<http://programmers.stackexchange.com/questions/37001/how-do-you-do-asp-net-performance-testing>

1.On the process/best practices: I recommend 2010 book "[.NET Performance Testing and Optimization - The Complete Guide](http://rads.stackoverflow.com/amzn/click/1906434409)", by Paul Glavich, Chris Farrell". It describes complete ASP.NET performance testing process and many best practice tips.

Microsoft's 2004 manual "Performance Best Practices at a Glance" in Chapter 16 "[Testing .NET Application Performance](http://msdn.microsoft.com/en-us/library/ff647788.aspx)" lays down main performance testing steps, but set of tools is outdated.

2.On the tools: It is essential to use at least one tool from of each of 3 groups.  
*Performance and memory profilers:*

* [CLRProfiler](http://www.microsoft.com/downloads/en/details.aspx?familyid=86ce6052-d7f4-4aeb-9b7a-94635beebdda&displaylang=en) (free);
* Visual Studio 2008 Team Edition (and VS 2010) has a built-in performance and memory profiler;
* [Red Gate's ANTS Memory and Performance Profilers](http://www.red-gate.com/products/dotnet-development/ants-performance-profiler);
* [Micro Focus DevPartner Studio Professional](http://microfocus.com/products/micro-focus-developer/devpartner/devpartner-studio-professional-edition.aspx);
* [dotTrace Performance and Memory profiler](http://www.jetbrains.com/profiler/) that you already know.

*HTTP traffic and client tools (free):*

* Firebug / Yslow for Firefox;
* HTTP Watch (Fermium);
* Internet Explorer Dev toolbar;
* dynaTrace AJAX Edition;
* Fiddler.

*Load testing tools:*

* VSTS 2008 (and 2010) that you mentioned;
* [NeoLoad](http://www.neotys.com/product/overview-neoload.html);
* My company has a free Fiddler extension called [StresStimulus](http://stresstimulus.stimulustechnology.com/) that is not nearly as complete as the two above, but is handy for quick and easy load testing.

ASP.NET brings the Trace handler with it that can be used to trace performance of your applications. See here: <http://msdn.microsoft.com/en-us/library/3xxk09t8(v=VS.85).aspx>

Just a pointer to a tool I'm using here in the office: Red Gate's [ANTS Performance Profiler](http://www.red-gate.com/products/dotnet-development/ants-performance-profiler/).

I use it for both C# and SQL profiling.

<http://geekswithblogs.net/robp/archive/2009/03/13/speedy-c-part-4-using---and-understanding---clr.aspx>

<http://www.codeproject.com/Articles/30648/Demo-on-CLR-Profiling>

<https://chrome.google.com/webstore/category/ext/11-web-development?hl=en-GB>

Google Chrome 🡪 extensions🡪 web Developer How to: View the ASP.NET Performance Counters Available on Your Computer

**.NET Framework 4**

You can see available system-level performance counters for ASP.NET using Windows performance-counter tools.

### To view the available ASP.NET counters

1. In Windows, click **Start**, and then click **Run**.
2. In the **Run** dialog box, enter **perfmon** and then click **OK**.
3. In the **Performance** dialog box, select **System Monitor** in the tree.
4. In the status pane toolbar, click the **View Report** button.
5. In the toolbar, click the **Add** button.

The **Add Counters** dialog box appears.

1. Under **Performance object**, select the ASP.NET object to track, select **All counters**, then click **Add**.

For information about each of the ASP.NET performance counters, see [Performance Counters for ASP.NET](http://msdn.microsoft.com/en-us/library/fxk122b4(v=vs.100).aspx).

<http://msdn.microsoft.com/en-us/library/xhcx5a20(v=vs.100).aspx>

<http://msdn.microsoft.com/en-us/library/3xxk09t8(v=vs.100).aspx>

<http://msdn.microsoft.com/en-us/library/aa373083%28v=vs.85%29.aspx>

asp.net performance counters

<http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters>

<http://www.codeproject.com/Articles/45292/NET-Best-Practice-No-1-Detecting-High-Memory-cons>

<http://www.codeproject.com/Articles/8590/An-Introduction-To-Performance-Counters>

