Overriding a Test Object metho

In this recipe, we will see how to write a new implementation of a metl
for a TO.

Getting ready

From the **File** menu, navigate to **New** | **Function Library...** or use t
Alt + *Shift* + *N* shortcut. Name the new function library as
`FR_RegFunc.vbs`.

How to do it...

As always, with programming, the task needs to be addressed in an
orderly fashion. Therefore, there is a series of implementation steps
follow:

1. Analyze the requirement, which means ask questions. For example:
   - What are the missing functions?
   - To which object class is it relevant?
2. Design the solution.
3. Code the function.
4. Test the function.
5. Register the function to the required object class.

In the following example, we will write a function in `FR_RegFunc.vbs` that overrides the `WinEdit_Set` method. The new method will try to set the field, and if an error occurs, it will check if there is a modal pop-up message that has opened in the Flight Reservation application (refer to Chapter 1, *Data-driven Test*). If it has opened, the method will close it and the flow may continue.

```
'If there is a problem when trying to set a WinEdit control with text, th
Function FR_WinEdit_Set(obj, text)
    On error resume next
    reporter.Filter = rfDisableAll 'Disable automatic reporting
    obj.set text
    reporter.Filter = rfEnableAll 'Enable automatic reporting
    'If the operation failed
    If err.number <> 0  Then
        'Report a warning so the test does not fail
        reporter.ReportEvent micWarning, "Set on " & obj.GetTOProperty("n
            err.number & ": " & err.description
        'An error was found, check if a popup dialog is open
        If obj.GetTOProperty("parent").Dialog("ispopupwindow:=true").exis
            'Report and Close popup
            Reporter.ReportEvent micDone, "Popup dialog found", "Closing
            obj.GetTOProperty("parent").Dialog("ispopupwindow:=true").Win
            'TODO: Decide which implementation is more suitable
            '1. We can try to set the field again
            '2. Return the control to the calling Action (as we do here)
            '3. Other
        End If
    End If
End Function
```

We then run `Action1` with the following lines of code:

```
'Register the overriding method
RegisterUserFunc "WinEdit", "Set", "FR_WinEdit_Set"
'Try to set the Agent Name field in the FR Login dialog
Dialog("Login").WinEdit("Agent Name").Set "mercury"
'Unregister the overriding method
UnregisterUserFunc "WinEdit", "Set"
```

Note

It is important to note that RegisterUserFunc is used during runtime to load the custom function for native method mappings. However, it is also possible to do this while designing via the UFT GUI, by navigating to **Design | Function Definition Generator** from the menu. This will provide you with autocomplete/intellisense in your test. Of course, the function library containing the custom functions must be available in the test's resources.

How it works...

The implemented custom method takes two arguments, namely, obj and text. The first is for the TO, WinEdit and the second is for the text to be entered. First, to obtain full control over the flow, we disable VBScript's native runtime error handling mechanism with On Error Resume Next. Second, to avoid the test being marked as failed automatically, we disable UFT's automatic event reporting by assigning Reporter.Filter = rfDisableAll to the Filter property of the Reporter object. Next, we set the value and restore Filter to its default value Reporter.Filter = rfEnableAll.

Mode	Description
0 or rfEnableAll	This is the default value. All reported events are displayed in the run results.
1 or rfEnableErrorsAndWarnings	This mode displays events with a warning or failed status in the run results.
2 or rfEnableErrorsOnly	This mode displays events with a failed status in the run results.
3 or rfDisableAll	This mode does not display any events in the run results.

In the following example, we will demonstrate all Filter properties of the Reporter object combinations from the preceding table:

```
Reporter.ReportEvent micPass, "Step 1", "Passed"
Reporter.ReportEvent micFail, "Step 2", "Failed"

'Disable all the Results
Reporter.Filter = rfDisableAll
Reporter.ReportEvent micPass, "Step 3", "Passed"
Reporter.ReportEvent micFail, "Step 4", "Failed"

'Enable Result Display
Reporter.Filter = rfEnableAll
Reporter.ReportEvent micWarning, "Step 5", "Warning"

'Enable only Errors and Warnings
Reporter.Filter = rfEnableErrorsAndWarnings
Reporter.ReportEvent micPass, "Step 6", "Passed"
Reporter.ReportEvent micFail, "Step 7", "Failed"
Reporter.ReportEvent micWarning, "Step 8", "Warning"
```

If an error of any kind occurs, it will be caught by the If err.number <> 0 Then clause. Then, our custom exception handling will be executed. In the preceding example, we report all types of warnings, but a specific implementation may select one type, depending on the requirements. For instance, the error may occur under controlled conditions (such as negative test(s)), and hence, our implementation should be more complex to cover such situations. In any case, it is recommended to leave the custom function as simple as possible.

The next step is to check if the parent container (window or dialog) has a child (owned) pop-up dialog open, which, it is reasonable to assume, is modal and therefore obstructs the target WinEdit, causing the error. If this is the case, then we report our findings and click on OK on the pop-up dialog WinButton.

There's more...

At this stage one may ask, what now? How do we decide on the correct implementation? As mentioned earlier, this depends on the requirements. For example, if the previous custom method is meant to be a recovery scenario, then we might want to add the following code to close the pop-up code that ensures WinEdit is actually assigned the text passed to the function. In such a case, our function code would change to:

```
'Continued…
```

```
obj.GetTOProperty("parent").Dialog("ispopupwindow:=true").WinButton("text
obj.Set text
```

It is not recommended to use a recursion, for example, with the following:

```
Call FR_WinEdit_Set(obj, text)
```

However, it is possible to shorten the syntax:

```
obj.set text
```

There are two limitations that must be taken into account when using the `RegisterUserFunc`:

- Number of arguments
- Interoperability of registered functions

Number of function arguments

When defining a function that overrides a method, it must have the same signature. This means that the overriding function cannot have a number of arguments different from the original method that is overridden. A workaround is to have one of the mandatory arguments sent as an array or, even better, as a dictionary. This way, you can have a customized version of the method that, in practice, is able to operate with a different number of arguments. It is even possible to design it in such a way that the custom method will treat items of the array or dictionary as optional.

Interoperability of registered functions

When a registered function includes a call to another registered function, be careful and use the correct syntax. To call a registered function so that no changes to existing calls should be carried out, we usually put a statement such as:

```
call obj.[native method]([arg1], [...], [argn])
```

To avoid a VBScript runtime error (`Type Mismatch`) during your run session, when a call from one overriding method to another is required, the limitations can be overcome by coding the call as follows:

```
call [custom method](obj, [arg1], [...], [argn])
```

See also

The following articles on www.advancedqtp.com (http://www.advancedqtp.com) also discuss `RegisterUserFunc` in depth:

- An article by Yaron Assa at http://www.advancedqtp.com/a-fresh-look-on-registeruserfunc (http://www.advancedqtp.com/a-fresh-look-on-registeruserfunc)
- An article by Meir Bar-Tal at http://www.advancedqtp.com/override-the-object-exist-property (http://www.advancedqtp.com/override-the-object-exist-property)
- An article by Meir Bar-Tal http://www.advancedqtp.com/limitations-of-registeruserfunc (http://www.advancedqtp.com/limitations-of-registeruserfunc)