



## Tracking User Actions

Tracking of user activities is a common requirement for many business applications. [Data Aquarium Framework](#) support Microsoft ASP.NET Membership via an advanced user management and login/logout user interface components. You can quickly create business rules to track user actions.

### SAMPLE APPLICATION

Generate a [Data Aquarium](#) project with the membership option enabled. Here is a typical screen shot of a *Northwind* database sample after the user with the name *user* has signed in.

The screenshot shows a web browser window displaying an application interface. The browser title is "Orders - Data Aquarium Framework - Windows Internet Explorer". The address bar shows "http://localhost:29580/Journal/default.aspx". The page content includes a navigation menu with "Data Controllers", "Master/Detail Extravaganza", and "Database Lookups". Below this is a header for "MYCOMPANY" and a welcome message: "Welcome user, Today is Tuesday, April 14, 2009 | My Account | Logout | Help".

The main content area displays a list of orders. A filter is applied: "A filter has been applied. Ship City is equal to London." The table below shows the filtered orders:

Customer Company Name	Employee Last Name	Order Date	Required Date	Shipped Date	Ship Via Company	Freight	Ship Name	Ship Address	Ship City
B's Beverages	King	8/26/1996	9/23/1996	8/28/1996	Federal Shipping	\$22.77	B's Bever		
Seven Seas Imports	Buchanan	11/21/1996	12/19/1996	11/26/1996	Federal Shipping	\$288.43	Seven Sea	Leipzig	
Eastern Connection	Devolo	11/26/1996	1/7/1997	12/4/1996	Speedy Express	\$71.97	Eastern Connecto	Lille	
Seven Seas Imports	Devolo	12/9/1996	1/6/1997	12/13/1996	Federal Shipping	\$22.21	Seven Sea	Lisboa	
Seven Seas Imports	Fuller	12/19/1996	1/16/1997	12/20/1996	Speedy Express	\$34.86	Seven Sea	London	
Eastern Connection	Devolo	1/1/1997	1/29/1997	1/16/1997	Federal Shipping	\$83.93	Eastern Connecto	Luleå	
Consolidated Holdings	Callahan	2/4/1997	3/18/1997	2/7/1997	United Package	\$9.21	Consolida	Lyon	
Consolidated Holdings	Fuller	3/3/1997	3/31/1997	3/18/1997	Speedy Express	\$6.17	Consolida	Madrid	
B's Beverages	Fuller	3/11/1997	4/8/1997	3/18/1997	Federal Shipping	\$45.59	B's Bever	Mannheim	
Seven Seas Imports	Callahan	3/12/1997	4/9/1997	3/19/1997	Speedy Express	\$4.20	Seven Sea	Marseille	
								México D.F.	
								Montréal	

At the bottom of the table, there is a footer: "Previous | Page: 1 2 3 4 | Next" and "Items per page: 10, 15, 20, 25 | Showing 1-10 of 33 items | Refresh". The page footer includes "© 2009 MyCompany. All rights reserved." and the browser status bar shows "Internet | Protected Mode: On" and "100%".

## TASK 1

You want to keep a journal of user activities. The built-in .NET diagnostics facility will play a role of a journal where we will be keeping all activity records.

## SOLUTION

Create a business rules class `Class1` and create method `OrdersAfterUpdate` as shown below. Link the business rules class to `~/Controllers/Orders.xml` data controller as explained [here](#).

### C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using MyCompany.Data;

public class Class1 : BusinessRules
{
    [ControllerAction("Orders", "Update", ActionPhase.After)]
    protected void OrdersAfterUpdate(int orderId, FieldValue shipAddress)
    {
        System.Diagnostics.Debug.WriteLine(String.Format(
            "Order #{0} has been updated by '{1}' on {2}",
            orderId, Context.User.Identity.Name, DateTime.Now));
        if (shipAddress.Modified)
            System.Diagnostics.Debug.WriteLine(String.Format(
                "Address has changed from '{0}' to '{1}'.",
                shipAddress.OldValue, shipAddress.NewValue));
    }
}
```

### VB

