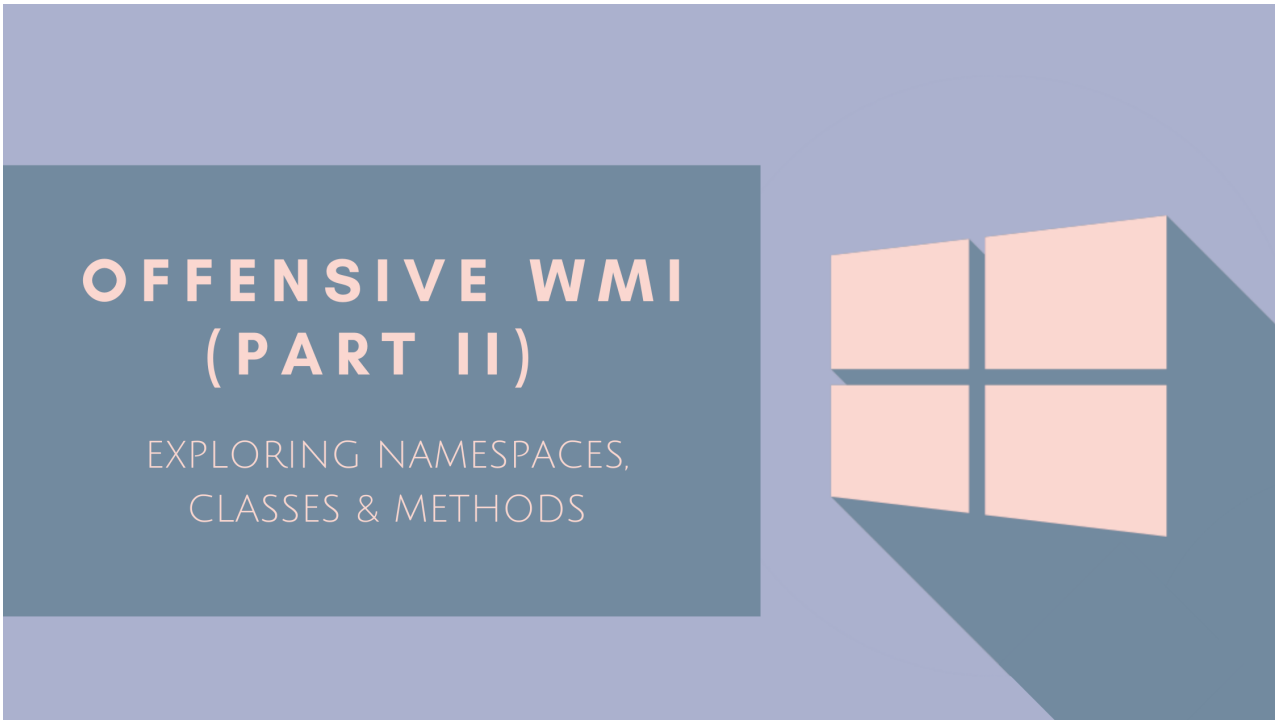
Offensive WMI - Exploring Namespaces, Classes & Methods (Part 2)

0xinfection.github.io/posts/wmi-classes-methods-part-2

September 5, 2021

2021-09-05



This blog post is the second part of the “Offensive WMI” series (the first is here), and this article will be focusing on the 3 major components in WMI that we’d be majorly dealing with. Throughout the article, we’ll be using both WMI and CIM cmdlets interchangeably so that we’re well-versed with both cmdlet types.

Namespaces

Let’s recall what namespaces are in simple terms:

> A **namespace** organizes information similar to folders in a filesystem. However, instead of physical locations (like on a disk), they are more logical in nature.

All namespaces in WMI are instances of the `__Namespace` system class. To get a list of all namespaces under the `root` namespace, we can query the same class using:

```
Get-WmiObject -Namespace root -Class __Namespace
```

```
PS C:\Users\pew> Get-WmiObject -Namespace root -Class __Namespace


__GENUS          : 2
__CLASS          : __NAMESPACE
__SUPERCLASS     : __SystemClass
__DYNASTY        : __SystemClass
__RELPATH        : __NAMESPACE.Name="subscription"
__PROPERTY_COUNT : 1
__DERIVATION     : {__SystemClass}
__SERVER         : DESKTOP-3PABHIK
__NAMESPACE      : ROOT
__PATH           : \\DESKTOP-3PABHIK\ROOT:__NAMESPACE.Name="subscription"
Name             : subscription
PSComputerName   : DESKTOP-3PABHIK

__GENUS          : 2
__CLASS          : __NAMESPACE
__SUPERCLASS     : __SystemClass
__DYNASTY        : __SystemClass
__RELPATH        : __NAMESPACE.Name="DEFAULT"
__PROPERTY_COUNT : 1
__DERIVATION     : {__SystemClass}
__SERVER         : DESKTOP-3PABHIK
__NAMESPACE      : ROOT
__PATH           : \\DESKTOP-3PABHIK\ROOT:__NAMESPACE.Name="DEFAULT"
Name             : DEFAULT
PSComputerName   : DESKTOP-3PABHIK

__GENUS          : 2
__CLASS          : __NAMESPACE
__SUPERCLASS     : __SystemClass
__DYNASTY        : __SystemClass
__RELPATH        : __NAMESPACE.Name="CIMV2"
```

The output contains a lot of info, so to filter out the noise, we can use Powershell's `select`:

```
Get-WmiObject -Namespace root -Class __Namespace | select name
```

```
PS C:\Users\pew> Get-WmiObject -Namespace root -Class __Namespace | select name

name
----
subscription
DEFAULT
CIMV2
msdtc
Cli
SECURITY
SecurityCenter2
RSOP
PEH
StandardCimv2
WMI
directory
Policy
Interop
Hardware
ServiceModel
SecurityCenter
Microsoft
Appv
```

Now we have a list of namespaces on our system. All these namespaces will be referred to as `root\<namespace>`, e.g. `root\DEFAULT`, `root\CIMV2`, etc, since they are namespaces under the root namespace.

