Chris Greaves
60 Canon Bayley Road; PO BOX 1452; Bonavista NL; CANADA A0C 1B0
709-218-7927

T:\Greaves\Products\DEVEL\OnTim\WhatFAQEnhances.doc

Contents

This note addresses some (and only some) of the issues involved in creating applications with independent scheduling events, to be controlled by a single OnTime statement in a template which is acting as a scheduler.

The Background

Since 1967 I have held that Computers are good at doing boring and repetitive tasks. Since around 1994 I have held that if I want to do it, I should be able to program a computer to do it. Since 2014 I have recognized than Alan M. Turing said it better with his description of a Turing Machine.

I have multiple applications that each depend on the Application.OnTime method, yet Microsoft Word (2003) allows only one OnTime event at a time. This poses a problem “How to overcome the limitation of only on OnTime event?

The solution is to wrest the OnTime event out of the hands of the feckless Microsoft Word and write a better OnTime processor.

My solution is to police all OnTime events through a single procedure which maintains control over a text file. The text file grows as claimant application append their timing requests to the file, and the policing procedure takes its direction from the file.

In essence “Take a Number”, or “Your call is important to us, please stay on the line”.

For these essays I decide to start with the policing procedure “OnTimerProcess” and three stripped-down slave macros “ClockMacro”, “SaverMacro” and “PlayerMacro”.

Build and Testing Three Slave Macros

Delete (and hence rebuild) Normal.dot.

Clear the Startup and Templates folders.

In the new Normal.dot a single module “modTimer” with code for the timer and three slaves: Clock, Saver and Player. All three macros work in that they speak, save, and play.

Then test that each macro appends a recall statement to the data file. The data file will be cluttered during this testing process.

The Three Slave Macros

```
Sub ClockMacro()
    Dim strDateTime As String ' Execute our primary task
    strDateTime = Format(Now(), "hh:mm:ss")
    Call SpeakString(strDateTime)
    Dim dtRandom As Date ' Queue ourselves
    dtRandom = Date + TimeSerial(Hour(Now()), lngcClockIntervalMinutes +
(lngcClockIntervalMinutes * Int(Minute(Now()) /
lngcClockIntervalMinutes)), 0)
    Call AppendFileData(strGetDataFileName, "Normal.modTimer.ClockMacro"
& strcDelimiter & dtRandom)
End Sub

Sub SaverMacro()
    Documents.Save (True) ' Execute our primary task
    Dim dtRandom As Date ' Queue ourselves
    dtRandom = Date + TimeSerial(Hour(Now()), lngcClockIntervalMinutes +
(lngcClockIntervalMinutes * Int(Minute(Now()) /
lngcClockIntervalMinutes)), 0)
    Call AppendFileData(strGetDataFileName, "Normal.modTimer.SaverMacro"
& strcDelimiter & dtRandom)
End Sub
```

```
Sub PlayerMacro()
    Dim strShell As String ' Execute our primary task
    strShell = """" & strcMediaPlayer & """ """ &
Application.StartupPath & "\" & strcAudioTrack & """"
    Shell strShell
    Dim dtRandom As Date ' Queue ourselves
    dtRandom = Date + TimeSerial(Hour(Now()), lngcClockIntervalMinutes +
(lngcClockIntervalMinutes * Int(Minute(Now()) /
lngcClockIntervalMinutes)), 0)
    Call AppendFileData(strGetDataFileName,
"Normal.modTimer.PlayerMacro" & strcDelimiter & dtRandom)
End Sub
```

Benchmark saved as Normal001.DOT.

Simple OnTime Host for Slaves and Timer

```
Sub OnTimerProcess()
'
'   Collect the tasks from the list
'
    Dim strMacroDateTime As String
    strMacroDateTime = strGetFileData(strGetDataFileName)
    Call blnKillFile(strGetDataFileName) ' Voiding the file allows salve
macros to continue their appenadages.
    Dim strAr() As String
    strAr = Split(strMacroDateTime, vbCrLf)
    Call QSort(strAr, LBound(strAr), UBound(strAr), False)
'
'   Locate and run all tasks up to NOW()
'
    Dim dtNextEvent As Date
    dtNextEvent = dtProcessEventsList(strAr)
'
'   Recall ourselves at the next pending event
'
    If dtNextEvent < Now() Then ' we are already late for the next call!
        dtNextEvent = Now() + TimeSerial(0, 0, 5)
    Else ' we can afford to wait until the future dtEvent
    End If
    Application.OnTime when:=dtNextEvent, Name:=strcTimerMacroName,
tolerance:=lngcTolerance

End Sub
```

