Using cmdlet definition XML to create Windows PowerShell cmdlets

CDXML specification

March 16, 2012

Abstract

This paper provides information about using cmdlet definition XML (CDXML) to create Windows PowerShell cmdlets for Windows® "8" Consumer Preview and Windows Server® "8" Beta. It provides guidelines for developers who create Windows PowerShell cmdlets for WMI providers. It assumes that the reader has some familiarity with Windows PowerShell and WMI.

This information applies to the following operating systems:

Windows® "8" Consumer Preview

Windows Server® "8" Beta

Windows Server 2008 R2\*

Windows 7\*

Windows Server 2008\*

\*With [Windows Management Framework 3.0](http://www.microsoft.com/download/en/details.aspx?id=28998)

**Disclaimer**: This document is provided “as-is”. Information and views expressed in this document, including URL and other Internet website references, may change without notice.

Some information relates to pre-released product which may be substantially modified before it’s commercially released. Microsoft makes no warranties, express or implied, with respect to the information provided here. You bear the risk of using it.

Some examples depicted herein are provided for illustration only and are fictitious. No real association or connection is intended or should be inferred.

This document does not provide you with any legal rights to any intellectual property in any Microsoft product. You may copy and use this document for your internal, reference purposes.

© 2012 Microsoft. All rights reserved.



Contents

[Introduction 3](#_Toc319695848)

[Glossary 3](#_Toc319695849)

[References 4](#_Toc319695850)

[Overview 4](#_Toc319695851)

[Overview of a CIM class 4](#_Toc319695852)

[Overview of CIM operations 6](#_Toc319695853)

[Cmdlet definition using CDXML 7](#_Toc319695854)

[Scenarios 7](#_Toc319695855)

[Getting object instances through query and filtering 7](#_Toc319695856)

[Performing operations on instances 8](#_Toc319695857)

[Creating a new instance. 9](#_Toc319695858)

[Performing class operations 9](#_Toc319695859)

[Handling WhatIf and Confirm in cmdlets 10](#_Toc319695860)

[Working with computers that run on server software other than Windows Server 10](#_Toc319695861)

[High-level structure of cmdlet definition XML 11](#_Toc319695862)

[CDXML examples 12](#_Toc319695863)

[Getting object instances through query and filtering 12](#_Toc319695864)

[Performing operations on instances 15](#_Toc319695865)

[Creating a new instance 17](#_Toc319695866)

[Performing class operations 18](#_Toc319695867)

[Appendix A - cmdlets-over-objects.xsd 19](#_Toc319695868)

# Introduction

The “CDXML: Cmdlet Definition XML” document describes the file format and the semantics of CDXML files. A CDXML file defines the mapping between Windows PowerShell cmdlets and CIM class operations or methods. Windows PowerShell cmdlet developers use CDXML files to define cmdlets that call a CIM Object Manager (CIMOM) server (such as WMI in Windows) to manage the server.

Cmdlets defined by using CDXML communicate with remote CIM servers by using the WsMan protocol (implemented by the WinRM service in Windows). This enables management of end-points that don’t have Windows PowerShell installed on them; for example, a computer running Windows Server with Windows PowerShell remoting disabled, or a computer that is running on server software other than Windows Server; for example, a CIM server like NanoWbem.

## Glossary

The following terms are defined in MS-GlOS:

Common Information Model (CIM)

Common Information Model (CIM) class

Common Information Model (CIM) instance

Common Information Model (CIM) method

Common Information Model (CIM) namespace

Common Information Model (CIM) object

Common Information Model (CIM) path

Common Information Model (CIM) property

Common Information Model (CIM) relative path

The following terms are defined in MSDN Library documentation.

|  |  |
| --- | --- |
| Term | Description |
| Windows PowerShell | A task automation framework consisting of a [command-line](http://en.wikipedia.org/wiki/Command-line_interface) [shell](http://en.wikipedia.org/wiki/Shell_(computing)) and associated [scripting language](http://en.wikipedia.org/wiki/Scripting_language) built on top of, and integrated with, the [.NET Framework](http://en.wikipedia.org/wiki/.NET_Framework). |
| Cmdlet | A single-feature command that manipulates objects in Windows PowerShell. The names of cmdlets follow the verb-noun format, and there is a recommended set of common verbs. |
| Verb | The first element of the cmdlet name separated with a hyphen (-) from the second element. |
| Noun | The second element of the cmdlet name. |
| WMI | Windows Management Instrumentation (WMI) is the infrastructure for managing Windows operating system resources. WMI implements the DMTF management standards, CIM and WS-Man.  WMI is a Microsoft implementation of a CIM Object Manager (CIMOM). |
| CIM | Common Information Model (CIM) is the DMTF standard [DSP0004] for describing the structure and behavior of the managed resources such as storage, network, or software components. Declarations for CIM resources are encoded in one of two languages: Management Object Framework (MOF) or CIM-XML. |
| CIM provider | An executable that can return or set information about a given managed resource object, as described in [DMTF-DSP0004]. |
| WQL query | The WMI Query Language (WQL) is a subset of the ANSI Structured Query Language (ANSI SQL) that is used by WMI to query for managed resources. |
| WS-Man/WinRM | Windows Remote Management (WinRM) is the Microsoft implementation of [WS-Management (WS-Man) protocol](http://msdn.microsoft.com/en-us/library/aa384470(v=VS.85).aspx), a standard Simple Object Access Protocol (SOAP)-based, firewall-friendly protocol that allows hardware and operating systems, from different vendors, to interoperate. |

## References

* Introduction to Windows PowerShell
* Windows PowerShell SDK
* WMI provider development
* WMI glossary on MSDN: <http://msdn.microsoft.com/en-us/library/aa394561(v=VS.85).aspx>
* WMI Protocol documentation MS-WMI
* [MSDN-WQL] Microsoft Corporation, "Querying with WQL", <http://msdn.microsoft.com/en-us/library/aa392902.aspx>
* MS-GLOS
* DMTF Generic CIM Operations Spec: DSP0223
* PS module design guidelines

# Overview

A cmdlet is a task-oriented-abstraction of a management action performed on entities (managed resources) in the system. The verb part of the cmdlet name denotes the action to be performed, and the noun part of the cmdlet name denotes the entity on which the action is to be performed. A cmdlet in Windows PowerShell version 3.0 can be implemented in one of three ways:

1. As a .NET class that derives from the System.Management.Automation.Cmdlet class
2. As a Windows PowerShell script advanced function
3. As a CIM class method
4. As a XAML workflow

Before we discuss the structure and use of CDXML files, we need to introduce the core CIM class definition concepts because the CDXML elements refer to them.

## Overview of a CIM class

This section provides an overview of CIM class declaration concepts for declarations that are references in CDXML. For a more detailed explanation, see the WMI provider development documentation.

