These are some notes I have gathered on querying objects concerning process, modules, threads in PowerShell. This paper is primarily concerned with comparing changes in loaded processes, modules, and threads over time. My notes concern these TypeNames in PowerShell 2.0:

- System.Diagnostics.Process
- System.Diagnostics.ProcessModuleCollection
- System.Management.ManagementObject#root\cimv2\Win32_Process
- System.Management.ManagementObject#root\cimv2\Win32_Thread

PS (Get-Process)

The 'Get-Process' cmdlet (alias 'ps') for the .NET class System.Diagnostics.Process has both Module and Thread collection derived classes. The 'ps' cmdlet has 51 properties including the ProcessModule and the ProcessThread Collections:

```powershell
$a = Get-Process
$a | gm | ? {$_._MemberType -eq "Property"}
```

<table>
<thead>
<tr>
<th>Name</th>
<th>MemberType</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BasePriority</td>
<td>Property</td>
<td>System.Int32 BasePriority {get;}</td>
</tr>
<tr>
<td>Container</td>
<td>Property</td>
<td>System.ComponentModel.IContainer Container {get;}</td>
</tr>
<tr>
<td>EnableRaisingEvents</td>
<td>Property</td>
<td>System.Boolean EnableRaisingEvents {get;set;}</td>
</tr>
<tr>
<td>ExitCode</td>
<td>Property</td>
<td>System.Int32 ExitCode {get;}</td>
</tr>
<tr>
<td>ExitTime</td>
<td>Property</td>
<td>System.DateTime ExitTime {get;}</td>
</tr>
<tr>
<td>Handle</td>
<td>Property</td>
<td>System.IntPtr Handle {get;}</td>
</tr>
<tr>
<td>HandleCount</td>
<td>Property</td>
<td>System.IntPtr HandleCount {get;}</td>
</tr>
<tr>
<td>HasExited</td>
<td>Property</td>
<td>System.Boolean HasExited {get;}</td>
</tr>
<tr>
<td>Id</td>
<td>Property</td>
<td>System.Int32 Id {get;}</td>
</tr>
<tr>
<td>MachineName</td>
<td>Property</td>
<td>System.String MachineName {get;}</td>
</tr>
<tr>
<td>MainModule</td>
<td>Property</td>
<td>System.Diagnostics.ProcessModule MainModule {get;}</td>
</tr>
<tr>
<td>MainWindowHandle</td>
<td>Property</td>
<td>System.IntPtr MainWindowHandle {get;}</td>
</tr>
<tr>
<td>MainWindowTitle</td>
<td>Property</td>
<td>System.String MainWindowTitle {get;}</td>
</tr>
<tr>
<td>MinWorkingSet</td>
<td>Property</td>
<td>System.IntPtr MinWorkingSet {get;set;}</td>
</tr>
<tr>
<td>Modules</td>
<td>Property</td>
<td>System.Diagnostics.ProcessModuleCollection Modules {get;}</td>
</tr>
<tr>
<td>NonpagedSystemMemorySize</td>
<td>Property</td>
<td>System.Int32 NonpagedSystemMemorySize {get;}</td>
</tr>
<tr>
<td>NonpagedSystemMemorySize64</td>
<td>Property</td>
<td>System.Int64 NonpagedSystemMemorySize64 {get;}</td>
</tr>
<tr>
<td>PageFile</td>
<td>Property</td>
<td>System.Int32 PageFile {get;}</td>
</tr>
<tr>
<td>PageFile64</td>
<td>Property</td>
<td>System.Int64 PageFile64 {get;}</td>
</tr>
<tr>
<td>PeakPageFile</td>
<td>Property</td>
<td>System.Int32 PeakPageFile {get;}</td>
</tr>
<tr>
<td>PeakPageFile64</td>
<td>Property</td>
<td>System.Int64 PeakPageFile64 {get;}</td>
</tr>
</tbody>
</table>