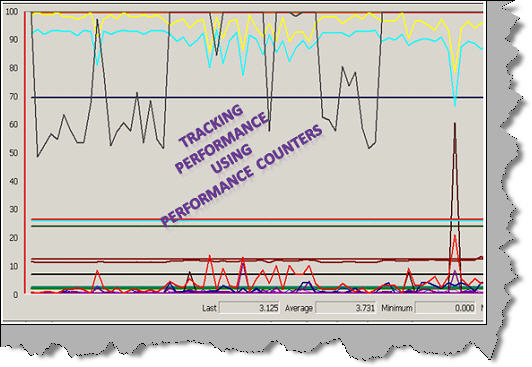
<http://msdn.microsoft.com/en-us/library/fxk122b4(v=vs.100).aspx>11

.NET Best Practice No: 3:- Using performance counters to gather performance data

|  |  |
| --- | --- |
| [My other best practices articles](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#My other best practices articles)  [Is this Article worth reading ahead?](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Is this Article worth reading ahead)  [Introduction: - My application performance is the best like a rocket J](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Introduction: - My application performance is the best like a rocket J)  [Thanks Javier and Michael](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Thanks Javier and Michael)  [At the end of the day its count, calculate and display](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#At the end of the day its count, calculate and display)  [Performance counters are not magicians](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Performance counters are not magicians)  [Types of measures in application](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Types of measures in application)  [Example on which performance counter will be tested](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Example on which performance counter will be tested)  [Adding our first instantaneous performance counter in 4 steps](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Adding our first instantaneous performance counter in 4 steps)  [Creating more sensible counter](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Creating more sensible counter)  [Average performance counters](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Average performance counters)  [Rate performance counters](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Rate performance counters)  [Performance counters left](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Performance counters left)  [Adding counters by C# code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Adding counters by Csharp code)  [Let’s ease some pain using Performance counter helper](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters" \l "Let’s ease some pain using Performance counter helper)  [Do not use it in production](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Do not use it in production)  [Conclusion](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Conclusion)  [Source code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Source code)  [Other Practices](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#Other Practices)  [My FAQ articles](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters#My FAQ articles) | **Improve garbage collector performance using finalize/dispose pattern**  [http://www.codeproject.com/KB/aspnet/DOTNETBestPractices3/BestPractises.png](http://iplayerhd.com/playerframe/progressive/69634f91-f2f5-4150-941b-e2da25b6b141.aspx?autostart=false&width=768&height=432&cbartype=bottom) |

### Is this Article worth reading ahead ?

This article discusses how we can use performance counter to gather data from an application. So we will first understand the fundamentals and then we will see a simple example from which we will collect some performance data.



### Introduction: - My application performance is the best like a rocket J

Let us start this article by a small chat between customer and developer.

**Scenario 1**

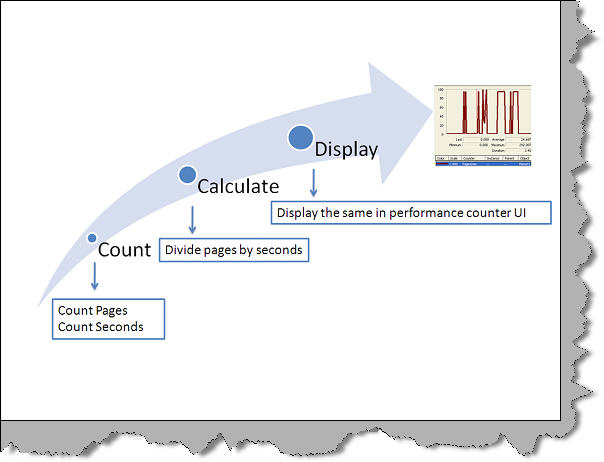
Customer: - How’s your application performance?  
Subjective developer: - Well it’s speedy, it’s the best …huuh aaa ooh it’s a like rocket.  
  
**Scenario 2**  
Customer: - How’s your application performance?  
Quantitative developer: - With 2 GB RAM , xyz processor and 20000 customer records the customer screen load in 20 secs.  
  
I am sure the second developer looks more promising than the first developer. In this article we will explore how we can use performance counters to measure performance of an application. So let’s start counting 1,2,3,4….  
  
Please feel free to download my free 500 question and answer eBook which covers .NET , ASP.NET , SQL Server , WCF , WPF , WWF@ [http://www.questpond.com](http://www.questpond.com/).    
  