A CIM class is an object oriented abstraction of the entity being managed through a CIMOM infrastructure like WMI. A class can have zero or more properties and methods that are either defined in the class itself or in one of its parent classes. The CIM class methods can be grouped along two dimensions:

1. Intrinsic versus. extrinsic methods

Intrinsic methods, discussed in more detail later in this paper, are the predefined methods that each class should implement.

Extrinsic methods are user-defined methods that the class implements in addition to the intrinsic methods.

1. Instance versus. static methods

Instance methods are called on an instance of a managed entity and therefore, always get that instance as one of its arguments.

Static methods are called on the class and may or may not get a managed entity instance as an argument.

Class declarations are written in the MOF specification language. Below is a simplified example declaration of the Win32\_Process class that represents the Windows processes. The class has a number of Windows-specific properties in addition to the properties it inherits from the standard CIM\_Process class.

The class declaration also defines one static method called Create and four instance methods Terminate, GetOwner, GetOwnerSid, and SetPriority.

class Win32\_Process : CIM\_Process

{

[Override("KernelModeTime")] uint64 KernelModeTime = NULL;

unt32 Priority = NULL;

[Override("UserModeTime")] uint64 UserModeTime = NULL;

string ExecutablePath;

uint32 MaximumWorkingSetSize;

uint32 MinimumWorkingSetSize;

uint32 PageFaults;

PageFileUsage;

uint32 PeakPageFileUsage;

uint32 PeakWorkingSetSize;

uint32 ProcessId;

uint32 QuotaNonPagedPoolUsage;

uint32 QuotaPagedPoolUsage;

uint32 QuotaPeakNonPagedPoolUsage;

uint32 QuotaPeakPagedPoolUsage;

string WindowsVersion;

uint32 ThreadCount;

uint32 HandleCount;

uint32 ParentProcessId;

uint32 SessionId;

uint64 PrivatePageCount;

ToSubClass] uint64 PeakVirtualSize;

uint64 VirtualSize;

uint64 ReadOperationCount;

uint64 WriteOperationCount;

uint64 OtherOperationCount;

uint64 ReadTransferCount;

uint64 WriteTransferCount;

uint64 OtherTransferCount;

string CommandLine;

[Static] uint32 Create ([In] string CommandLine, [In] string CurrentDirectory, [In] Win32\_ProcessStartup ProcessStartupInformation, [Out] uint32 ProcessId);

uint32 Terminate ([In] uint32 Reason);

uint32 GetOwner ([Out] string User, [Out] string Domain);

uint32 GetOwnerSid ([Out] string Sid);

uint32 SetPriority ([In] sint32 Priority);

};

## Overview of CIM operations

The table below lists the operations that a client can perform on a CIM class. The first five operations (GetInstance, EnumerateInstance, CreateInstance, ModifyInstance, and DeleteInstance) are the intrinsic class methods, and the QueryInstance operation is like an enumeration with a filter.

The InvokeMethod operation is a generic mechanism to invoke the extrinsic methods by passing the method name and the method arguments in the invocation payload.

|  |  |
| --- | --- |
| Operation | Description |
| GetInstance | Gets a specific instance of a CIM class |
| EnumerateInstances | Enumerates all instances of a CIM class |
| QueryInstances | Returns the object instances that match a query.  WMI server only supports WQL queries. For more information about WQL, see [Querying with WQL](http://msdn.microsoft.com/en-us/library/aa392902.aspx) (<http://msdn.microsoft.com/en-us/library/aa392902.aspx>). |
| EnumerateAssociatedInstances | Enumerates instances related to a given instance through association. |
| CreateInstances | Creates a new instance of CIM class on the server. This method returns the newly created instance. |
| ModifyInstance | Modifies an instance that exists on server. If the instance does not exist, an error is returned. |
| DeleteInstance | Deletes an existing instance. |
| InvokeMethod | Invokes an extrinsic method on the class. This can be an instance or a static method. |

More details about the class methods can be found in the following locations:

* WMI v2 client API SDK documentation
* WinRM protocol documentation

In most cases, one Windows PowerShell cmdlet corresponds to one CIM class method. The following table shows a typical mapping between common Windows PowerShell verbs and CIM operations.

|  |  |
| --- | --- |
| Windows PowerShell verb | CIM operation |
| Get | GetInstance, EnumerateInstances, QueryInstances, EnumerateAssociatedInstances |
| Set, Update | ModifyInstance |
| Remove | DeleteInstance |
| New, Add | CreateInstance |
| Other verbs | Extrinsic methods (may be static or instance) |

Note: This table is only a recommendation. You can implement a cmdlet to call a static method for cmdlets like Get, Set, or Remove as well.

# Cmdlet definition using CDXML

A CDXML document contains mapping between one or more Windows PowerShell cmdlets and one CIM class. Typically, these cmdlets would have a common noun that corresponds to the class name, but it is possible to have multiple nouns mapped to the same CIM class.

A CDXML file can be imported into Windows PowerShell like any other module by using import-module cmdlet – for example:

Import-module .\Win32Process.cdxml

We recommend that you package Windows PowerShell modules by using a module manifest (.psd1). A module manifest may include various module types like .psm1, .dll or .cdxml. For more information, see the Windows PowerShell module manifest design guidelines.

## Scenarios

The following scenarios illustrate the structure of the CDXML documents.

### Getting object instances through query and filtering

In this scenario, a cmdlet retrieves zero, one, or many instances of a class. There are several ways to retrieve class instances from a CIM server:

* GetInstance
* QueryInstances
* EnumerateInstances.
* EnumerateAssociatedInstances

Currently CDXML does not support GetInstance for retrieving a particular instance. Server filtering can be done through QueryInstances; however, CDXML only supports WQL queries.

A GET cmdlet definition based on Enumerate and Query requires the following information:

* Metatdata that describes the cmdlet properties
  + Verb
  + Noun
  + HelpURI
* Cmdlet parameters for filtering the results
  + Cmdlet parameter Name and Type
  + Cmdlet parameter attributes (Mandatory, ValueFromPipeline, Position, and so on)
  + CIM class property used for filtering
  + Type of filter query expressions to be formed (for example, RegularQuery or MinQuery)
* For each parameter, the form of its query expression can be further specialized into one of the following types:
* RegularQuery—compares an instance property value with the corresponding cmdlet argument using either the “equal” or “like” operator.
* MinQuery—compares an instance property value with the corresponding cmdlet argument value using the “less than or equal” operator.
* MaxQuery—compares an instance property value with the corresponding cmdlet argument value using the “greater than or equal” operator.
* ExcludeQuery—compares an instance property value with the corresponding cmdlet argument value and filters out instances that match.

The information above applies to the GET cmdlet when working with “instance providers”; that is, a CIM provider that returns instances through QueryInstance or EnumerateInstances intrinsic operations. How cmdlets get data from a static CIM method is discussed in more detail later in this paper.

These CDXML elements are used to describe the information outlined above.

