Using Classes in VBScript

Description: SalesLogix v6 opens up a whole new door for SalesLogix developers. Now, with VBScript, you have in your arsenal a new set of tools with the ability to use all that VBScript has to offer - all from inside of SalesLogix. One of the best new things we can now use (in my opinion) is using classes in scripts. This article will cover how to use classes in VBScript for making some routine tasks easier to use and more reusable.

Category: SalesLogix VBScript Articles
Author: Ryan Farley
Submitted: 11/14/2002

Stats: *Article has been read 18700 times* **Rating:** *- 4.9 out of 5 by 19 users*

Using Classes in VBScript

SalesLogix v6 opens up a whole new door for SalesLogix developers. Now, with VBScript, you have in your arsenal a new set of tools with the ability to use all that VBScript has to offer - all from inside of SalesLogix. One of the best new things we can now use (in my opinion) is using classes in scripts. This article will cover how to use classes in VBScript for making some routine tasks easier to use and more reusable.

Introduction

Let's take a look at what a class or object really is *(If you are already familiar with the concept of a class or object, you may want to skip over this section).* First, let's look at a couple of terms:

- **What is an object?**
  An object is basically a "code" container. It contains variables, functions, etc, and is normally used to represent an entity of some kind.
- **What is an entity?**
  An entity is literally a "thing". Ok, not too descriptive, is it? I suppose what I mean about a "thing" is some sort of idea or maybe a tangible item. Like a "product", or even the "sale" of a product could be this "thing" or entity. An entity could be a history item, or even a person.

Let's explore the idea of having a "sale" entity. You would need to know several things about this entity, such as which product they purchased, who purchased it, who the sales person was, etc. It may seem simple enough to get all of this data, but all of these pieces that make up this "sale" entity could potentially be stored in different tables - each with possibly it's own query to the database, which is all still doable, but wouldn't it be nice to work with one single item that represented all of this information? Helloooo, object. Your code will be cleaner, it will be easier to use, and you can reuse it whenever you need to represent this "sale" entity. If you need to change a calculation or something in the sale object, you change it only in the class that makes up the object and it is affected everywhere it is used.

More About Objects

An object, or really the class that makes up, or describes, the object, mainly contains two things. Properties and Methods.

- **Properties**
  Properties are really just facts, or attributes that the entity has. For example, in our example above, the "sale" entity would have several properties such as Sale Date, Sales Person, Product, etc.
- **Methods**
  Methods are really actions in the entity that can yield results or change something about the entity. For example, in our "sale" entity, there could potentially be methods to process a sale, which would deduct the product from inventory, and charge the customer. Possibly calculate sales tax (for new sales). Or maybe to add payments against the sale. Methods in terms of VBScript are either subroutines or functions.

Example syntax for properties & methods would look something like this:

```
'property
objSale.SalesDate = Now

'method
amtRemaining = objSale.AddPayment("25.99")
```

Creating Classes

Ok, enough talk already, let's look at some code! Creating classes are really simple, what might take some time is getting yourself to think about code as separate, reusable entities. Obviously, since this site is dedicated to SalesLogix development, from here on we'll focus on how to do this in respect to SalesLogix (although most of what is covered here will still work in other VBScript environments).

In the SalesLogix Architect, let's create a new VBScript plugin. We don't need a Sub Main or anything like that (so take that out of the VBScript window). The class does not need an entry point since it is not "invoked" or "run". Instead it is "instanciated". Ideally, how you will want to build classes in SalesLogix, is to create them as a separate VBScript plugin. Although you could create the class in the code behind a form, it is not reusable that way. If we create it in a separate VBScript plugin, we can simply "include" it in the script we want to use the class in by using the cool new feature to include scripts (see Using Classes section later on in this article). For our example that follows, we'll focus on building a class to add a history item in SalesLogix. So thinking in terms of objects and entities, the "History Item" is our entity. It will have properties for things like the ContactID it is for, the date, etc. We'll make the assumption that it will be used for adding history items to contacts only

Basic syntax of a class is to name the class with a 'Class' and 'End Class'. Here's an example.

```
Class History
End Class
```

It may not look like much (because it really isn't as of yet), but we could create an instance of the class like this:

```
'create and instanciate the History object
Dim objHistory
Set objHistory = New History

'we could do things with it here, but we haven't defined any of that yet

'let's get rid of it
Set objHistory = Nothing
```

Now we'll add some properties to it. But first, there are a few things to know about properties. There are 3 kinds of properties in VBScript. Let, Set, & Get properties. Let's take a look at what those mean.

