

Forum:[Hack Forums](#)

Thread: [Calculate Task Per Minute?](#)

-PUBG- - September 16, 2018, 3:25 pm

Hello, I am just trying to see what the best way of calculating checks per minute on a multi-tasked function would be

Thanks for any help :)

Razagas - September 17, 2018, 1:10 am

Without giving you a complete source and doing all the work for you a very simple way of doing this, without a resource consuming timer is as follows.

1. Create a DateTime object and TasksCompleted int when you first click whatever button starts your tasks.
2. With each run increase the TasksCompleted int, subtract DateTime.Now from your original DateTime.
3. Once the resulting TimeSpan object is 00:01:00, reset your original DateTime object to the current time and TasksCompleted after passing the information to wherever you want to keep track of it.
4. You now know how many task instances completed in a minute TimeSpan, without using a terrible CPU intensive timer.

0xDEAD10CC - September 17, 2018, 10:14 am

I would advise using the Stopwatch class instead. You can use a threading timer for as granular as you want to check the statistic.

Ex:

Code timer ms elapsed 30000ms = $30000/1000/60 = 0.5$ m elapsed

>> TASKS/m = TASKS / m elapsed

i.e. 100 tasks complete
= $100 / 0.5 = 200$ tasks/m

I don't know why you'd use DateTime when it can change, or why you'd reset it. Just use the Stopwatch class and calculate its running timer until it's complete, then stop it and re-calculate if necessary.

Razagas - September 17, 2018, 2:21 pm

(Sep 17, 2018 10:14 AM)0xDEAD10CC Wrote: [\[CITE\]](#) I would advise using the Stopwatch class instead. You can use a threading timer for as granular as you want to check the statistic.

Ex:

Code timer ms elapsed 30000ms = $30000/1000/60 = 0.5$ m elapsed

>> TASKS/m = TASKS / m elapsed

i.e. 100 tasks complete
= $100 / 0.5 = 200$ tasks/m

I don't know why you'd use DateTime when it can change, or why you'd reset it. Just use the Stopwatch class and calculate its running timer until it's complete, then stop it and re-calculate if

necessary.

If you look into how stopwatch operates, it uses `DateTime.Now()` as the `Start()` method and `DateTime.Now()` as the `Stop()` method. This is a viable solution but it is not a timer and you should not be explaining it as a timer. Timers are bad practice, consume excessive cpu for a simple calculation and lead to poor habits later on. The stopwatch class is a shortcut of the method I outlined above, and offers a better understanding of solving "complex" problems with skills that will be more valuable later on than just "use a stopwatch."

Furthermore to help you understand how `DateTime` works, The reason you'd use `DateTime` is because it does not change once you set a local variable to it. If I call `DateTime.Now()` to be a local variable "X", X will not change from the static `DateTime` object that I defined when I instanced it. You would then compare X to `DateTime.Now()` upon task completion to receive your `TimeSpan` object which is what `StopWatch` returns. The reason you need to reset your initial X, or reinitialize it, is to grab the new `DateTime.Now()` of when the task is ran again, to get a reference point.

Basically, for a new programmer trying to learn how Timespans work, and date/time objects, they shouldn't use stopwatch. It is a higher level implementation of comparing `DateTime` objects to produce a `TimeSpan` object. `StopWatch` is perfectly fine for an experienced coder, and I do advocate its use later on, but for now I believe he can benefit a lot more from learning about problem solving, the "`DateTime`" class, the "`TimeSpan`" class and later on how to refine it using higher level classes like `StopWatch`.

Edit: Worth taking note also is `DateTime.UtcNow()` would be the better option, assuming you don't care to report execution timestamps in a local time format, or don't mind reformatting to local time after. I can't post links yet, so I can't link to the performance chart showing the difference between `DateTime.UtcNow()` vs stopwatch but feel free to Google it. It's roughly 80% faster and less CPU intensive.

0xDEAD10CC - September 17, 2018, 3:07 pm

(Sep 17, 2018 02:21 PM)Razagas Wrote: [\[CITE\]](#) If you look into how stopwatch operates, it uses `DateTime.Now()` as the `Start()` method and `DateTime.Now()` as the `Stop()` method. This is a viable solution but it is not a timer and you should not be explaining it as a timer. Timers are bad practice, consume excessive cpu for a simple calculation and lead to poor habits later on. The stopwatch class is a shortcut of the method I outlined above, and offers a better understanding of solving "complex" problems with skills that will be more valuable later on than just "use a stopwatch."

