Timer Comparison Tests

Over the years, I have used various functions to measure time intervals: **Timer, GetSystemTime, GetTickCount**.

Each of these can give times to millisecond precision though I normally round to 2 d.p. (centiseconds).
This is because each function is based on the system clock which is normally updated 64 times per second – approximately every 0.0156 seconds

When I started my series of speed comparison tests, I initially used the GetSystemTime function.
However, some occasional inconsistencies led me to revert to the very simple Timer function.

Recently I was alerted to the **timeGetTime** API by AWF member **ADezii** with these comments taken from the Access 2000 Developers Handbook pg 1135-1136:
If you're interested in measuring elapsed times in your Access Application, you're much better off using the timeGetTime() API Function instead of the Timer() VBA Function. There are 4 major reasons for this decision:
***1.** timeGetTime() is more accurate. The Timer() Function measure time in 'seconds' since Midnight in a single-precision floating-point value, and is not terribly accurate. timeGetTime() returns the number of 'milliseconds' that have elapsed since Windows has started and is very accurate.*
***2.** timeGetTime() runs longer without 'rolling over'. Timer() rolls over every 24 hours. timeGetTime() keeps on ticking for up to 49 days before it resets the returned tick count to 0.*
***3.** Calling timeGetTime() is significantly faster than calling Timer().*
***4.** Calling timeGetTime() is no more complex than calling Timer(), once you've included the proper API declaration*

Part of this comment is no longer accurate in that the Timer function can measure to milliseconds.
However, as I had never used the timeGetTime API, I decided to compare the results obtained using each of the methods using two simple tests:

- Looping through a simple square root calculation repeatedly (20000000 times)
- Measuring the time interval after a specified time delay setup using the Sleep API (1.575 s)

I also added a Stopwatch class to the timer comparison tests (again thanks to **ADezii** for this code)

Obviously, as with any timer tests, other factors such as background windows processes, and overall CPU load will lead to some natural variation. To minimise the effects of those, I avoided running any other applications at the same time and ran each test 10 times. Furthermore, the test order was randomised each time ... just in case. The average times were calculated along with the minimum/maximum times and standard deviation for each test.

As the results are all based on the system clock, I expected the results to be similar in each case.
However, it seemed reasonable that certain functions would be more efficient to process

For these tests, the main requirement is certainly **not** to determine which gives the smallest time.
Here the aim is to achieve consistency so that repeated tests should provide a small **standard deviation**

1. Test Results

These are the average results for test A – calculation loop (old desktop PC with 32-bit Access & 4GB RAM):

Average Results

Workstation	Test	Test Type	Run Count	Loop Count	Min Time	Max Time	Std Dev	Average Time
COLIN-PC	A	GetSystemTime	10	20000000	1.773	2.086	0.098	1.808
COLIN-PC	A	GetTickCount	10	20000000	1.781	1.813	0.011	1.801
COLIN-PC	A	Stopwatch Class	10	20000000	1.741	1.767	0.009	1.753
COLIN-PC	A	TimeGetTime	10	20000000	1.773	1.789	0.008	1.784
COLIN-PC	A	Timer	10	20000000	1.852	1.859	0.004	1.855

Record: 1 of 5 No Filter Search

As expected, the results were similar, but the **GetSystemTime** function produced much greater variation
The **Timer** function had the smallest variation but also its average times were slightly larger than those obtained by other methods.
The **Stopwatch** class seemed to run fastest of all but with slightly larger variation than timeGetTime or Timer

For comparison, I repeated the tests on a laptop with 64-bit Access & 8GB RAM:

Average Results

Workstation	Test	Test Type	Run Count	Loop Count	Min Time	Max Time	Std Dev	Average Time
COLIN-LAPTOP	A	GetSystemTime	10	20000000	0.373	1.983	0.391	1.262
COLIN-LAPTOP	A	GetTickCount	10	20000000	1.359	1.500	0.047	1.420
COLIN-LAPTOP	A	Stopwatch Class	10	20000000	1.078	1.282	0.060	1.139
COLIN-LAPTOP	A	TimeGetTime	10	20000000	1.125	1.314	0.067	1.216
COLIN-LAPTOP	A	Timer	10	20000000	1.047	1.281	0.076	1.120

Record: 1 of 5 No Filter Search

As you would expect, each of the times are faster but once again **GetSystemTime** produced some variation.
GetTickCount had the smallest variation but the times were also consistently slower.
The other 3 methods were perhaps broadly comparable.

