Default Field Values in Custom Action Handlers

A common requirement in many data management applications is to compliment data manually entered by users with some calculated values. You may be required to perform some complicated financial computations, lookup additional field values in the database according to some custom logic, or automatically supply audit trail by recording the time stamps and user ID in the database records. Some of these tasks can be encapsulated in database table triggers. If this is not an option then it is time to create a custom action handler.

We will illustrate this with the unit price lookup by improving on the custom action handler example presented in Aquarium Express Primer.

The primer is demonstrating a lookup of *unit price* for inserted *order details* records of *Northwind* database. Basically it is suggesting allowing insertion of the *order details* record without a price and executing a follow-up update of the new record. This is resulting in two database operations - SQL *insert* will be followed by *update* statement.

We will improve on that by supplying a *unit price* prior to the insertion of *order details* record.

Open *~/Controllers/OrderDetails.aspx* and change the definition of *createForm1* as follows:

```
<view id="createForm1" type="Form" commandId="command1" label="New Order
Details">
  <headerText>Please fill this form and click OK button to create a new order
details record. Click Cancel to return to the previous screen.</headerText>
  <categories>
    <category headerText="New Order Details">
      <description>Complete the form. Make sure to enter all required
fields.</description>
      <dataFields>
        <dataField fieldName="OrderID" aliasFieldName="OrderCustomerID" />
        <dataField fieldName="ProductID" aliasFieldName="ProductProductName"
/>
        <dataField fieldName="UnitPrice" dataFormatString="c" columns="15"
hidden="true"/>
        <dataField fieldName="Quantity" columns="15" />
        <dataField fieldName="Discount" columns="15" />
      </dataFields>
    </category>
  </categories>
</view>
```

The primer suggests removing the *UnitPrice* data field from view *createForm1*. We are restoring the data field and instead marking it up as "hidden". Hidden data fields are not rendered in the standard user interface when displayed in a web browser, which meets our objective as outline in the primer. On the other hand, hidden fields are available for manipulation in custom action handlers. Note that you can still display any hidden fields if you are using custom form templates.

Now you can remove the price lookup code from method *AfterSqlActon* in *Class1*.

Create a new method *BeforeSqlAction* as shown in example.

Here is *C#* implementation of *BeforeSqlAction*.

```
protected override void BeforeSqlAction(ActionArgs args, ActionResult result)
{
    if (args.CommandName == "Insert" && args["UnitPrice"].Value == null)
    {
        double price = 0;
        using (SqlText findPrice = new SqlText(
            "select UnitPrice from Products where ProductId = @ProductID"))
        {
            findPrice.AddParameter("@ProductId", args["ProductID"].Value);
            price = Convert.ToDouble(findPrice.ExecuteScalar());
        }
        args["UnitPrice"].NewValue = price;
        args["UnitPrice"].Modified = true;
    }
}
```

VB.NET implementation looks very much the same.

```
Protected Overrides Sub BeforeSqlAction(ByVal args As
MyCompany.Data.ActionArgs, _
                                            ByVal result As
MyCompany.Data.ActionResult)
        If args.CommandName = "Insert" And args("UnitPrice").Value Is Nothing
Then
            Dim price As Double = 0
            Using findPrice As SqlText = New SqlText( _
                "select UnitPrice from Products where ProductId =
@ProductID")
                findPrice.AddParameter("@ProductID", args("ProductID").Value)
                price = Convert.ToDouble(findPrice.ExecuteScalar())
            End Using
            args("UnitPrice").NewValue = price
            args("UnitPrice").Modified = True
        End If
    End Sub
```

As you can see we are finding the price of the ordered product and then assigning it to the new value of *UnitPrice* in the argument. It is important to mark the field value as modified. Otherwise the field will not be included in the *SQL* statement generated by data controller.

Custom action handlers allow complex calculations to be implemented in a high level language such as C# or *VB.NET*, which may be the only option if calculations involve resources outside of the database server. The same custom action handler is uniformly used in all pages of your application that are displaying the data with the help of the controllers that are referring to your custom action handler via *actionHandlerType* attribute. Custom action handlers become centralized stateless business logic layer rule repositories of your Web 2.0 applications written with ASP.NET and AJAX.