All information and methods subject to revision and not guaranteed. Use at your own risk.
Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

```csharp
PeakPagedMemorySize64     Property   System.Int64 PeakPagedMemorySize64 {get;}
PeakVirtualMemorySize      Property   System.Int32 PeakVirtualMemorySize {get;}
PeakVirtualMemorySize64    Property   System.Int64 PeakVirtualMemorySize64 {get;}
PeakWorkingSet             Property   System.Int32 PeakWorkingSet {get;}
PeakWorkingSet64           Property   System.Int64 PeakWorkingSet64 {get;}
PriorityBoostEnabled       Property   System.Boolean PriorityBoostEnabled {get;set;}
PriorityClass              Property   System.Diagnostics.ProcessPriorityClass PriorityClass {get;set;}
PrivateMemorySize          Property   System.Int32 PrivateMemorySize {get;}
PrivateMemorySize64        Property   System.Int64 PrivateMemorySize64 {get;}
PrivilegedProcessorTime    Property   System.TimeSpan PrivilegedProcessorTime {get;}
ProcessName                Property   System.String ProcessName {get;}
ProcessorAffinity          Property   System.IntPtr ProcessorAffinity {get;set;}
Responding                 Property   System.Boolean Responding {get;}
SessionId                  Property   System.Int32 SessionId {get;}
Site                       Property   System.ComponentModel.ISite Site {get;set;}
StandardError              Property   System.IO.StreamReader StandardError {get;}
StandardInput              Property   System.IO.StreamWriter StandardInput {get;}
StandardOutput             Property   System.IO.StreamReader StandardOutput {get;}
StartInfo                  Property   System.Diagnostics.ProcessStartInfo StartInfo {get;set;}
StartTime                  Property   System.DateTime StartTime {get;}
SynchronizingObject        Property   System.ComponentModel.ISynchronizeInvoke SynchronizingObject {get;set;}
Threads                    Property   System.Diagnostics.ProcessThreadCollection Threads {get;}
TotalProcessorTime         Property   System.TimeSpan TotalProcessorTime {get;}
UserProcessorTime          Property   System.TimeSpan UserProcessorTime {get;}
VirtualMemorySize          Property   System.Int32 VirtualMemorySize {get;}
VirtualMemorySize64        Property   System.Int64 VirtualMemorySize64 {get;}
WorkingSet                 Property   System.Int32 WorkingSet {get;}
WorkingSet64               Property   System.Int64 WorkingSet64 {get;}
```

We can get at all this process information with the 'ps' alias for defaults fields or we can call customized fields by name or by alias with the 'select-object' cmdlet. For example, the alias CPU below is 'TotalProcessorTime.TotalSeconds':

```powershell
ps | Select Name,ID,VM,WS,CPU | ft -auto
```

We can get at all this process information with the 'ps' alias for defaults fields or we can call customized fields by name or by alias with the 'select-object' cmdlet. For example, the alias CPU below is 'TotalProcessorTime.TotalSeconds':

```powershell
ps | Select Name,ID,VM,WS,CPU | ft -auto
```

The 'sort-object' cmdlet can be applied to the pipeline as well:

```
ps | Select Name,ID,VM,WS,CPU | Sort ID | ft -auto
```

All information and methods subject to revision and not guaranteed. Use at your own risk.
We can also use a 'foreach' language command to help us powerfully invoke the `System.Diagnostics.ProcessModuleCollection` and `System.Diagnostics.ProcessThreadCollection`. Note that the 'foreach' language command is distinct from the Powershell's 'foreach-object' cmdlet. Please see help 'about_foreach' for more information.

```
foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.MainModule}
```

(Or alternatively)

```
foreach ($id in ( Get-Process | ? {$_.Name} )) {write $id.MainModule}
```

### Size(K) ModuleName FileName
--- ---------- --------
80 ApMsgFwd.exe C:\Program Files\Apoint\ApMsgFwd.exe
32 Apntex.exe C:\Program Files\Apoint\Apntex.exe
176 Apoint.exe C:\Program Files\Apoint\Apoint.exe
112 BDTUpdateService.exe C:\Program Files (x86)\Spyware Doctor\BDT\BDTUpdateService.exe

We can find "module information" (as opposed to "process information") about all currently loaded "MainModules" (e.g. binaries) with:

```
$findGP=foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.MainModule}
$findGP  | ft *  | more
```

