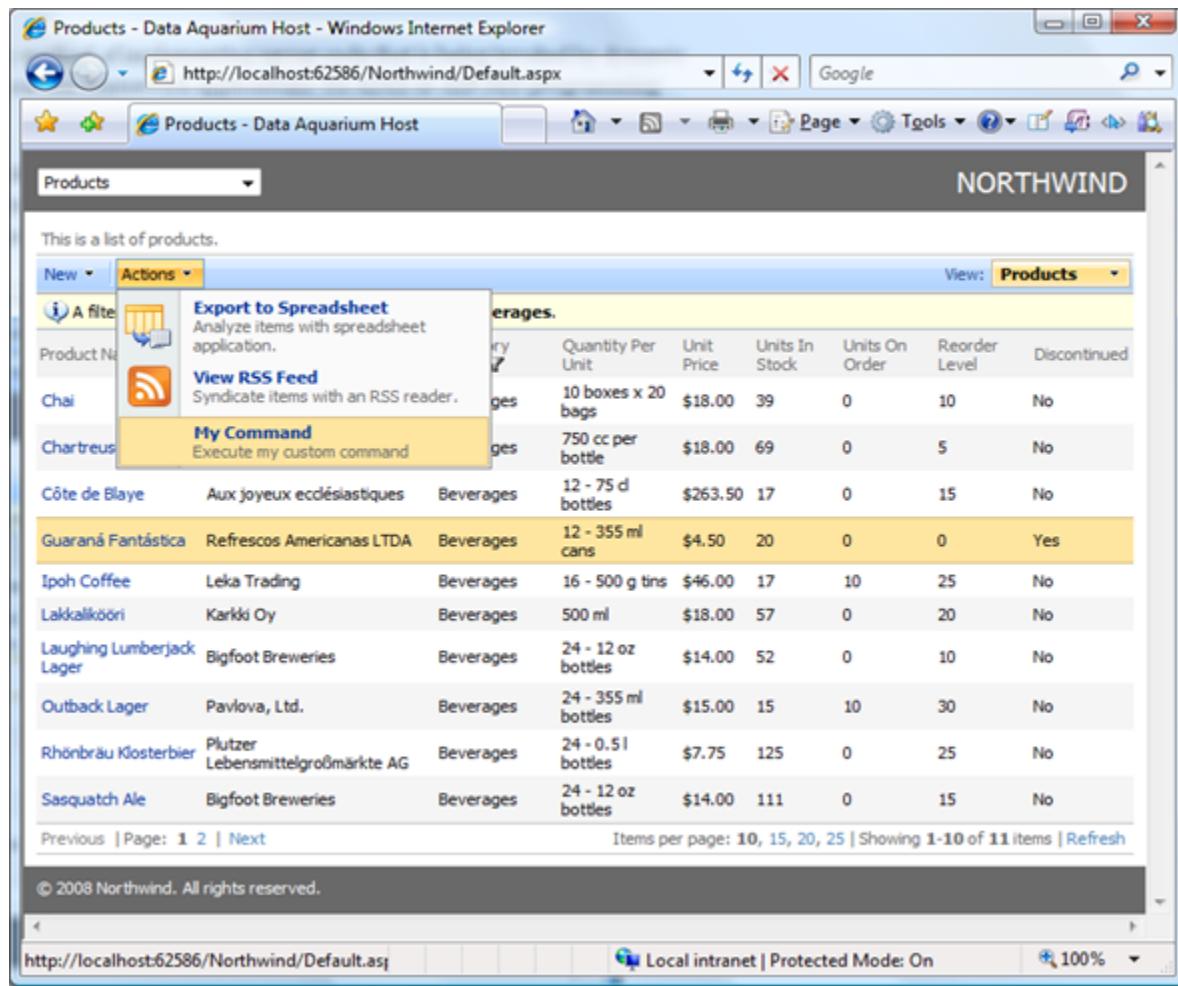


Creating Custom Action Handlers in Data Aquarium Applications

Custom actions provide the easy way of implementing server code that is being invoked by dynamic AJAX user interface of [Data Aquarium Framework](#) applications. No AJAX or ASP.NET programming experience is required.