```

Imports Microsoft.VisualBasic
Imports MyCompany.Data

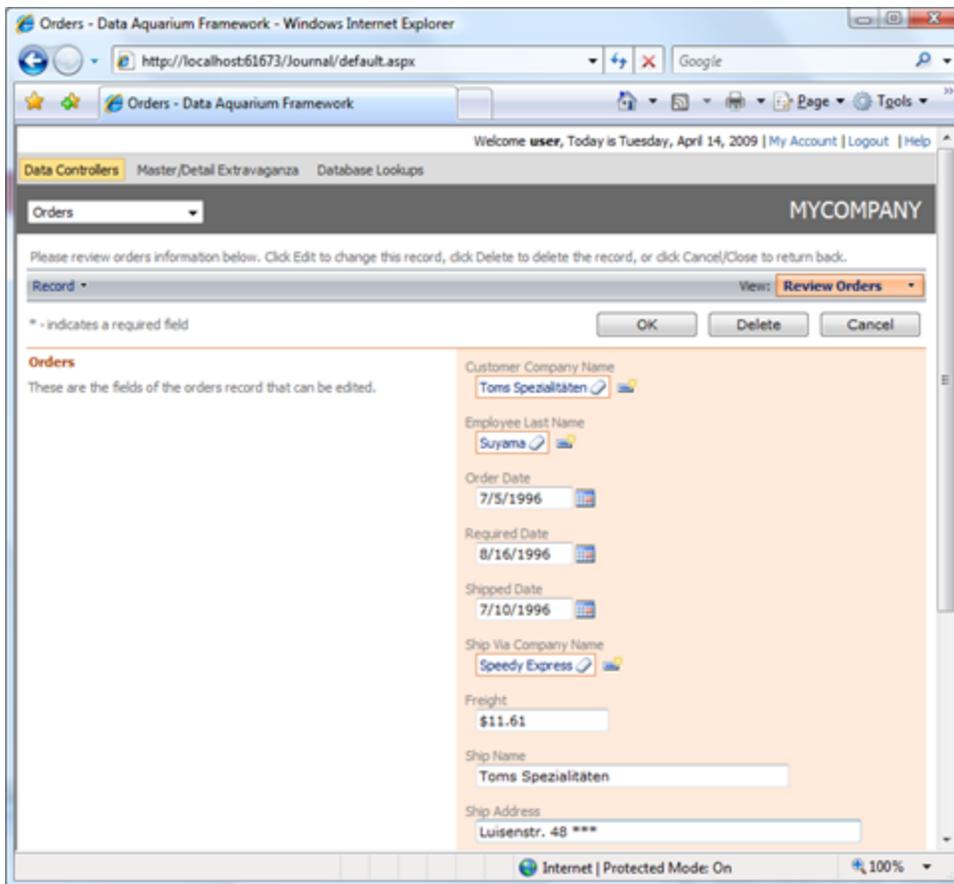
Public Class Class1
    Inherits BusinessRules

    <ControllerAction("Orders", "Update", ActionPhase.After)> _
    Protected Sub OrdersAfterUpdate(ByVal orderId As Integer, _
                                    ByVal shipAddress As FieldValue)
        System.Diagnostics.Debug.WriteLine(String.Format( _
            "Order #{0} has been updated by '{1}' on {2}", _
            orderId, Context.User.Identity.Name, DateTime.Now))
        If (shipAddress.Modified) Then
            System.Diagnostics.Debug.WriteLine(String.Format( _
                "Address has changed from '{0}' to '{1}'.", _
                shipAddress.OldValue, shipAddress.NewValue))
        End If
    End Sub

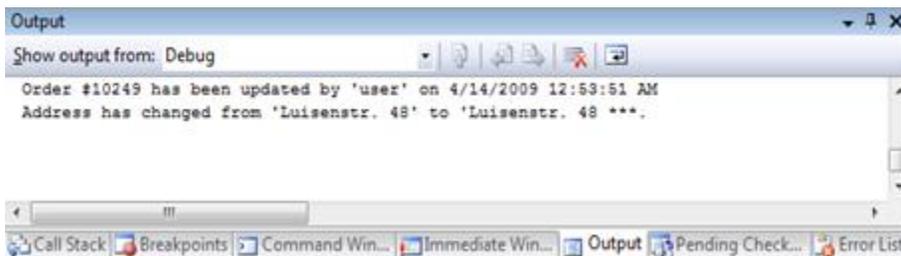
End Class

```

Open application in a web browser and select *Orders* data controller from the drop down in the top left corner. Start editing any order in the grid or form view and make sure to change *Ship Address* field. This field is the last visible field in the screen shot.



Hit *OK* button and the business method rule will intercept the action as soon as a successful database update has been completed. The first line of code will report the order ID and the user's identity. The second line of code will detect the change in the address field.



Property *Context* provides business rules developers with the same *Request*, *Response*, *User*, *Application*, *Session*, and *Server* properties that are available to web form developers.

The first two properties shall not be used since they provide information related to a current web service request and cannot be used to influence the user interface presentation.

Use the other properties as you if you were writing a typical web form.

Replace *System.Diagnostics.Debug* with a business object that is designed to keep track of user activities in a permanent data store such as a database table.

## TASK 2

All records in a database of orders must be marked with a reference to a user. User information will be utilized to filter data and for data analysis and reporting purposes.

## SOLUTION

Alter table *[Northwind].[dbo].[Orders]* to include new field *UserName* by executing the following *SQL* statement.

