Performance Counters

VMware® Infrastructure 3 SDK

ESX has been instrumented using an extensive system of counters that collect data at runtime about performance for all aspects of the system—CPU, disk, memory, network, and so on. The data in the counters is accessible through the PerformanceManager managed object. PerformanceManager also allows you to modify the interval over which data is collected.

A complete description of how to use the PerformanceManager managed object can be found in *VMware Infrastructure SDK Programming Guide*. Refer to that document for more information about working with performance counters.

This technical note provides additional information about performance counters available in ESX. The following tables provide information on commonly used performance counters. Each table contains the following columns:

- **Name**—name of the counter.
- **Entity**—managed objects from which the performance counter is collected, for example, VirtualMachine or ResourcePool. The performance counter is collected for each device (CPU, NIC, and so on), for each instance of the managed object, or for both.
- **Unit/Description**—the unit of the performance counter, for example, milliseconds, KB, or percentage.
- **Example**—an example or illustration of the performance counter.
- **Additional Information**—any additional notes about the performance counter.

Table 1. CPU Performance Counters

Name	Entity	Unit/Description	Example	Additional Information
usage	HostSystem VirtualMachine	**Unit:** Percentage The CPU utilization over the interval of collection.	A value of 100% represents complete usage of all processor cores on the system. For example, two VMs using 50% of a four-core system are completely using two cores.	■ This metric makes it convenient to compare two hosts with different speeds ■ CPU usage can be measured only on VM level ■ The value of this counter can be changed significantly by performing the operations on the HostSystem like creating a Snapshot or removing snapshots for all the VMs.

usagemhz	HostSystem
	Virtualmachine
	ComputeResource
	ResourcePool
		Unit: MHz (million cycles per second)
		The CPU utilization over the interval of collection.
			The maximum possible value is the frequency of the processors multiplied by the number of cores.
			`cpu.usage = cpu.usagemhz / (num of cores * cpu frequency) * 100%`
			For example, a VM using 4000 MHz on a system with four 2 GHz processors is using 50% of the CPU.
			`(4000 / (4 * 2000)) *100% = 50%`
				■ This metric makes it convenient to compare two hosts with different speeds
				■ CPU usage in MHz can be measured for each VM and each instance of virtual CPU.
				■ The value of this counter can be changed significantly by performing the operations on the HostSystem like creating a Snapshot or removing a snapshot for all the VMs.
				■ Physical CPU used by ESXServer and VirtualMachines can be verified by HostSystem.summary.quickStats.overallCpuUsage and VirtualMachine.summary.quickStats.overallCpuUsage
				■ **Note:** quickStats don't have an interval. It is the sample of the value of a particular performance counter at the time the value was captured by the VI Service.

system	VirtualMachine	**Unit:** Millisecond The time spent in VMkernel during the last update interval. Some CPU time is used by the system itself for internal purposes.	N/A	■ This metric can be measured for each instance of a virtual CPU.

Table 1. CPU Performance Counters

Name	Entity	Unit/Description	Example	Additional Information
wait	VirtualMachine	**Unit:** Milliseconds The time spent waiting for hardware or VMkernel lock thread locks during the last update interval.	N/A	■ This metric can be measured for each instance of a Virtual CPU.
ready	VirtualMachine	**Unit:** Milliseconds The time to spend waiting for CPU(s) to become available in the past update interval.	The total time for a Virtual Machine that has been spent in the ready state can be found by the average for the total number of virtual CPUs. `%ready = (readyTime * 100) / statsIntervalPeriod InMilliSeconds`	■ This metric can be measured for each instance of virtual CPU. ■ Sometimes, the %ready goes over 100. That signifies that the VM is starving and does not have access to the CPU. However, it is fine if this value goes above 100 only for a short period of time. If %ready is above 100 for a considerable amount of time, then as VM has been starving for too long a period. ■ The cpu.ready counter is reported per CPU instance
extra	VirtualMachine	**Unit:** Milliseconds The time above the statically calculated entitlement. Entitlement is the share of processing time that a virtual machine should get as a result of its vCPU count and assigned shares.	N/A	■ This metric can be measured for each instance of a virtual CPU.
used	HostSystem VirtualMachine	**Unit:** Milliseconds CPU used.	N/A	■ This metric can be measured for each instance of a virtual CPU.

