Measuring time (MercuryTimers

The `MercuryTimers` object enables measurement of time between an two operations. Unlike the native VBScript `Timer` function, the `MercuryTimers` object supports the utilization of multiple `timer_name` time measurement transactions in parallel. In essence, the `Timers` object is a kind of Dictionary that can store different `Timer` objects, each with a unique key. This can be useful to measure the time elaps at different levels of the run session, and hence, identify possible bottlenecks caused by specific blocks of code.

The `MercuryTimers` object provides the following methods to use with `Timer` object:

- `Start`: This starts measuring time in milliseconds
- `Stop`: This stops measuring time
- `Continue`: This continues to measure time from the moment th timer stopped
- `Reset`: This resets the timer to zero.

It also provides the `ElapsedTime` property, which can be used to report, as well as synchronization.

Getting ready

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

How to do it...

Suppose we wish to measure the time taken to perform a complex task, such as a call to a function that validates the data in a Web table. The syntax to instantiate a `Timer` object with the `MercuryTimers` collection is as follows:

```
Set timer_var = MercuryTimers.Timer("timer_name")
```

However, it is possible to instantiate the `Timer` object by simply invoking its `Start` method:

```
MercuryTimers.Timer("timer_name").Start
```

All methods listed previously are used with this syntax.

In the following example, we have added time measurements to the previously discussed procedure to close a process by its ID:

```
MercuryTimers.Timer("Notepad").Start

SystemUtil.Run "notepad.exe"
pID_Notepad = Window("regexpwndtitle:=Notepad").GetROProperty("process id

Print MercuryTimers.Timer("Notepad").ElapsedTime

Print pID_Notepad
Print SystemUtil.CloseProcessByid(pID_Notepad)
Print MercuryTimers.Timer("Notepad").ElapsedTime

MercuryTimers.Timer("Notepad").Stop
```

How it works...

As mentioned previously, the `MercuryTimers` object is actually a collection of zero or more `Timer` objects, which we can instantiate during

runtime. Each object named `Timer` contributes to the clarity of the code (as opposed to variables using the native VBScript `Timer` function) and work in parallel to the script (asynchronous mode). This means, once we instantiate a `Timer` object and invoke its `Start` method, it works in the background, and the script can continue to run. Using the `ElapsedTime` property, we can check or report the state of affairs. The `Continue`, `Reset`, and `Stop` methods are self-evident and do not require further explanation with regard to their function. However, it is important to note when we might use them. Suppose that we wish to isolate the net time of a function A that calls another auxiliary, function B. We might then wish to start a timer in function A, and stop it just before calling function B (which would have its own timer), then resume the timer in function A after returning from function B.

PREV
Managing proce…

NEXT
Resolving file loc…