Cmdlet Metadata : <Cmdlet> and <CmdletMetadata>

Cmdlet Filterting Parameters : <GetCmdletParameters>

For more information about these elements, see the CDXML schema documentation.

### Performing operations on instances

In this scenario, a cmdlet that corresponds to an instance method, intrinsic or extrinsic, is applied to a set of instances of a class that have been retrieved by using a query constructed based on the query parameters.

A CIM query operation may return 0 or many instances matching the constructed query. After the instances are retrieved, the CIM Instance operation can be performed on those instances. The following table shows a typical mapping between cmdlet verbs and CIM operations.

|  |  |
| --- | --- |
| Windows PowerShell verb | CIM operation |
| Set, Update | ModifyInstance |
| Remove | DeleteInstance |
| Other approved verbs | Extrinsic Instance method operations |

A cmdlet definition based on a CIM Instance operation requires three pieces of information:

* Cmdlet Metadata element.
* Query Parameters element, which defines the properties and query expressions used to filter the instances
* Method elements, which define the mapping between the cmdlet parameters and the corresponding method. For Method elements, you'll also need the following information:
  + For both extrinsic and intrinsic methods—The name of the method (for intrinsic methods , use cim: as the prefix).
  + For extrinsic methods—Method parameters that are used as cmdlet parameters.
    - A method can have more parameters than are used by a cmdlet.
  + For extrinsic methods—Method output.

The following CDXML elements are used to describe the information outlined above:

Cmdlet Metadata : <Cmdlet> and <CmdletMetadata>

Cmdlet Filterting Parameters : <GetCmdletParameters>

Method invoked by cmdlet : <Method>

For more information about these elements, see the CDXML schema documentation.

### Creating a new instance.

A new class instance is typically created by using the CreateInstance intrinsic method.

To create a new class instance, you'll need the following information:

* The name of the Windows PowerShell verb (new or add)
* The name of the CIM provider method (CreateInstance or extrinsic static method)
* If the CIM provider requires certain properties (or method parameters in the case of an extrinsic method) to be filled in for a newly created instance, these must be mandatory cmdlet parameters.

It is possible to create new class instances by using an extrinsic class method (aka a Static method) as well. From the CDXML perspective, the CreateInstance intrinsic method is like a class method, as described below.

### Performing class operations

Class operations (aka Static methods in MOF) are different than instance operations and cmdlets mapped to static methods are defined in a different section than those mapped to instance methods. The two main differences are (i) static methods don’t have query parameters because they cannot be applied to instances, and (ii) it is possible to map a single cmdlet to multiple static methods by creating a separate parameter set for each method.

A cmdlet definition for invoking a static method has the following elements:

* Cmdlet metadata element, which is the same as in the previous scenarios.
* Method element. For Method elements, you'll also need the following information:
  + The name of the method.
  + The method parameters. Each parameter appears as a cmdlet parameter.
    - A method can have more parameters than are used by a cmdlet.
  + The method output.

A separate static method can be called for each parameter set of the cmdlet. This is not possible with cmdlets that are mapped to instance methods.

These CDXML elements are used to describe the information outlined above:

Cmdlet Metatadata: <Cmdlet> and <CmdletMetadata>

Method Signature: <Method>

### Handling WhatIf and Confirm in cmdlets

A Windows PowerShell cmdlet that changes the system state must support WhatIf and Confirm parameters. A cmdlet run with WhatIf will tell the user what changes this cmdlet will make. Similarly, A cmdlet run with the confirm parameter will generate a confirmation prompt for the user before making the change. A full description of Windows PowerShell prompting behavior is available on MSDN Library.

A key concern for implementing WMI class WhatIf orconfirm semantics in Windows PowerShell is whether the target WMI provider natively supports this behavior. A provider written using the new MI provider API may call MI\_ShouldProcess to prompt the user before making changes. If this is the case, the cmdlet defined in CDXML just needs to declare that it supports Whatif or confirm. For WMI providers that don’t support MI\_ShouldProcess natively (for example, all legacy providers under root\cimv2 namespace), CDXML will also declare that WhatIf or confirm should be handled locally in the cmdlet, without relying on the WMI provider.

### Working with computers that run on server software other than Windows Server

There are some subtle differences between CIM Object Manager (CIMOM) servers that impact the user experience. A developer who is using CDXML to create cmdlets that manage a server running on server software other than Windows Server should be aware of the server configuration and capabilities.

* Query capability—If the server does not support server filtering using WQL then EnumerateInstance operations can be used instead of QueryInstances. In this case, the filtering happens only on the client.
* SchemaRetrievalSupport—Windows Server "8" Beta implements class schema retrieval (EnumerateClasses and GetClass) operations that provide a rich client experience. Not all end-points support this functionality.
* Handling ShouldProcess (Whatif or confirm)—This should be implemented on the client instead of relying on a provider to do it. This is also applicable to WMI providers that have not implemented MI\_ShouldProcess (for example, all legacy WMI providers in the root/cimv2 namespace).
* Changing the namespace at cmdlet runtime.

These CDXML elements are used to describe the information outlined above:

<CmdletAdapterPrivateData>

## 

## High-level structure of cmdlet definition XML

The structure of a CDXML document is shown in the table below. The document is divided into two major sections (elements):

* One for defining cmdlets that map to instance methods (the InstanceCmdlets element), and
* One for defining cmdlets that map to static methods (the StaticCmdlets element).

The instance cmdlets element is further divided into these sections:

* Definition of parameters that are used to retrieve class instances that will be passed to instance methods (the GetCmdletParameters element)
* Definition of the Get-<noun> cmdlet (the GetCmdlet element), and
* A sequence of instance cmdlet mappings (the Method element).