Table 1. CPU Performance Counters

Name	Entity	Unit/Description	Example	Additional Information

guaranteed	VirtualMachine	**Unit:** Milliseconds
		The amount of the reservation time that the virtual machine used in the past update interval.
		For example, a virtual machine using 2000MHz on a system with four 2GHz processors is using 25% of the CPU. There are 80,000ms available in a 20,000ms update interval, which means that 20,000ms of time is reserved. A virtual machine using half of its available cycles has 10,000ms guaranteed time.
			cpu.guaranteed is the consumed reservation time of CPU resources for a virtual machine.
			`cpu.guaranteed = cpu.used * interval length / cpu frequency`
			For example, if a virtual machine reserves 100MHz on a 1 GHz processor and uses 50 MHz during an interval of 20,000ms, the guaranteed time is 1000 milliseconds.
			`(50 * 20,000 / 1000) = 1000 milliseconds.`
			If cpu.usagemhz is 200 MHz during an interval of 20,000ms, the guaranteed time is 2000 milliseconds, although the used time is 4000 milliseconds.
			`(100 * 20,000 / 1000) = 2000 milliseconds.`
				■ This metric can be measured for each instance of a virtual CPU.

reservedCapacity	Host	**Unit:** MHz (million cycles per second) The value in MHz of the reservation property of the host's resource pool, or the sum of the reservation properties of the resource pool's (immediate) children, whichever is larger. Note that the children's sum of reservations can be larger than that of the parent only if the parent is marked as reservationExpandable.		■ This metric can be measured for the host.

Table 2. Disk Performance Counters

Name	Entity	Unit/Description	Example	Additional Information

kernelwritelatency	HostSystem
	VirtualMachine
	(per disk basis)
		Unit: Milliseconds
		The average time spent in the ESX Server VMKernel per write or the overhead introduced by the VMKernel.
		This counter can be calculated as:
		`Kernel Write Latency = Total Wrirte Latency – Device Write Latency.`
			`G-----K-----H-----D`
			`|---------->|`
			`Kernel Write Latency`
			■ G—Guest O/S
			■ K—Kernel
			■ H—HBA
			■ D—Device
				N/A

devicewritelatency	HostSystem
	Virtualmachine
	(per disk basis)
		Unit: Milliseconds
		The average time taken to complete a write from the physical device. That is, the time from the HBA to service an IO request.
			`G-----K-----H-----D`
			`|---->|`
			`Device Write Latency`
			■ G—Guest O/S
			■ K—Kernel
			■ H—HBA
			■ D—Device
				N/A

totalWriteLatency	HostSystem
	VirtualMachine
	(per disk basis)
		Unit: Milliseconds
		The average amount of time taken for a write from the perspective of a guest operating system.
			`G-----K-----H-----D`
			`|---------->|`
			`Total Write Latency`
			■ G—Guest O/S
			■ K—Kernel
			■ H—HBA
			■ D—Device
				■ This metric can be calculated as the sum of Kernel Write Latency and Physical Device Write Latency.

read	HostSystem VirtualMachine (per disk basis)	**Unit:** KBps The disk read rate. It indicates the amount of data read in the performance interval.	N/A	■ This metric can be calculated by multiplying blocksRead and blockSize.
write	HostSystem VirtualMachine (per disk basis)	**Unit:** KBps The disk write rate. It indicates the amount of data written to disk in the performance interval	N/A	■ This metric can be calculated by multiplying blocksWritten and blockSize.
usage	HostSystem VirtualMachine (per disk basis)	**Unit:** KBps The sum of Disk Read and Disk Write on all disks instances of the host or virtual machine, as applicable to two entities host and virtual machines.	N/A	■ For hosts, this metric can be represented on a per Virtual Machine basis as a stacked graph.