- **Let**
  The *Let* property is your standard "pass something to the class" property. For example, if our History item has a "property" or "attribute" of completed date (which it will), then this completed data would be a Let property of the class. It would "Let" us give this value to the class.
- **Set**
  The *Set* property is very similar to the Let property, with the exception that it is used for objects. If we have a property in our class to hold an ADO Recordset or another class object, then we would need to "Set" the reference using a Set Property. In our example, we will not use a Set property because all of our values will be strings or numbers which are used as Let properties (since they are 'value' types, not referencing an object)
- **Get**
  Last but not least is the *Get* property. This is the one you'll use often. It is used to "get" a value from the class. For example, in our history class, we could add a Get property to get the ID of the newly created history item.

Another thing to note is scope (this applies to both properties and methods). VBScript Class members can be private or public. What this means is whether or not it is visible outside the class, or if it is hidden within the class for only the class to use. If you want something to be accessed outside the class you mark it public. Our History class will have a public "Add" method to add the history values to the history table. It will also have a private function to create a new history ID. Our class needs this internally and we don't want to expose this to the outside (if they want an ID they can create one themselves, it really wouldn't apply to our history entity).

Ready? Let's add some properties to out class. For our example, we'll keep things simple and only add the minimal number of required fields to our class for adding history items. They way a property works is to publicly access (ie: from outside the class) a private member or variable. We need to create the private variables (denoted with a 'm_' prefix) and then expose them through properties.

```
Private m_type
Private m_conId
Private m_startDT
Private m_completedDT
Private m_userID
Private m_desc
Private m_notes
```

```
Private m_historyID

'Here are the "writable" Let properties
Public Property Let HistoryType(ByVal val)
        m_type = val
End Property

Public Property Let ContactID(ByVal val)
        m_conId = val
End Property

Public Property Let StartDate(ByVal val)
        m_startDT = val
End Property

Public Property Let CompletedDate(ByVal val)
        m_completedDT = val
End Property

Public Property Let UserID(ByVal val)
        m_userID = val
End Property

Public Property Let Description(ByVal val)
        m_desc = val
End Property

Public Property Let Notes(ByVal val)
        m_notes = val
End Property

'Here are the "readable" Get properties
Public Property Get HistoryID
        HistoryID = m_historyID
End Property
```

Doesn't that look nice? We could make a property both readable and writable by adding both a Let and a Get for the same property name, but we won't do that in our example. Let's say we want to initialize some values, like the userid. A class has two methods that fire when the class is created and when the class is terminated. If you add these subroutines to your class they will automatically run when those two events happen. Here's what those special methods look like.

```
Private Sub Class_Initialize()
Private Sub Class_Terminate()
```

In our class we'll add a Class_Initialize to set the userid to the current user. The code using the class could always change it later using the property, but this way it does not need to be set, unless it is for a user other than the current one.

```
Private Sub Class_Initialize()
        m_userID = Application.BasicFunctions.CurrentUserID
End Sub
```