Test the OnTimer macro, running just one slave macro. Check that the slave macro runs every one second (set by its Constant)

Once each of the three slaves has been proved in isolation, flush the time text file and run all three slave macros once. Check that they repeat just once every minute.

Close all open documents and check that all three slave macros continue to run at one-minute intervals when no documents are active.

Open a document and check that all three slave macros continue to run at one-minute intervals when a document is active

Leave the saver at 1, set the player at 2, set the clock at 3-minute intervals.

Benchmark saved as Normal002.DOT. This is an important milestone because it proves that the concept of a single operating system can be built around the scarce and limiting resources of a single OnTime event in Microsoft Word.[1]

[1] It has been my contention for twenty years that I should be able to do anything I want to do on a digital computer. Alan M. Turing said it with greater elegance.

We Begin our March to Independence

Our objective here is to isolate the timer in one document ("Timer.doc") and run slaves from different documents ("Saver.doc", "Clock.doc", "Player.doc"). Justification arises from the hundreds of hours invested in applications such as MRUse and Player, and other applications, all of which use the OnTime method but until now have not been able to function in parallel.

Move to Separate Modules Within Normal.Dot

We begin the trek with all four macros in a single module in the Normal.DOT template.

In theory three lines such as "`AppendFileData(strGetDataFileName, "Normal.modTimer.ClockMacro" & strcDelimiter & dtRandom`" need to have the module name changed to that of the specific slave procedure and that slave procedure be moved to its module.

We copy the Clock macro from the module modTimer to the module modClock. We should continue to hear the clock speak out at 1-minute intervals.

We make the change, the clock chimes; we disable the Clock macro in modTimer and make the changes one by one to the remaining two slave macros.

With all three slave macros within their own modules we have begun our slaves march away from the cozy nest in Normal.dot modTimer.

Benchmark saved as Normal003.DOT.

Timer Operating on Schedule

We implement a simple scheme for processing the time text file.

With Clock=1, Player=3 and Saver=7 minute intervals, the Timer happily raises each task at is approximately allotted time.

```
Normal.modClock.ClockMacro  02/09/2020 11:30:00 AM
Normal.modPlayer.PlayerMacro    02/09/2020 11:30:00 AM
Normal.modSaver.SaverMacro  02/09/2020 11:33:00 AM
```

Benchmark saved as Normal004.DOT.

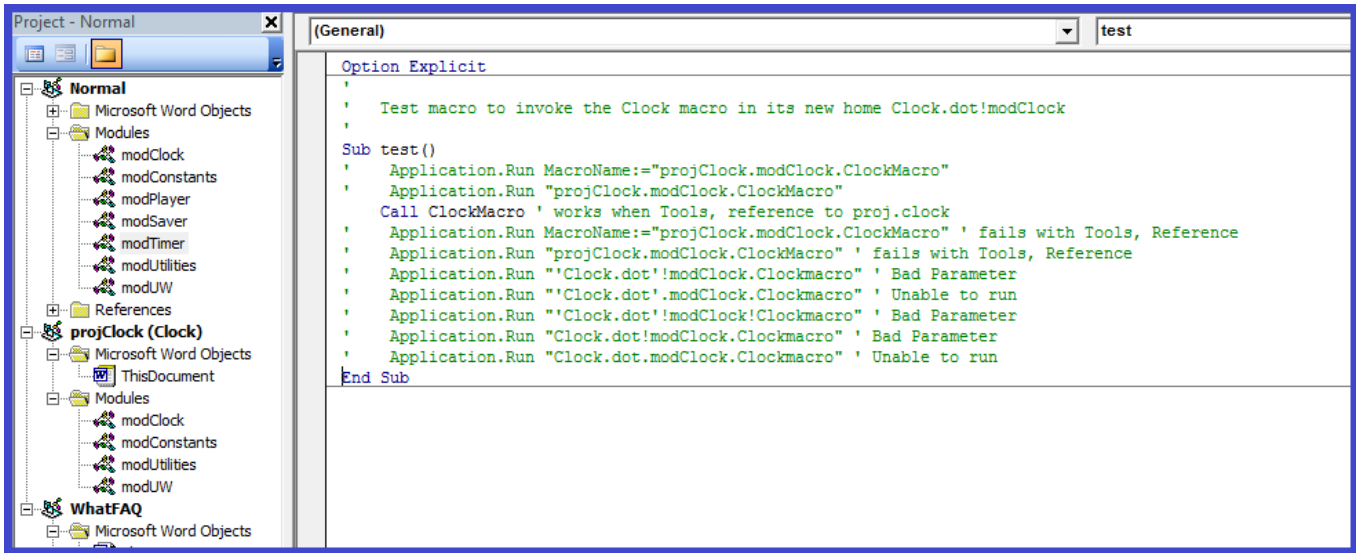
Move to Separate Files Within the Templates Folder