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CDXML element | | | | | | | | Description | | |
| [Windows PowerShellMetadata](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_PowerShellMetadata) | | | | | | | | The root element for a CDXML document. | | |
|  | [Class](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_Class) | | | | | | | Describes the CIM class. Class will have the full path of CIM Class (Namespace+ClassName). | | |
|  |  | [InstanceCmdlets](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_InstanceCmdlets) | | | | | | Describes the cmdlets mapped to instance operations. | | |
|  |  |  | [GetCmdletParameters](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_GetCmdletParameters) | | | | | A global set of query parameters used by all instance cmdlets. The cmdlet can define its own set of query parameters. | | |
|  |  |  | [GetCmdlet](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_GetCmdlet) | | | | | Describes the metadata for Get Cmdlet.  Get Cmdlet is treated differently than other instance cmdlets.  This is optional; it is used only when overriding. | | |
|  |  |  | [Cmdlet](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_Cmdlet) | | | | | One <cmdlet> node that corresponds to each cmdlet defined by this CDXML. | | |
|  |  |  |  | | CmdletMetadata | | | Describes the metadata for the cmdlet. This includes Noun and Verb for the cmdlet, as well as aliases and flags for specifying ConfirmImpact Settting. | | |
|  |  |  |  | | Method | | | The name of the CIM Instance method invoked by the cmdlet. | | |
|  |  |  |  | |  | Parameters | | | Describes the parameters of the method. | |
|  |  |  |  | |  |  | Parameter | | | Individual parameters of the method. This defines the name and type of the parameter and Windows PowerShell-specific attributes like Mandatory, PipeLine binding, and so on. |
|  |  |  |  | | GetCmdletParameters | | | Specified when a cmdlet replaces the global definition. | | |
|  |  | [StaticCmdlets](file:///C:\Users\osajid\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\OX1PLT2Q\CDXML%20-%20Cmdlet%20Definition%20XML%200%204.docx#_StaticCmdlets) | | | | | | Describes cmdlets mapped to a static method. | | |
|  |  |  | Cmdlet | | | | |  | | |
|  |  |  |  | CmdletMetadata | | | |  | | |
|  |  |  |  | Method | | | | Describes static methods called by the cmdlet for a static method. There can be one method per parameter set. | | |
|  |  |  |  | |  | Parameters | | | The collection of parameters for this method. | |
|  |  |  |  | |  |  | Parameter | | | Describes one parameter of the method and how it appears in the cmdlet signature. |
|  |  | CmdletAdapterPrivateData | | | | | | | | Used to override the default behavior for CDXML processing. |
|  | Enums | | | | | | | | | Defines the enumerations to be used by cmdlets. |

## CDXML examples

This section describes the detailed structure of the CDXML document by showing the content of the CDXML documents that correspond to the scenarios introduced earlier in this paper.

### Getting object instances through query and filtering

#### Example 1: Simple cmdlet to get instances of Win32\_Process

|  |
| --- |
| Get-Win32Process [[-Name] <string[]>]  Get-Win32Process -ProcessId <UInt32[]> |

For the cmdlet signature above, we have the following information:

* The noun for the cmdlet is “Win32Process”.
* The cmdlet is only querying for instances (it’s a Get cmdlet).
* It’s a regular query expression, where Win32\_Process instances that match an ID or Name are returned.
* The Name parameter is a string; it accepts wildcards and is optional.
* The default ParamaterSet is ByName

|  |
| --- |
| <PowerShellMetadata xmlns="http://schemas.microsoft.com/cmdlets-over-objects/2009/11">  <!--  ClassName attribute defines the CIM Class in namespace/class format.  In this example, root/cimv2 is the namespace and Win32\_process is the WMI class.  -->  <Class ClassName="root/cimv2/Win32\_Process">  <Version>2.0.0.0</Version> <DefaultNoun>Win32Process</DefaultNoun>  <InstanceCmdlets>  <!--  Global definition for query parameters. Not used in this example because we have a single cmdlet only.  -->  <GetCmdletParameters />  <!--  Definition of Get-Process cmdlet  -->    <GetCmdlet>  <CmdletMetadata Verb="Get" HelpUri="http://link.to.online.help" />  <GetCmdletParameters DefaultCmdletParameterSet="ByName">  <QueryableProperties>  <Property PropertyName="Name">  <Type PSType="string" />  <RegularQuery AllowGlobbing="true">  <CmdletParameterMetadata IsMandatory="false" Position="0"  ValueFromPipelineByPropertyName="true" CmdletParameterSets="ByName" />  </RegularQuery>  </Property>  <Property PropertyName="ProcessId">  <Type PSType="uint32" />  <RegularQuery>  <CmdletParameterMetadata IsMandatory="true" Aliases="ID PID"  CmdletParameterSets="ById" />  </RegularQuery>  </Property>  </QueryableProperties>  </GetCmdletParameters>  </GetCmdlet>  </InstanceCmdlets  </Class>  </PowerShellMetadata> |

Using this sample CDXML in Windows PowerShell

#Save the sample file as CDXML (for example, Win32Process.cdxml)#Import this file in PS

Import-module .\win32Process.cdxml

#check the imported cmdlets

Get-command –module winprocess\* -syntax

#run the cmdlet

Get-Win32Process

Get-Win32Process –id 0

Get-Win32Process –id 0,10,11

Get-Win32Process –Name c\*

Get-Win32Process –Name a\*,csrss.exe

#### Example 2: Get cmdlet with different query types

|  |
| --- |
| Get-Win32Process [[-Name] <string[]>] [-ExcludeName <string[]>] [-MinWorkingSet <UInt64>] [-MaxWorkingSet <UInt64>]  Get-Win32Process -ProcessId <UInt32[]> |

|  |
| --- |
| <PowerShellMetadata xmlns="http://schemas.microsoft.com/cmdlets-over-objects/2009/11">  <Class ClassName="root\cimv2\Win32\_Process">  <Version>2.0.0.0</Version>  <DefaultNoun>Win32Process</DefaultNoun>  <InstanceCmdlets>  <GetCmdletParameters />  <GetCmdlet>  <CmdletMetadata Verb="Get" HelpUri="http://link.to.online.help" />  <GetCmdletParameters DefaultCmdletParameterSet="ByName">  <QueryableProperties>  <Property PropertyName="Name">  <Type PSType="string" />  <RegularQuery AllowGlobbing="true">  <CmdletParameterMetadata IsMandatory="false" Position="0" ValueFromPipelineByPropertyName="true" CmdletParameterSets="ByName" />  </RegularQuery>  <ExcludeQuery AllowGlobbing="true">  <CmdletParameterMetadata PSName="ExcludeName" CmdletParameterSets="ByName" />  </ExcludeQuery>  </Property>  <Property PropertyName="ProcessId">  <Type PSType="uint32" />  <RegularQuery>  <CmdletParameterMetadata IsMandatory="true" Aliases="ID PID"  CmdletParameterSets="ById" />  </RegularQuery>  </Property>  <Property PropertyName="WorkingSetSize">  <Type PSType="Uint64" />  <MinValueQuery>  <CmdletParameterMetadata PSName="MinWorkingSet" CmdletParameterSets="ByName" />  </MinValueQuery>  <MaxValueQuery>  <CmdletParameterMetadata PSName="MaxWorkingSet" CmdletParameterSets="ByName" />  </MaxValueQuery>  </Property>  </QueryableProperties>  </GetCmdletParameters>  </GetCmdlet>  </InstanceCmdlets>  </Class>  </PowerShellMetadata> |

Using this sample CDXML in Windows PowerShell

#Save the sample file as CDXML (for example, Win32Process.cdxml)

#Import this file in PS

Import-module .\win32Process.cdxml

#check the imported cmdlets

Get-command –module winprocess\* -syntax

#run the cmdlet

Get-Win32Process

Get-Win32Process –id 0

Get-Win32Process –id 0,10,11

Get-Win32Process –Name c\*

Get-Win32Process –Name a\*,csrss.exe

Get-Win32Process –Name c\* -ExcludeName csrss\*

Get-Win32Process -MinWorkingSet 2MB -MaxWorkingSet 5MB

#### Example 3: Getting associated objects

This example shows how we can get objects that are “associated” to a given object. We want to get a list of processes that are using a given file. In CIM, the relationship between Process and File is represented through an “association” class, CIM\_ProcessExecutable.