### Size Company        FileVersion    ProductVersion Description    Product        ModuleName     FileName          BaseAddress ModuleMemorySize EntryPointAddr
--- -------        -----------    -------------- -----------    -------        ----------     --------          ----------- -------------- --------------
80 Alps Electr... 7, 0, 0, 20    7, 0, 0, 20    ApMsgFwd       ApMsgFwd       ApMsgFwd.exe   C:\Program ...        4194304          81920        4205648
32 Alps Electr... 7.0.1.29       7.0.1.29       Alps Pointi... Alps Pointi... Apntex.exe     C:\Program ...        4194304          32768        4205088
176 Alps Electr... 7.0.7.156      7.0.7.156      Alps Pointi... Alps Pointi... Apoint.exe     C:\Program ...        4194304         180224        4243984
112 Threat Expe... 2, 0, 6, 15    2, 0, 6, 15    Browser Def... Threat Expe... BDTUpdateSe... C:\Program ...        4194304         114688        4228632
40 1.0.0.0        1.0.0.0        CCP            CCP            CCP.exe        C:\Program ...         589824          40960         612350
944 Google Inc.    0.0.0.0        0.0.0.0        Google Chrome  Google Chrome  chrome.exe     C:\Users\Ad...         1900544         966656        2188746

The `get-Process` cmdlet also access `System.Diagnostics.ProcessThreadCollection`. Usually, this is done with 'ps':

```
$modules=ps
$modules | % {$_Thread}
```

But we can also do this with the 'foreach' language command.

```
foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.Threads}
```

### BasePriority CurrentPriority IdealProcessor PriorityBoostEnabled PriorityLevel
: 6        : 11        : 3324        : True        : Normal

All information and methods subject to revision and not guaranteed. Use at your own risk.
PrivilegedProcessorTime : 00:00:04.6800300
StartAddress : 2001135024
StartTime : 7/21/2010 10:21:45 AM
ThreadState : Wait
TotalProcessorTime : 00:00:06.3960410
UserProcessorTime : 00:00:01.7160110
...
The syntax below allows the user to take full advantages of Powershell's flexible query, member coupling, and .NET classes reach. Below we are storing to and then displaying from the variable $FWAPI all ModuleNames (binaries) that import "FirewallAPI.dll":

```powershell
$FWAPI=foreach ($id in ( Get-Process | ? {$_.Modules -match "FirewallAPI.dll"}) ) {write $id.MainModule}
```

Below we are storing to and then displaying from the variable $FWAPI all ModuleNames (binaries) that import "FirewallAPI.dll":

```powershell
$FWAPI=foreach ($id in ( Get-Process | ? {$_.Modules -match "FirewallAPI.dll"}) ) {write $id.MainModule}
```

The syntax below allows the user to take full advantages of Powershell's flexible query, member coupling, and .NET classes reach. Below we are storing to and then displaying from the variable $FWAPI all ModuleNames (binaries) that import "FirewallAPI.dll":

```powershell
$FWAPI=foreach ($id in ( Get-Process | ? {$_.Modules -match "FirewallAPI.dll"}) ) {write $id.MainModule}
```

Now, we query every module (dll) loaded by the process svchost.exe:

```powershell
$SVChost_lm=foreach ($id in ( Get-Process | ? {($_.Name -match "svchost")}) {write $id.Modules}
```

Now, we query every module (dll) loaded by the process svchost.exe:

```powershell
$SVChost_lm=foreach ($id in ( Get-Process | ? {($_.Name -match "svchost")}) {write $id.Modules}
```

All information and methods subject to revision and not guaranteed. Use at your own risk.
All information and methods subject to revision and not guaranteed. Use at your own risk.
Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

To compare the listing of every executable and their modules over time we can use "compare-object" (alias "diff"):

$a=foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.Name,$id.ID,$id.Modules}
sleep -seconds 10
# Start notepad now
$b=foreach ($id in ( Get-Process | ? {$_.Modules} )) {write $id.Name,$id.ID,$id.Modules}
$c=diff $a $b
$c | fl * | out-file modulelog

Raw output like the following is produced:

InputObject : notepad
SideIndicator : =>
InputObject : 1712
SideIndicator : =>
SideIndicator : =>

We can 'unroll' all the 'diff' object with

$d=$c | % {$_.InputObject}
$d

notepad
1712
Size(K) ModuleName                                             FileName
------- ----------                                             --------
188 notepad.exe                                         C:\Windows\system32\notepad.exe
1560 ntdll.dll                                          C:\Windows\system32\ntdll.dll
1204 kernel32.dll                                       C:\Windows\system32\kernel32.dll
...

or with selection and/or sorting of module and process specific properties:

$d=$c | % {$_.InputObject} | Select ModuleName,FileName,ModuleMemorySize,EntryPointAddress | Sort EntryPointAddress
$d | ft -wrap -auto

ModuleName                                          FileName                                            ModuleMemorySize                                    EntryPointAddress
----------                                          --------                                            ----------------                                    -----------------
ntdll.dll                                           C:\Windows\system32\ntdll.dll                       1597440                                             0
kernel32.dll                                        C:\Windows\system32\kernel32.dll                    1232896                                             2002105520
USER32.dll                                          C:\Windows\system32\USER32.dll                      839680                                              2003343380
notepad.exe                                         C:\Windows\system32\notepad.exe                     192512                                              4282241460
smum64.dll                                          C:\Program Files (x86)\Spyware Doctor\smum64.dll    282624                                              8791685212184...
...

We can use the semantics below to create an object ("$FindLots") that displays process name, process ID, process filename followed by the modules and threads that belong to that process.

All information and methods subject to revision and not guaranteed. Use at your own risk.
Looking at Processes, Modules, and Threads with Powershell 2.0

```powershell
$FindLots=foreach ($id in ( Get-Process | ? {$_.Name} )) {$id.Name,$id.ID,$id.Path,$id.Modules,$id.Threads}
$FindLots | gm | findstr TypeName

TypeName: System.String
TypeName: System.Int32
TypeName: System.Diagnostics.ProcessModuleCollection
TypeName: System.Diagnostics.ProcessThreadCollection

This type of 'hybrid' object produces output that first lists the process name, PID and full path to the binary followed by all the modules and then all the threads loaded by the respective process:

$FindLots | ft

ApMsgFwd
3320
C:\Program Files\Apoint\ApMsgFwd.exe

<table>
<thead>
<tr>
<th>Size(K)</th>
<th>ModuleName</th>
<th>FileName</th>
</tr>
</thead>
<tbody>
<tr>
<td>80</td>
<td>ApMsgFwd.exe</td>
<td>C:\Program Files\Apoint\ApMsgFwd.exe</td>
</tr>
<tr>
<td>1560</td>
<td>ntdll.dll</td>
<td>C:\Windows\system32\ntdll.dll</td>
</tr>
<tr>
<td>1204</td>
<td>kernel32.dll</td>
<td>C:\Windows\system32\kernel32.dll</td>
</tr>
<tr>
<td>1056</td>
<td>ADVAPI32.dll</td>
<td>C:\Windows\system32\ADVAPI32.dll</td>
</tr>
<tr>
<td>1292</td>
<td>RPCRT4.dll</td>
<td>C:\Windows\system32\RPCRT4.dll</td>
</tr>
<tr>
<td>820</td>
<td>USER32.dll</td>
<td>C:\Windows\system32\USER32.dll</td>
</tr>
<tr>
<td>400</td>
<td>GDI32.dll</td>
<td>C:\Windows\system32\GDI32.dll</td>
</tr>
</tbody>
</table>

BasePriority : 6
CurrentPriority : 8
Id : 3324
IdealProcessor : 
PriorityBoostEnabled : True
PriorityLevel : Normal
PrivilegedProcessorTime : 00:00:02.9484189
...

Comparing 'hybrid' (mixed 'TypeName') objects produces some problematic results since the original objects have to be reset to process and thread based arrays and then compared.