  
Courtesy :- [http://scoutbase.org.uk](http://scoutbase.org.uk/)  

### Thanks Javier and Michael

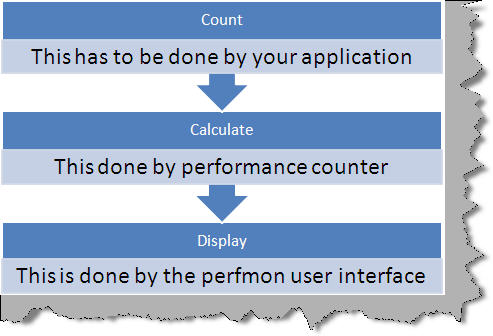
I really do not have the intellectual to write something on performance counters. But reading the below articles I was able to manage something. So first let me thank these guys and then we can move ahead in the article.  
  
Thanks a bunch Javier Canillas for creating the performance counter helper , it really eases of lot of code<http://perfmoncounterhelper.codeplex.com/>    
  
Thanks Michael Groeger for the wonderful article, I took the code of counter creation from your article<http://www.codeproject.com/KB/dotnet/perfcounter.aspx>    
  
I also picked up lot of pieces from <http://msdn.microsoft.com/en-us/library/system.diagnostics.performancecounter.aspx>  

### At the end of the day its count, calculate and display

Any performance evaluation works on count, calculate and display. For instance if you want to count how many pages in memory where processed per second we first need to count number of pages and also how many seconds where elapsed. Once we are finished with counting we then need to calculate i.e. divide the number of pages by seconds elapsed. Finally we need to display the data of our performance.



Now that we know it’s a 3 step process i.e. count, calculate and display. The counting part is done by the application. So the application needs to feed in the data during the counting phase. Please note the data is not automatically detected by the performance counters , some help needs to be provided by the application. The calculation and display is done by the performance counter and monitor.

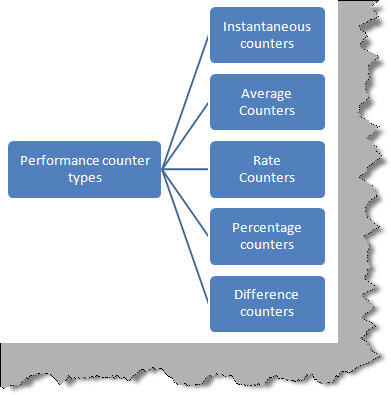


### Performance counters are not magicians

If application does not provide counter data performance counters cannot measure by himself. Performance counter cannot measure applications which do not feed performance data. In other words the application needs to feed in counter data by creating performance counter objects.

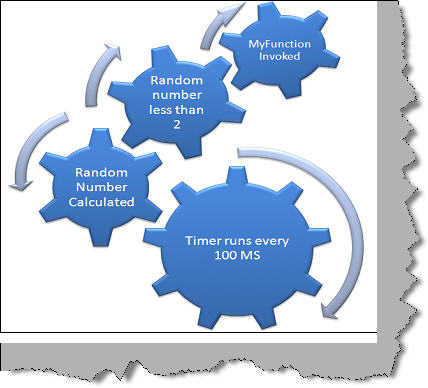
### Types of measures in application

Almost all application performance measurements fall in to one of the below six categories.  
  
**Instantaneous values: -** Many times we just want to measure the most recent value. For instance we just want to measure how many customer records where processed? , how much RAM memory has been used etc. These types of measures are termed as instantaneous or absolute values. Performance counter supports these measurement types by using instantaneous counters.  
  
**Average values: -** Sometimes instant / recent values do not really show the real picture. For instance just saying that application consumed 1 GB space is not enough. But if we can get some kind of average data consumption like 10 MB data was consumed per 1000 records probably you can get more insight of what is happening inside the application. Performance counter supports these kinds of measurement types by using average performanance counters like AverageBase, AverageTimer32, AverageCount64 etc.  
  
**Rate values: -** There are situations when you want to know the rate of events with respect to time. For example you would like to how many records where processed per second. Rate counters help us to calculate these kinds of performance metrics.  
  
**Percentage values: -** Many times we would like to see values as percentages for comparison purposes. For example you want to compare performance data between 2 computers. Comparing direct values will not be a fair comparison. So if we can have % values from both computers then the comparison can make more sense. If we want to compare values between different performance counters, percentage is much better option rather than using absolute values.   
  
For example if you want to compare how much RAM is utilized as compared to hard disk space. Comparing 1 GB ram usage with 50 GB hard disk usage is like comparing apples with oranges. If you can express these values as percentages then comparison will be fair and justifiable. Percentage performance counters can help us to express absolute values as percentages.  
  