|  |
| --- |
| Get-Win32Process –ProcessFile <CimInstance#CIM\_DataFile> |

Here we want to find the list of processes for a given logon session (instance of Win32\_LogonSession class). The Name and Id parameters have been removed to keep the example simple.

|  |
| --- |
| <PowerShellMetadata xmlns="http://schemas.microsoft.com/cmdlets-over-objects/2009/11">  <Class ClassName="root\cimv2\Win32\_Process">  <Version>2.0.0.0</Version>  <DefaultNoun>Win32Process</DefaultNoun>  <InstanceCmdlets>  <GetCmdletParameters />  <GetCmdlet>  <CmdletMetadata Verb="Get" HelpUri="http://link.to.online.help" />  <GetCmdletParameters DefaultCmdletParameterSet="ByName">  <QueryableAssociations>  <Association Association="CIM\_ProcessExecutable" ResultRole="Dependent" SourceRole="Antecedent">  <AssociatedInstance>  <Type PSType="Microsoft.Management.Infrastructure.CimInstance" ETSType="Microsoft.Management.Infrastructure.CimInstance#root\cimv2\CIM\_DataFile" />  <CmdletParameterMetadata PSName="ProcessFile" ValueFromPipeline="true" CmdletParameterSets="ByName ByLiteralName" />  </AssociatedInstance>  </Association>  </QueryableAssociations>  </GetCmdletParameters>  </GetCmdlet>  </InstanceCmdlets>  </Class>  </PowerShellMetadata> |

Using this sample CDXML in Windows PowerShell

#Save the sample file as CDXML (for example, Win32Process.cdxml)

#Import this file in PS

Import-module .\win32Process.cdxml

#check the imported cmdlets

Get-command –module win32process\* -syntax

#run the cmdlet

Get-Win32Process

#Create a local instance of CIM\_File class to pass to Get-Win32Process

$file = New-CimInstance -ClassName CIM\_DataFile -Namespace root/cimv2 -Property @{Name="C:\Windows\System32\wininet.dll"} -Key Name -ClientOnly

#NOTE: In most of the cases, there would be another cmdlet to get instances of CIM\_DataFile

Get-Win32Process –ProcessFile $file

### Performing operations on instances

In this section, we will see examples of CDXML where cmdlet invokes an instance method.

|  |
| --- |
| Get-Win32Process [[-Name] <string[]>]  Get-Win32Process -ProcessId <UInt32[]>  Stop-Win32Process [-Name] <string[]> [[-Reason] <UInt32>] [-WhatIf] [-Confirm]  Stop-Win32Process -ProcessId <UInt32[]> [[-Reason] <UInt32>] [-WhatIf] [-Confirm]  Stop-Win32Process -InputObject <CimInstance#Win32\_Process[]>[[-Reason] <UInt32>] [-WhatIf] [-Confirm]  Debug-Win32Process [-Name] <string[]>  Debug-Win32Process -ProcessId <UInt32[]>  Debug-Win32Process -InputObject |

Key points to note:

1. Name and ProcessId are the parameters used for filtering the instances of Win32\_Process.
2. There is a slight difference in the usage of the Name parameter. It is optional in Get-Win32Process but mandatory in Stop-Win32Process and Debug-Win32Process.

As a general Windows PowerShell convention, there should not be any mandatory parameter in the default parameter set for a Get cmdlet. However, we want to make it mandatory in other cmdlets . It's common to have a set of cmdlets follow a similar pattern for query parameters. For example, look at the syntax of the Windows PowerShell Get-Service , Set-Service, Start-Service, and Stop-Service cmdlets; the parameters for Set, Start, and Stop are the same. To minimize duplication of common parameters, CDXML allows you to define a global set of Query Parameters that is shared by all instance cmdlets. Individual cmdlets can define their own query parameters that override the global definition. It is very likely that a Get cmdlet would have slightly different parameter attributes because its default parameter set may not have any required parameters.

1. Stop-Win32Process supports whatif and confirm. As per Windows PowerShell guidelines, any cmdlet that changes the state of the system should support WhatIf or confirm. Because the Win32\_Process WMI class does not natively support ShouldProcess, Whatif or confirm must be handled on the client.

Note: If you are a provider developer, the new MI Provider development API natively supports ShouldProcess. For more information, see the MI API reference.

|  |
| --- |
| <PowerShellMetadata xmlns="http://schemas.microsoft.com/cmdlets-over-objects/2009/11">  <Class ClassName="root\cimv2\Win32\_Process">  <Version>2.0.0.0</Version>  <DefaultNoun>Win32Process</DefaultNoun>  <InstanceCmdlets>  <!--  Global definition for query parameters used by Stop-win32Process and Get-Win32ProcessOwner cmdlets  -->  <GetCmdletParameters DefaultCmdletParameterSet="ByName">  <QueryableProperties>  <Property PropertyName="Name">  <Type PSType="string" />  <RegularQuery AllowGlobbing="true">  <CmdletParameterMetadata IsMandatory="true" Position="0" ValueFromPipelineByPropertyName="true" CmdletParameterSets="ByName" />  </RegularQuery>  </Property>  <Property PropertyName="ProcessId">  <Type PSType="uint32" />  <RegularQuery>  <CmdletParameterMetadata IsMandatory="true" Aliases="ID PID"  CmdletParameterSets="ById" />  </RegularQuery>  </Property>  </QueryableProperties>  </GetCmdletParameters>  <GetCmdlet>  <CmdletMetadata Verb="Get"/>  <!--  Definition of query parameters used by Get-win32Process  It does not use the global definition because the Get cmdlet should not have mandatory parameters in the default parameter set  -->  <GetCmdletParameters DefaultCmdletParameterSet="ByName">  <QueryableProperties>  <Property PropertyName="Name">  <Type PSType="string" />  <RegularQuery AllowGlobbing="true">  <CmdletParameterMetadata IsMandatory="false" Position="0" ValueFromPipelineByPropertyName="true" CmdletParameterSets="ByName" />  </RegularQuery>  </Property>  <Property PropertyName="ProcessId">  <Type PSType="uint32" />  <RegularQuery>  <CmdletParameterMetadata IsMandatory="true" Aliases="ID PID"  CmdletParameterSets="ById" />  </RegularQuery>  </Property>  </QueryableProperties>  </GetCmdletParameters>  </GetCmdlet>  <Cmdlet>  <CmdletMetadata Verb="Stop" ConfirmImpact="Medium" />  <Method MethodName="Terminate">  <Parameters>  <Parameter ParameterName="Reason">  <Type PSType="Uint32" />  <CmdletParameterMetadata IsMandatory="false" Position="1" />  </Parameter>  </Parameters>  </Method>  </Cmdlet>  <Cmdlet>  <CmdletMetadata Verb="Debug" />  <Method MethodName="AttachDebugger">  <ReturnValue>  <Type PSType="int" />  <CmdletOutputMetadata>  <ErrorCode />  </CmdletOutputMetadata>  </ReturnValue>  </Method>  </Cmdlet>    </InstanceCmdlets>  <CmdletAdapterPrivateData>  <Data Name="ClientSideShouldProcess" />  </CmdletAdapterPrivateData>  </Class>  </PowerShellMetadata> |

### Creating a new instance

There are two methods that are commonly used to create an instance:

* CreateInstance intrinsic method
* Static method

This section provides an example of the CreateInstance intrinsic method. An example of a static method is covered in the next section.