We are working within the templates folder because we chose to begin our trek from Normal.dot. Our reasoning is that Normal.dot might provide the most likely place for initial success, especially with code examples from the Word2003 help files.

An alternate starting location is from the StartUp folder, our preferred location because we think of startup applications as the core of our designs on and with Word2003. For example, Playr049 and MRUse593 are stalwarts of our Startup folder.

We create a new Clock.dot in our templates folder; we know that Word knows of this folder because it has stored Normal.dot in the folder.

We drag copies of our Clock module (and our constants module) from Normal.dot to Clock.dot, and we disable the code in Normal.modClock.



Results of my first trial with the clock module in a separate document.

Application.Run MacroName:="projClock.modClock.ClockMacro"	
Application.Run "projClock.modClock.ClockMacro"	
Call ClockMacro	' works when Tools, reference to proj.clock
Application.Run MacroName:="projClock.modClock.ClockMacro"	' fails with Tools, Reference
Application.Run "projClock.modClock.ClockMacro"	' fails with Tools, Reference
Application.Run "'Clock.dot'!modClock.Clockmacro"	' Bad Parameter
Application.Run "'Clock.dot'.modClock.Clockmacro"	' Unable to run
Application.Run "'Clock.dot'!modClock!Clockmacro"	' Bad Parameter
Application.Run "Clock.dot!modClock.Clockmacro"	' Bad Parameter
Application.Run "Clock.dot.modClock.Clockmacro"	' Unable to run

I was all ready to document my successes and failures when the first two lines in the above table, lines that had failed to work initially suddenly worked! Was it the implementation of the Tools references? I resolved to return and try again with more stringent control, and documentation.

Test01

```
Sub Test01()
On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
Application.Run "'Clock.dot'.modClock.Clockmacro"
Application.Run "'Clock.dot'!modClock!Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
Application.Run "Clock.dot.modClock.Clockmacro"
End Sub
```

My first test is with NINE Application.Run statements. All statements fail because (1) the Clock macro was previously and subsequently shown to operate (and announce the time) and (2) the On Error statement causes continued execution. Doubt this? Disable the On Error and run it manually.

Test02

```
Sub Test02()
'    On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
Application.Run "'Clock.dot'.modClock.Clockmacro" ' Unable
Application.Run "'Clock.dot'!modClock!Clockmacro" ' Unable
Application.Run "Clock.dot!modClock.Clockmacro"
Application.Run "Clock.dot.modClock.Clockmacro" ' Unable
End Sub
```

My second tests disable the On Error and Tools references the Clock.dot. Three of the nine statements fail. These three are my experimental syntax of the Help files examples.