Table 3. Memory Performance Counters

Name	Entity	Unit/Description	Example	Additional Information
swapin	HostSystem VirtualMachine ResourcePool	**Unit:** KB The total amount of memory that has been swapped in or read from the swap space.	A VM has 200MB of target memory that must be swapped out. In this case: **swaptarget = 200000** Suppose that the VM swaps out 100 MB, then swaps in 30MB, and later swaps out 50 MB. In this case: **swapin = 30000** **swapped = 120000** (100MB - 30 MB + 50 MB) **swapout = 150000** (100MB+50MB which is the cumulative memory that has been swapped out)	N/A
swapout	HostSystem Virtualmachine ResourcePool	**Unit:** KB The total amount of pages (physical) that have been written to the swap space.		N/A
swaptarget	VirtualMachine ResourcePool	**Unit:** KB The amount of memory that can be swapped.		N/A
swapped	VirtualMachine ResourcePool	**Unit:** KB The amount of memory that is swapped.		N/A
swapunreserved	HostSystem ComputeResource	**Unit:** KB The amount of unreserved swap space.	N/A	■ This counter is available to ESX 2.x where a single swap file is used for all virtual machines. On ESX 3.x and beyond a swap file is generated on a per virtual machine basis and there is no need for this counter.
swapused	HostSystem ComputeResource	**Unit:** KB The amount of memory used for swap space.	N/A	■ This metric can be referred to as the number of physical pages that have been swapped.
vmmemctl	HostSystem VirtualMachine ResourcePool	**Unit:** KB The total amount of ballooned memory.	N/A	■ This metric can be referred to as the amount of memory reclaimed due to ballooning. It uses machine pages rather than physical pages because ballooned pages are 1:1 mapped.
consumed	HostSystem VirtualMachine ResourcePool	**Unit:** KB The amount of memory that is not free.	N/A	For VirtualMachine: ■ The amount of host memory consumed by the virtual machine for guest memory. For HostSystem: ■ This counter can be calculated as: `total memory of host – free memory` ■ It includes memory reserved for the service console. Note that the entire memory reserved for the service console is considered as used here. ■ This counter refers to the machine pages.

Retrieving Performance Data

The following steps outline the basic process for retrieving the performance data using the VI Perl Toolkit. For full code, please refer to "Full Listing of viperformance.pl" on page 8. You may also refer the performance samples shipped with the VI Perl Toolkit.

1 Get a Managed Object Reference to a Performance Manager

```
my $perfmgr_view = Vim::get_view(mo_ref => Vim::get_service_content()->perfManager);
```

2 Get Performance Metric ID's using QueryAvailablePerfMetric which retrieves available performance metrics for the specified Managed Entity between the optional beginTime and endTime

```
my $availmetricid = $perfmgr_view->QueryAvailablePerfMetric(entity => $entity);
```

3 Create a Filtered List of Performance Metric ID's for the desired type, using the GroupInfo. In this example, type is set to "cpu", but you could also set the type to "disk" to retrieve disk metrics or "mem" to retrieve memory metrics.

```
my $type = "cpu";

   my $counters;
   my @filtered_list;
   my $perfCounterInfo = $perfmgr_view->perfCounter;

   foreach (@$perfCounterInfo) {
      my $key = $_->key;
      $all_counters->{ $key } = $_;
      my $group_info = $_->groupInfo;
      if ($group_info->key eq $type) {
         $counters->{ $key } = $_;
      }
   }

   foreach (@$availmetricid) {
      if (exists $counters->{$_->counterId}) {
        my $metric = PerfMetricId->new (counterId => $_->counterId,
                                        instance => (Opts::get_option('instance') || ''));
         push @filtered_list, $metric;
      }
   }
```

4 Create PerfQuerySpec, passing the managed entity, for which the performance data has to be retrieved, filtered list of performance metric id's, intervalId, maxSample incase of real-time data

```
$perf_query_spec = PerfQuerySpec->new(entity => $host,
                                      metricId => @filtered_list,
                                      'format' => 'csv',
                                      intervalId => shift @$intervals,
                                      maxSample => Opts::get_option('samples'));
```

5 Invoke QueryPerf, passing the PerfQuerySpec created above, for fetching the performance statistics.

```
$perf_data = $perfmgr_view->QueryPerf( querySpec => $perf_query_spec);
```

