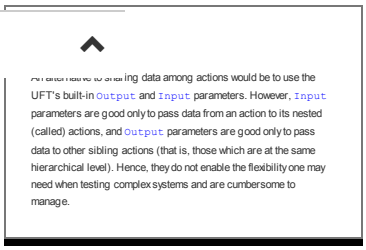
Using a global dictionary for fast shared data access

Using a DataTable is a generally good practice because spreadsheet data is easy to create, visualize, and maintain. This is because MS Excel lies behind the `DataTable`, which is, as mentioned before, a wrapper to the Excel COM object. Other advantages of using the `DataTable` include its full integration with the test and action iterations mechanism and with the results report, in which one can visualize each iteration, along with the input data.

This is all good for the retrieval of input data that is prepared during design time. However, using the `DataTable` for sharing between actions has two main drawbacks during runtime:

- Repeated writes and reads may hinder performance when it comes to a large number of iterations and a large number of parameters, as is quite often the case with many information systems.
- Sharing data with `GlobalSheet` is very difficult to implement. For example, suppose we need to store the `CustomerID` given by the system upon customer creation. In `GlobalSheet`, it will be stored at the current row. Though we may set the exact row using the `DataTable` method, that is, `SetCurrentRow (<rownumber>)`, it is still a question of how to ensure that at a later stage, an action that needs a `CustomerID` would know the correct row number.



A better approach is to have the data that must be shared and stored in the `Dictionary` object of a global scope. A `Dictionary` object is actually a hash table with a capacity to store values of different types, such as strings, numbers, Booleans, arrays, and references to objects (including other nested `Dictionary` objects, which is a powerful, yet very advanced technique that is out of scope here). Each value is stored with a unique key by which it can be accessed later.

Getting ready

In UFT, create a new function library by navigating to **File** | **New** | **Function Library** (or use the key shortcut *Alt* + *Shift* + *N*) and save it as `UFT_Globals.vbs`. It is recommended to save it in a folder, which would be shared later by all tests.

Navigate to **File** | **Settings** and attach the function library to the test.

How to do it...

As any public variable declared in a function library attached to a test can be accessed by any action, we will define a global variable and two functions to initialize `initGlobalDictionary` and dispose `disposeGlobalDictionary`:

```
Dim GlobalDictionary

Function initGlobalDictionary()
    If not (lcase(typename(GlobalDictionary)) = "dictionary") Then
        Set GlobalDictionary = CreateObject("Scripting.Dictionary")
    End If
End Function
Function disposeGlobalDictionary()
    Set GlobalDictionary = nothing
End Function
```

The `initGlobalDictionary()` function will check if the public variable `GlobalDictionary` was not initialized earlier, and then set it with a reference to a new instance of a `Dictionary` object, as mentioned in the previous code. The `disposeGlobalDictionary()` function is given for the sake of completeness, as in any case, memory is released when the test stops. However, we may wish to empty the `GlobalDictionary` variable in certain cases, so it is recommended to include this function as well.

Now, in `Action1` (or whichever action runs first in our test), we will write the following code:

```
If cint(Environment("TestIteration")) = 1 and cint(Environment("ActionIte
    call initGlobalDictionary()
End If
```

The previous code will ensure that the `GlobalDictionary` variable is instantiated only once at the beginning of the run session. If we need a new instance for every test iteration, then we just need to change the code to the following lines of code, so that we get a new instance only at the start of the first `Action1` iteration:

```
If CInt(Environment("ActionIteration")) = 1 Then
    call initGlobalDictionary()
End If
```

With our test set up this way, we can now use this global object to share data as in the following example. Create a new `Action2` DataTable and make it run after `Action1` (at the end of the test). Now, write the following code in `Action1`:

```
GlobalDictionary.Add "CustomerID", "123456789"
Print Environment("ActionName") & ": " & GlobalDictionary("CustomerID")
```

In `Action2`, write the following code:

```
Print Environment("ActionName") & ": " & GlobalDictionary("CustomerID")
```

It is strongly recommended to remove a key from the dictionary when it is no longer required:

```
GlobalDictionary.Remove "CustomerID"
```

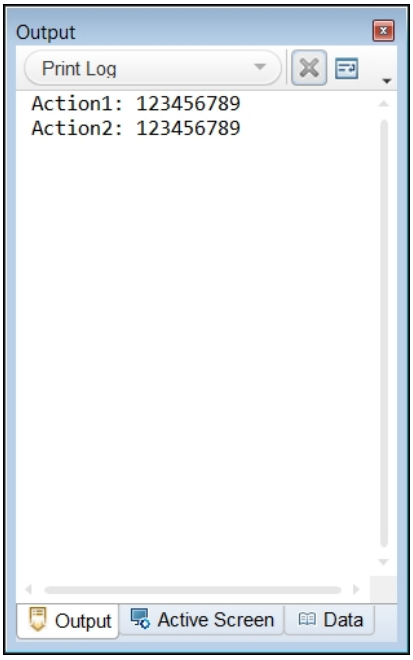
Alternatively, to remove all keys from the dictionary altogether at the end of a test iteration or at the beginning of a test iteration greater than the first, use the following line of code:

```
GlobalDictionary.RemoveAll
```

As mentioned earlier, keys must be unique and if we use the same keys in each test iteration, it would cause a runtime error with the first key found to exist in the dictionary. Another way, as mentioned earlier, is to call the `disposeGlobalDictionary` at the end of each test iteration and the `initializeGlobalDictionary()` method at the start.

How it works...

When you run this test, in `Action1`, it first creates a new `Dictionary` instance and assigns a reference to the public variable `GlobalDictionary`. Then, it adds a new key `CustomerID` with the value `123456789`, and prints the action name from the Environment built-in runtime variables (`"Action1"`) and the value, by referring to the `CustomerID` key we just added. Then, it executes `Action2`, where it again prints in the same manner as in `Action1`. However, as the `ActionName` Environment variable is dynamic, it prints `"Action2"`. This is to prove that `Action2` actually has access to the key and value added in `Action1`. The output of this test is as shown in the screenshot:



See also

Refer to the *Using a global dictionary for fast shared code access* recipe.