Test03

```
Sub Test03()
'    On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
'    Application.Run "'Clock.dot'.modClock.Clockmacro"
'    Application.Run "'Clock.dot'!modClock!Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
'    Application.Run "Clock.dot.modClock.Clockmacro"
End Sub
```

My third test is a repeat of the second, but this time I disable bad statements. Note that (1) On Error remains disabled (2) the macro name has been changed to TEST03 and (3) all six statements appear to be valid with a Tools reference to Clock.dot

Test04

```
Sub Test04()
On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
'    Application.Run "'Clock.dot'.modClock.Clockmacro"
'    Application.Run "'Clock.dot'!modClock!Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
'    Application.Run "Clock.dot.modClock.Clockmacro"
End Sub
```

For the fourth test I remove the Tools reference to Clock.dot and enable the On Error. The macro TEST04 runs to completion with nary a peep. Not one of the six statements executed successfully.

Test05

```
Sub Test05()
'    On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
'    Application.Run "'Clock.dot'.modClock.Clockmacro"
'    Application.Run "'Clock.dot'!modClock!Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
'    Application.Run "Clock.dot.modClock.Clockmacro"
End Sub
```

For my fifth test the On Error statement is disabled and I have restored the Tools reference to Clock.dot. All six statements were successful.

Test06

```
Sub Test06()
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
End Sub
```

For my sixth test I exit WinWord.exe, reload it, eliminate known problem statements and run all six statements successfully. I have retained the Tools, references to Clock.dot

Test07

```
Sub Test07()
On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
End Sub
```

The seventh test had me disable the Tools references; not a peep.

I suppose that:-

(1) These six statements might be generally useful

(2) There may be other statements forms that could be used, but I have not yet come across them.

(3) A reference to the application template (Clock.dot) from Normal.dot appears to be essential. This does not appeal to me. I do not want to register every application willy-nilly

(4) I may obtain better results by using the Startup Folder for application templates.

Benchmark saved as Normal005.DOT.

Move to Separate Files Within the Startup Folder

TEST08 sees me move the application Clock.dot from the Templates folder to the Startup folder. I open the Clock.dot and run Clock macro to satisfy that it produces an audible signal then close the Clock.dot template.

Test08

```
Sub Test08()
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
End Sub
```

The eight test runs all six candidate statements successfully.

Test09

```
Sub Test09()
On Error Resume Next
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run MacroName:="projClock.modClock.ClockMacro"
Application.Run "projClock.modClock.ClockMacro"
Application.Run "'Clock.dot'!modClock.Clockmacro"
Application.Run "Clock.dot!modClock.Clockmacro"
End Sub
```

TEST09 sees me quit Word, move Clock.dot out of the Startup folder. All six statements failed, evidenced by On Error Resume Next and not a sound from the Clock macro.

Test10

TEST10 is a simple confirmation test: I quit Word, move Clock.dot back to the startup folder and hear six statements speak the time.

I suppose that:-

(5) An application in the Startup folder can have its Application.Run from Normal.dot.

How about running OnTime?

Test11

```
Sub Test11()
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="projClock.modClock.ClockMacro"
''''    Application.OnTime (when:=Now(),
Name:="projClock.modClock.ClockMacro")
''''    Application.OnTime (Now(), "projClock.modClock.ClockMacro")
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="modClock.ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5), Name:="ClockMacro"
End Sub
```

For TEST11, the Clock.dot is in the startup folder, but the application is not opened, nor is it Tools, Referenced from Normal.dot.

The statements highlighted in red indicate syntax errors. These two statements are parenthesized versions of the first statement.

I suppose that:-

(6) Despite the syntax shown at the top of the help files (*expression*.**OnTime(*When*, *Name*, *Tolerance*)**, parentheses cannot be used to surround the parameters of the OnTime command

The three statements that execute use successfully broader specifications of the macro. The third statement might well pick up a ClockMacro in some other application in the Startup folder.

Test12

```
Sub Test12()
'    Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="projClock.modClock.ClockMacro"
'    Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="modClock.ClockMacro"
'    Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="ClockMacro"
Application.OnTime Name:="projClock.modClock.ClockMacro", when:=Now() +
TimeSerial(0, 0, 5)
End Sub
```

For TEST12 I introduce the second of my six statements. Note that the first three statements are disabled because “Word can maintain only one background timer set by **OnTime**. If you start another timer before an existing timer runs, the existing timer is canceled.”. I could leave them enabled and allow Word’s OnTime to trample over itself, but I prefer to keep things squeaky-clean for now.