> **NOTE:** One weirdly intriguing fact is that the default namespace in WMI is not `root\DEFAULT` but rather `root\CIMV2` (it has been like this since Windows 2000).

The same can be achieved using the CIM cmdlet `Get-CimInstance`, where there is no need of :

```
Get-CimInstance -Namespace root -ClassName __Namespace
```

```
PS C:\Users\pew> Get-CimInstance -Namespace root -ClassName __Namespace

Name            PSComputerName
----            --------------
subscription
DEFAULT
CIMV2
msdtc
Cli
SECURITY
SecurityCenter2
RSOP
PEH
StandardCimv2
WMI
directory
Policy
Interop
Hardware
ServiceModel
SecurityCenter
Microsoft
Appv
```

Okay, now that's sorted, what about nested namespaces? We already saw that there are several namespaces under the `root` namespace. We can simply write up a script that recursively gets us the namespaces (from PSMag):

```
Function Get-WmiNamespace {
    Param (
        $Namespace='root'
    )
    Get-WmiObject -Namespace $Namespace -Class __NAMESPACE | ForEach-Object {
            ($ns = '{0}\{1}' -f $_.__NAMESPACE,$_.Name)
            Get-WmiNamespace $ns
    }
}
```

```
PS C:\Users\pew> Function Get-WmiNamespace {
>>     Param (
>>         $Namespace='root'
>>     )
>>     Get-WmiObject -Namespace $Namespace -Class __NAMESPACE | ForEach-Object {
>>             ($ns = '{0}\{1}' -f $_.__NAMESPACE,$_.Name)
>>             Get-WmiNamespace $ns
>>     }
>> }
PS C:\Users\pew> Get-WmiNamespace
ROOT\subscription
ROOT\subscription\ms_409
ROOT\DEFAULT
ROOT\DEFAULT\ms_409
ROOT\CIMV2
ROOT\CIMV2\mdm
ROOT\CIMV2\mdm\dmmap
```

> **NOTE:** The classes and namespaces may vary from machine to machine depending upon the hardware available, applications installed, and many other factors.

Classes

Now that we have a list of namespaces available to make use of, let's take a look at classes. So what are classes?

> A **WMI class** represents a specific item in your system. It could be anything ranging from system processes to hardware (e.g. a network card), services, etc.

Now, classes are divided into 3 major categories (this is a requirement of the CIM standard):

- **Core classes**: They apply to all areas of management and provide few basic functionalities. You'll usually see them starting with double underscores (e.g. `__SystemSecurity`).
- **Common classes**: These are extensions of core classes, and apply to specific management areas. You'll identify one when you see a class prefixed with `CIM_` (e.g. `CIM_TemperatureSensor`).
- **Extended classes**: These are extra additions to common classes based on tech stacks. (e.g. `Win32_Process`).

Classes are further divided into these types:

- **Abstract classes**: These are templates to define new classes.
- **Static classes**: Mostly used to store data.
- **Dynamic classes**: These are retrieved from a provider and represents a WMI managed resource. We're mostly interested in classes of this type.
- **Association classes**: Describes relationships between classes and managed resources.

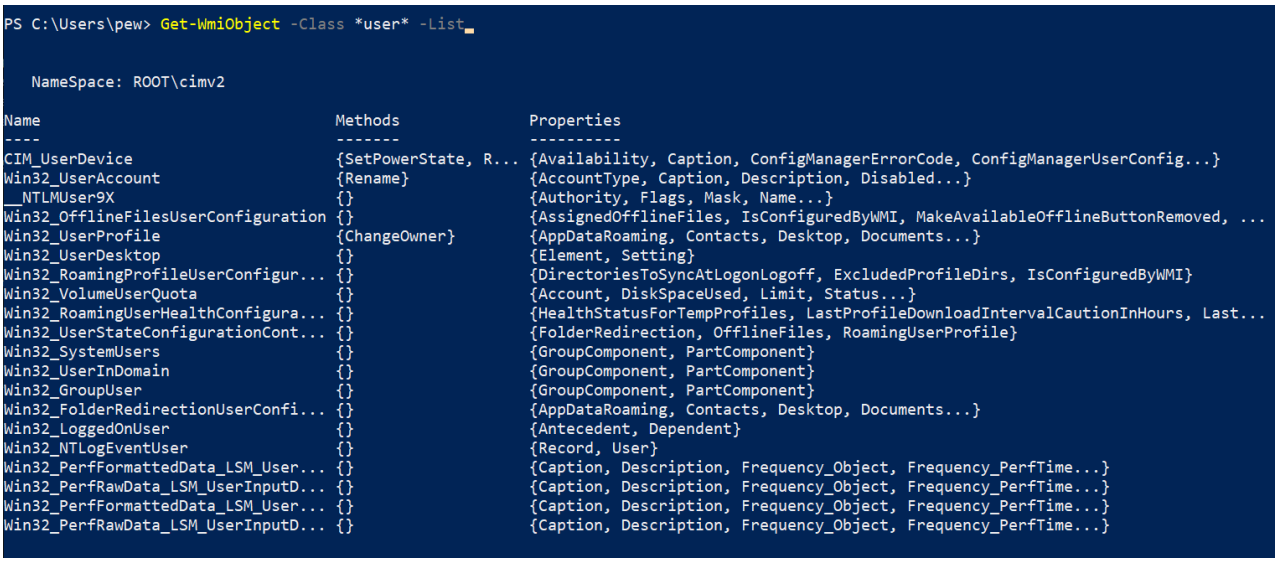
Listing Classes

Enough theory. Let's try to find some classes. Once again, we can use the `Get-WmiObject` cmdlet to list the available classes:

```
Get-WmiObject -Class * -List
```

