

Yaron Assa

Automation Expert and QTP Programmer

Event interrupt in QTP

yassa · Friday, March 18th, 2011

Motivation

One of the more frustrating facts about the script-engine implementation in QTP is that you cannot bind it to external object's events. This means that you cannot get a "notification" from an external object or program about a change (e.g. an error has occurred, a logon session has expired, or a process has completed).

Native VBScript has the ability to register to an object's events, allowing you to interrupt the normal script flow with a response to a specific event. This can be done via the `ConnectObject` command of the `WScript` global object (which is **not** available from within QTP). For example, with `ConnectObject` support, you can have your script react to Word documents being opened and closed by the user:

```
Set objWord = WScript.CreateObject("Word.Application")
WScript.ConnectObject objWord, "objWord_"

objWord.visible = True
blnWordVisible = True

Do While blnWordVisible
    WScript.Sleep 500
Loop

Sub objWord_Quit
    blnWordVisible = False
    WScript.Echo "You quit Word."
End Sub

Sub objWord_DocumentChange
    WScript.Echo "You switched documents."
End Sub
```

This ability can be extremely valuable in testing services, databases and GUI applications, but sadly QTP offers no parallel ability, or even anything that comes close to it. You can rig something up using Recovery Scenarios, but it's cumbersome, slow and extremely limited.

Workaround

Luckily, there is a relatively simple workaround which allows you to interrupt QTP script flow from an external .Net object.

The workaround consists of two parts:

1. An interrupt “Hook” in QTP, which allows the external object to “Jump” the QTP code execution to a certain function or procedure.
2. An object that listens to events or initiates them, and calls the hook in response. For simplicity, I’ll use a .Net object, but it can be any object you can instantiate from within QTP (with, for example, a CreateObject command).

The Interrupt Hook

The interrupt hook is basically just a pointer to a subprocedure (a sub or a function). You can generate such a pointer using a simple GetRef command:

```
Public Sub DemoHook(sParam1)
    MsgBox sParam1 'Demonstrate interrupt functionality
End Sub

Dim oHook
Set oHook = GetRef("DemoHook")
```

In the example above, the oHook variable contains a pointer to a subprocedure that will serve as our interrupt hook. Naturally you can make the hook as complicated as you’d like, as long as you remember not to take for granted any resource outside its scope. It can be called at any time, even before the resource you were counting on was initialized and loaded.

Now if we can send oHook to an external object that can bind to events, it will “ring the bell” and activate oHook when needed.

The Event listener object

As I’ve said, the event listener can be any type of object, in practically any type of programming language. It can even be written in (real) VBScript, as it has an event-hooking capability.

In this case I’ve written a simple object in C#, and to keep things really simple, the event it listens to is a timer tick. Meaning that every 30 seconds, this timer will tick, the C# object will catch that event, and will initiate the QTP interrupt hook.

In order for our object to call the QTP hook, we need to pass that hook to our C# code. I’ve build an Init method for the object, which takes hook and stores it.

The C# code is as follows:

```
using System;
```

```

namespace EventHook_Demo
{
    public class EventHook
    {
        /// <summary>
        /// Our timer
        /// </summary>
        private System.Timers.Timer _Timer = new System.Timers.Timer
(5000);

        /// <summary>
        /// The hook back to QTP
        /// </summary>
        private object _QTPHook = null;

        public EventHook()
        {
            //Basic configuration
            _Timer.AutoReset = true;
            _Timer.Elapsed += React_To_Tick;
        }

        /// <summary>
        /// Fire QTP hook every time the timer ticks.
        /// </summary>
        public void React_To_Tick(object sender, System.Timers.
ElapsedEventArgs e)
        {
            //Call the QTP hook

            try
            {

//Send over the elapsed time since we've started listening
                _QTPHook.GetType().InvokeMember("",
, System.Reflection.BindingFlags.InvokeMethod,
                null, _QTPHook, new object
[] {e.SignalTime.ToShortTimeString() });
            }
            catch
            {
                //In case QTP has stopped listening to us
                _Timer.Stop();
            }
        }

        /// <summary>
        /// Start the timer and listening to its events.
    }
}

```

```

    /// </summary>
    public void StartListening()
    {
        _Timer.Start();
    }

    /// <summary>
    /// Stop the timer
    /// </summary>
    public void StopListening()
    {
        _Timer.Stop();
    }

    /// <summary>
    /// Sends over the QTP hook
    /// </summary>
    public void Init(object QTPHook)
    {
        _QTPHook = QTPHook;
    }
}

```

The important command is the InvokeMember method - it makes the QTP hook activate, and send over a parameter. Not only can you send information from the object to QTP (as we've sent the SignalTime parameter), but you can also return information from the QTP hook to the object. In our example it will be stored in the ReturnValue variable, and you could cast it to its real type (i.e. string, int, or anything else).

Combining the parts

Now all that's left is to build a QTP test that creates the event-hook object, initializes it with a QTP hook, and runs for enough time for the object to interrupt the test flow:

```

Option Explicit

Public Function DemoHook(sParam1)
    MsgBox sParam1 'Demonstate interrupt funcnality
    DemoHook = "Answer"
End Function

Dim oHook
Dim oEventObject
Set oHook = GetRef("DemoHook")

Set oEventObject = DotNetFactory.CreateInstance(

```

```
"EventHook_Demo.EventHook", "C:\EventHook Demo.dll")

oEventObject.Init(oHook)
oEventObject.StartListening()

'Regular test flow comes here
'We'll simulate it with a loop for 2 minutes

Dim iTimer
iTimer = Timer
Do
    Wait 1
Loop Until (Timer-iTimer > 120) 'Loop for 120 seconds
```

When you run the above test (and place the relevant C# dll in c:\, of course), the test flow will enter the loop, and the “jump” to the DemoHook function every 30 seconds.

Just as we’ve hooked to the timer tick event, we could hook to almost any event of any object, allowing us to create responsive and flexible automation code. This may require us to code more intensive information exchange between QTP and our event-hook object, but when applied in the right circumstances, it’s worth it.

If you wish, you can download the C# dll and try it out for yourself:

Event Interrupt Example (ver 1)

An example for QTP event interrupt article

A small remark: since we’ve used the persistent DotNetFactory object to create our C# class, stopping the test will not actually stop the C# object operation. It will remain active until you complete close QTP. Using CreateObject or other instantiation methods will not cause this problem.

Final Thoughts

Using external objects, we can enhance QTP, and compensate for its lack in programmatic capabilities. This approach offers interesting possibilities and I shall explorer them in future posts.

This entry was posted on Friday, March 18th, 2011 at 3:21 pm and is filed under [QTP Techniques](#)

You can follow any responses to this entry through the [Comments \(RSS\)](#) feed. You can leave a response, or [trackback](#) from your own site.