I note with increasing despondency that although Application has only two methods of running a user macro (Run and OnTime), it manages to invent two descriptions for the only shared parameter (“Name” and “MacroName”). So far the position of named parameters is irrelevant.

Test13

```
Sub Test13()
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="projClock.modClock.ClockMacro"
''''    Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
"projClock.modClock.ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="projClock.modClock.ClockMacro"
''''    Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
"projClock.modClock.ClockMacro"
''''    Application.OnTime  when:=Now() + TimeSerial(0, 0, 5),
"'Clock.dot'!modClock.Clockmacro"
''''    Application.OnTime  when:=Now() + TimeSerial(0, 0, 5),
"Clock.dot!modClock.Clockmacro"
End Sub
```

For TEST13 I bring in the six statements from TEST08, these being the six candidates that passed the tests. I modify these six for the OnTime method. Four of the statements immediately disqualify themselves with Syntax errors.

I suppose that:-

(7) The “Name” sentinel is a mandatory sentinel for the OnTime method, but not for the Run method. I hope that you are paying attention to all these rules.

Test14

```
Sub Test14()
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="projClock.modClock.ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="modClock.ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5), Name:="ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="projClock.modClock.ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5),
Name:="modClock.ClockMacro"
Application.OnTime when:=Now() + TimeSerial(0, 0, 5), Name:="ClockMacro"
End Sub
```

Test14 provides six useable variants on two basic candidates for the OnTime event

Test15

TEST15 is a repeat of TEST14 or any successful test.

I want to check that a startup application can CALL a function in the OnTimer module in Normal.dot

I delete the OnTimer text file from the startup folder and re-run TEST14, which has six OnTime calls.

```
Normal.modClock.ClockMacro  02/09/2020 4:27:00 PM
```

The clock speaks the time just once which is what we expect from six rapid-fire OnTime statements, and the OnTimer.txt text file has but on entry, indicating that the Clock macro was entered only once.

(The possibility of multiple events in the communications file is dealt with elsewhere)

At this point I know that:-

(1) I can RUN a Startup application macro from NORMAL.DOT

(2) I can run a Startup application macro ONTIME from NORMAL.DOT

(3) I can run a Normal Template function from an application template in the Startup folder.

At this point we have made progress.

We believe that we can effect a simple OnTime operating system under the constraints:-

(1) The operating system is housed in Normal.dot

(2) The slave applications are in the Startup folder

We can see our next two objectives:-

(3) The operating system is housed in some template other than Normal.dot

(4) The slave applications are in some folder other than the Startup folder.

Benchmark saved as Normal006.DOT.

The Story so Far

We left our gallant heroes – Normal.dot in Templates and Clock.dot in Startup – chatting with each other across a gulf. Normal.dot had been archived as Normal006.dot. The question now is: Can Normal(006).dot operate successfully as an operating system if it is moved from Templates to Startup? If so, then the timer and slaves can be shown to operate from within Startup. We will have come a long way from the four procedures huddled in a cramped module in Normal.dot (Simple OnTime Host for Slaves and Timer)

As we have trekked, some of our companions have dropped by the wayside; we might explore these failed members to see if they succeed in a different environment. OnTime and Run do seem inconsistent at times.

Appendix: Reply to Hans

[quote]I get the same error, [/quote]Hans, thanks for this confirmation; at least I know I am not yet a Senile etc.

[quote]…but why do you want to use two different documents for this? [/quote]I don't, actually. I want to use dozens, ten at least.

"Timer" is a stripped down operating system which polices tasks to be run at different times.

"Player is just one such slave task. I have others :grin: such as "Saver", "Clock", "Record" and so on. Including Word documents to launch Excel workbooks, DOS batch files, System32 commands and so on.

"Clock" wants to be woken up as regular as clockwork on every five (or ten, or …) minute interval so that it can announce the time.

"Saver" will save all open documents every one, two, three or four minutes. (A facsimile of my MRUse application_

"Player", ("BigPlayer" when it is all dressed up), grabs the Duration of an MP3 file from the metadata, issues the file to WinAmp, and then requests Timer that it (Player) be woken up after lngDuration seconds, at which event Player grabs a different track, duration, plays it and goes to sleep again. Player's sleep times are based on a file's duration, and appear as random lengths to Timer.