This example uses the Win32\_Environment class, defined in the root\cimv2 namespace, because it implements the CreateInstance method.

|  |
| --- |
| New-EnvironmentVariable -Name <string> -UserName <string> -VariableValue <string> |

|  |
| --- |
| <PowerShellMetadata xmlns="http://schemas.microsoft.com/cmdlets-over-objects/2009/11">  <Class ClassName="ROOT\cimv2\Win32\_Environment">  <Version>1.0.0.0</Version>  <DefaultNoun>EnvironmentVariable</DefaultNoun>  <StaticCmdlets>  <Cmdlet>  <CmdletMetadata Verb="New" />  <Method MethodName="cim:CreateInstance">  <Parameters>  <Parameter ParameterName="Name">  <Type PSType="System.String"/>  <CmdletParameterMetadata IsMandatory="true"/>  </Parameter>  <Parameter ParameterName="UserName">  <Type PSType="System.String"/>  <CmdletParameterMetadata IsMandatory="true"/>  </Parameter>  <Parameter ParameterName="VariableValue">  <Type PSType="System.String"/>  <CmdletParameterMetadata IsMandatory="true"/>  </Parameter>  </Parameters>  </Method>  </Cmdlet>  </StaticCmdlets>  </Class>  </PowerShellMetadata> |

Note: No instance cmdlets have been defined in this example to keep the CDXML short. It is okay to have instance and static definitions in the same CDXML (for example, we can define Get or Set cmdlets in the same CDXML).

### Performing class operations

|  |
| --- |
| Start-Win32Process -Path <string> [-WorkingDirectory <string>] |

|  |
| --- |
| <PowerShellMetadata xmlns="http://schemas.microsoft.com/cmdlets-over-objects/2009/11">  <Class ClassName="root\cimv2\Win32\_Process">  <Version>2.0.0.0</Version>  <DefaultNoun>Win32Process</DefaultNoun>  <StaticCmdlets>  <Cmdlet>  <CmdletMetadata Verb="Start" />  <Method MethodName="Create">  <Parameters>  <Parameter ParameterName="CommandLine">  <Type PSType="string" />  <CmdletParameterMetadata IsMandatory="true" Position="0" />  </Parameter>  <Parameter ParameterName="CurrentDirectory">  <Type PSType="string" />  <CmdletParameterMetadata IsMandatory="false" Position="1" />  </Parameter>  <Parameter ParameterName="ProcessId">  <Type PSType="int" />  <CmdletOutputMetadata />  </Parameter>  </Parameters>  </Method>  </Cmdlet>  </StaticCmdlets>  </Class>  </PowerShellMetadata> |

Note: No instance cmdlets have been defined in this example to keep the CDXML short. It is okay to have instance and static definitions in the same CDXML (for example, we can define Get or Stop-Win32Process cmdlets in the same CDXML).

# Appendix A - cmdlets-over-objects.xsd

<?xml version="1.0" encoding="utf-8"?>

<!-- ##################################################################

(c) Microsoft. All rights reserved.

################################################################### -->

<!DOCTYPE schema [

<!ENTITY csharpIdentifierLetterCharacterRegex "\p{Lu}\p{Ll}\p{Lt}\p{Lm}\p{Lo}\p{Nl}">

<!ENTITY csharpIdentifierFirstCharacterRegex "&csharpIdentifierLetterCharacterRegex;\_">

<!ENTITY csharpIdentifierOtherCharacterRegex "&csharpIdentifierLetterCharacterRegex;\p{Nd}\_">

<!ENTITY csharpIdentifierRegex "[&csharpIdentifierFirstCharacterRegex;][&csharpIdentifierOtherCharacterRegex;]{0,100}">

<!ENTITY csharpNamespaceRegex "(&csharpIdentifierRegex;)(\.&csharpIdentifierRegex;){0,10}">

<!ENTITY clrTypeSpecialCharacterRegex "\,\=\ \+\&amp;\\*\[\]\.\\\-\&quot;">

<!ENTITY clrTypeRegex "[&csharpIdentifierOtherCharacterRegex;&clrTypeSpecialCharacterRegex;]{1,1000}">

<!ENTITY powerShellVerbCharacterRegex "\p{Ll}\p{Lu}\p{Lt}\p{Lo}\p{Lm}">

<!ENTITY powerShellNounCharacterRegex "&powerShellVerbCharacterRegex;\p{Nd}">

<!ENTITY powerShellAliasCharacterRegex "&powerShellNounCharacterRegex;\_-">

<!ENTITY powerShellCmdletParameterCharacterRegex "&powerShellNounCharacterRegex;\_">

<!ENTITY powerShellVerbRegex "[&powerShellVerbCharacterRegex;]{1,100}">

<!ENTITY powerShellNounRegex "[&powerShellNounCharacterRegex;]{1,100}">

<!ENTITY powerShellAliasRegex "[&powerShellAliasCharacterRegex;]{1,100}">

<!ENTITY powerShellCmdletParameterRegex "[&powerShellCmdletParameterCharacterRegex;]{1,100}">

<!ENTITY powerShellParameterSetRegex "[^ ,;]{1,100}">

]>

<!--

csharpIdentifierRegex based on C# Language Specification 3.0, section 2.4.2 Idenfifiers

- the C# spec allows \p{Pc}\p{Cf}\p{Mn}\{Mc} in csharpIdentifierOtherCharacterRegex,

but those characters classes seem risky wrt code injection attacks, so they were

removed above.

- '\_' (ASCII 95) is the only character in \p{Pc} class out of the first 0..255 characters

- No characters in the first 0..255 characters belong to \p{Cf}, \p{Mn}, \p{Mc}

clrTypeRegex and clrTypeSpecialCharacterRegex are based on http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

clrTypeSpecialCharacterRegex includes the following characters;

- ',' - part of ReflectionDimension and TypeName productions in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '=' - part of AssemblyProperty production in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- ' ' - unspecified in the docs, but apparently allowed (i.e. between NamespaceTypeName and AssemblyNameSpec in the TypeName production

- '+' - part of NestedTypeName production in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '&amp;' - part of ReferenceTypeSpec production in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '\*' - part of ReflectionDimension production in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '[' and ']' - part of ArrayTypeSpec production in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '.' - part of ReflectionDimension and NamespaceTypeName productions in BNF at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '\' - an escape character specified at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx

- '-' - required for Culture property of an Assembly.