Finally, I used a Windows tablet with 2GB RAM. Clearly, its only just adequate for running Access and struggles with any complex processing. The times were inevitably a LOT slower but the pattern in the results were similar

Average Results

Workstation	Test	Test Type	Run Count	Loop Count	Min Time	Max Time	Std Dev	Average Time
COLIN-TABLET	A	GetSystemTime	10	20000000	5.271	5.847	0.181	5.722
COLIN-TABLET	A	GetTickCount	10	20000000	5.687	5.719	0.012	5.706
COLIN-TABLET	A	Stopwatch Class	10	20000000	5.685	5.746	0.018	5.719
COLIN-TABLET	A	TimeGetTime	10	20000000	5.693	5.723	0.008	5.706
COLIN-TABLET	A	Timer	10	20000000	5.680	5.727	0.016	5.705

Record: 1 of 5 No Filter Search

The second set of tests were done with a **specified time delay** of 1.575s. For these tests, we would expect each value to be slightly larger than the time delay to allow for processing the timer functions. There should also be less variation between the different PCs

These are the average results for test B – on the desktop PC with 4GB RAM:

Average Results

Workstation	Test	Test Type	Run Count	Delay Time	Min Time	Max Time	Std Dev	Average Time
COLIN-PC	B	GetSystemTime	10	1.575	1.456	2.448	0.285	1.647
COLIN-PC	B	GetTickCount	10	1.575	1.578	1.594	0.007	1.590
COLIN-PC	B	Stopwatch Class	10	1.575	1.581	1.600	0.007	1.590
COLIN-PC	B	TimeGetTime	10	1.575	1.581	1.601	0.007	1.592
COLIN-PC	B	Timer	10	1.575	1.578	1.609	0.008	1.591

Record: 1 of 5 No Filter Search

Once again **GetSystemTime** was MUCH less consistent with some large results.
Even more worryingly there were a couple of values significantly less than 1.575 seconds.
All the other functions were similar both in terms of variation and average times obtained

Here are the average results using the laptop with 8GB RAM:

Average Results

Workstation	Test	Test Type	Run Count	Delay Time	Min Time	Max Time	Std Dev	Average Time
COLIN-LAPTOP	B	GetSystemTime	10	1.575	1.578	1.594	0.007	1.590
COLIN-LAPTOP	B	GetTickCount	10	1.575	1.578	1.594	0.005	1.592
COLIN-LAPTOP	B	Stopwatch Class	10	1.575	1.578	1.596	0.008	1.586
COLIN-LAPTOP	B	TimeGetTime	10	1.575	1.593	1.596	0.001	1.594
COLIN-LAPTOP	B	Timer	10	1.575	1.594	1.594	0.000	1.594

Record: 1 of 5 No Filter Search

In this case, all tests seemed reliable but it was noticeable that **timeGetTime** had minimal variation.
Even better, the **Timer** function had 10 identical results – no variation whatsoever!