**Difference values: -** Many times we would like to get difference performance data , for instance how much time was elapsed from the time application started, how much hard disk consumption was done by the application from the time it started etc. In order to collect these kinds of performance data we need to record the original value and the recent value. To get final performance data we need to subtract the original value from the recent value. Performance counter provides difference counters to calculate such kind of performance data.  
  
So summarizing there are 5 types of performance counters which can satisfy all the above counting needs. Below figure shows the same in a pictorial format.



### Example on which performance counter will be tested

In this complete article we will be considering a simple counter example as explained below. In this example we will have a timer which generates random number every 100 milliseconds. These random numbers are later checked to see if it’s less than 2. Incase its less than 2 then function ‘MyFunction’ is invoked.



Below is the code where the timer runs every 100 milliseconds and calculates random number. If the random number is smaller than 2 we invoke the function ‘MyFunction’.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

private void timer1\_Tick(object sender, EventArgs e)

{

*// Generate random number between 1 to 5.*

Random objRnd = new Random();

int y = objRnd.Next(1, 5);

*// If random number is less than 2 call my Function*

if (y > 2)

{

MyFunction();

}

}

Below is the code for ‘MyFunction’ which is invoked when the value of random number is less than 2. The method does not do anything as such.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

private void MyFunction()

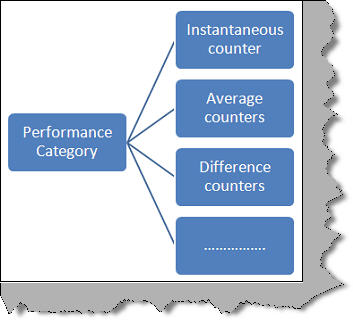
{

}

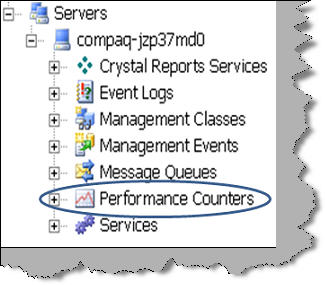
All our performance counters example in this article will use the above defined sample. 

### Adding our first instantaneous performance counter in 4 steps

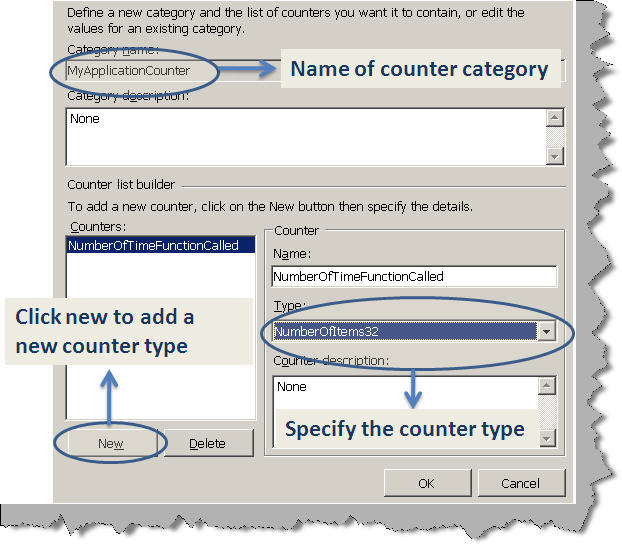
Before we go in to in depth of how to add performance counters, let’s first understand the structure of performance counters. When we create performance counters it needs to belong to some group. So we need to create a category and all performance counters will lie under that category.



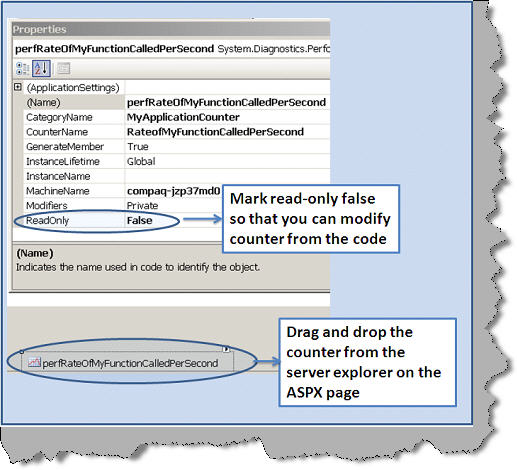
We will like to just count how many times ‘MyFunction’ was called. So let’s create an instantaneous counter called as 'NumberOfTimeFunctionCalled'. Before we move ahead let’s see how many different types of instantaneous counters are provided by performance counters:-   
  
Below definitions are taken from <http://msdn.microsoft.com/en-us/library/system.diagnostics.performancecountertype.aspx>.   
  