RFC-1766 defines the format of a culture identifier as:

Language-Tag = Primary-tag \*( "-" Subtag )

Primary-tag = 1\*8ALPHA

Subtag = 1\*8ALPHA

- '&quot;' - unspecified in the docs, but used in the examples at http://msdn.microsoft.com/en-us/library/yfsftwz6.aspx (i.e. Culture="")

-->

<xsd:schema

elementFormDefault="qualified"

xmlns:xsd="http://www.w3.org/2001/XMLSchema"

xmlns:cps="http://schemas.microsoft.com/cmdlets-over-objects/2009/11"

targetNamespace="http://schemas.microsoft.com/cmdlets-over-objects/2009/11"

>

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

This schema defines the format of Windows PowerShell CIM Modules.

A Windows PowerShell CIM Module defines a set of cmdlets that interact with a CIM class.

A Windows PowerShell CIM Module needs to be saved in a file with ".cdxml" extension.

A ".cdxml" file can be imported into a Windows PowerShell session directly by Import-Module cmdlet,

or by referring to the ".cdxml" file from NestedModules or RootModule entry of

a module manifest (a ".psd1" file) and then importing the ".psd1" file by Import-Module cmdlet.

</xsd:documentation>

</xsd:annotation>

<xsd:element name="PowerShellMetadata">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Class" type="cps:ClassMetadata" />

<xsd:element name="Enums" type="cps:EnumMetadata" minOccurs="0" />

</xsd:sequence>

</xsd:complexType>

<xsd:key name="UniqueNamesOfEnums">

<xsd:selector xpath=".//cps:Enums/cps:Enum" />

<xsd:field xpath="@EnumName" />

</xsd:key>

<xsd:unique name="UniqueCmdletAliases">

<xsd:selector xpath=".//cps:CmdletMetadata" />

<xsd:field xpath="@Aliases" />

</xsd:unique>

</xsd:element>

<!-- ############################################################ -->

<!-- # ENUM ##################################################### -->

<xsd:complexType name="EnumMetadata">

<xsd:sequence>

<xsd:element name="Enum" maxOccurs="unbounded">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Value" maxOccurs="unbounded">

<xsd:complexType>

<xsd:attribute name="Name" type="cps:EnumValueName" use="required" />

<xsd:attribute name="Value" type="xsd:integer" use="required" />

</xsd:complexType>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="EnumName" type="cps:EnumTypeName" use="required">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

EnumName attribute specifies the name of a .NET enum.

This is the name to use in a PSType attribute.

The name should include a namespace to avoid naming conflicts

(i.e. the name should be "Networking.MyEnum" rather than "MyEnum").

The system will prefix the name of the enum with the following namespace: "Microsoft.PowerShell.Cmdletization.GeneratedTypes"

(i.e. "Networking.MyEnum" will become "Microsoft.PowerShell.Cmdletization.GeneratedTypes.Networking.MyEnum").

When referring to the enum in types.ps1xml and format.ps1xml files, one has to use the full, prefixed name of the enum.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="UnderlyingType" type="cps:PowerShellTypeIdentifier" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Underlying type of the enum.

C# Language Specification allows (in section 4.1.9 "Enumeration types") only the following

underlying types:

byte (System.Byte),

sbyte (System.SByte),

short (System.Int16),

ushort (System.UInt16),

int (System.Int32),

uint (System.UInt32),

long (System.Int64),

ulong (System.UInt64).

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="BitwiseFlags" type="xsd:boolean" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

BitwiseFlags attribute specifies if the .NET enum will be decorated with a System.FlagsAttribute.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<xsd:key name="UniqueNamesOfEnumValues">

<xsd:selector xpath=".//cps:Value" />

<xsd:field xpath="@Name" />

</xsd:key>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<!-- ############################################################ -->

<!-- # CLASS #################################################### -->

<xsd:complexType name="ClassMetadata">

<xsd:sequence>

<xsd:element name="Version" type="cps:VersionString">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Version element is semantically equivalent to the ModuleVersion entry in a module manifest (psd1) file.

</xsd:documentation>

</xsd:annotation>

</xsd:element>

<xsd:element name="DefaultNoun" type="cps:Noun">

<xsd:annotation>

<xsd:documentation>

DefaultNoun element specified the default noun for cmdlets defined in this document.

</xsd:documentation>

</xsd:annotation>

</xsd:element>

<xsd:element name="InstanceCmdlets" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="GetCmdletParameters" type="cps:GetCmdletParameters">

<xsd:unique name="UniqueCmdletParameterNamesUnderGetCmdletParametersElement\_1">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@PSName" />

</xsd:unique>

<xsd:unique name="UniqueCmdletParameterAliasesUnderGetCmdletParametersElement\_1">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@Aliases" />

</xsd:unique>

</xsd:element>

<xsd:element name="GetCmdlet" type="cps:GetCmdletMetadata" minOccurs="0" />

<xsd:element name="Cmdlet" type="cps:InstanceCmdletMetadata" minOccurs="0" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="StaticCmdlets" minOccurs="0">

<xsd:complexType>

<xsd:choice maxOccurs="unbounded">

<xsd:element name="Cmdlet" type="cps:StaticCmdletMetadata">

<xsd:unique name="UniqueParameterSetNamesInStaticCmdlet">

<xsd:selector xpath=".//cps:Method" />

<xsd:field xpath="@CmdletParameterSet" />

</xsd:unique>

<xsd:keyref name="DefaultCmdletParameterSetKeyRef" refer="cps:UniqueParameterSetNamesInStaticCmdlet">

<xsd:selector xpath=".//cps:CmdletMetadata" />

<xsd:field xpath="@DefaultCmdletParameterSet" />

</xsd:keyref>

</xsd:element>

</xsd:choice>

</xsd:complexType>

</xsd:element>

<xsd:element name="CmdletAdapterPrivateData" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Data" maxOccurs="unbounded">

<xsd:complexType>

<xsd:simpleContent>

<xsd:extension base="xsd:string">

<xsd:attribute name="Name" type="cps:NonEmptyString" use="required" />

</xsd:extension>

</xsd:simpleContent>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:key name="UniqueNamesOfPrivateDataElements">

<xsd:selector xpath=".//cps:Data" />

<xsd:field xpath="@Name" />

</xsd:key>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="CmdletAdapter" type="cps:PowerShellTypeIdentifier" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

CmdletAdapter attribute specifies which .NET class is responsible for translating

cmdlet invocations into queries and method invocations.

If this attribute is ommited, then by default the cmdlets are translated into WMI queries and method invocations.

The class specified here has to be derived from Microsoft.PowerShell.Cmdletization.CmdletAdapter class.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="ClassName" type="cps:NonEmptyString" use="required">

<xsd:annotation>

<xsd:documentation>

ClassName attribute specified the class that the cmdlets work against.