```powershell
$a=foreach ($id in ( Get-Process | ? {$_.Name} )) {$id.Name,$id.ID,$id.Path,$id.Modules,$id.Threads}
sleep -seconds 10
# start mspaint
$b=foreach ($id in ( Get-Process | ? {$_.Name} )) {$id.Name,$id.ID,$id.Path,$id.Modules,$id.Threads}
$aa=$a | ? {$_.Item}
$bb=$b | ? {$_.Item}
$cc=compare-object $aa $bb

The resulting objects lose the linkage between process and module when they become 'ParameterizedProperties':

$bb | gm | findstr Property
```

All information and methods subject to revision and not guaranteed. Use at your own risk.
Looking at Processes, Modules, and Threads with Powershell 2.0 All Rights Reserved Ryan M. Ferris RMF Network Security July 2010

Item ParameterizedProperty System.Diagnostics.ProcessModule Item(int index) {get;}
Count Property System.Int32 Count {get;}
Item ParameterizedProperty System.Diagnostics.ProcessThread Item(int index) {get;}
Count Property System.Int32 Count {get;}

However, we still have list of binaries, modules, and threads that have changed:

```
$cc
```

<table>
<thead>
<tr>
<th>InputObject</th>
<th>SideIndicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>{1808, 3132, 3140, 3240...}</td>
<td>=&gt;</td>
</tr>
<tr>
<td>{3676, 3896, 3912, 1264...}</td>
<td>=&gt;</td>
</tr>
<tr>
<td>{228, 236, 240, 304...}</td>
<td></td>
</tr>
<tr>
<td>{640, 636, 748, 1008...}</td>
<td>=&gt;</td>
</tr>
</tbody>
</table>
...

```
$cc | % {$.InputObject} | more
```

<table>
<thead>
<tr>
<th>BasePriority</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>CurrentPriority</td>
<td>10</td>
</tr>
<tr>
<td>Id</td>
<td>1808</td>
</tr>
<tr>
<td>IdealProcessor</td>
<td></td>
</tr>
<tr>
<td>PriorityBoostEnabled</td>
<td>True</td>
</tr>
<tr>
<td>PriorityLevel</td>
<td>Normal</td>
</tr>
<tr>
<td>PrivilegedProcessorTime</td>
<td>00:00:01.1700075</td>
</tr>
<tr>
<td>StartAddress</td>
<td>4209661616</td>
</tr>
<tr>
<td>StartTime</td>
<td>7/30/2010 10:55:03 AM</td>
</tr>
<tr>
<td>ThreadState</td>
<td>Wait</td>
</tr>
</tbody>
</table>
...

```
$cc | % {$.InputObject} | findstr ModuleName
```

<table>
<thead>
<tr>
<th>ModuleName</th>
<th>mspaint.exe</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileName</td>
<td>C:\Windows\system32\mspaint.exe</td>
</tr>
<tr>
<td>BaseAddress</td>
<td>4285464576</td>
</tr>
<tr>
<td>ModuleMemorySize</td>
<td>614400</td>
</tr>
<tr>
<td>EntryPointAddress</td>
<td>4285797576</td>
</tr>
</tbody>
</table>
...

```
$cc | % {$.InputObject} | findstr /V "IdealProcessor" | findstr Id
```

| Id | 228 |
| Id | 236 |
| Id | 240 |
| Id | 304 |
...

All information and methods subject to revision and not guaranteed. Use at your own risk.
GWMI (Windows Management Instrumentation)

A comparison between aliases ps ("get-process") and gwmi ("get-WMIObject") shows that gwmi uses the PID of each process as a handle to display much the same process information as ps. WMI is a comprehensive database of Windows objects of all types. A full discussion is beyond the scope of this document. However, WMI’s database and management format does make time slice comparisons straightforward.

