Using global variables (Environment)

The `Environment` object can be best described as a kind of **Dictiona** that holds pairs of keys and values. Unlike the native Dictionary, it h extended capacities such as the following:

- UFT features that contain built-in variables, which can return useful values during runtime, such as `ActionName`, `ActionIteration`, `TestName`, `TestIteration`, `TestDir`, and `OS`
- Defining test internal variables (that is, persistent) or external variables (see the following point)
- Loading a set of variables from an external XML file, preset, or dynamically used during runtime
- **Automation Object Model** (**AOM**) support, which enables the addition of internal variables dynamically before launching a tes run session
- The scope of the object is global and is loaded automatically wh UFT opens, as is true for all reserved objects

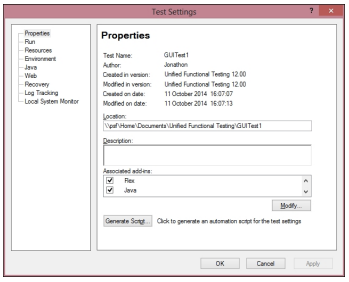
Getting ready

From the **File** menu, navigate to **New** | **Test**, or use the *Ctrl + N* shortcut.

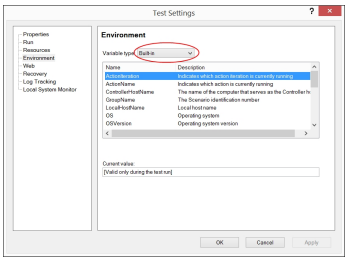
How to do it…

Proceed with the following steps:

1. For built-in variables:
    1. Navigate to **File | Settings**. The **Test Settings** dialog will open, as shown in the following screenshot. The **Environment** item (tab) is circled in red:



    2. Select **Environment**, as shown in the previous screenshot, and the dialog will display the built-in variables list. Scroll to explore the variables. Take note that the list here is a field labelled **Current value**, which will show the variable's value only if it is not a runtime determined value. Examples for the latter are `ActionIteration` and `ActionName`, while the variables `OS`, `TestName`, and `TestDir` are examples for such values that UFT can retrieve, independent of its running state:



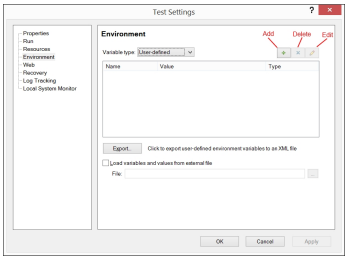
Retrieving the values of built-in variables during runtime is done using code such as the following:

```
Print Environment("ActionIteration")
```

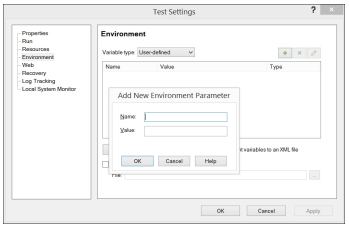
This prints the value of the current `Action` iteration (when runs with input data from the local `DataSheet`). It is possible, of course, to use such variables to control the flow, as is shown in the *Importing an Excel file to a test* recipe in Chapter 1, *Data-driven Tests*.

Built-in variables are read-only.

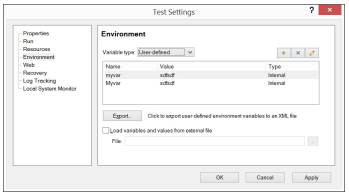
2. For user-defined variables:
    1. You can define your own `Environment` variables according to the requirements or needs. It is important to keep in mind that, being an object having global scope, the `Environment` object is very useful to store configuration data that is used across tests (for example, website URL, super username and encrypted password, and so on). Though technically feasible, it is not really recommended to use this object as a means to store runtime data that needs to be shared across actions. For that purpose, using a globally defined Dictionary would be much more suitable.
    2. From the **Variable type** list, select **User-defined**. The following screen will be displayed. The main buttons used to edit the variables list (Add, Delete and Edit) are labeled in the following screenshot:



To add, click on the **+** icon. The **Add New Environment Parameter** dialog will pop up. Enter a variable name and value in the appropriate fields, and click on the **OK** button:

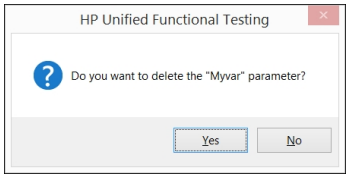


Add two variables, `myvar` and `Myvar`. The next screenshot shows that variable names are case sensitive:

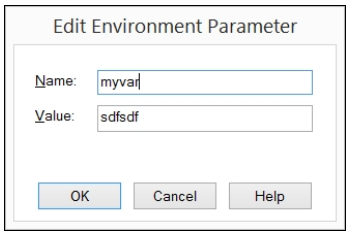


As you can see, the third column labeled **Type** indicates that the variables are `Internal`. This means that the variables are specific to the current test.