**NumberOfItems32:-** An instantaneous counter that shows the most recently observed value.   
  
**NumberOfItems64:-** An instantaneous counter that shows the most recently observed value. Used, for example, to maintain a simple count of a very large number of items or operations. It is the same as NumberOfItems32 except that it uses larger fields to accommodate larger values.  
  
**NumberOfItemsHEX32:-**An instantaneous counter that shows the most recently observed value in hexadecimal format. Used, for example, to maintain a simple count of items or operations.  
  
**NumberOfItemsHEX64:-** An instantaneous counter that shows the most recently observed value. Used, for example, to maintain a simple count of a very large number of items or operations. It is the same as NumberOfItemsHEX32 except that it uses larger fields to accommodate larger values.  
  
**Step 1 Create the counter: -** For our current scenario ‘NumberOfItems32’ will suffice. So let’s first create ‘NumberOfItems32’ instantaneous counter. There are two ways to create counters one is through the code and the other is using the server explorer of VS 2008. The code approach we will see later. For the time we will use server explorer to create our counter. So open your visual studio à click on view à server explorer and you should see the performance counters section as shown in the below figure. Right click on the performance counters section and select create new category. 



When we create a new category you can specify the name of the category and add counters in to this category. For the current example we have given category name as ‘MyApplication’ and added a counter type of ‘NumberOfItem32’ with name ‘NumberOfTimeFunctionCalled’.



**Step 2 Add the counter to your visual studio application: -** Once you have added the counter on the server explorer, you can drag and drop the counter on the ASPX page as shown below.   
**You need to mark ‘ReadOnly’ value as false so that you can modify the counter value from the code.**



**Step 3 Add the code to count the counter: -** Finally we need to increment the counter. We have first cleared any old values in the counter during the form load. Please note that counter values are stored globally so they do not do reset by themselves we need to do it explicitly. So in the form load we have cleared the raw value to zero.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

private void Form1\_Load(object sender, EventArgs e)

{

perfNumberOfTimeFunctionCalled.RawValue = 0;

}

Whenever the function is called we are incrementing the value by using ‘Increment’ method. Every call to the increment function increases the number by 1.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

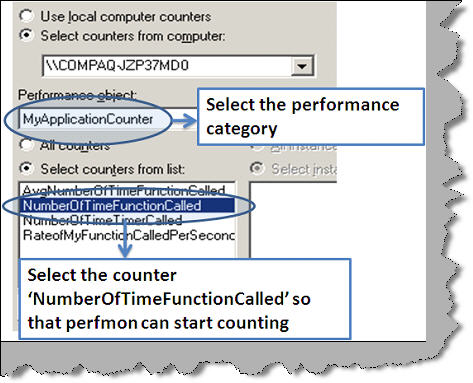
private void MyFunction()

{

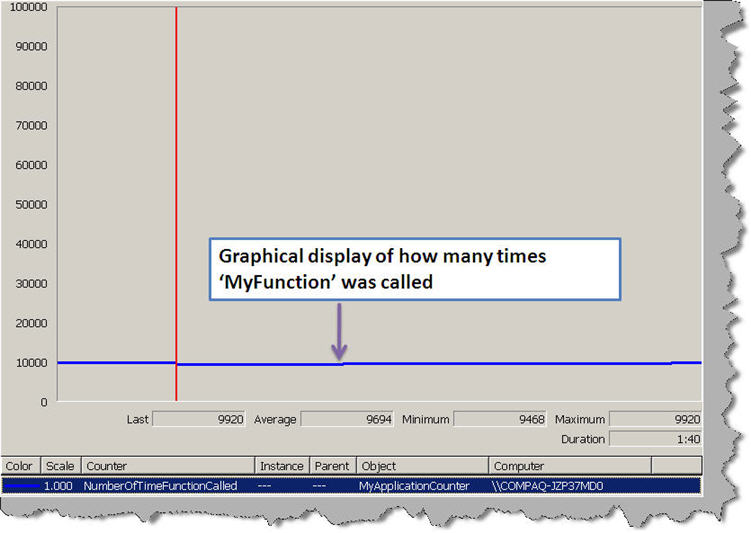
perfNumberOfTimeFunctionCalled.Increment();