Suppose you have generated ASP.NET application based on [Data Aquarium Framework](#) with Code OnTime Generator for the Northwind database. If you run this web application and select *Products* in the drop down at the top of the page, and click on *Actions* menu option then a view similar to the one displayed in the picture will be displayed.



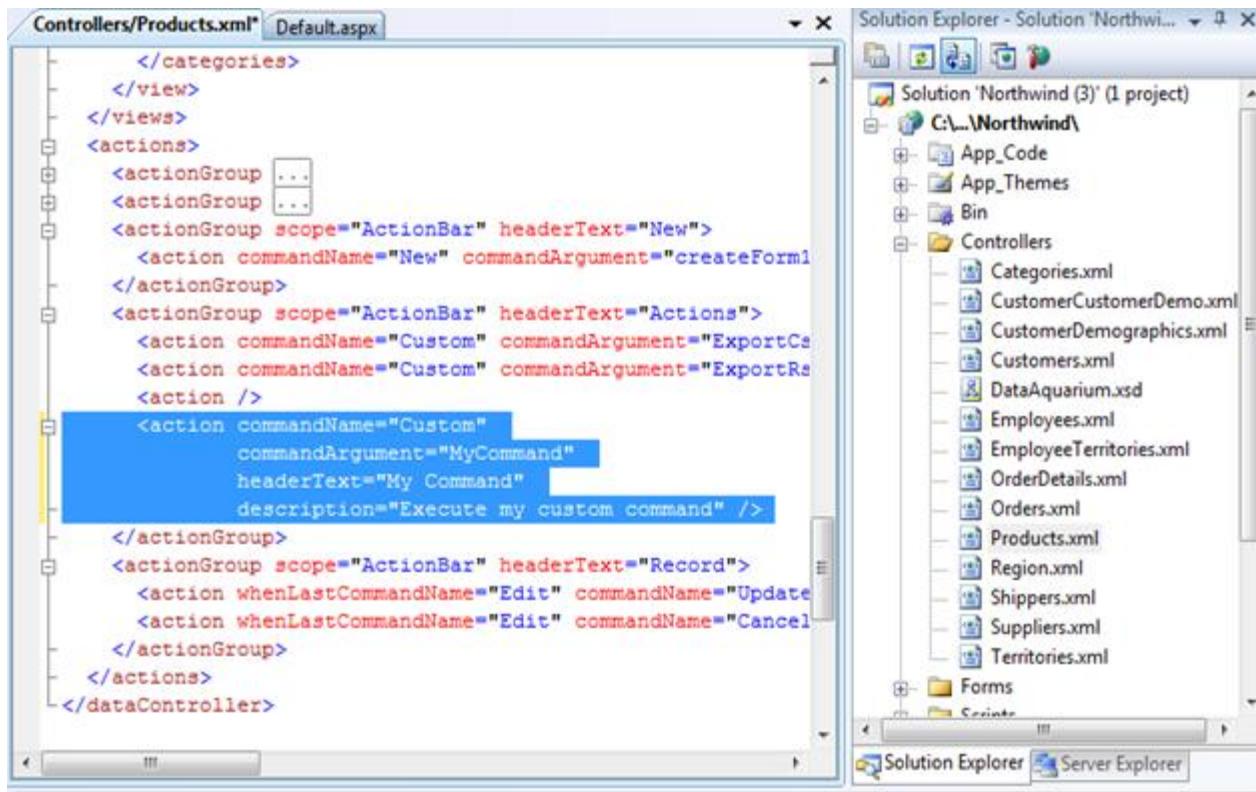
The screenshot shows a Windows Internet Explorer window displaying a Data Aquarium application. The title bar reads "Products - Data Aquarium Host - Windows Internet Explorer" and the address bar shows "http://localhost:62586/Northwind/Default.aspx". The main content area has a header "NORTHWIND" and a sub-header "This is a list of products.". Below this is a table of product data with columns: Product Name, Supplier, Category, Quantity Per Unit, Unit Price, Units In Stock, Units On Order, Reorder Level, and Discontinued. An "Actions" menu is open over the first row, showing options: "Export to Spreadsheet", "View RSS Feed", and "My Command". The "My Command" option is highlighted with a yellow background. At the bottom of the page, there are navigation links for "Previous", "Page: 1 2 Next", and "Items per page: 10, 15, 20, 25 | Showing 1-10 of 11 items | Refresh". The footer contains the copyright notice "© 2008 Northwind. All rights reserved." and the URL "http://localhost:62586/Northwind/Default.aspx".