This will list all the classes above, but for the sake of an example, let's say we are interested in the users on the system. We can narrow down to our specific use case using the following command, which lists all available classes for fetching/manipulating user information:

```
Get-WmiObject -Class *user* -List
```



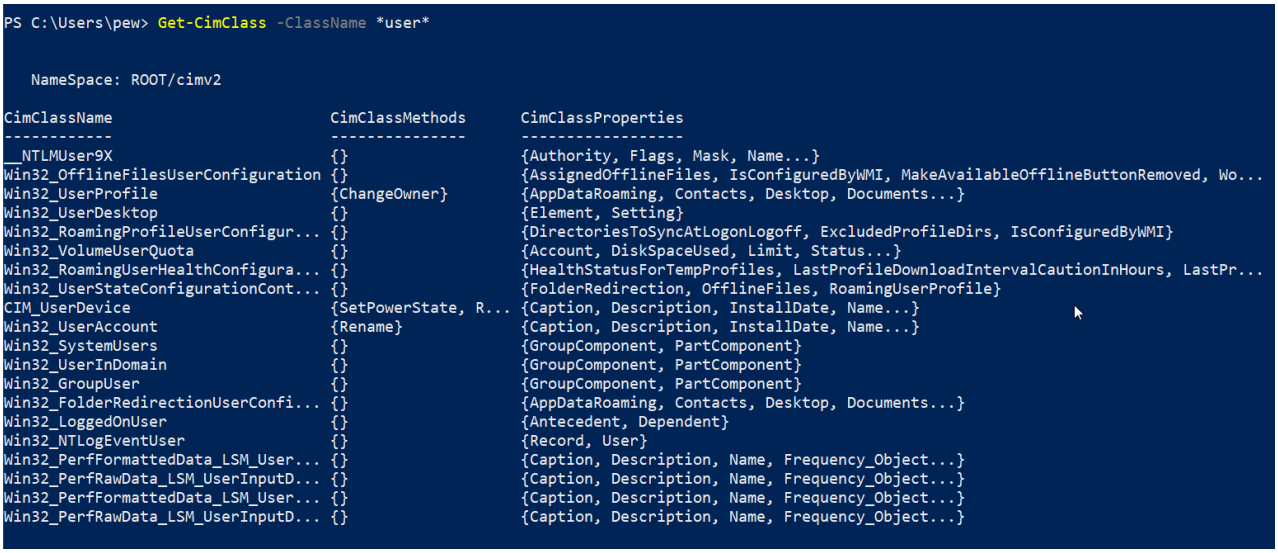
```
PS C:\Users\pew> Get-WmiObject -Class *user* -List

   NameSpace: ROOT\cimv2

Name                                Methods              Properties
----                                -------              ----------
CIM_UserDevice                      {SetPowerState, R... {Availability, Caption, ConfigManagerErrorCode, ConfigManagerUserConfig...}
Win32_UserAccount                   {Rename}             {AccountType, Caption, Description, Disabled...}
__NTLMUser9X                        {}                   {Authority, Flags, Mask, Name...}
Win32_OfflineFilesUserConfiguration {}                   {AssignedOfflineFiles, IsConfiguredByWMI, MakeAvailableOfflineButtonRemoved, ...
Win32_UserProfile                   {ChangeOwner}        {AppDataRoaming, Contacts, Desktop, Documents...}
Win32_UserDesktop                   {}                   {Element, Setting}
Win32_RoamingProfileUserConfigur... {}                   {DirectoriesToSyncAtLogonLogoff, ExcludedProfileDirs, IsConfiguredByWMI}
Win32_VolumeUserQuota               {}                   {Account, DiskSpaceUsed, Limit, Status...}
Win32_RoamingUserHealthConfigura... {}                   {HealthStatusForTempProfiles, LastProfileDownloadIntervalCautionInHours, Last...
Win32_UserStateConfigurationCont... {}                   {FolderRedirection, OfflineFiles, RoamingUserProfile}
Win32_SystemUsers                   {}                   {GroupComponent, PartComponent}
Win32_UserInDomain                  {}                   {GroupComponent, PartComponent}
Win32_GroupUser                     {}                   {GroupComponent, PartComponent}
Win32_FolderRedirectionUserConfi... {}                   {AppDataRoaming, Contacts, Desktop, Documents...}
Win32_LoggedOnUser                  {}                   {Antecedent, Dependent}
Win32_NTLogEventUser                {}                   {Record, User}
Win32_PerfFormattedData_LSM_User... {}                   {Caption, Description, Frequency_Object, Frequency_PerfTime...}
Win32_PerfRawData_LSM_UserInputD... {}                   {Caption, Description, Frequency_Object, Frequency_PerfTime...}
Win32_PerfFormattedData_LSM_User... {}                   {Caption, Description, Frequency_Object, Frequency_PerfTime...}
Win32_PerfRawData_LSM_UserInputD... {}                   {Caption, Description, Frequency_Object, Frequency_PerfTime...}
```