To delete, select the second variable and click on the **x** icon. The following dialog will appear:



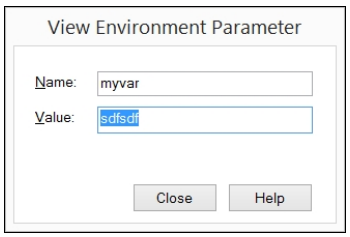
To edit, in order to change the value of a variable, select it and click on the edit icon. The **Edit Environment Parameter** dialog will appear:



Edit the value as per your will and click on **OK** to approve. Your changes will be kept and seen on the variables list in the **Test Settings** dialog.

> **Note**
>
> Though the **Name** field appears to be enabled, it is actually a read-only field. This means that one can only change the value of an `Environment` variable, not its name. This is important to avoid problems related to existing code, which already refer to previously defined `Environment` variables. Still, if one wishes to make such a name change, the option of adding a new variable and deleting the old one is always open. Just keep in mind that you may need to make changes to your code.

3. To export:
    1. The **Environment** object can be stored as an XML file. This is useful in order to make general configuration settings available across tests and even test environments or platforms. Click on the **Export** button, select where you want to store the file from the **Save Environment Variable File** dialog that opens, and click on the **Save** button.
4. To import:
    1. As mentioned earlier, it is possible to reuse values previously exported to an XML file. Just mark the checkbox labeled with load variables and values from an external file, and click on the **Browse...** button on the right. Then, select the file from the **Open Environment Variable File** dialog and click on the **Open** button. Please note that for all variables that were loaded from the file, the **Type** column now has the value of `External`. Consistent with the logic of reusability, these variables cannot be changed from within the **Test Settings** dialog. They are read-only. You can check this by clicking on the edit icon or instead, by using the **Edit Environment Parameter** dialog, you get the **View Environment Parameter** dialog. The fields are both read-only:



It is also possible to import an XML file during runtime. The syntax is as follows:

```
Dim sFilePathname = "C:\Automation\Config\Env_1.xml"

Environment.LoadFromFile sFilePathname
```

Also, if you load it before the test starts to run (using the AOM with an external VBS file, for instance), then the optional `Boolean` argument, `KeepLoaded`, is also required:

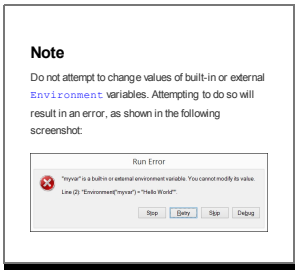
```
Environment.LoadFromFile sFilePathname, True
```

Otherwise, the variables and values will be lost later.

5. Runtime creation and update:

    1. Though not recommended (as I said, it is better to use a global Dictionary for runtime data sharing), it is technically possible to create user-defined variables during runtime. It is even possible to update their values. These would, of course, disappear from memory when the run is over. The code is—not surprisingly—similar to that of a Dictionary:

```
Environment("MyVarName") = "MyVarValue"
```



6. Retrieving values during runtime:

    This can be easily done with code as follows:

```
Print Environment("MyVarName")
```

    If the variable does not exist, an error, as shown in the following screenshot, will pop up:

How it works…

The previous section was quite thorough in describing the workings of the `Environment` object, so here we will summarize.

We have seen the main uses of the `Environment` object as a way to define variables that are required during the run session. We have described how to add new persistent variables and delete/edit existing ones using the test settings during design time. We have also explained how, during runtime, one can create new variables and change their values, as well as how to retrieve the values of any **Environment** variable (be it Internal, External, or runtime).

Finally, we discussed the features of Export and Import, stressing that this is how we attain reusability of required configuration variables across tests.

See also

- Assa, Y. (2008) *Reserved Objects as an Env Object Replacement*, at http://www.advancedqtp.com/reserved-objects-as-an-env-object-replacement (http://www.advancedqtp.com/reserved-objects-as-an-env-object-replacement)

- Vainstein, D. (2008) *Viewing and Editing Environment Complex Parameter Values*, at http://www.advancedqtp.com/viewing-and-editing-environment-complex-parameter-values/ (http://www.advancedqtp.com/viewing-and-editing-environment-complex-parameter-values/)