"Record" will wake up and record the temperature (or survey what you are doing right now or …)

Unlike MSWord (with one OnTime event) or Excel (ten events), Timer can accommodate as many scheduled tasks as you like **and** remembers them across reboots of Word or Windows **and** can be implemented as a simple module in any existing Word (or Excel or PPT or Outlook or …) application.

[quote]BTW, in PlayerMacro, you should open the text file for Output, not for Input, since you want to write a file to it. [/quote]Thanks. I *am* senile after all! I have changed it to Open Output,

(later) Mystery Solved, sort of.

I have previously noticed that (Word2003) **Application.OnTime** works to completion only if the target document is active. For example, my big baby, the proper player, wakes up and does WinAmp, but only if that player document is Active. If I Alt-Tab to my memoirs and start documenting my day, Memoirs.doc is active and Player.doc is not active, and the OnTime code does not find the Player macro, shrugs its shoulders, and slips off to the pub, without telling anybody.

The Application.Run appears to exhibit the same behaviour.

I set a breakpoint on each of the “intFile = FreeFile” statements (one in Timer, one in Player) and then re-ran the Player macro from VBE, first making sure that the Timer document was active.

When I arrived at the breakpoint in Timer, I made the Player.doc active before F5-ing Timer.

Ten seconds later I was at the breakpoint in Player; made Timer.doc active and F5, worked like a charm, as long as I remembered to make each target document active with an Alt-Tab. (so much for automation).

I had previously circumvented this activation problem with BigPlayer by dropping BigPlayer.doc into the startup folder.

Dropping Timer and Player into the startup folder did not resolve today's local problem. I must manually switch between active documents for the scheme to work, even if the two documents are in the startup folder.

My help files suggest *Application.Run "***My Document.doc'!***ThisModule.ThisProcedure"*. I cannot understand the weird syntax; the single quotes obviate the need for a change of delimiter from period to exclamation.

The help files also say "If you specify the document name, your code can only run macros in documents related to the current context— not just any macro in any document.", which might be Microsoft's way of saying "This won't work if the target document is not active".

If so that suggests to me that back in pre-1997 the programmer assumed that the target macro would always be in the ActiveDocument. That there would be only one OnTime event, and that the target macros had better be in the OnTime project.

Which leaves the puzzle of why we would bother to specify the document name (in any form)

I tried four variations on **Application.Run "'T:\Greaves\Startup\Word\Timer.doc'!modTimer.Timer Macro"**

(1) & (2) With and without the document path

(3) & (4) Using the recommended exclamation mark and the wistful period.

All four combinations failed.

This is something to sleep on.

If I load the FullName document name into the text file I can automate the Document.Activate process to let the macro run, but that will cause me to type some of my memoirs into whichever application is woken up at that time.

(later still) "`Application.Run MacroName:="Normal.Module2.Macro1"` suggests that it might work in Normal. I move modTimer to my Normal.dot and close my Timer.dot and change the description in my Application.Run.

(still later still) I have embarked on a trek, starting with all four micros in one module in Normal.dot, and gradually weaning them further and further away to see how far I can go while maintaining an OpSys with three slave applications that wake up on time.

Sulks

Chris

Appendix: On the Design of Microsoft Office

That Microsoft Office was designed badly is not open for argument.

At the top design level separate teams were established for each application (Word, Excel, PowerPoint etc) and at the second level applications included patched-in third party code (for example the saveasHTML code in Word97); at the third level each contributing third-party had its own standards. So we see so-called un-trappable errors in MSWord, and varying limits (ten OnTime events in Excel but only one in Word). There is no limit on the number of OnTime events in PowerPoint because PowerPoint(2003) does not provide an OnTime method.

We are not surprised to see two statements near-identical to the user named “Run” and “OnTime” (both cause a user-macro to be run) rather than “Run” and “RunOnTime”).

We are not surprised to see two statements near-identical to the user, one where the macro is identified as “Name:=” and the other identified as “MacroName:=”.

Likewise we should not be surprised at the inconsistent syntax of "'Clock.dot'.modClock.Clockmacro".

Think “hundreds of programmers not talking with each other” and then you’ll have no problems working ~~with~~ around Microsoft Office products.

Index