The same can be achieved with the `Get-CimClass` cmdlet also:

```
Get-CimClass -ClassName *user*
```



```
PS C:\Users\pew> Get-CimClass -ClassName *user*

   NameSpace: ROOT/cimv2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
__NTLMUser9X                        {}                   {Authority, Flags, Mask, Name...}
Win32_OfflineFilesUserConfiguration {}                   {AssignedOfflineFiles, IsConfiguredByWMI, MakeAvailableOfflineButtonRemoved, Wo...
Win32_UserProfile                   {ChangeOwner}        {AppDataRoaming, Contacts, Desktop, Documents...}
Win32_UserDesktop                   {}                   {Element, Setting}
Win32_RoamingProfileUserConfigur... {}                   {DirectoriesToSyncAtLogonLogoff, ExcludedProfileDirs, IsConfiguredByWMI}
Win32_VolumeUserQuota               {}                   {Account, DiskSpaceUsed, Limit, Status...}
Win32_RoamingUserHealthConfigura... {}                   {HealthStatusForTempProfiles, LastProfileDownloadIntervalCautionInHours, LastPr...
Win32_UserStateConfigurationCont... {}                   {FolderRedirection, OfflineFiles, RoamingUserProfile}
CIM_UserDevice                      {SetPowerState, R... {Caption, Description, InstallDate, Name...}
Win32_UserAccount                   {Rename}             {Caption, Description, InstallDate, Name...}
Win32_SystemUsers                   {}                   {GroupComponent, PartComponent}
Win32_UserInDomain                  {}                   {GroupComponent, PartComponent}
Win32_GroupUser                     {}                   {GroupComponent, PartComponent}
Win32_FolderRedirectionUserConfi... {}                   {AppDataRoaming, Contacts, Desktop, Documents...}
Win32_LoggedOnUser                  {}                   {Antecedent, Dependent}
Win32_NTLogEventUser                {}                   {Record, User}
Win32_PerfFormattedData_LSM_User... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_LSM_UserInputD... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfFormattedData_LSM_User... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_LSM_UserInputD... {}                   {Caption, Description, Name, Frequency_Object...}
```

> **NOTE:** For a list of all Win32 classes, you can refer to Microsoft's documentation on classes. The Win32 provider provides classes for 4 different categories: Computer System Hardware Classes, Operating System Classes, Performance Counter Classes and WMI Service Management Classes.

Remember that we talked about *dynamic* classes being the ones that provide us instances? To get only the dynamic classes, we can make use of the `-QualifierName` switch of `Get-CimClass` cmdlet:

```
Get-CimClass -ClassName *user* -QualifierName dynamic
```

```
PS C:\Users\pew> Get-CimClass -ClassName *user* -QualifierName dynamic_


   NameSpace: ROOT/cimv2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
Win32_UserAccount                   {Rename}             {Caption, Description, InstallDate, Name...}
Win32_OfflineFilesUserConfiguration {}                   {AssignedOfflineFiles, IsConfiguredByWMI, MakeAvailableOfflineButtonRemoved, ...
Win32_UserProfile                   {ChangeOwner}        {AppDataRoaming, Contacts, Desktop, Documents...}
Win32_UserDesktop                   {}                   {Element, Setting}
Win32_RoamingProfileUserConfigur... {}                   {DirectoriesToSyncAtLogonLogoff, ExcludedProfileDirs, IsConfiguredByWMI}
Win32_VolumeUserQuota               {}                   {Account, DiskSpaceUsed, Limit, Status...}
Win32_RoamingUserHealthConfigura... {}                   {HealthStatusForTempProfiles, LastProfileDownloadIntervalCautionInHours, Last...
Win32_UserStateConfigurationCont... {}                   {FolderRedirection, OfflineFiles, RoamingUserProfile}
Win32_SystemUsers                   {}                   {GroupComponent, PartComponent}
Win32_UserInDomain                  {}                   {GroupComponent, PartComponent}
Win32_GroupUser                     {}                   {GroupComponent, PartComponent}
Win32_FolderRedirectionUserConfi... {}                   {AppDataRoaming, Contacts, Desktop, Documents...}
Win32_LoggedOnUser                  {}                   {Antecedent, Dependent}
Win32_NTLogEventUser                {}                   {Record, User}
Win32_PerfFormattedData_LSM_User... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_LSM_UserInputD... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfFormattedData_LSM_User... {}                   {Caption, Description, Name, Frequency_Object...}
Win32_PerfRawData_LSM_UserInputD... {}                   {Caption, Description, Name, Frequency_Object...}
```

So far so good. What's next? Querying the classes to get the juicy stuff out of them.

Fetching Classes

We're interested in the `Win32_UserAccount` class this time. Fetching data is simple, we can simply:

```
Get-WmiObject -Class Win32_UserAccount
```

```
PS C:\Users\pew> Get-WmiObject -Class Win32_UserAccount


AccountType : 512
Caption     : DESKTOP-3PABHIK\Administrator
Domain      : DESKTOP-3PABHIK
SID         : S-1-5-21-3057680761-1860298131-55431140-500
FullName    :
Name        : Administrator

AccountType : 512
Caption     : DESKTOP-3PABHIK\DefaultAccount
Domain      : DESKTOP-3PABHIK
SID         : S-1-5-21-3057680761-1860298131-55431140-503
FullName    :
Name        : DefaultAccount

AccountType : 512
Caption     : DESKTOP-3PABHIK\Guest
Domain      : DESKTOP-3PABHIK
SID         : S-1-5-21-3057680761-1860298131-55431140-501
FullName    :
Name        : Guest

AccountType : 512
Caption     : DESKTOP-3PABHIK\pew
Domain      : DESKTOP-3PABHIK
SID         : S-1-5-21-3057680761-1860298131-55431140-1001
FullName    :
Name        : pew
```

> **TIP**: To get a more verbose output, you can pipe the above command into Powershell's `Format-List` or `fl`, something like: `Get-WmiObject -Class Win32_UserAccount | fl *` which will get you everything the class has to offer.

