Using method overriding to support object subtypes

In some cases, we may need to implement different versions of the same native method to reflect different subtypes of a particular cont class. To achieve this, we can use method overriding with `RegisterUserFunc` and reroute our test flow dynamically. For examp suppose that **Application Under Test** (**AUT**) uses different types of input controls, such as a password (encrypted) field, specially formal fields for ID, date, and email, or even just for text, as opposed to numeric only data. In such cases, we might wish to use specific code validate the input data or even (in a very advanced implementation) t generate it randomly.

Let us take, for example, two input fields (`WebEdit`) having different class HTML native attributes. We will write a custom version of the `WebEdit_Set` method (actually a delegate) that will reroute the functic to execute a specific piece of code for the given element class.

Getting ready

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

In an external HTML editor, create an HTML file and name it `WebEdit_Subtype.html`. Write the following HTML code and save the file:

```
<html>
  <head>
    <title>AdvancedQTP - Overriding TO Methods Example
    </title>
  </head>
  <body>
    <div><span role="label">Username:</span><input class="normal" type="t
    <div><span role="label">Password:</span><input class="pwd" type="pass
    <div><span role="label">Email:</span><input class="normal" type="text
    <div><span role="label">Tel:</span><input class="not_defined" type="t
  </body>
  <footer>
  </footer>
</html>
```

Open the file in Internet Explorer and add the four input fields to the local OR. At the end of every stage, do save the test.

How to do it...

Perform the following steps:

1. In the function library, we will write the following code:

```
Function WebEdit_SetEx(obj, str)
    'Reroute the flow based on the obj class
    Select case lcase(obj.GetTOProperty("class"))
        Case "normal"
            print "Regular input"
            obj.set str
        Case "pwd", "enc"
            print "Password input"
            obj.SetSecure
        case else
            print "N/A - Assuming regular input"
            obj.set str
    End Select
End Function
```

2. In `Action`, we will write our code as usual:

```
RegisterUserFunc "WebEdit", "Set", "WebEdit_SetEx"

with Browser("AdvancedQTP - Overriding").Page("AdvancedQTP - Overri
    .WebEdit("username").Set "Username"
    .WebEdit("password").Set "Password123"
    .WebEdit("email").Set "email@email.com"
    .WebEdit("tel").Set "(01) 23 456 789"
End with
UnregisterUserFunc "WebEdit", "Set"
```

The flow has rerouted behind the scenes, as explained in the previous recipes. The print log in the **Output** pane, for the four input fields defined in our HTML page, are:

Regular input

Password input

Regular input

N/A - Assuming regular input

How it works...

The `WebEdit_Set` method was overridden in a different way. While entering the function, the code checks the type of control that invoked the method (the sender) by retrieving the value of its class attribute. According to the result of this inquiry, the function enters the piece of code that was written to handle the specific control subtype. In the case of the `Password` field, the `SetSecure` method is invoked instead of `Set`, to enter the value. In this case, we could have also used the `Crypt.Encrypt` method with `Set`, as follows:

```
Function WebEdit_SetEx(obj, str)
    'Reroute the flow based on the obj class
    Select case lcase(obj.GetTOProperty("class"))
        Case "normal"
            print "Regular input"
            obj.set str
        Case "pwd", "enc"
            print "Password input"
            obj.set Crypt.Encrypt(str)
        case else
            print "N/A - Assuming regular input"
            obj.set str
    End Select
End Function
```

There's more...

Another possible application of `RegisterUserFunc` is to have several versions of a method to support different requirements. The specific custom method to use should be selected during runtime from any Action or function, by calling `RegisterUserFunc` and `UnregisterUserFunc`. This may cover cases where we are required to support the following:

- Different versions of the testware framework
- Different configurations of AUT
- The behavior of a TO with context-dependent or dynamically changing behavior

However, if we do change the method registrations during runtime, then we must be extremely cautious, as it may have an impact on the test results. A good practice to meet this risk would be to implement a tracking mechanism, for instance, with a `GlobalDictionary` object (refer to Chapter 1, *Data-driven Tests*) from which you would be able to retrieve any class and method, which are the current effective and registered custom methods. Accordingly, it would be possible to validate that the correct method is registered within a given context.