Furthermore to help you understand how `DateTime` works, The reason you'd use `DateTime` is because it does not change once you set a local variable to it. If I call `DateTime.Now()` to be a local variable "X", X will not change from the static `DateTime` object that I defined when I instanced it. You would then compare X to `DateTime.Now()` upon task completion to receive your `TimeSpan` object which is what `StopWatch` returns. The reason you need to reset your initial X, or reinitialize it, is to grab the new `DateTime.Now()` of when the task is ran again, to get a reference point.

Basically, for a new programmer trying to learn how Timespans work, and date/time objects, they shouldn't use stopwatch. It is a higher level implementation of comparing `DateTime` objects to produce a `TimeSpan` object. `StopWatch` is perfectly fine for an experienced coder, and I do advocate its use later on, but for now I believe he can benefit a lot more from learning about problem solving, the "`DateTime`" class, the "`TimeSpan`" class and later on how to refine it using higher level classes like `StopWatch`.

Edit: Worth taking note also is `DateTime.UtcNow()` would be the better option, assuming you don't care to report execution timestamps in a local time format, or don't mind reformatting to local time

after. I can't post links yet, so I can't link to the performance chart showing the difference between `DateTime.UtcNow()` vs stopwatch but feel free to Google it. It's roughly 80% faster and less CPU intensive.

In most cases the high performance counter is available which means it uses [QueryPerformanceCounter](#), which depends on the success of `QueryPerformanceFrequency`...

Why would you re-invent the wheel and do the calculations for time intervals if the `Stopwatch` class does it for you? Your big paragraph doesn't change common sense... He wants to check calculate tasks per minute, this doesn't mean he wants to check every minute, which means you will need a timer to invoke the calculation callback depending on how frequently you need this calculation to be performed to get those statistics. How do you think the Windows file transfer estimation works - do you think they check every nanosecond wasting CPU cycles on calculating the time it takes to move the file? How granular do you really think humans are able to see in terms of how often that calculation is performed to give us the result?

Just the fact that you're saying timers are bad practice makes me think you're one of those members who has little clue on how programming in the real world works. What you fail to understand is that the solution here really depends on how he's doing the calculations. He can either invoke a callback once a task is complete if you wanted to avoid threading the calculation but one batch of tasks might take 1 minute and another might take 10 which means you're not going to get a consistent refresh of the statistics if you DON'T thread it. Excessive CPU? You must be doing some very poor threading in your own programs - go take your lack of programming experience somewhere else.

The `Stopwatch` class does not use `DateTime`, read MSDN:

Quote: The `Stopwatch` measures elapsed time by counting timer ticks in the underlying timer mechanism. If the installed hardware and operating system support a high-resolution performance counter, then the `Stopwatch` class uses that counter to measure elapsed time. Otherwise, the `Stopwatch` class uses the system timer to measure elapsed time. Use the `Frequency` and `IsHighResolution` fields to determine the precision and resolution of the `Stopwatch` timing implementation.

Therefore it's also more suited for time measurement, especially if it's using a high resolution timer.

>> The reason you'd use `DateTime` is because it does not change once you set a local variable to it

What do you think the `DateTime.Now` get property returns if you change the system time...? ? Again - you have absolutely zero clue. UTC or local time is irrelevant here, he's calculating number of tasks within a certain period of time; nobody needs to know whether it is 07:00 UTC or 15:00 MST... And you still need to retrieve the `DateTime` to calculate the delta after the variable has been stored; any difference in system time between these reference points is going to skew that calculation. And it isn't a function, there is no "`DateTime.Now()`". I've been programming much longer with .NET than you guaranteed.

Don't believe me? Read the very first SO link I dug up just to re-iterate what I was initially saying: <https://stackoverflow.com/questions/2923283/stopwatch-vs-using-system-datetime-now-for-timing-events>

What sense would it make for every worker thread executing a task to be calculating the tasks/minute statistic? That seems very conflicting especially if these tasks are executed on multiple threads already, which is why it makes plenty more sense to have a threading timer execute on a separate thread with periodic callbacks to update the statistics based on some completion task pool. If you don't understand this concept, then that is not my fault...

Quote:2. With each run increase the TasksCompleted int, subtract DateTime.Now from your original DateTime.