The CIM cmdlet `Get-CimInstance` can also be used to fetch the same info:

```
Get-CimInstance -ClassName Win32_UserAccount
```

```
PS C:\Users\pew> Get-CimInstance -ClassName Win32_UserAccount

Name            Caption                 AccountType          SID                     Domain
----            -------                 -----------          ---                     ------
Administrator   DESKTOP-3PABHIK\Admini... 512                  S-1-5-21-3057680761-18... DESKTOP-3PABHIK
DefaultAccount  DESKTOP-3PABHIK\Defaul... 512                  S-1-5-21-3057680761-18... DESKTOP-3PABHIK
Guest           DESKTOP-3PABHIK\Guest     512                  S-1-5-21-3057680761-18... DESKTOP-3PABHIK
pew             DESKTOP-3PABHIK\pew       512                  S-1-5-21-3057680761-18... DESKTOP-3PABHIK
WDAGUtilityAc... DESKTOP-3PABHIK\WDAGUt... 512                  S-1-5-21-3057680761-18... DESKTOP-3PABHIK
```

Now we have a list of all user accounts on the system!

Let's turn our attention to the processes running on the system. The class `Win32_Process` gives us a list of processes running on the system:

```
Get-WmiObject -Class Win32_Process
```

It is not uncommon for a lot of processes to be running on a system that might make your terminal keep scrolling endlessly! To avoid that we can use the `-Filter` switch to get a specific process we are looking for (here we've picked `lsass.exe`):

```
Get-WmiObject -Class Win32_Process -Filter 'name="lsass.exe"'
```

```
PS C:\Users\pew> Get-WmiObject -Class Win32_Process -Filter 'name="lsass.exe"'


__GENUS                 : 2
__CLASS                 : Win32_Process
__SUPERCLASS            : CIM_Process
__DYNASTY               : CIM_ManagedSystemElement
__RELPATH               : Win32_Process.Handle="720"
__PROPERTY_COUNT        : 45
__DERIVATION            : {CIM_Process, CIM_LogicalElement, CIM_ManagedSystemElement}
__SERVER                : DESKTOP-3PABHIK
__NAMESPACE             : root\cimv2
__PATH                  : \\DESKTOP-3PABHIK\root\cimv2:Win32_Process.Handle="720"
Caption                 : lsass.exe
CommandLine             :
CreationClassName       : Win32_Process
CreationDate            : 20210905130813.559825-420
CSCreationClassName     : Win32_ComputerSystem
CSName                  : DESKTOP-3PABHIK
Description             : lsass.exe
ExecutablePath          :
ExecutionState          :
Handle                  : 720
HandleCount             : 1142
InstallDate             :
KernelModeTime          : 2031250
MaximumWorkingSetSize   :
MinimumWorkingSetSize   :
Name                    : lsass.exe
OSCreationClassName     : Win32_OperatingSystem
OSName                  : Microsoft Windows 10 Enterprise Evaluation|C:\Windows|\Device\Harddisk0\Partition2
OtherOperationCount     : 1406
OtherTransferCount      : 757670
PageFaults              : 5701
PageFileUsage           : 6300
ParentProcessId         : 536
PeakPageFileUsage       : 6328
```

The CIM cmdlet alternative `Get-CimInstance` gives a shorter, more comprehensive output in this case (and it also supports the `-Filter` switch):

```
Get-CimInstance -ClassName Win32_Process
```

```
PS C:\Users\pew> Get-CimInstance -ClassName Win32_Process

ProcessId Name                         HandleCount WorkingSetSize VirtualSize
--------- ----                         ----------- -------------- -----------
0         System Idle Process          0           8192           8192
4         System                       2058        143360         3985408
108       Registry                     0           73183232       85819392
368       smss.exe                     53          1163264        2203359711232
464       csrss.exe                    436         5312512        2203412115456
540       wininit.exe                  162         7036928        2203387621376
560       csrss.exe                    290         5066752        2203417415680
640       winlogon.exe                 277         12439552       2203420164096
680       services.exe                 580         9830400        2203384905728
692       lsass.exe                    1125        18735104       2203417661440
820       svchost.exe                  1273        30228480       2203464220672
844       fontdrvhost.exe              32          3829760        2203386961920
848       fontdrvhost.exe              32          3375104        2203386687488
940       svchost.exe                  957         12361728       2203400003584
992       svchost.exe                  258         8298496        2203397320704
384       dwm.exe                      857         70197248       2203619950592
1040      svchost.exe                  109         5513216        2203386359808
1048      svchost.exe                  144         6172672        2203394641920
1136      svchost.exe                  383         15532032       2203423784960
1172      svchost.exe                  246         14356480       2203435802624
1184      svchost.exe                  216         10125312       2203397558272
1192      svchost.exe                  227         12058624       2203401801728
1276      svchost.exe                  121         7348224        2203387666432
1328      svchost.exe                  221         9637888        2203397136384
1416      svchost.exe                  383         19050496       2203426344960
1496      svchost.exe                  126         7921664        2203388583936
1596      svchost.exe                  212         7593984        2203391565824
1652      svchost.exe                  169         7917568        2203395842048
1660      svchost.exe                  152         5935104        2203388870656
```