Product Name	Supplier	Category	Quantity Per Unit	Unit Price	Units In Stock	Units On Order	Reorder Level	Discontinued
Chai	Spicy yet subversive. Analyze items with spreadsheet application.	Beverages	10 boxes x 20 bags	\$18.00	39	0	10	No
Chartreuse	Syndicate items with an RSS reader.	Beverages	750 cc per bottle	\$18.00	69	0	5	No
Côte de Blaye	Aux joyeux ecdésiastiques	Beverages	12 - 75 cl bottles	\$263.50	17	0	15	No
Guaraná Fantástica	Refrescos Americanas LTDA	Beverages	12 - 355 ml cans	\$4.50	20	0	0	Yes
Ipoú Coffee	Léka Trading	Beverages	16 - 500 g tins	\$46.00	17	10	25	No
Lakkalikööri	Karkki Oy	Beverages	500 ml	\$18.00	57	0	20	No
Laughing Lumberjack Lager	Bigfoot Breweries	Beverages	24 - 12 oz bottles	\$14.00	52	0	10	No
Outback Lager	Pavlova, Ltd.	Beverages	24 - 355 ml bottles	\$15.00	15	10	30	No
Rhönbräu Klosterbier	Plutzer Lebensmittelgrößmärkte AG	Beverages	24 - 0.5 l bottles	\$7.75	125	0	25	No
Sasquatch Ale	Bigfoot Breweries	Beverages	24 - 12 oz bottles	\$14.00	111	0	15	No

Let's write some custom server code, which will execute when the *My Command* action selected. We will learn how to implement custom server code that will validate the entered data just before it is submitted to the database. We will also write server code to invoke the client-side script to interact with the AJAX views.

Specifying Custom Actions

Open your project in *Visual Studio 2008* or *Visual Web Developer Express 2008*, expand *Controllers* folder, and open *Products.xml* data controller descriptor. Scroll to the bottom of the file and find custom command with *MyCommand* argument.



This is the only entry, which is needed to have your custom command displayed on the action bar of the views managed by the *Products* data controller. Set up your own custom header text and description to reflect the purpose of the action.

You can create additional action groups with the scope of *ActionBar*, *Form*, and *Grid*. The *ActionBar* action are displayed as menu option in the action bar. The *Form* action is displayed as a button in the data entry form. The *Grid* action will be displayed as an

option in the grid view row context menu, which pops up when you click on the drop down arrow next to the value in the first column of the row.

You can additional supply the context of previously executed command via *whenLastCommandName* attribute. For example, you might want certain actions to be available only when user has started editing the record, or when a new record is being created.

You can also specify user roles to automatically show/hide actions based on users security credentials. This feature is integrated with ASP.NET security infrastructure and required no coding at all.