}

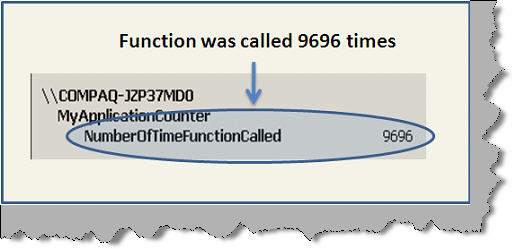
Step 4 View the counter data: - Now that we have specified the counter in the application which increments every time ‘MyFunction’ function is called. It’s time to use performance monitor to display the performance counter. So go to start à run and type ‘perfmon’. You will see there are lots of by default performance counters. For clarity sake we will remove all the counters for now and add our performance counter i.e. ‘NumberofTimeFunctionCalled’. 



You can now view the graphical display as shown in the below figure. Ensure that your application is running because application emits data which is then interpreted by the performance monitor.



Above view is a graphical view of the same. To view the same in textual format you use the view report tab provided by performance monitor. You can see the report shows that ‘MyFunction’ was called 9696 times from the time application started.



### Creating more sensible counter

In the previous section we have measured how many times ‘MyFunction’ was called. But this performance count does not really show any kind of measure. It would be great if we can also see the count of how many times the timer was called. Then later we can compare between the numbers of time timer was called and ‘MyFunction’ was called.  
  
So create an instantaneous counter and increment this counter when the timer fires as shown in the below code.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

private void timer1\_Tick(object sender, EventArgs e)

{

perfNumberOfTimeTimerCalled.Increment();

Random objRnd = new Random();

int y = objRnd.Next(1, 5);

if (y > 2)

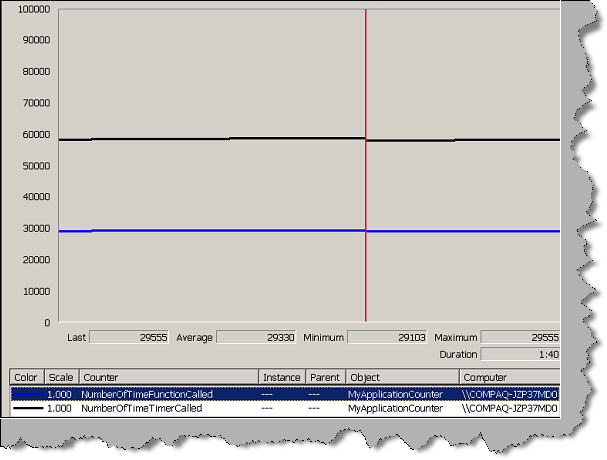
{

MyFunction();

}

}

You can see both the counters in t he below graph the blue line showing the number of times ‘MyFunction’ was called and the black one showing number of times timer called.

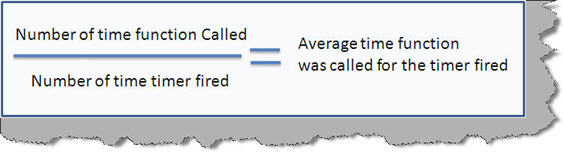


If we look in to the report view we can see for how many times the timer fired and how many times was ‘MyFunction’ called.

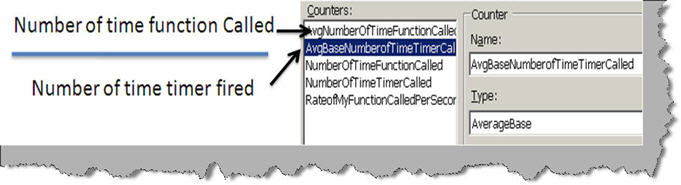


### Average performance counters

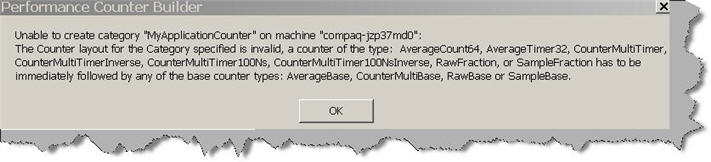
In the previous section we had counted two counters one which says how many times did the timer fire and the other says how many times ‘MyFunction’ was called. If we can have some kind of average data which says how many times was ‘MyFunctionCalled’ for the number of times timer called it will be make more sense.  
  