Now all we really have left is to create the public "Add" method to create a row in the history table. There's going to be a few other things happening here to get some other values here, such as the accountid and account name for the contactid set in the object.

```
Public Sub Add
Dim cn
```

```
    Set cn = Application.GetNewConnection
    m_historyID = GetNewID("history")

    cn.Execute GetInsertSQL()

    cn.Close
    Set cn = Nothing
End Sub

Private Function GetInsertSQL()
Dim sql

    sql = "insert into history (historyid, type, accountid, accountname, "
    sql = sql & "contactid, contactname, startdate, userid, timeless, duration,
"
    sql = sql & "description, notes, longnotes, "
    sql = sql & "createuser, createdate, completeduser, completeddate) "
    sql = sql & "values ('"
    sql = sql & m_historyID & "', "
    sql = sql & "'" & m_type & "', "
    sql = sql & "'" & GetContactValue(m_conId, "accountid") & "', "
    sql = sql & "'" & GetContactValue(m_conId, "account") & "', "
    sql = sql & "'" & m_conId & "', "
    sql = sql & "'" & GetContactValue(m_conId, "fullname") & "', "
    sql = sql & "'" & m_startDT & "', "
    sql = sql & "'" & m_userID & "', "
    sql = sql & "'T', "
    sql = sql & "'0', "
    sql = sql & "'" & Replace(m_desc, "'", "''") & "', "
    sql = sql & "'" & Replace(Mid(m_notes, 1, 254), "'", "''") & "', "
    sql = sql & "'" & Replace(m_notes, "'", "''") & "', "
    sql = sql & "'" & Application.BasicFunctions.CurrentUserID & "', "
    sql = sql & "'" & Now & "', "
    sql = sql & "'" & m_userID & "', "
    sql = sql & "'" & m_completedDT & "')"

    GetInsertSQL = sql
End Function

Private Function GetContactValue(ByVal conid, ByVal field)
Dim val

    val = GetValue(field, "contact", "contactid = '" & conId & "'")
    GetContactValue = Replace(val, "'", "''")
End Function

Private Function GetNewID(ByVal table)
Dim rs
Dim cn

    Set cn = Application.GetNewConnection
    Set rs = cn.Execute("slx_dbids('" & table & "', 1)")

    GetNewID = rs.Fields(0).Value & ""

    rs.Close
    Set rs = Nothing
    cn.Close
    Set cn = Nothing
End Function
```

```
Private Function GetValue(ByVal field, ByVal table, ByVal where)
Dim rs
Dim cn

    Set cn = Application.GetNewConnection
    Set rs = cn.Execute("select " & field & " from " & table & " where " &
where)

    If Not (rs.BOF Or rs.EOF) Then
        GetValue = rs.Fields(0).Value & ""
    Else
        GetValue = ""
    End If

    rs.Close
    Set rs = Nothing
    cn.Close
    Set cn = Nothing
End Function
```

That may seem like a lot, but really I just wanted to make the demo functional. If you don't understand everything that is going on there then don't worry about it. The purpose of this article is to demonstrate how to create & use classes in VBScript. There will be other articles to cover ADO basics, VBScript basics, & how to create SalesLogix IDs, etc. For now, let's move on to how to use the class. This is where you'll love it because things could not get any simpler on that end.

Using Classes

Now that we understand how to make a class, let's take a look at how to use it. Create a new Contact form and add a picklist named pklDesc (for the To-Do Regarding picklist), a memo named memNotes for the notes, and a button to instanciate the class and add the history item. In the button's Click event, place the following code:

```
Sub btnAddHistoryClick(Sender)
Dim objHistory

    'instanciate the history object
    Set objHistory = New History

    With objHistory
         'set the properties
         .HistoryType = "262147"
         .ContactID = Application.BasicFunctions.CurrentContactID
         .StartDate = Now
         .CompletedDate = Now
         .UserID = Application.BasicFunctions.CurrentUserID
         .Description = pklDesc.Text
         .Notes = memNotes.Text

         'call the history object's add method to insert the history item
         .Add

         MsgBox "History item added with ID '" & .HistoryID & "'"
    End With

    'dereference the history object
    Set objHistory = Nothing
```

```
    pklDesc.Text = ""
    memNotes.Text = ""
End Sub
```

Now look at that. Enough to bring tears to my eyes. Isn't that clean? Now whenever you want to add a history item, you could use this class and have the same *simple* syntax you see above. All you have to do is add the script, instanciate the object, set it's properties and away you go. I like it!

Including The Class VBScript

We've talked a lot about how we will use the class and they only way to go is to include the script. Just to make sure everyone knows what we mean by this, let's take a minute before we end to look at that. SalesLogix v6 has a great new feature to include a VBScript so you can use it in another script (or script behind a Form). To do this, all you have to do is click the "Include Script" button on the top right of the code window and select the VBScript that has your class in it. Take a look at this screenshot:

Easy enough. Now you can include the class VBScript whenever you need to add history items. Cool. One thing to note about adding Include Scripts. When you include a script, the script name will be automatically added as a comment at the top of your code window. This makes it easy to see that you are including a script without showing the Included Scripts pane of the code window.

Wrapping it up

Knowing how to take advantage of things available to you in VBScript will allow you to make your life as a SalesLogix developer much simpler. With classes, you can encapsulate complex or redundant code to make performing those tasks easier. The best part of all is it begs for code reuse. You can develop your own "toolbox" bundle containing all your classes that you use each time you develop in the Architect. Gotta love that!

Until next time, happy coding.
-Ryan