Do it this way, and as I already mentioned, if task #1 takes 5 seconds and you calculate a stat of 12 tasks/minute, task #2 goes on 5 mins and doesn't complete, then you have to wait those 5 minutes to see an accurate statistic update in the program reflect what the calculated tasks/minute result is -- ONLY after that 5 mins has completed. If he was to use your method he could see no change in results for X amount of time, then have it instantly jump to a completely different calculated result at an inconsistent rate without it being a smooth running time calculation. Depending on how fast the calculations take place, if he was rendering the results on a UI it may even create the inadvertent result of the calculation being entirely unreadable until all tasks have completed as well because unlike with a timer you simply cannot control how frequently the statistics are re-calculated and subsequently re-rendered on the UI.

Sorry - but you are entirely wrong on this one.

Razagas - September 17, 2018, 5:39 pm

1. UTC vs Local is very relevant, lookup the performance difference between DateTime.UtcNow vs DateTime.Now you'll learn a very startling truth.

2. You're correct that I simplified how stopwatch works. It takes a snapshot of QueryPerformanceCounter upon start and another upon stop and calculates the result, once again you can look this up. I simplified it down to a DateTime argument but the truth is they both return a TimeSpan. So you should look that up as well.

3. Timers are bad practice because they spawn a thread that is constantly counting ticks to run a loop, they can be used properly but the average programmer online is going to use for shitty reasons.

4. You shouldn't call someone's way of doing something wrong. That is how I know you are trying to come off as some superior programmer. Get over yourself bud. I get it you have a lot of internet points, but these are both valid solutions that help the OP.

5. You constantly tried to personally attack my skillset, yet you know nothing of my work. You are not helping the discussion in any way, shape, or form by arguing that I have no idea what I'm talking about. I have a bachelor's in CS with an emphasis in Software Engineering, I think I'm qualified enough to give OP advice.

6. UI blocking? The guy asked how to calculate the time difference between events, do you really want to go into asynchronous programming?

7. You say that there is no DateTime.now() and claim to be some super advanced 1337 programmer. Guess what buddy? Those are parathensis for DateTime formatting, Google it.

8. You constantly mention threading, it's 2018 buddy learn to async. You're entire 2 paragraphs on 'BUT WHAT IF ONE TAKES 5-10 MINUTES' becomes a moot point once if you used asynchronous methods, maybe one day you'll learn how to use those.

You need to ignore my account creation date, posts, reputation, etc. I've used the .net framework for over 10 years now, I'm not some kid learning. I am a full time software engineer, I have a degree in this field, I could give 2 shits about your shiny internet points. Now let the OP learn something instead of trying to bash people and speak nonsensical bullshit to make yourself seem "super

educated" your ability to copy/paste tutorials and code challenges to seem like some kind of "C# guru" does not impress me. I'm not 13. Learn some fucking respect.

ppaida - September 18, 2018, 6:30 am

If you are multi threading this, make sure your stop watch is using

```
Code Interlocked.Increment(ref counter).  
Ints are not threadsafe.
```

0xDEAD10CC - September 18, 2018, 12:19 pm

(Sep 17, 2018 05:39 PM)Razagas Wrote: [\[CITE\]](#) 1. UTC vs Local is very relevant, lookup the performance difference between `DateTime.UtcNow` vs `DateTime.Now` you'll learn a very startling truth.

2. You're correct that I simplified how stopwatch works. It takes a snapshot of `QueryPerformanceCounter` upon start and another upon stop and calculates the result, once again you can look this up. I simplified it down to a `DateTime` argument but the truth is they both return a `TimeSpan`. So you should look that up as well.

3. Timers are bad practice because they spawn a thread that is constantly counting ticks to run a loop, they can be used properly but the average programmer online is going to use for shitty reasons.

4. You shouldn't call someone's way of doing something wrong. That is how I know you are trying to come off as some superior programmer. Get over yourself bud. I get it you have a lot of internet points, but these are both valid solutions that help the OP.

5. You constantly tried to personally attack my skillset, yet you know nothing of my work. You are not helping the discussion in any way, shape, or form by arguing that I have no idea what I'm talking about. I have a bachelor's in CS with an emphasis in Software Engineering, I think I'm qualified enough to give OP advice.

6. UI blocking? The guy asked how to calculate the time difference between events, do you really want to go into asynchronous programming?

7. You say that there is no `DateTime.now()` and claim to be some super advanced 1337 programmer. Guess what buddy? Those are parathensis for `DateTime` formatting, Google it.