The results using the 2GB tablet were

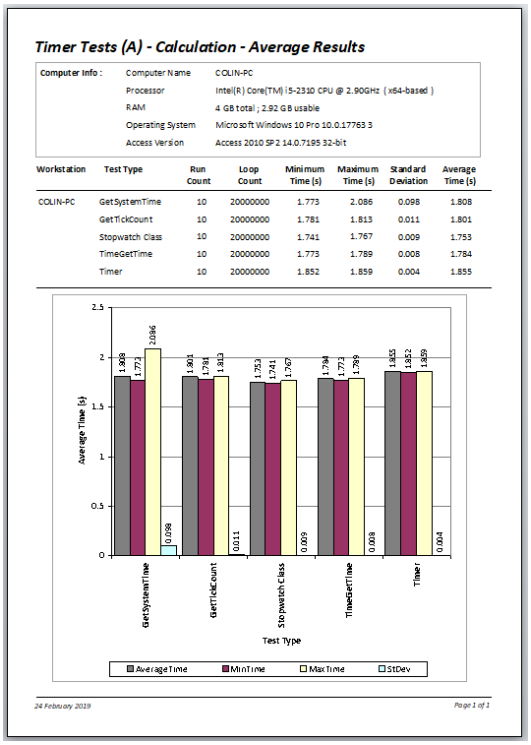
Average Results

Workstation	Test	Test Type	Run Count	Delay Time	Min Time	Max Time	Std Dev	Average Time
COLIN-TABLET	B	GetSystemTime	10	1.575	0.811	1.837	0.422	1.429
COLIN-TABLET	B	GetTickCount	10	1.575	1.563	1.579	0.006	1.575
COLIN-TABLET	B	Stopwatch Class	10	1.575	1.575	1.577	0.001	1.576
COLIN-TABLET	B	TimeGetTime	10	1.575	1.575	1.577	0.001	1.576
COLIN-TABLET	B	Timer	10	1.575	1.570	1.578	0.003	1.577

Record: 1 of 5 No Filter Search

Once again, **GetSystemTime** produced significant variation and a couple of results that were clearly incorrect. In this test **Stopwatch class, timeGetTime** and **Timer** were again the most consistent

The example application also includes reports with charts. These are for the desktop PC:

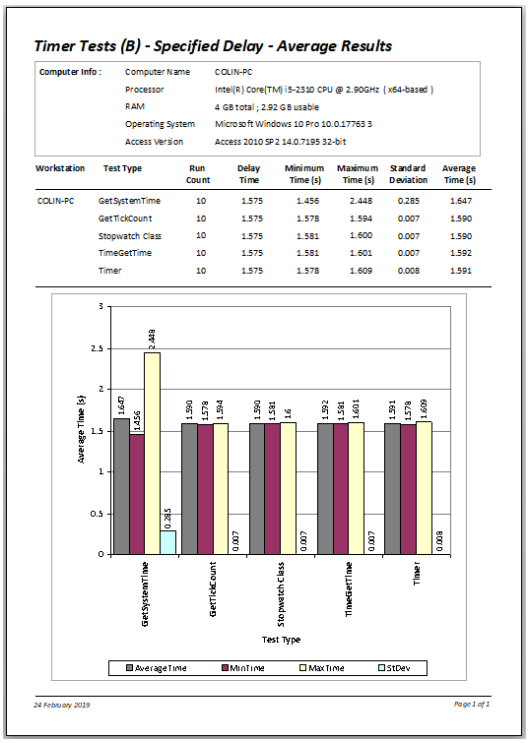


Timer Tests (A) - Calculation - Average Results

Computer Info :		
	Computer Name	COLIN-PC
	Processor	Intel(R) Core(TM) i5-2310 CPU @ 2.90GHz (x64-based)
	RAM	4 GB total ; 2.92 GB usable
	Operating System	Microsoft Windows 10 Pro 10.0.17763 3
	Access Version	Access 2010 SP2 14.0.7195 32-bit

Workstation	Test Type	Run Count	Loop Count	Minimum Time (s)	Maximum Time (s)	Standard Deviation	Average Time (s)
COLIN-PC	GetSystemTime	10	20000000	1.773	2.086	0.098	1.808
	GetTickCount	10	20000000	1.781	1.813	0.011	1.801
	Stopwatch Class	10	20000000	1.741	1.767	0.009	1.753
	TimeGetTime	10	20000000	1.773	1.789	0.008	1.784
	Timer	10	20000000	1.852	1.859	0.004	1.855