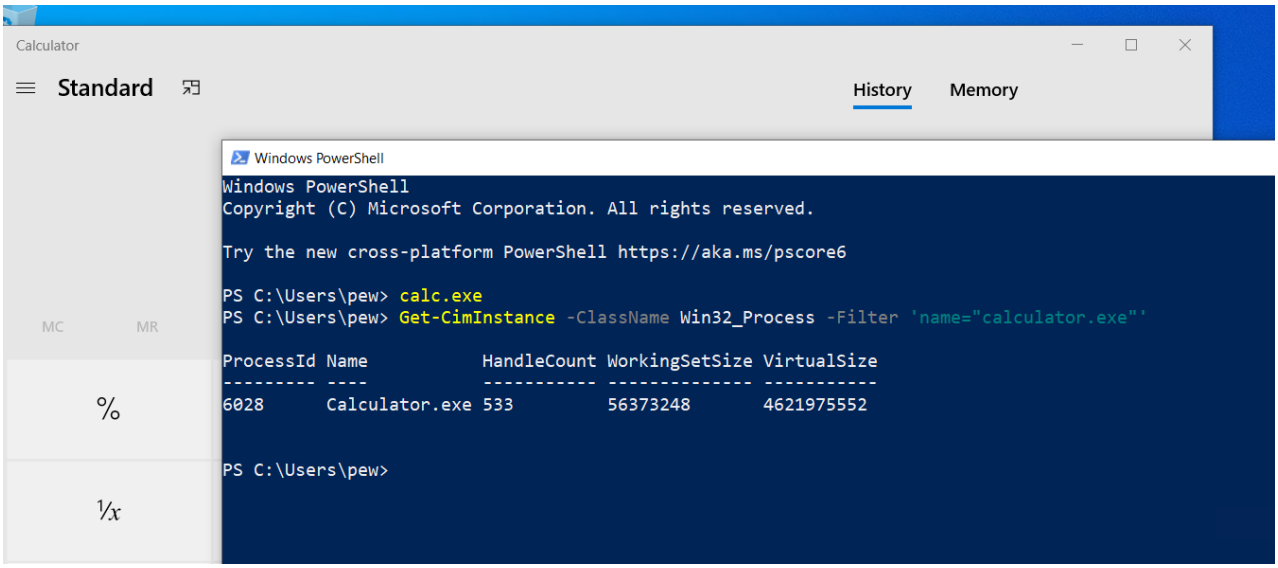
An idiomatic expression doing the same with WQL is as below:

```
Get-WmiObject -Query 'select * from win32_process where name="lsass.exe"'
```

Okay, now we know about listing, fetching and filtering instances of classes in WMI. Let's look at how removing instances works in WMI.

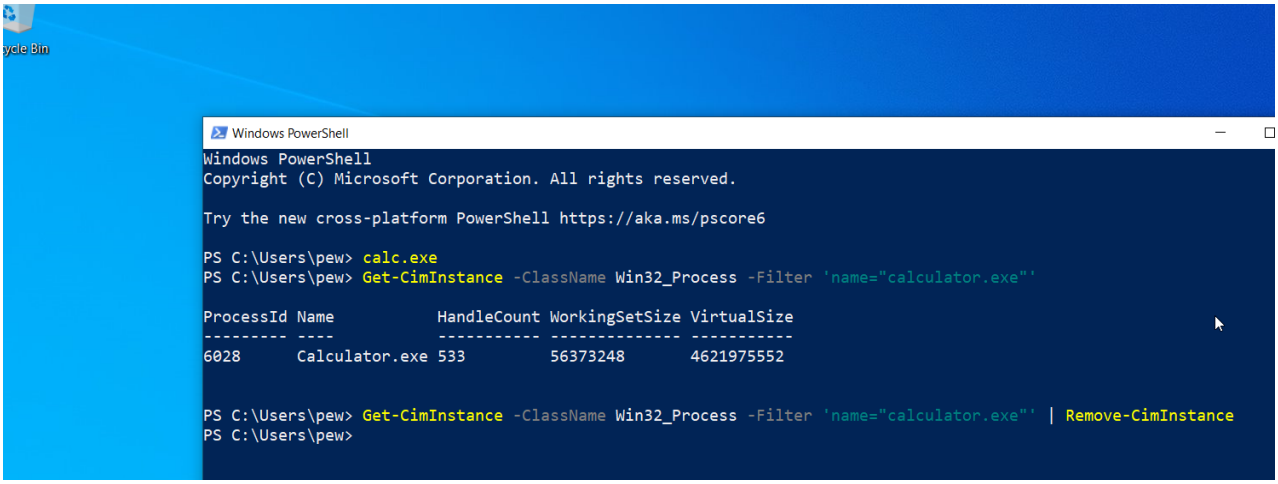
Removing Class Instances

The `Remove-WmiObject` (for WMI cmdlets) and `Remove-CimInstance` (for CIM cmdlets) are two cmdlets that have the capabilities of removing instances. You can pipe the output of a relevant command to the cmdlets. For a quick demo, let's run our favourite calculator app and list the process.



What happens if we pipe the command to `Remove-CimInstance`? The process gets killed!

```
Get-CimInstance -ClassName Win32_Process -Filter 'name="calculator.exe"' | Remove-CimInstance
```



This is extremely useful when messing around with Registry, or better, in a situation where we've created our own classes for storing our payloads and stuff – we can simply use the cmdlet to list all items under the class and thereby cleaning them up all in one go.

Methods

Methods are ways provided to manipulate WMI objects. If you scroll up to the place where we listed all the classes available, you'll notice a column called `Methods` which lists available methods.