8. You constantly mention threading, it's 2018 buddy learn to async. You're entire 2 paragraphs on 'BUT WHAT IF ONE TAKES 5-10 MINUTES' becomes a moot point once if you used asynchronous methods, maybe one day you'll learn how to use those.

You need to ignore my account creation date, posts, reputation, etc. I've used the .net framework for over 10 years now, I'm not some kid learning. I am a full time software engineer, I have a degree in this field, I could give 2 shits about your shiny internet points. Now let the OP learn something instead of trying to bash people and speak nonsensical bullshit to make yourself seem "super educated" your ability to copy/paste tutorials and code challenges to seem like some kind of "C# guru" does not impress me. I'm not 13. Learn some fucking respect.

In the context of his problem, timezones are entirely irrelevant. I never needed to look anything up to correct you on your false over-simplification of how the `Stopwatch` class works. Why should I look up all these other classes and topics when my suggested solution is already much more appropriate than yours? Would that not already imply that I am educated on the topics? ?

>> they can be used properly but the average programmer online is going to use for shitty reasons.

Don't suggest that OP don't use them in this case then because you are incapable of using them correctly. Not everyone is a self-proclaimed programmer full of inaccurate knowledge like yourself.

>> You shouldn't call someone's way of doing something wrong

Okay, how about I call it "naive" then? I made an example proving why your suggestion was a bad idea to help OP create a better program. I don't need to know anything about your work, your suggestions and ideas speak for themselves.

>> You say that there is no `DateTime.now()` and claim to be some super advanced 1337 programmer. Guess what buddy? Those are parathensis for `DateTime` formatting, Google it.

There's no `DateTime.now()` either because the `Now` property starts with a capital N and it isn't a function. And formatting is with `ToString()` or string interpolation:

CS bachelor's - sure. Did you find your degree in a garbage bin somewhere? ? You've had plenty of chances to prove yourself now and I'm still seeing nothing. Maybe it would be wiser to listen to people that actually know what they're talking about?

>> You constantly mention threading, it's 2018 buddy learn to `async`. You're entire 2 paragraphs on 'BUT WHAT IF ONE TAKES 5-10 MINUTES' becomes a moot point once if you used asynchronous methods

Find an `async` timer. Doesn't work that way with timers and `await/async`... Nothing works the way you're describing actually. You know very little, I'm not going to argue anymore with you because you lack the experience to even comprehend the proof I'm giving you.

If you think you have any clue what you're saying - then show us your "asynchronous methods" involving a consistent periodic callback that updates the tasks/minute calculation without having to wait for a task to complete. Your solution has no way of calculating tasks/minute based on time but is more of a result calculation. If he is using this calculation to display on a UI, if a task completes every ~1ms then the user is not even going to be able to read that label because it's being changed too fast. You fail to see how a threading timer would prevent this fast calculation rate and provide a way to throttle the calculation callback time interval.

edit: Just as I thought, no code from you to back your suggestion up.

-PUBG- - September 24, 2018, 8:22 pm

I am threading this and I am not sure what the safe way to go would be especially when wanting to update the per minute checks every second showing what it would be.. If that makes sense.

0xDEAD10CC - September 24, 2018, 8:34 pm

(Sep 24, 2018 08:22 PM)RapidHost™ Wrote: [\[CITE\]](#) I am threading this and I am not sure what the safe way to go would be especially when wanting to update the per minute checks every second showing what it would be.. If that makes sense.

Then thread a callback for a timed interval that does the calculations outside of the UI thread, then joins the UI thread only to update the calculated results. I posted a decent strategy above, and what you definitely don't want to do is what Razagas suggested above; he doesn't even know the difference between a method call and a property..

-PUBG- - September 24, 2018, 8:40 pm

(Sep 24, 2018 08:34 PM)0xDEAD10CC Wrote: [\[CITE\]](#) Then thread a callback for a timed interval that does the calculations outside of the UI thread, then joins the UI thread only to update the calculated results. I posted a decent strategy above, and what you definitely don't want to do is what Razagas suggested above; he doesn't even know the difference between a method call and a property..

Are you speaking about your reply saying

Code timer ms elapsed 30000ms = $30000/1000/60 = 0.5$ m elapsed

>> TASKS/m = TASKS / m elapsed

i.e. 100 tasks complete
= $100 / 0.5 = 200$ tasks/m
If so I will give that a go..

0xDEAD10CC - September 24, 2018, 9:04 pm

That was just a calculation example, I'm talking about the method used to make the calculations as well.