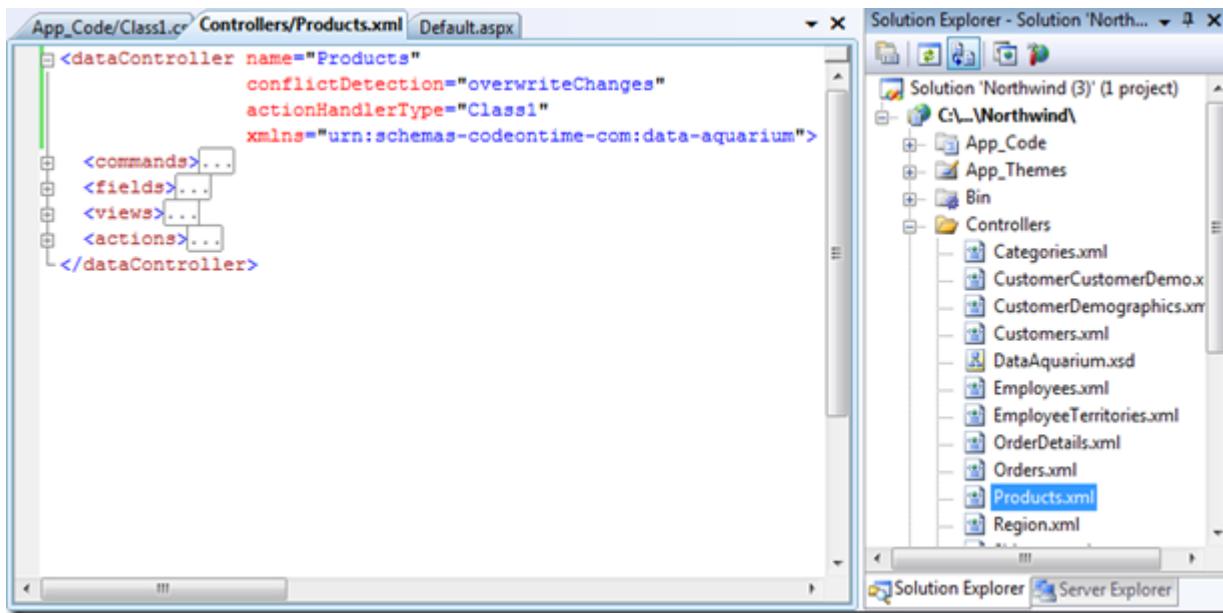
Create Server Code to Handle Custom Actions

Add a class to the *App_Code* folder of your application. Specify that you are using *Northwind.Data* namespace if you have entered *Northwind* as a default namespace when you have generated the code with Code OnTime Generator. Also specify that the class will inherit it's functionality from the *ActionHandlerBase* class that is a part of [Data Aquarium Framework](#).

The screenshot shows a Visual Studio interface. The code editor window is open with the file 'App_Code/Class1.cs' selected. The code defines a class named 'Class1' that inherits from 'ActionHandlerBase'. The Solution Explorer window on the right shows a project named 'Northwind' with three files: 'Default.aspx', 'ScriptHost.ashx', and 'web.config'. The 'Class1.cs' file is highlighted in the Solution Explorer.

```
1 using System;
2 using System.Data;
3 using System.Configuration;
4 using System.Linq;
5 using System.Web;
6 using System.Web.Security;
7 using System.Web.UI;
8 using System.Web.UI.HtmlControls;
9 using System.Web.UI.WebControls;
10 using System.Web.UI.WebControls.WebParts;
11 using System.Xml.Linq;
12
13 using Northwind.Data;
14
15 public class Class1 : ActionHandlerBase
16 {
17     public Class1()
18     {
19     }
20 }
```

