Implementing a class

In this recipe, you will learn the following:

- The basic concepts and the syntax required by VBScript to implement a class
- The different components of a class and interoperation
- How to implement a type of generic constructor function for VBScript classes
- How to use a class during runtime

Getting ready

From the **File** menu, navigate to **New** | **Function Library...**, or use *Alt* + *Shift* + *N* shortcut. Name the new function library `cls.MyFirstClass.vbs` and associate it with your test.

How to do it...

We will build our `MyFirstClass` class from the ground up. There are several steps one must follow to implement a class; they are follows:

1. Define the class as follows:

```
Class MyFirstClass
```

2. Next, we define the class fields. Fields are like regular variables, but encapsulated within the namespace defined by the class. The fields can be private or public. A private field can be accessed only by class members. A public field can be accessed from any block of code. The code is as follows:

```
Class MyFirstClass
  Private m_sMyPrivateString
  Private m_oMyPrivateObject
  Public m_iMyPublicInteger
End Class
```

Note

It is a matter of convention to use the prefix `m_` for class member fields; and `str` for `string`, `int` for `integer`, `obj` for `Object`, `flt` for `Float`, `bln` for `Boolean`, `chr` for `Character`, `lng` for `Long`, and `dbl` for `Double`, to distinguish between fields of different data types. For examples of other prefixes to represent additional data types, please refer to sites such as https://en.wikipedia.org/wiki/Hungarian_notation.

Hence, the private fields' `m_sMyPrivateString` and `m_oMyPrivateObject` will be accessible only from within the class methods, properties, and subroutines. The public field `m_iMyPublicInteger` will be accessible from any part of the code that will have a reference to an instance of the `MyFirstClass` class; and it can also allow partial or full access to private fields, by implementing public properties.

Note

By default, within a script file, VBScript treats as public identifiers such as function and subroutines and any constant or variable defined with `Const` and `Dim` respectively, even if not explicitly defined. When associating function libraries to UFT, one can limit access to specific globally defined identifiers, by preceding them with the keyword `Private`.

The same applies to members of a class, function, sub, and property. Class fields must be preceded either by `Public` or `Private`; the public scope is not assumed by VBScript, and failing to precede a field identifier with its access scope will result in a syntax error. Remember that, by default, VBScript creates a new variable if the explicit option is used at the script level to force explicit declaration of all variables in that script level.

3. Next, we define the class properties. A property is a code structure used to selectively provide access to a class' private member fields. Hence, a property is often referred to as a getter (to allow for data retrieval) or setter (to allow for data change).

   A property is a special case in VBScript; it is the only code structure that allows for a duplicate identifier. That is, one can have a `Property Get` and a `Property Let` procedure (or `Property Set`, to be used when the member field actually is meant to store a reference to an instance of another class) with the same identifier. Note that `Property Let` and `Property Set` accept a mandatory argument. For example:

```
Class MyFirstClass
  Private m_sMyPrivateString
  Private m_oMyPrivateObject
   Public m_iMyPublicInteger

  Property Get MyPrivateString()
    MyPrivateString = m_sMyPrivateString
  End Property

  Property Let MyPrivateString(ByVal str)
    m_sMyPrivateString = str
  End Property

  Property Get MyPrivateObject()
    Set MyPrivateObject = m_oMyPrivateObject
  End Property

  Private Property Set MyPrivateObject(ByRef obj)
    Set m_oMyPrivateObject = obj
  End Property
End Class
```

   The public field `m_iMyPublicInteger` can be accessed from any code block, so defining a getter and setter (as properties are often referred to) for such a field is optional. However, it is a good practice to define fields as private and explicitly provide access through public properties. For fields that are for exclusive use of the class members, one can define the properties as private. In such a case, usually, the setter (`Property Let` or `Property Set`) would be defined as private, while the getter (`Property Get`) would be defined as public. This way, one can prevent other code components from making changes to the internal fields of the class to ensure data integrity and validity.

4. Define the class methods and subroutines. A method is a function, which is a member of a class. Like fields and properties, methods (as well as subroutines) can be `Private` or `Public`. For example:

```
Class MyFirstClass
  '… Continued
  Private Function MyPrivateFunction(ByVal str)
    MsgBox TypeName(me) & " - Private Func: " & str
    MyPrivateFunction = 0
  End Function

  Function MyPublicFunction(ByVal str)
    MsgBox TypeName(me) & " - Public Func: " & str
    MyPublicFunction = 0
  End Function

  Sub MyPublicSub(ByVal str)
    MsgBox TypeName(me) & " - Public Sub: " & str
  End Sub
End Class
```

Note

Keep in mind that subroutines do not return a value. Functions by design should not return a value, but they can be implemented as a subroutine. A better way is to, in any case, have a function return a value that tells the caller if it executed properly or not (usually zero (`0`) for no errors and one (`1`) for any fault). Recall that a function that is not explicitly assigned a value function and is not explicitly assigned a value, will return empty, which may cause problems if the caller attempts to evaluate the returned value.