In order to get these kinds of metrics Average performance counters can be used. So for our scenario we need to count the number of time function was called and number of time the timer fired. Then we need to divide them to find on a average how many times was the function for the timer fired.



We need to add two counters one for the numerator and the other for the denominator. For the numerator counter we need to add ‘AverageCount64’ type of counter while for the denominator we need to add ‘AverageBase’ type of counter.



You need to add the ‘AverageBase’ counter after the ‘AverageCount64’ type counter or else you will get an error as shown below.



For every timer tick we increment the number of time timer called counter.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

private void timer1\_Tick(object sender, EventArgs e)

{

perfAvgNumberofTimeTimerCalled.Increment();

Random objRnd = new Random();

int y = objRnd.Next(1, 5);

if (y > 2)

{

MyFunction();

}

}

For every function call we increment the number of time function called counter.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

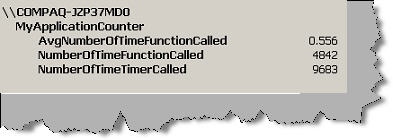
private void MyFunction()

{

perfNumberOfTimeFunctionCalled.Increment();

}

If you run the application in the view report mode you should see something as shown below. You can see on a average ‘MyFunction’ is called 0.5 times.

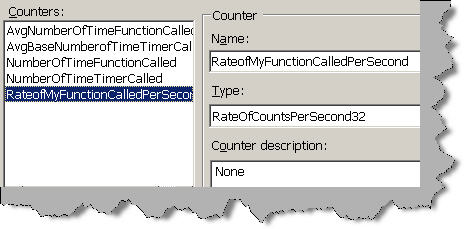


If you do the calculation you will get the same figure which is been calculated by the performance monitor.



### Rate performance counters

From our sample we would now like to find out the rate of ‘MyFunction’ calls with respect to time. So we would like know how many calls are made every second. So browse to the server explorer and add ‘rateofCountsPerSecond32’ counter as shown in the below figure. Increase this counter every time ‘MyFunction’ is called.



If you run the application you should be able to see the ‘RateofMyFunctionCalledPerSecond’ value. Below is a simple report which shows the rate of counter data which was ran for 15 seconds. The total call made in this 15 second was 72. So the average call is 5 ‘MyFunction’ calls per second.



### Performance counters left

We have left percentage counters and difference counters as they are pretty simple and straightforward. In order to maintain this article to the point and specific I have excused both these counter types.

### Adding counters by C# code

Till now we have added the performance counter using server explorer. You can also add the counter by code. The first thing is we need to import System.Diagnostics namespace.  
  
We then need to create object of ‘CounterCreationDataCollection’ object.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

CounterCreationDataCollection Mycounters = new CounterCreationDataCollection();

Create our actual counter and specify the counter type. 

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

CounterCreationData totalOps = new CounterCreationData();

totalOps.CounterName = "Numberofoperations";

totalOps.CounterHelp = "Total number of operations executed";

totalOps.CounterType = PerformanceCounterType.NumberOfItems32;

Mycounters.Add(totalOps);

Finally create the counter inside a category. Below code snippet is creating the counter in ‘MyCategory’ category.

http://www.codeproject.com/images/minus.gif Collapse | [Copy Code](http://www.codeproject.com/Articles/42001/NET-Best-Practice-No-3-Using-performance-counters)

PerformanceCounterCategory.Create("MyCategory","Sample category for Codeproject", Mycounters);

### Let’s ease some pain using Performance counter helper

Its quiet a pain to write the counter creation code. You can use performance counter helper to ease and make your code smaller. You can find the performance counter helper at <http://perfmoncounterhelper.codeplex.com/> ,

### Do not use it in production

Oh yes, use it only when you are doing development. If you are using in production ensure that there is an enabling and disabling mechanism or else it will affect your application performance.

### Conclusion

• Use performance counters to measure application data.  
• Performance counters comes in various categories like instantaneous, average , rate etc.  
• Performance counters should not be used in production. In case it’s used should have a disabling mechanism.  
• Performance counter cannot measure by itself application needs to provide data so that performance monitors can calculate and display the data.