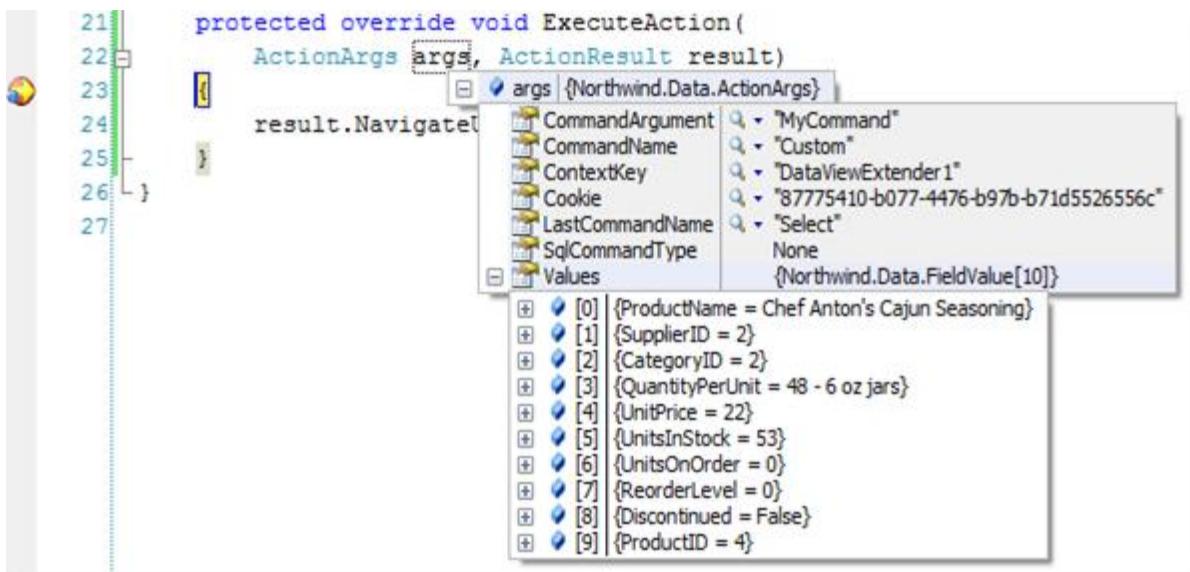
Now we need to hook the custom action handler *Class1* into the [Data Aquarium Framework](#). Scroll to the top of the *Products* data controller file and enter the class name as a value of the *actionHandlerType* attribute as show in the picture.



Override the *ExecuteAction* method in *Class1*:

```
protected override void ExecuteAction(ActionArgs args, ActionResult result)
{
    if (args.CommandName == "Custom" && args.CommandArgument == "MyCommand")
        result.NavigateUrl = "http://www.microsoft.com";
}
```

If the action is selected in the form view then the current record information is provided in the action argument values. If your action has been specified in the scope of the grid then the current row field values are passed alone to your code.



Use command name and argument to process multiple actions within the same action handler. Last command name can also be of use if you need to further alter the action behavior.

Create Server Code to Handle Data Manipulation Actions

Let's add some data validation code to prevent users from making changes to some sensitive information that we care about. For example, we will raise an exception if a user is trying to change the *Chai* product.

Override the *BeforeSqlAction* method in the *Class1*. Notice that we are using *Linq* to query the values of the action arguments.

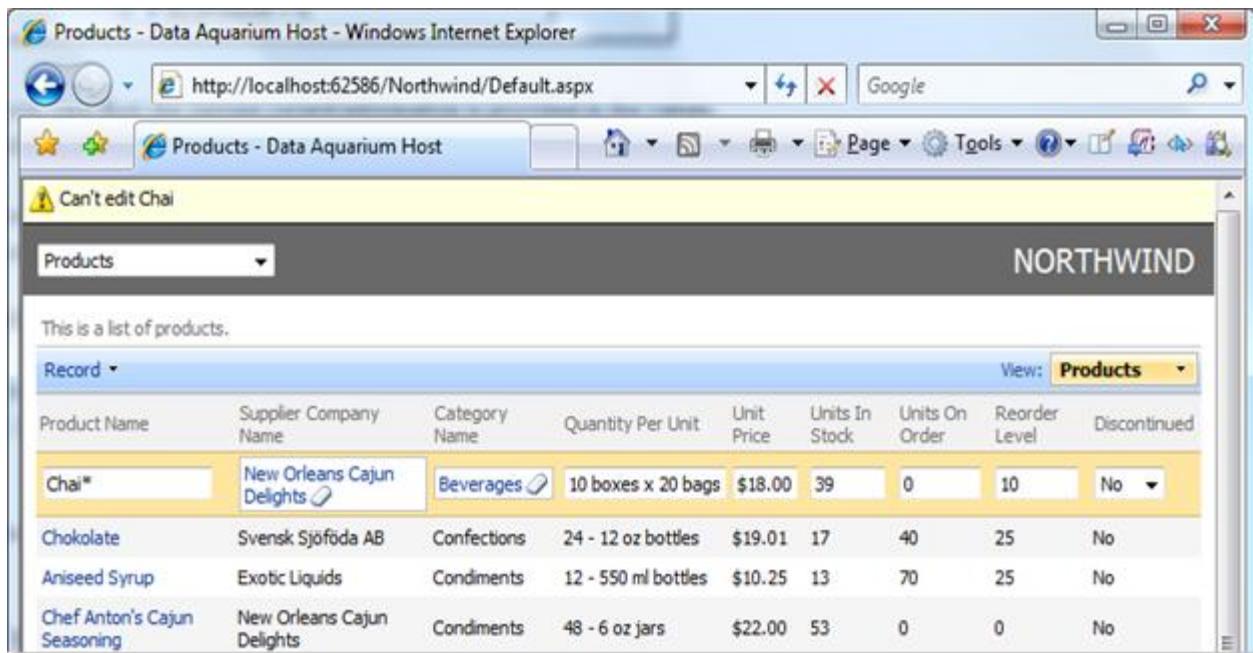
```

protected override void BeforeSqlAction(ActionArgs args, ActionResult result)
{
    if (args.CommandName == "Update")
    {
        string s = (string)
            from c in args.Values
            where c.Name == "ProductName"
            select c.OldValue).First();

        if (s == "Chai")
            throw new Exception("Can't edit Chai");
    }
}