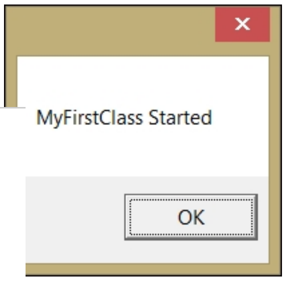
5. Now, we define how to initialize the class when a VBScript object is instantiated:

```
Set obj = New MyFirstClass
```

The `Initialize` event takes place at the time the object is created. It is possible to add code that we wish to execute every time an object is created. So, now define the standard private subroutine `Class_Initialize`, sometimes referred to (albeit only by analogy) as the constructor of the class. If implemented, the code will automatically be executed during the `Initialize` event. For example, if we add the following code to our class:

```
Private Sub Class_Initialize
  MsgBox TypeName(me) & " started"
End Sub
```

Now, every time the `Set obj = New MyFirstClass` statement is executed, the following message will be displayed:


MyFirstClass Started

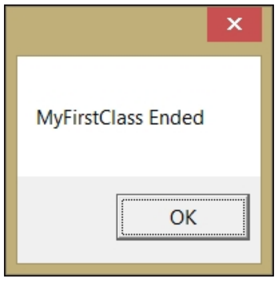
ine how to finalize the class. We finalize a class when a ;cript object is disposed of (as follows), or when the script exits current scope (such as when a local object is disposed when a function returns control to the caller), or a global object is disposed (when UFT ends its run session):

```
Set obj = Nothing
```

The `Finalize` event takes place at the time when the object is removed from memory. It is possible to add code that we wish to execute, every time an object is disposed of. If so, then define the standard private subroutine `Class_Terminate`, sometimes referred to (albeit only by analogy) as the destructor of the class. If implemented, the code will automatically be executed during the `Finalize` event. For example, if we add the following code to our class:

```
Private Sub Class_Terminate
  MsgBox TypeName(me) & " ended"
End Sub
```

Now, every time the `Set obj = Nothing` statement is executed, the following message will be displayed:


MyFirstClass Ended

7. Invoking (calling) a class method or property is done as follows:

```
'Declare variables
Dim obj, var

'Calling MyPublicFunction
obj.MyPublicFunction("Hello")

'Retrieving the value of m_sMyPrivateString
var = obj.MyPrivateString

'Setting the value of m_sMyPrivateString
obj.MyPrivateString = "My String"
```

Note that the usage of the public members is done by using the syntax `obj.<method or property name>`, where `obj` is the variable holding the reference to the object of class. The dot operator (`.`) after the variable identifier provides access to the public members of the class. Private members can be called only by other members of the class, and this is done like any other regular function call.

8. VBScript supports classes with a default behavior. To utilize this feature, we need to define a single default method or property that will be invoked every time an object of the class is referred to, without specifying which method or property to call. For example, if we define the public method `MyPublicFunction` as default:

```
Public Default Function MyPublicFunction(ByVal str)
  MsgBox TypeName(me) & " - Public Func: " & str
  MyPublicFunction = 0
End Function
```

   Now, the following statements would invoke the `MyPublicFunction` method implicitly:

```
Set obj = New MyFirstClass

obj("Hello")
```

   This is exactly the same as if we called the `MyPublicFunction` method explicitly:

```
Set obj = New MyFirstClass

obj.MyPublicFunction("Hello")
```

   Note

   Contrary to the usual standard for such functions, a default method or property must be explicitly defined as `public`.

9. Now, we will see how to add a constructor-like function. When using classes stored in function libraries, UFT (know as QTP in previous versions), cannot create an object using the `New` operator inside a test Action.

   In general, the reason is linked to the fact that UFT uses a wrapper on top of WSH, which actually executes the VBScript (VBS 5.6) code. Therefore, in order to create instances of such a custom class, we need to use a kind of constructor function that will perform the `New` operation from the proper memory namespace. Add the following generic constructor to your function library:

```
Function Constructor(ByVal sClass)
  Dim obj

  On Error Resume Next

  'Get instance of sClass
  Execute "Set obj = New [" & sClass & "]"
  If Err.Number <> 0 Then
    Set obj = Nothing
    Reporter.ReportEvent micFail, "Constructor", "Failed to create
  End If

  Set Constructor = obj
End Function
```

   We will then instantiate the object from the UFT Action, as follows:

```
Set obj = Constructor("MyFirstClass")
```

   Consequently, use the object reference in the same fashion as seen in the previous line of code:

```
obj.MyPublicFunction("Hello")
```

How it works...

As mentioned earlier, using the internal public fields, methods, subroutines, and properties is done using a variable followed by the dot operator and the relevant identifier (for example, the function name).

As to the constructor, it accepts a string with the name of a class as an argument, and attempts to create an instance of the given class. By using the `Execute` command (which performs any string containing valid VBScript syntax), it tries to set the variable `obj` with a new reference to an instance of `sClass`. Hence, we can handle any custom class with this function. If the class cannot be instantiated (for instance, because the string passed to the function is faulty, the function library is not associated to the test, or there is a syntax error in the function library), then an error would arise, which is gracefully handled by using the error-handling mechanism (as described in Chapter 6, *Event and Exception Handling*), leading to the function returning nothing. Otherwise, the function will return a valid reference to the newly created object.

See also

The following articles at www.advancedqtp.com (http://www.advancedqtp.com) are part of a wider collection, which also discuss classes and code design in depth:

- An article by Yaron Assa at http://www.advancedqtp.com/introduction-to-classes (http://www.advancedqtp.com/introduction-to-classes)
- An article by Yaron Assa at http://www.advancedqtp.com/introduction-to-code-design (http://www.advancedqtp.com/introduction-to-code-design)
- An article by Yaron Assa at http://www.advancedqtp.com/introduction-to-design-patterns (http://www.advancedqtp.com/introduction-to-design-patterns)