```powershell
$i=ps
$j=gwmi win32_process
diff $i $j
```

<table>
<thead>
<tr>
<th>InputObject</th>
<th>SideIndicator</th>
</tr>
</thead>
<tbody>
<tr>
<td>\RMFVISTA\root\cimv2:Win32_Process.Handle=&quot;0&quot;</td>
<td>=&gt;</td>
</tr>
<tr>
<td>\RMFVISTA\root\cimv2:Win32_Process.Handle=&quot;4&quot;</td>
<td>=&gt;</td>
</tr>
<tr>
<td>\RMFVISTA\root\cimv2:Win32_Process.Handle=&quot;524&quot;</td>
<td>=&gt;</td>
</tr>
<tr>
<td>\RMFVISTA\root\cimv2:Win32_Process.Handle=&quot;656&quot;</td>
<td>=&gt;</td>
</tr>
<tr>
<td>... System.Diagnostics.Process (ApMsgFwd)</td>
<td>&lt;=</td>
</tr>
<tr>
<td>System.Diagnostics.Process (Apoint)</td>
<td>&lt;=</td>
</tr>
<tr>
<td>System.Diagnostics.Process (audiodg)</td>
<td>&lt;=</td>
</tr>
<tr>
<td>System.Diagnostics.Process (BDTUpdateService)</td>
<td>&lt;=</td>
</tr>
<tr>
<td>... Note how the 'Handle' field is the same as the process ID in the gwmi query below:</td>
<td></td>
</tr>
</tbody>
</table>

```powershell
$j=gwmi win32_process
$j | Select ProcessId, Name, Handle, HandleCount | Sort ProcessId, HandleCount | ft -auto
```

<table>
<thead>
<tr>
<th>ProcessId</th>
<th>Name</th>
<th>Handle</th>
<th>HandleCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>System Idle Process</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>System</td>
<td>4</td>
<td>1729</td>
</tr>
<tr>
<td>176</td>
<td>Tcpview.exe</td>
<td>176</td>
<td>308</td>
</tr>
<tr>
<td>432</td>
<td>svchost.exe</td>
<td>432</td>
<td>1439</td>
</tr>
<tr>
<td>512</td>
<td>smss.exe</td>
<td>512</td>
<td>28</td>
</tr>
<tr>
<td>632</td>
<td>wmpnscfg.exe</td>
<td>632</td>
<td>133</td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Comparing process command lines is straightforward with WMI:

```powershell
$a=foreach ($id in ( gwmi win32_process | ? {$_._HANDLE} )) {write $id.CommandLine}
sleep -seconds 10
# Start notepad now
$b=foreach ($id in ( gwmi win32_process | ? {$_._HANDLE} )) {write $id.CommandLine}
$c=diff $a $b
```

<table>
<thead>
<tr>
<th>InputObject</th>
<th>SideIndicator</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

All information and methods subject to revision and not guaranteed. Use at your own risk.
"C:\Windows\system32\notepad.exe" => "C:\Windows\system32\notepad.exe"

GWMI also has a win32_thread component with a feature not so easily access in System.Diagnostics.Process: a relative path link from each thread to process handle or 'PID'. This makes linking threads with their processes more straightforward.

$j=gwmi win32_thread
$j | Select ProcessHandle, Handle, StartAddress | Sort ProcessHandle, Handle | ft -auto

ProcessHandle Handle StartAddress
------------- ------ ------------
0             0          54986864
0             1          54986864
1004          1392     1997105712
1004          1688     1997105712
1004          2160     1997105712
...

$j | %{$__RELPATH}

Win32_Thread.Handle="0",ProcessHandle="0"
Win32_Thread.Handle="1",ProcessHandle="0"
Win32_Thread.Handle="8",ProcessHandle="4"
Win32_Thread.Handle="16",ProcessHandle="4"
Win32_Thread.Handle="20",ProcessHandle="4"
Win32_Thread.Handle="24",ProcessHandle="4"
...

Because WMI's win32_thread class links process handles (PID) explicitly with thread handles, we can definitively see which processes have changed or created new thread handles in the diff below.

$a=foreach ($id in ( gwmi win32_thread | ? {$_.HANDLE} )) {write $id.ProcessHandle}
sleep -seconds 10
# Start notepad now
$b=foreach ($id in ( gwmi win32_thread | ? {$_.HANDLE} )) {write $id.ProcessHandle}
$c=diff $a $b
$c | % {ps -id $_.inputobject} | Select ProcessName,ID | ft -auto

ProcessName     Id
-----------     --
services       748
svchost        664
dschtos        700
dschtos        1288
explorer       3088
SearchIndexer  3676
notepad        4116

All information and methods subject to revision and not guaranteed. Use at your own risk.