```

Locate the *Chai* record in the *Products* screen, change any field of the record, and select *Save* in the action bar, in the grid context menu, or in the form edit view. The following error message is displayed at the top of the screen to the end user.



If an exception is raised before the execution of the SQL command then the SQL command is canceled. You can also cancel command by invoking the *Cancel* method of *result* parameter. This may be useful if you would like to execute your own data update

instead of relying on the automatic dynamic SQL generation feature of [Data Aquarium Framework](#). You can use *Values* property of the action argument and inspect individual fields via their *Name*, *Value*, *NewValue*, and *OldValue* properties.

Create Server Code to Invoke Client Java Script

You can supply a custom client-side Java Script expression, which will be evaluated upon the completion of execution of your server code. This works for both custom and data manipulation actions.

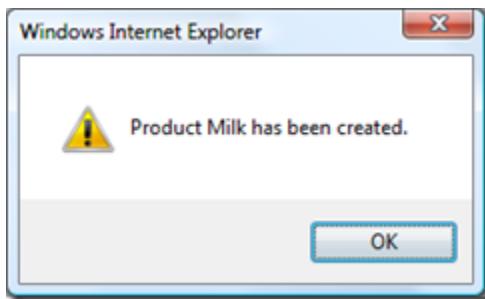
Suppose you want to allow users to enter multiple products with the minimum number of clicks. When user selects the *New* action from the action bar and enters the first product you want the data view to stay in the *New Products* form until the user decides to cancel.

Enter the following method in the *Class1*.

```
protected override void AfterSqlAction(ActionArgs args, ActionResult result)
{
    if (args.CommandName == "Insert")
    {
        result.ClientScript = String.Format(@""
            alert('Product {0} has been created.');
            $find('{1}').executeCommand(
                {{commandName:'New', commandArgument: 'createForm1'}});
            (from c in args.Values
            where c.Name == "ProductName"
            select c.NewValue).First(),
            args.ContextKey);
    }
}
```

Our custom code will kick in whenever the *Insert* command is executed and will assign a Java Script expression to be evaluated when the result is returned to the client-side data view. The *alert* method call will display a confirmation telling the user that the record

has been created indeed. The `$find` method call will find the client side Ajax component identified by `ContextKey` passed in the action arguments. Method `executeCommand` belongs to the `DataView` JavaScript class and will execute the client side command `New`, which will result in the `createForm1` view to be displayed again. From the user perspective the *New Products* form simply remains in place when the record creation confirmation is dismissed.



Conclusion

Exceptionally flexible server-side programming support in applications based on [Data Aquarium Framework](#) provides great number of customization options to programmers with any degree of experience with ASP.NET and AJAX.

It allows real separation of the business logic from the presentation.

Your web application sends asynchronous JSON requests to the stateless server application that are being processed with all the power of the Microsoft.NET framework without any need to know ASP.NET or AJAX programming techniques. Start being productive now.