Handling pop-up dialogs

It is common to encounter pop-up dialogs that open up while using software applications. Mostly, these are application modals, which mean that no further operations can be performed within the application context until the dialog is closed. Some can be system modals, meaning that no further operations can be performed on the machine until the dialog is closed. Quite often, these dialogs offer various options presented as buttons, such as **OK**, **Approve**, **Submit**, **Apply**, **Cancel**, **Ignore**, and **Retry**. This variety needs to be managed in a very accurate fashion, as the choice made affects the rest of the test flow substantially. Moreover, sometimes another pop-up dialog may show up as a direct result of a given choice. Such an event may be delayed a bit, for example, due to server-side validation, and hence it is of utmost importance to detect it in a reliable yet efficient way.

The basic problem with pop-up dialogs is that, quite often, their appearance is unexpected. For instance, if there is some script error as a result of a bug, then a dialog will appear, but our script would not know how to handle it unless we put that logic or intelligence into the code. If we fail to do so, then our script will make a futile attempt to continue the normal flow, and hence, precious time and resources would be lost. On the other hand, in such a case, we would like our script to detect such an error dialog and report that a problem may have been found. Perhaps we would like to exit that specific action or test, or even halt the whole run session.

pected pop-up dialogs is using the UFT built-in . However, in my view, this practice is not ormance issues and implementation not be covered here. Instead, we will suggest a generic technique that is very simple to implement and can be custom tailored to any specific requirement that may arise.

Getting ready

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

How to do it...

If we refrain from using the recovery scenario feature as I recommended, then the question remains, how can we have any pop-up dialog appearance covered with the least amount of code? If we take the risk of such dialogs too seriously, then we may end up with our code cluttered with `If-Then-End If` statements, just to check that our application context is normal and no pop-up dialog is opened.

The approach I will advocate here assumes that the risk of unexpected pop-up dialogs (for instance, due to bugs) for a mature application is minimal. So instead of listening to pop-up dialogs all the time (as is the case for a recovery scenario), we will check if there is a pop-up dialog open, just in case an operation fails. For example, if we try to click on an object on a web page while a pop-up dialog is open, a runtime error will be thrown by UFT. To prevent a UFT pop-up dialog from opening and hence pausing the run session, we will catch the error inside our code. After the pop-up dialog is handled (closed), our test will continue, stop, or reroute the flow according to the analysis of the situation. Here, we shall assume that the dialog is not consequential to the flow, and that just closing it solves the problem.

To implement our solution, we need to do two basic things:

- Write one generic function, `DialogHandler()`, which can detect

and handle any open dialog.

- Catch an error in certain methods (where the presence of a pop-up dialog would affect the flow) and invoke the `dialogHandler()` method. Here we will be using the `RegisterUserFunc` technique explained in detail in Chapter 4, *Method Overriding*.

We will then write the following method to handle any pop-up dialog in our `library Web_Functions.vbs` function:

```
Function handleDialog()
    Dim sMessageText

    'A popup dialog can be directly accessed but in some cases its parent
    'For instance: Browser("micclass:=Browser").Dialog("regexpwndclass:=#
    With Dialog("regexpwndclass:=#32770")
        'Check if a Dialog exists
        If .Exist(0) Then
            'Focus on the Dialog
            .Activate
            'Get the static text
            sMessageText= .Static("regexpwndtitle:=.+").GetTOProperty("te
            'Click on the OK button (can be parameterized in case of need
            .WinButton("text:=OK").Click
            'Check again to verify that the Dialog was closed
            If not .Exist(0) Then
                Reporter.ReportEvent micPass, "handleDialog", "Dialog wit
                handleDialog=true
            else
                Reporter.ReportEvent micFail, "handleDialog", "Dialog wit
                handleDialog=false
            End If
        else
            'No dialog was found so we return true
            handleDialog=true
        End If
    End With
End Function
```

As an example, we will write the following overriding method in our `Web_RegisteredFunctions.vbs` function library:

```
Function WebEdit_Set(obj, text)
    On error resume next
    'Try
    obj.set text
    'Catch
    if err.number <> 0 then
        'If there's a dialog open that is handled then retry.
        if handleDialog() then
            obj.set text
        else
            Reporter.ReportEvent micFail, "WebEdit_Set", "An error occur
            'Stop the run session (or handle otherwise)
            ExitTest()
        End if
    End if
End Function
```

Of course, here we assume that closing the dialog is a good enough solution, but this may not be the case. If a script error caused the browser to open a pop-up dialog, then it may reopen. In such a case, a more sophisticated scheme would be required, which is out of the scope of this basic recipe. Another thing that is worth noting is that if the `HandleDialog()` method does not find any dialog open, it is up to the calling function or action to check for other possible problems that caused the error. As mentioned earlier, the modal dialog may be implemented as a `Div` element, so the inline descriptive programming-based description would not fit.

> **Note**
>
> The previous function serves only as an example of how to implement the approach outlined in this recipe. Of course, the same logic should be implemented for each operation (click, double-click, and so on) that can be blocked by a pop-up dialog.

We will register the previous `WebEdit_Set` method before starting the test flow and unregister it at the end of the flow (refer to Chapter 4, *Method Overriding*):

```
RegisterUserFunc "WebEdit", "Set", "WebEdit_Set"

'Test Flow goes here...

UnregisterUserFunc "WebEdit", "Set"
```

How it works...

The `HandleDialog()` function uses a generic description to identify a dialog and close an open one. Of course, this is a simplified version and may need to be expanded. For instance, to make the function able to also handle application modal pop-up dialogs built on web `Div` elements

with JavaScript, one should add suitable working code with a matching description. In addition, the function is built on the assumption that there is an **OK** button to close the dialog. This, however, may not be the case, and dialogs with more than a single button would require a more elaborate method.

The overriding `WebEdit_Set(obj, text)` method is an example of how to achieve the effect of detecting an obstructing open modal dialog. First, we disable the automatic runtime mechanism for error handling with `On Error Resume Next`. Next, we try to perform the operation on the input field. If the operation fails, the error is trapped and `HandleDialog()` is invoked.

PREV
Managing multipl…

NEXT
Downloading a fil…

Recommended / Queue / Recent / T … y.safaribooksonline.com/) /
Sign Out