6 Print the values fetched by the QueryPerf

```
 foreach (@$perf_data) {
    Display $_->value
}
```

Full Listing of viperformance.pl

```
#!/usr/bin/perl –w
#
# Copyright (c) 2007 VMware, Inc.  All rights reserved.

use strict;
use warnings;

use FindBin;
use lib "$FindBin::Bin/../";

use VMware::VIRuntime;
use AppUtil::HostUtil;
use AppUtil::VMUtil;

$SIG{__DIE__} = sub{Util::disconnect()};
$Util::script_version = "1.0";

sub retrieve_performance;

my %opts = (
   'host' => {
      type => "=s",
      help => "Name of the host",
      required => 1,
   },
   'countertype' => {
      type => "=s",
      help => "Counter type [cpu | mem | net | disk | sys]",
      required => 1,
   },
   'interval' => {
      type => "=i",
      help => "Interval in seconds",
      required => 0,
   },
   'instance' => {
      type => "=s",
      help => "Name of instance to query",
      required => 0,
   },
   'samples' => {
      type => "=s",
      help => "Number of samples to retrieve",
      required => 0,
      default => 10,
   },
   'out' => {
      type => "=s",
      help => "Name of file to hold output",
      required => 0,
   },
);

Opts::add_options(%opts);
Opts::parse();
Opts::validate(\&validate);

Util::connect();

my $all_counters;
retrieve_performance();

Util::disconnect();

sub retrieve_performance() {
   my $host = Vim::find_entity_view(view_type => "HostSystem",
```

```
                                    filter => {'name' => Opts::get_option('host')});
   if (!defined($host)) {
      Util::trace(0,"Host ".Opts::get_option('host')." not found.\n");
      return;
   }

   my $perfmgr_view = Vim::get_view(mo_ref => Vim::get_service_content()->perfManager);

   my @perf_metric_ids = get_perf_metric_ids(perfmgr_view=>$perfmgr_view,
                                             host => $host,
                                             type => Opts::get_option('countertype'));

   my $perf_query_spec;
   if(defined Opts::get_option('interval')) {
      $perf_query_spec = PerfQuerySpec->new(entity => $host,
                                            metricId => @perf_metric_ids,
                                            'format' => 'csv',
                                            intervalId => Opts::get_option('interval'),
                                            maxSample => Opts::get_option('samples'));
   }
   else {
      my $intervals = get_available_intervals(perfmgr_view => $perfmgr_view,
                                              host => $host);
      $perf_query_spec = PerfQuerySpec->new(entity => $host,
                                            metricId => @perf_metric_ids,
                                            'format' => 'csv',
                                            intervalId => shift @$intervals,
                                            maxSample => Opts::get_option('samples'));
   }

   if(defined Opts::get_option('out')) {
      my $filename = Opts::get_option('out');
      open(OUTFILE, ">$filename");
   }
   my $perf_data;
   eval {
       $perf_data = $perfmgr_view->QueryPerf( querySpec => $perf_query_spec);
   };
   if ($@) {
      if (ref($@) eq 'SoapFault') {
         if (ref($@->detail) eq 'InvalidArgument') {
            Util::trace(0,"Specified parameters are not correct");
         }
      }
      return;
   }
   if (! @$perf_data) {
      Util::trace(0,"Either Performance data not available for requested period "
                    ."or instance is invalid\n");
      my $intervals = get_available_intervals(perfmgr_view=>$perfmgr_view,
                                              host => $host);
      Util::trace(0,"\nAvailable Intervals\n");
      foreach(@$intervals) {
         Util::trace(0,"Interval " . $_ . "\n");
      }
      return;
   }
   foreach (@$perf_data) {
      print_log("Performance data for: " . $host->name . "\n");
      my $time_stamps = $_->sampleInfoCSV;
      my $values = $_->value;
      foreach (@$values) {
         print_counter_info($_->id->counterId, $_->id->instance);
         print_log("Sample info : " . $time_stamps);
         print_log("Value: " . $_->value . "\n");
      }
   }
}
```

```
sub print_counter_info {
   my ($counter_id, $instance) = @_;
   my $counter = $all_counters->{$counter_id};
   print_log("Counter: " . $counter->nameInfo->label);
   if (defined $instance) {
      print_log("Instance : " . $instance);
   }
   print_log("Description: " . $counter->nameInfo->summary);
   print_log("Units: " . $counter->unitInfo->label);
}

sub get_perf_metric_ids {
   my %args = @_;
   my $perfmgr_view = $args{perfmgr_view};
   my $entity = $args{host};