```
alter table Orders
add UserName varchar(50)
go
```

Modify command *command1* in the data controller *~/Controllers/Orders.xml* to select the new field twice. Once the field is selected under its own name and the other time we are selecting this very field under alias *UserNameReadOnly*. The reason for that is explained later.

```
<command id="command1" type="Text">
  <text>
    <![CDATA[
select
  "Orders"."OrderID" "OrderID"
  .....
  ,"Orders"."UserName" "UserName"
  ,"Orders"."UserName" "UserNameReadOnly"
```

```

from "dbo"."Orders" "Orders"
.....
]]>
    </text>
</command>

```

Add two field definitions for *UserName* instances in *SQL* statement to the list of data controller fields.

```

<fields>
.....
    <field name="UserName" type="String" label="User Name"/>
    <field name="UserNameReadOnly" type="String" label="User Name"
readOnly="true"/>
</fields>

```

Let's add the new read-only version of the field and a hidden version of the field to the list of data fields of views *grid1* and *editForm1*.

```

<dataField fieldName="UserNameReadOnly"/>
<dataField fieldName="UserName" hidden="true"/>

```

Modify view *createForm1* to include field *UserName* as a single hidden field.

```

<dataField fieldName="UserName" hidden="true"/>

```

We will silently assign a user name when a new record is created to the data controller field *UserName*. The captured value will be displayed when users review existing records but will be drawn from *UserNameReadOnly* for display purposes instead.

Add the following business rule to class *Class1*.

C#

```

[ControllerAction("Orders", "Update", ActionPhase.Before)]
[ControllerAction("Orders", "Insert", ActionPhase.Before)]

```

```
protected void OrdersBeforeUpdate(FieldValue userName)
{
    userName.NewValue = Context.User.Identity.Name;
    userName.Modified = true;
}
```

## VB

```
<ControllerAction("Orders", "Update", ActionPhase.Before)> _
<ControllerAction("Orders", "After", ActionPhase.Before)> _
Protected Sub OrdersBeforeUpdate(ByVal userName As FieldValue)
    userName.NewValue = Context.User.Identity.Name
    userName.Modified = True
End Sub
```

This method will be automatically invoked whenever an order is about to be updated or inserted into database.

The first line will assign name of the currently logged-in user to the *UserName* field.

The second line will indicate that the field has actually changed. The framework is using *Modified* property of *FieldValue* instances to determine if a field shall be included in automatically generated SQL statement to update or insert a record.

You can also do an update on your own without relying on the framework. The best place for that sort of updates is in business rules methods with *ActionPhase.After* specifies as a parameter of *ControllerAction* attribute.

Here is how the grid of orders will look if you update a few records. The right-most column is displaying the name of the user.

This is a list of orders.

Customer Company Name	Employee Last Name	Order Date	Required Date	Shipped Date	Ship Via Company Name	Freight	Ship Name	Ship Address	Ship City	User Name
Vins et alcools Chevalier	Buchanan	7/4/1996	8/1/1996	9/28/2008	Federal Shipping	\$322.38	Vins et alcools Chevalier	59 rue de l'Abbaye	Reims	user
	Suyama	7/5/1996	8/16/1996	7/10/1996	Speedy Express	\$11.61	Toms Spezialitäten	Luisenstr. 48 ***	Münster	n/a
	Peacock	7/8/1996	8/5/1996	7/12/1996	United Package	\$165.83	Hanari Carnes	Rua do Paço, 67	Rio de Janeiro	n/a
Victualles en stock	Leverling	7/8/1996	8/5/1996	7/15/1996	Speedy Express	\$41.34	Victualles en stock	2, rue du Commerce**	Lyon	admin

The two fields *UserName* and *UserNameReadOnly* are required since read-only fields are never transferred to the server from the client web page. Hidden fields are never displayed but always travel from the client to the server and back. By introducing two versions of the same field we overcome this limitation imposed by the framework's optimization logic.

## CONCLUSION

Business rules in [Data Aquarium Framework](#) provide an excellent place to universally track user activities.

Code OnTime LLC

<http://www.codeontime.com>