Listing Methods

To repeat our chore and list all available methods, we can do something like:

```
Get-CimClass -MethodName *
```

To filter out instances that allow us to perform a specific method, we can pass a method name, for example, `Create` (which is always interesting because it might allow us to create something):

```
Get-CimClass -MethodName Create
```

```
PS C:\Users\pew> Get-CimClass -MethodName create


   NameSpace: ROOT/cimv2

CimClassName                        CimClassMethods      CimClassProperties
------------                        ---------------      ------------------
Win32_Process                       {Create, Terminat... {Caption, Description, InstallDate, Name...}
Win32_ScheduledJob                  {Create, Delete}     {Caption, Description, InstallDate, Name...}
Win32_DfsNode                       {Create}             {Caption, Description, InstallDate, Name...}
Win32_BaseService                   {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_SystemDriver                  {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_Service                       {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_TerminalService               {StartService, St... {Caption, Description, InstallDate, Name...}
Win32_Share                         {Create, SetShare... {Caption, Description, InstallDate, Name...}
Win32_ClusterShare                  {Create, SetShare... {Caption, Description, InstallDate, Name...}
Win32_ShadowCopy                    {Create, Revert}     {Caption, Description, InstallDate, Name...}
Win32_ShadowStorage                 {Create}             {AllocatedSpace, DiffVolume, MaxSpace, UsedSpace...}
```

Further narrowing things down, to list available methods for a specific class, we need to use Powershell's `select` with the `-ExpandProperty` switch:

With `Get-WmiObject`:

```
Get-WmiObject -Class Win32_Process -List | select -ExpandProperty Methods
```

With `Get-CimClass`:

```
Get-CimClass -ClassName Win32_Process | select -ExpandProperty CimClassMethods
```

```
PS C:\Users\pew> Get-WmiObject -Class Win32_Process -List | select -ExpandProperty Methods


Name           : Create
InParameters   : System.Management.ManagementBaseObject
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Process
Qualifiers     : {Constructor, Implemented, MappingStrings, Privileges...}

Name           : Terminate
InParameters   : System.Management.ManagementBaseObject
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Process
Qualifiers     : {Destructor, Implemented, MappingStrings, Privileges...}

Name           : GetOwner
InParameters   :
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Process
Qualifiers     : {Implemented, MappingStrings, ValueMap}

Name           : GetOwnerSid
InParameters   :
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Process
Qualifiers     : {Implemented, MappingStrings, ValueMap}

Name           : SetPriority
InParameters   : System.Management.ManagementBaseObject
OutParameters  : System.Management.ManagementBaseObject
Origin         : Win32_Process
Qualifiers     : {Implemented, MappingStrings, ValueMap}
```

> **NOTE**: Please note that the value passed to `select` statement is the name of the column which we got when listing the classes. If you're confused, scroll up to the paragraph where we listed a class and observe the output difference between WMI and CIM cmdlet output.

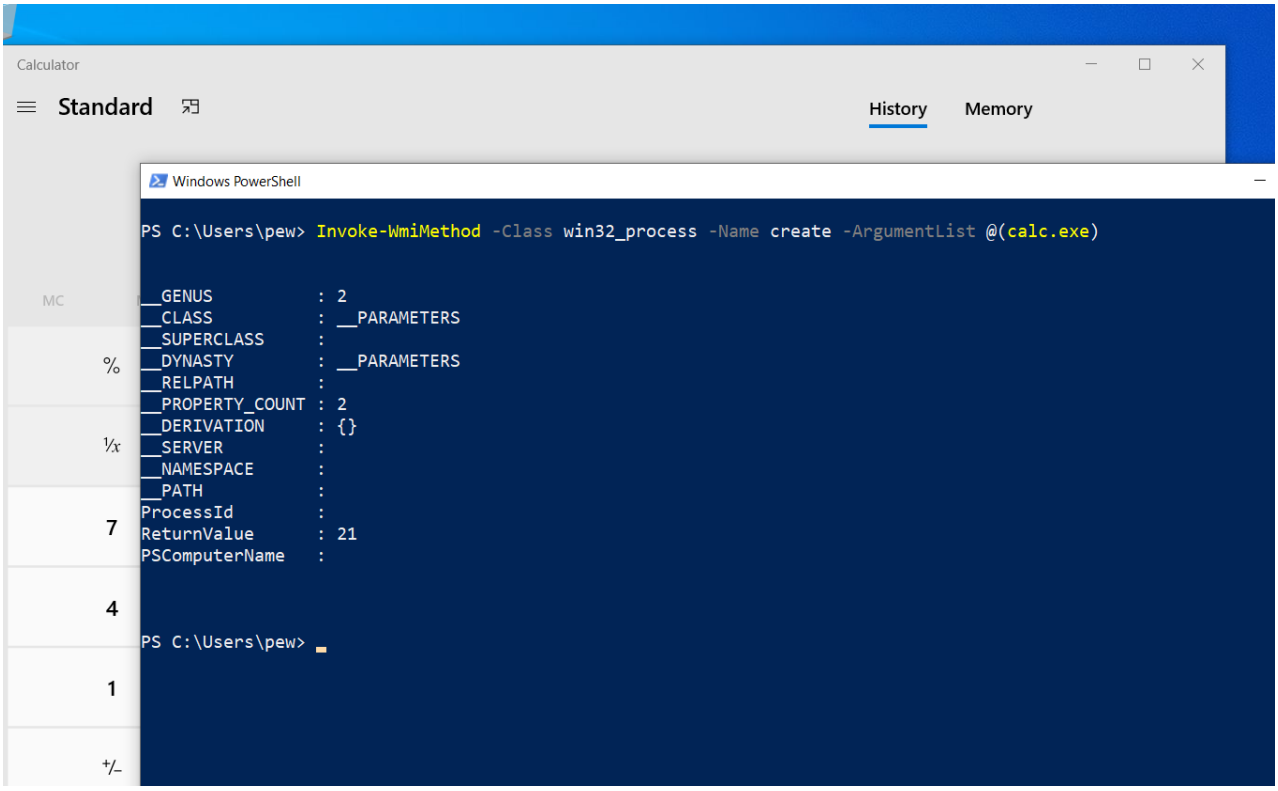
So, we have methods like `Create`, `Terminate`, `GetOwner`, `GetOwnerSid`, etc for the Win32_Process class. Great. Now let us see how we can use methods.