   my $type = $args{type};

   my $counters;
   my @filtered_list;
   my $perfCounterInfo = $perfmgr_view->perfCounter;
   my $availmetricid = $perfmgr_view->QueryAvailablePerfMetric(entity => $entity);

   foreach (@$perfCounterInfo) {
      my $key = $_->key;
      $all_counters->{ $key } = $_;
      my $group_info = $_->groupInfo;
      if ($group_info->key eq $type) {
         $counters->{ $key } = $_;
      }
   }

   foreach (@$availmetricid) {
      if (exists $counters->{$_->counterId}) {
         #push @filtered_list, $_;
         my $metric = PerfMetricId->new (counterId => $_->counterId,
                                         instance => (Opts::get_option('instance') || ''));
         push @filtered_list, $metric;
      }
   }
   return \@filtered_list;
}

sub get_available_intervals {
   my %args = @_;
   my $perfmgr_view = $args{perfmgr_view};
   my $entity = $args{host};

   my $historical_intervals = $perfmgr_view->historicalInterval;
   my $provider_summary = $perfmgr_view->QueryPerfProviderSummary(entity => $entity);
   my @intervals;
   if ($provider_summary->refreshRate) {
      push @intervals, $provider_summary->refreshRate;
   }
   foreach (@$historical_intervals) {
      push @intervals, $_->samplingPeriod;
   }
   return \@intervals;
}

sub validate {
   my $valid = 1;
   if (Opts::option_is_set('countertype')) {
      my $ctype = Opts::get_option('countertype');
      if(!(($ctype eq 'cpu') || ($ctype eq  'mem') || ($ctype eq 'net')
         || ($ctype eq 'disk') || ($ctype eq 'sys'))) {
         Util::trace(0,"counter type must be [cpu | mem | net | disk | sys]");
         $valid = 0;
```

```
      }
   }
   if (Opts::option_is_set('out')) {
     my $filename = Opts::get_option('out');
     if ((length($filename) == 0)) {
        Util::trace(0, "\n'$filename' Not Valid:\n$@\n");
        $valid = 0;
     }
     else {
        open(OUTFILE, ">$filename");
        if ((length($filename) == 0) ||
          !(-e $filename && -r $filename && -T $filename)) {
           Util::trace(0, "\n'$filename' Not Valid:\n$@\n");
           $valid = 0;
        }
     }
  }
  return $valid;
}

sub print_log {
   my ($prop) = @_;
   if (defined (Opts::get_option('out'))) {
      print OUTFILE  $prop."\n";
   }
   else {
      Util::trace(0, $prop." \n");
   }
}

__END__

=head1 NAME

viperformance.pl - Retrieves performance counters from a host.

=head1 SYNOPSIS

 viperformance.pl [options]

=head1 DESCRIPTION

This VI Perl command-line utility provides an interface to retrieve
performance counters from the specified host. Performance counters
shows these primary attributes: CPU Usage, Memory Usage, Disk I/O Usage,
Network I/O Usage, and System Usage.

=head1 OPTIONS

=head2 GENERAL OPTIONS

=over

=item B<Host>

Required. Name of the host.

=item B<countertype>

Required. Counter type [cpu | mem | net | disk | sys].

=item B<interval>

Optional. Interval in seconds.

=item B<samples>

Optional. Number of samples to retrieve. Default: 10
```

```
=item B<instance>

Optional. Name of instance to query. Default: Aggregate of all instance.
          Specify '*' for all the instances.

=item B<out>

Optional. Name of the filename to hold the output.

=back

=head1 EXAMPLES

Retrieve performance counter for countertype 'cpu' from host 'Host123'

 viperformance.pl --url https://<host>:<port>/sdk/vimService
                --username myuser --password mypassword
                --host Host123 --countertype cpu

Retrieve performance counter for countertype 'net' from host 'Host123'.
Let the interval be 30 seconds and the number of samples be 3.

 viperformance.pl --url https://<host>:<port>/sdk/vimService
                --username myuser --password mypassword
                --host Host123 --countertype net --interval 30
                --samples 3

Retrieve performance counter for countertype 'net' from host 'Host123' for
cpu instance 1.

 viperformance.pl --url https://<host>:<port>/sdk/vimService
                --username myuser --password mypassword
                --host Host123 --countertype net --interval 30
                --samples 3 --instance 1

Retrieve performance counter for countertype 'net' from host 'Host123' for
all the cpu instances.

 viperformance.pl --url https://<host>:<port>/sdk/vimService
                --username myuser --password mypassword
                --host Host123 --countertype net --interval 30
                --samples 3 --instance *

=head1 SUPPORTED PLATFORMS

All operations work with VMware VirtualCenter 2.0.1 or later.

All operations work with VMware ESX Server 3.0.1 or later.
```