24 February 2019 Page 1 of 1



Timer Tests (B) - Specified Delay - Average Results

Computer Info :		
	Computer Name	COLIN-PC
	Processor	Intel(R) Core(TM) i5-2310 CPU @ 2.90GHz (x64-based)
	RAM	4 GB total ; 2.92 GB usable
	Operating System	Microsoft Windows 10 Pro 10.0.17763 3
	Access Version	Access 2010 SP2 14.0.7195 32-bit

Workstation	Test Type	Run Count	Delay Time	Minimum Time (s)	Maximum Time (s)	Standard Deviation	Average Time (s)
COLIN-PC	GetSystemTime	10	1.575	1.456	2.448	0.285	1.647
	GetTickCount	10	1.575	1.578	1.594	0.007	1.590
	Stopwatch Class	10	1.575	1.581	1.600	0.007	1.590
	TimeGetTime	10	1.575	1.581	1.601	0.007	1.592
	Timer	10	1.575	1.578	1.609	0.008	1.591

24 February 2019 Page 1 of 1

2. *Conclusions*

Overall, I would suggest that both **Timer** and **TimeGetTime** are reliable. Each had minimal variation compared to the other methods
Bearing in mind that the **Timer** function is based on the time elapsed since midnight whereas **timeGetTime** runs for 49 days before resetting, **timeGetTime** should definitely be used if the timing tests are likely to cross midnight or last longer than 24 hours.

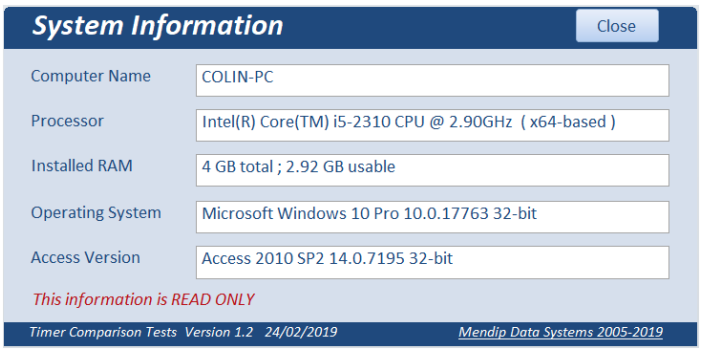
However, for smaller time intervals on a reasonably powerful PC, I don't think there is much advantage in one method compared to the other

Stopwatch class works well but requires additional code compared to the **Timer** or **TimeGetTime** methods
GetTickCount is satisfactory but perhaps not as reliable as other methods
GetSystemTime is unreliable and should not be used

NOTE: there are other methods that I haven't yet tested successfully including the **multimedia timer**.

3. *System Info*

The example app also includes a system information feature with data mostly obtained using WMI



4. Test Code

ID	Test	Timer Code	API etc
1	Timer	'start timer sngStart = Timer Select Case strTest Case "A" 'Calculation For Q = 1 To LC 'loop count dblSqr = Sqr(Q) Next Case "B" 'specified delay Sleep 1000 * TD 'time delay End Select 'stop timer sngEnd = Timer Me.txtTime1 = Round((sngEnd - sngStart), 3)	N/A
2	GetSystemTime	'start timer sngStart = GetCurrentSystemTime Select Case strTest Case "A" 'Calculation For Q = 1 To LC 'loop count dblSqr = Sqr(Q) Next Case "B" 'specified delay Sleep 1000 * TD 'time delay End Select 'stop timer sngEnd = GetCurrentSystemTime Me.txtTime2 = Round((sngEnd - sngStart), 3)	#If VBA7 Then Declare PtrSafe Sub GetSystemTime Lib "kernel32" (lpSystemTime As SYSTEMTIME) #Else Declare Sub GetSystemTime Lib "kernel32" (lpSystemTime As SYSTEMTIME) #End If
3	GetTickCount	'start timer - milliseconds StartTime = GetTickCount Select Case strTest Case "A" 'Calculation For Q = 1 To LC 'loop count dblSqr = Sqr(Q) Next Case "B" 'specified delay Sleep 1000 * TD 'time delay End Select stop timer -'milliseconds EndTime = GetTickCount Me.txtTime3 = (EndTime - StartTime) / 1000	#If VBA7 Then Declare PtrSafe Function GetTickCount Lib "kernel32" () As Long #Else Declare Function GetTickCount Lib "kernel32" () As Long #End If