### Source code

You can find the sample source code for the above performance counters discussed at you can download the source code from [here](http://www.codeproject.com/KB/aspnet/DOTNETBestPractices3/WindowsFormCounters.zip)

### Other Practice

For best Practices Part 1, click [here](http://www.codeproject.com/KB/aspnet/BestPrctice1.aspx)

For best Practices Part 2, click [here](http://www.codeproject.com/KB/aspnet/DONETBestPracticeNo2.aspx)

For best Practices Part 4, click [here](http://www.codeproject.com/KB/dotnet/DotNetBestPractices4.aspx)

For best Practices Part 5, click [here](http://www.codeproject.com/KB/dotnet/BestPractices5.aspx)

# <http://blog.infostretch.com/performance-monitoring-of-asp-net-application-part-1-performance-counters>

# Performance Monitoring of ASP.Net Application – Part 1 : Performance Counters

*by*PATEL AKHILKUMAR*on AUGUST 2, 2011*

**Preface:**

While working on many applications’ performance and load test projects, we have been testing quite a few web and desktop applications developed using .Net, Java, PHP etc. using different frameworks, models, technologies and libraries. One of the basic requirements on all these projects is to involve the stakeholders, who are either working directly or supporting other activities for the application development like architecture, leads, system admins and DBAs etc. Thus, the first challenge for performance test architecture would be of defining the roles and help areas for each of the stakeholders. We need to plan it in an efficient way so that it is not an expensive engagement and does not consume valuable time of various stakeholders. To achieve this, the performance architects should first prepare the list of required tools and define techniques for the stakeholders so that we can minimize their involvement and still get the best results with their help.

In this blog, we’ll talk about different tools and techniques based on the SUT hosting or setup environment. Like for Java, the set of the performance monitoring tools and techniques would be different than ASP.Net or PHP. In the same way, the set of tools may be different based on server architecture and environment. Also, we could get different requirements from different stakeholders for monitoring.

Based on all this, we suggest different tools, counters and techniques to the different groups within a project. I’ll share some details for ASP.Net technologies one by one through this series of blogs. First, let’s talk about the most common performance counters which would be helpful for monitoring and debugging of ASP.Net applications.

**Performance Counters:**

I am assuming that you are aware of performance counters, if not, you can get yourself acquainted with [performance counters here](http://msdn.microsoft.com/en-us/library/aa373083%28v=vs.85%29.aspx). As such, the monitoring of counters depends on the information that is required for resolving issues or optimizing the application. Based on the requirements, one can monitor the different performance counters into [perfmon](http://msdn.microsoft.com/en-us/library/aa645516%28v=VS.71%29.aspx" \t "_blank) – a performance monitoring utility.  Now that you are ready to starting the monitoring of counters – the important question is – which counters are you going to monitor? Following is the list of most common performance counters:

The following list provides definitions of most common performance counters. The term in bracket specifies the category of counter:

1. Application Restart (Asp.Net System) - The number of times that an application has been restarted. This counter increments with each Application\_end event but is reset if IIS is restarted.
2. Request Queued (ASP.Net System) – The number of requests waiting for service.
3. Worker Process Restarts (ASP.NET System) – The number of times a worker process has been restarted on the server.
4. Errors Total (Asp.Net Applications) – The total number of errors that occurred during the execution of HTTP requests.
5. Errors Total / Sec (Asp.Net Applications) – The number of errors per second.
6. Pipeline Instance Count (Asp.Net Applications) – The number of active pipeline instances.
7. Requests/Sec. (ASP.NET Applications) – The number of requests executed per second.
8. % CPU Utilization (Processor) – Current CPU utilization – the CPU should be utilized fully.
9. # of Exceptions Thrown (.Net CLR Exceptions) – The total number of exceptions thrown.
10. Context Switches/Sec (System) – A context switch occurs when the kernel switches the processor from one thread to another, for example, when a thread with a higher priority than the running thread becomes ready. Context switching activity is important for several reasons. A program that monopolizes the processor lowers the rate of context switches because it does not allow much processor time for the other processes threads. A high rate of context switching means that the processor is being shared repeatedly, for example, by many threads of equal priority. A high context-switch rate often indicates that there are too many threads competing for the processors on the system.

**Conclusion:**

The above list is just an introduction to basic performance counters. You might need to monitor additional performance counters as per the nature of applications and as per the requirements of different project teams. Now I will try to describe about some common tools which will be useful for ASP.Net application debugging and monitoring

<http://technet.microsoft.com/en-us/library/hh543994.aspx>