> **TIP**: To use a method, we need to know what parameters do we need to supply when calling the method. To list all available parameters, we can use a combination of Powershell or better just read the documentation.

Using Methods

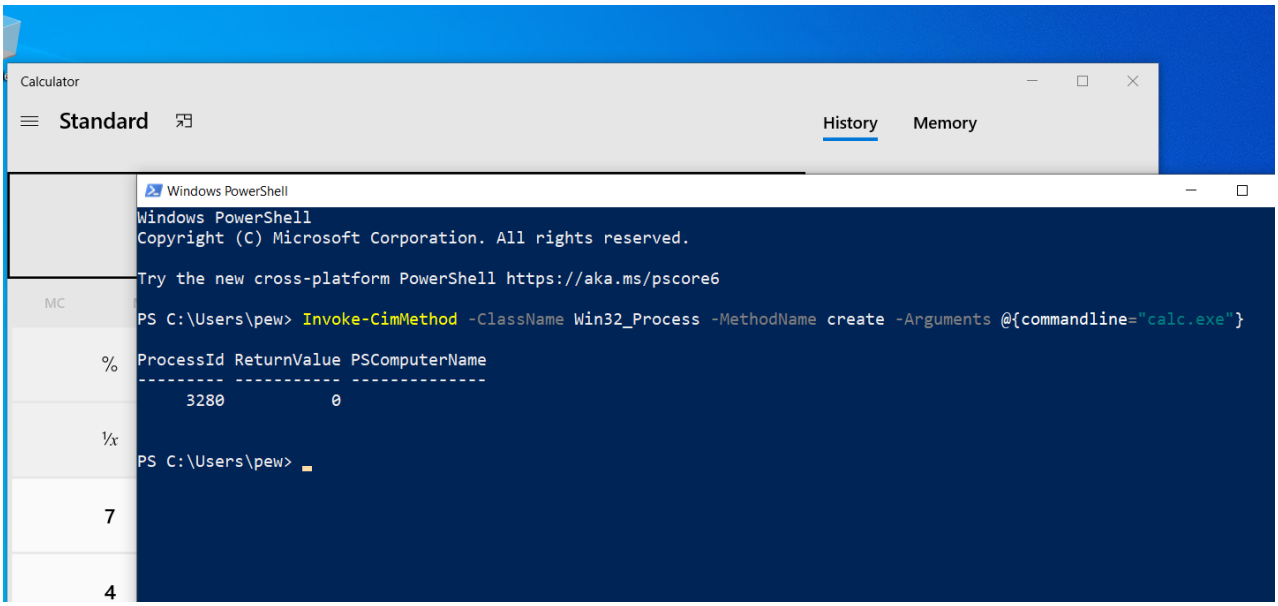
The `Invoke-WmiMethod` (for WMI) and `Invoke-CimMethod` (for CIM cmdlets) allows us to use the methods for a specific class. Let's try to spawn a calculator:

```
Invoke-WmiMethod -Class Win32_Process -Name Create -ArgumentList calc.exe
```



To use the CIM cmdlet, the syntax varies slightly:

```
Invoke-CimMethod -ClassName Win32_Process -MethodName create -Arguments
@{commandline="calc.exe"}
```



Well, now we know about spawning new processes!

Setting Properties of Objects

Last but not the least, we should take a look at updating instances of a class. However, it is important to keep in mind that the instance should be writable. With a bit of scripting, we can cook up a recipe for getting all writable properties of an class. Here's the script (sourced from PSMag):

```
$class = [wmiclass]'<class_name>'
$class.Properties | ForEach-Object {
      foreach ($qualifier in $_.Qualifiers) {
           if ($qualifier.Name -eq "Write") {
                $_.Name
           }
      }
}
```

For our example, we'll use the class `Win32_OperatingSystem`, which has a writable property called `Description` (essentially the description of the OS).

```
PS C:\Users\pew> Get-WmiObject -Class win32_operatingsystem | select description

description
-----------


PS C:\Users\pew>
```

Let us update the property name to `PewOS` using `Set-WmiInstance:

```
PS C:\> Get-WmiObject -Class Win32_operatingsystem | Set-WmiInstance -Arguments @{description="PewOS"}


SystemDirectory : C:\Windows\system32
Organization    :
BuildNumber     : 19043
RegisteredUser  : pew
SerialNumber    : 00329-20000-00001-AA627
Version         : 10.0.19043



PS C:\> Get-WmiObject -Class Win32_operatingsystem | select description

description
-----------
PewOS
```

The same could be achieved with `Set-CimInstance`, but that is left up to the reader to explore. :)

Conclusion

Whew, that was a long read! By now, we have a solid foundation of both the WMI and CIM cmdlets and how they can be used to achieve significant control over a system. So far thanks for being here, and I'll see you in the blog. Cheers! 🥂