Implementing a generic Login class

In this recipe, we will see how to implement a generic `Login` class. Th class captures both, the GUI structure and the processes that are common to all applications with regard to their user access module. It agnostic to the particular object classes, their technologies, and other identification properties. The class shown here implements the comm wrapper design pattern, as it encapsulates a process (`Login`) with the main default method (`Run`).

Getting ready

You can use the same function library `cls.Google.vbs` as in the previous recipe *Implementing a simple search class*, or create a new o (for instance, `cls.Login.vbs`) and associate it with your test.

How to do it...

1. In the function library, we will write the following code to define the class `Login`:

```
Class Login
  Private m_wndContainer  'Such as a Browser, Window, SwfWindow
  Private m_wndLoginForm  'Such as a Page, Dialog, SwfWindow
  Private m_txtUsername  'Such as a WebEdit, WinEdit, SwfEdit
  Private m_txtIdField  'Such as a WebEdit, WinEdit, SwfEdit
  Private m_txtPassword  'Such as a WebEdit, WinEdit, SwfEdit
  Private m_chkRemember  'Such as a WebCheckbox, WinCheckbox, SwfCh
  Private m_btnLogin    'Such as a WebEdit, WinEdit, SwfEdit
End Class
```

These fields define the test objects, which are required for any `Login` class, and the following fields are used to keep runtime data for the report:

```
  Public  Status  'As Integer
  Public  Info  'As String
```

The `Run` function is defined as a `Default` method that accepts a `Dictionary` as argument. This way, we can pass a set of named arguments, some of which are optional, such as `timeout`.

```
Public Default Function Run(ByVal ArgsDic)
    'Check if the timeout parameter was passed, if not assign it 1(
    If Not ArgsDic.Exists("timeout") Then ArgsDic.Add "timeout", 1(
    'Check if the client window exists
    If Not me.Container.Exist(ArgsDic("timeout")) Then
      me.Status  = micFail
      me.Info    = "Failed to detect login browser/dialog/window."
      Exit Function
    End If
    'Set the Username
    me.Username.Set ArgsDic("Username")
    'If the login form has an additional mandatory field
    If me.IdField.Exist(ArgsDic("timeout")) And ArgsDic.Exists("IdF
      me.IdField.Set ArgsDic("IdField")
    End If
    'Set the password
    me.Password.SetSecure ArgsDic("Password")
    'It is a common practice that Login forms have a checkbox to ke
    If me.Remember.Exist(ArgsDic("timeout")) And ArgsDic.Exists("Re
      me.Remember.Set ArgsDic("Remember")
    End If
    me.LoginButton.Click
  End Function
```

The `Run` method actually performs the login procedure, setting the username and password, as well as checking or unchecking the **Remember Me** or **Keep me Logged In** checkbox according to the argument passed with the `ArgsDic` dictionary.

The `Initialize` method accepts `Dictionary` just like the `Run` method. However, in this case, we pass the actual TOs with which we wish to perform the login procedure. This way, we can actually utilize the class for any `Login` form, whatever the technology used to develop it. We can say that the class is *technology agnostic*. The parent client dialog/browser/window of the objects is retrieved using the `GetTOProperty("parent")` statement:

```
Function Initialize(ByVal ArgsDic)
    Set m_txtUsername  = ArgsDic("Username")
    Set m_txtIdField  = ArgsDic("IdField")
    Set m_txtPassword  = ArgsDic("Password")
    Set m_btnLogin    = ArgsDic("LoginButton")
    Set m_chkRemember  = ArgsDic("Remember")
    'Get Parents
    Set m_wndLoginForm = me.Username.GetTOProperty("parent")
    Set m_wndContainer = me.LoginForm.GetTOProperty("parent")
  End Function
```

In addition, here you can see the following properties used in the class for better readability:

```
Property Get Container()
    Set Container = m_wndContainer
  End Property
  Property Get LoginForm()
    Set LoginForm = m_wndLoginForm
  End Property
  Property Get Username()
    Set Username = m_txtUsername
  End Property
  Property Get IdField()
    Set IdField = m_txtIdField
  End Property
  Property Get Password()
    Set Password = m_txtPassword
  End Property
  Property Get Remember()
    Set Remember = m_chkRemember
  End Property
  Property Get LoginButton()
    Set LoginButton = m_btnLogin
  End Property

  Private Sub Class_Initialize()
    'TODO: Additional initialization code here
  End Sub
  Private Sub Class_Terminate()
    'TODO: Additional finalization code here
  End Sub
```

We will also add a custom function to override the `WinEdit` and `WinEditor Type` methods:

```
Function WinEditSet(ByRef obj, ByVal str)
  obj.Type str
End Function
```

This way, no matter which technology the textbox belongs to, the `Set` method will work seamlessly.

2. To actually test the `Login` class, write the following code in the Test Action (this time we assume that the `Login` form was already opened by another procedure):

```
Dim ArgsDic, oLogin

'Register the set method for the WinEdit and WinEditor
RegisterUserFunc "WinEdit", "WinEditSet", "Set"
RegisterUserFunc "WinEditor", "WinEditSet", "Set"

'Create a Dictionary object
Set ArgsDic = CreateObject("Scripting.Dictionary")
'Create a Login object
Set oLogin = New Login

'Add the test objects to the Dictionary
With ArgsDic
  .Add "Username", Browser("Gmail").Page("Gmail").WebEdit("txtUserr
  .Add "Password", Browser("Gmail").Page("Gmail").WebEdit("txtPassw
  .Add "Remember", Browser("Gmail").Page("Gmail").WebCheckbox("chkF
  .Add "LoginButton", Browser("Gmail").Page("Gmail").WebButton("btr
End With

'Initialize the Login class
oLogin.Initialize(ArgsDic)

'Initialize the dictionary to pass the arguments to the login
ArgsDic.RemoveAll
With ArgsDic
  .Add "Username", "myuser"
  .Add "Password", "myencriptedpassword"
  .Add "Remember", "OFF"
End With

'Login
oLogin.Run(ArgsDic) 'or: oLogin(ArgsDic)

'Report result
Reporter.ReportEvent oLogin.Status, "Login", "Ended with " & GetSta

'Dispose of the objects
```

```
Set oLogin = Nothing
Set ArgsDic = Nothing
```

How it works...

Here, we will not delve into the parts of the code already explained in the *Implementing a simple search class* recipe. Let's see what we did in this recipe:

- We registered the custom function `WinEditSet` to the `WinEdit` and `WinEditor` TO classes using `RegisterUserFunc`. As discussed previously, this will make every call to the method set to be rerouted to our custom function, resulting in applying the correct method to the Standard Windows text fields.
- Next, we created the objects we need, a `Dictionary` object and a `Login` object.
- Then, we added the required test objects to `Dictionary`, and then invoked its `Initialize` method, passing the `Dictionary` as the argument.
- We cleared `Dictionary` and then added to it the values needed for actually executing the login (`Username`, `Password`, and the whether to remember the user or keep logged in checkboxes usually used in `Login` forms).
- called the `Run` method for the `Login` class with the newly ulated `Dictionary`.
- er, we reported the result by taking the `Status` and `Info` public ls from the `oLogin` object.
- he end of the script, we unregistered the custom function from lasses in the environment (`StdWin` in this case).