ID	Test	Timer Code	API etc
4	TimeGetTime	'start timer -time in milliseconds StartTime = TimeGetTime() Select Case strTest Case "A" 'Calculation For Q = 1 To LC 'loop count dblSqr = Sqr(Q) Next Case "B" 'specified delay Sleep 1000 * TD 'time delay End Select 'stop timer - time in milliseconds EndTime = TimeGetTime() Me.txtTime4 = (EndTime - StartTime) / 1000	#If VBA7 Then Public Declare PtrSafe Function TimeGetTime Lib "winmm.dll" Alias "timeGetTime" () As Long #Else Public Declare Function TimeGetTime Lib "winmm.dll" Alias "timeGetTime" () As Long #End If
5	Stopwatch class	'Create a New Instance of the Class Dim stpw As New StopWatch 'start timer If Not stpw.IsRunning Then stpw.StartTimer Select Case strTest Case "A" 'Calculation For Q = 1 To LC 'loop count dblSqr = Sqr(Q) Next Case "B" 'specified delay Sleep 1000 * TD 'time delay End Select 'stop timer stpw.StopTimer Me.txtTime5 = stpw.GetSecondsElapsed	Code in Stopwatch class module

5. *Some Useful links*

timeGetTime
https://docs.microsoft.com/en-us/windows/desktop/api/timeapi/nf-timeapi-timegettime
https://bytes.com/topic/access/insights/618175-timegettime-vs-timer

StopWatchClass
https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?redirectedfrom=MSDN&view=netframework-4.7.2

General
https://stackoverflow.com/questions/18346879/timer-accuracy-c-clock-vs-winapis-qpc-or-timegettime

Quote from that link:
Timer(), GetTickCount and timeGetTime() *are derived from a calibrated hardware clock. Resolution is not great, they are driven by the clock tick interrupt which ticks by default 64 times per second or once every 15.625 msec. You can use* ***timeBeginPeriod()*** *to drive that down to 1.0 msec. Accuracy is very good, the clock is calibrated from a NTP server, you can usually count on it not being off more than a second over a month.*

QPC *has a much higher resolution, always better than one microsecond and as little as half a nanosecond on some machines. It however has poor accuracy, the clock source is a frequency picked up from the chipset somewhere. It is not calibrated and has typical electronic tolerances. Use it only to time short intervals.*

Latency is the most important factor when you deal with timing. You have no use for a highly accurate timing source if you can't read it fast enough. And that's always an issue when you run code in user mode on a protected mode operating system. Which always has code that runs with higher priority than your code. Particularly device drivers are trouble-makers, video and audio drivers in particular. Your code is also subjected to being swapped out of RAM, requiring a page-fault to get loaded back. On a heavily loaded machine, not being able to run your code for hundreds of milliseconds is not unusual. You'll need to factor this failure mode into your design. If you need guaranteed sub-millisecond accuracy then only a kernel thread with real-time priority can give you that.

A pretty decent timer is the ***multi-media timer*** *you get from* ***timeSetEvent()****. It was designed to provide good service for the kind of programs that require a reliable timer. You can make it tick at 1 msec, it will catch up with delays when possible. Do note that it is an asynchronous timer, the callback is made on a separate worker thread so you have to be careful taking care of proper threading synchronization.*