Example: "root/cimv2/Win32\_Process"

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="ClassVersion" type="cps:NonEmptyString" use="optional">

<xsd:annotation>

<xsd:documentation>

ClassVersion attribute describes the version of the implementation of the class from the ClassName attribute.

Contents of the ClassVersion attribute are passed without interpretation inside

WMI's custom operation option named "MI\_OPERATIONOPTIONS\_PROVIDERVERSION".

WMI infrastructure will compare this value against the contents of the [ClassVersion] qualifier of the WMI class

and provide descriptive error message if it cannot invoke the WMI provider - i.e. if the client attempts to use a non-existant method, property or parameter).

If WMI infrastructure can invoke the WMI provider, then the provider is responsible for further versioning decisions.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<!-- ############################################################ -->

<!-- # CMDLETS ################################################## -->

<xsd:complexType name="InstanceCmdletMetadata">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Cmdlet element under InstanceCmdlets element defines a cmdlet that wraps an instance method.

Cmdlet parameters of a cmdlet defined this way are a sum of

1) cmdlet parameters defined through GetCmdletParameters elements

2) cmdlet parameters mapped to input parameters of the method defined by Method element

</xsd:documentation>

</xsd:annotation>

<xsd:sequence>

<xsd:element name="CmdletMetadata" type="cps:CommonCmdletMetadata" />

<xsd:element name="Method" type="cps:InstanceMethodMetadata">

<xsd:unique name="UniqueCmdletParameterNamesUnderMethodElement\_1">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@PSName" />

</xsd:unique>

</xsd:element>

<xsd:element name="GetCmdletParameters" type="cps:GetCmdletParameters" minOccurs="0">

<xsd:unique name="UniqueCmdletParameterNamesUnderGetCmdletParametersElement\_2">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@PSName" />

</xsd:unique>

<xsd:unique name="UniqueCmdletParameterAliasesUnderGetCmdletParametersElement\_2">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@Aliases" />

</xsd:unique>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="StaticCmdletMetadata">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Cmdlet element under StaticCmdlets element defines a cmdlet that wraps one or more static methods.

Cmdlet parameters of a cmdlet defined this way are mapped to input parameters of methods defined by Method element

Each wrapped method corresponds to a parameter set of the cmdlet.

</xsd:documentation>

</xsd:annotation>

<xsd:sequence>

<xsd:element name="CmdletMetadata">

<xsd:complexType>

<xsd:complexContent>

<xsd:extension base="cps:CommonCmdletMetadata">

<xsd:attribute name="DefaultCmdletParameterSet" type="cps:ParameterSetName" use="optional" />

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

</xsd:element>

<xsd:element name="Method" type="cps:StaticMethodMetadata" maxOccurs="unbounded">

<xsd:unique name="UniqueCmdletParameterNamesUnderMethodElement\_2">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@PSName" />

</xsd:unique>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="GetCmdletMetadata">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

GetCmdlet element defines cmdlet metadata for the cmdlet that queries for object instances.

If GetCmdlet element is ommited, then the default verb (&quot;Get&quot;) and noun (based on &lt;DefaultNoun&gt; element) are going to be used.

GetCmdlet element is typically used for one of the following items:

- To allow the Get cmdlet to have different GetCmdletParameters than other cmdlets (for example to make all parameters optional for Get cmdlet, but make some parameters mandatory for other cmdlets)

- To change the verb of the cmdlet (for example to use &quot;Find&quot; where appropriate)

- To define aliases for the cmdlet

</xsd:documentation>

</xsd:annotation>

<xsd:sequence>

<xsd:element name="CmdletMetadata" type="cps:CommonCmdletMetadata" />

<xsd:element name="GetCmdletParameters" type="cps:GetCmdletParameters" minOccurs="0">

<xsd:unique name="UniqueCmdletParameterNamesUnderGetCmdletParametersElement\_3">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@PSName" />

</xsd:unique>

<xsd:unique name="UniqueCmdletParameterAliasesUnderGetCmdletParametersElement\_3">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@Aliases" />

</xsd:unique>

</xsd:element>

</xsd:sequence>

</xsd:complexType>

<xsd:complexType name="CommonCmdletMetadata">

<xsd:attribute name="Verb" type="cps:Verb" use="required">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Verb attribute specifies the verb of the cmdlet.

Please refer to Cmdlet Design Guidelines for a list of approved verbs.

Verb attribute is equivalent to the verbName parameter of System.Management.Automation.CmdletAttribute constructor.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="Noun" type="cps:Noun" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Noun attribute specifies the noun of the cmdlet.

If the Noun attribute is ommited, then contents of the DefaultNoun element are used.

Noun attribute is equivalent to the nounName parameter of System.Management.Automation.CmdletAttribute constructor.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="Aliases" type="cps:NonEmptyListOfAliases" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Aliases attribute specifies a white-space separated list of aliases for the cmdlet.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="ConfirmImpact" type="cps:ConfirmImpact" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

ConfirmImpact attribute specifies the impact of the cmdlet.

ConfirmImpact attribute determines the default -Confirm and -WhatIf behavior.

ConfirmImpact attribute is equivalent to the ConfirmImpact property of System.Management.Automation.CmdletAttribute.

Presence of the ConfirmImpact attribute is equivalent to setting to true the SupportsShouldProcess property of System.Management.Automation.CmdletAttribute.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="HelpUri" type="xsd:anyURI" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

HelpUri attribute specifies the URI with the help content.

HelpUri attribute is used for the following help experience: Get-Help -Online &lt;cmdlet name&gt;

HelpUri attribute is equivalent to the HelpUri property of System.Management.Automation.CmdletAttribute

Example: "http://go.microsoft.com/fwlink/?LinkID=113309"

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<!-- ############################################################ -->

<!-- # METHODS ################################################## -->

<xsd:complexType name="StaticMethodMetadata">

<xsd:complexContent>

<xsd:extension base="cps:CommonMethodMetadata">

<xsd:sequence>

<xsd:element name="Parameters" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Parameter" type="cps:StaticMethodParameterMetadata" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

<xsd:key name="UniqueParameterNamesWithinStaticMethod">

<xsd:selector xpath=".//cps:Parameter" />

<xsd:field xpath="@ParameterName" />

</xsd:key>

<xsd:unique name="UniquePositionWithinStaticMethodParameters">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@Position" />

</xsd:unique>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="CmdletParameterSet" type="cps:ParameterSetName" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

CmdletParameterSet attribute specifies the name of a cmdlet parameter set associated with the static method.

If CmdletParameterSet is ommited, then the name of the cmdlet parameter set is auto-generated based on the name of the method.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="InstanceMethodMetadata">

<xsd:complexContent>

<xsd:extension base="cps:CommonMethodMetadata">

<xsd:sequence>

<xsd:element name="Parameters" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Parameter" type="cps:InstanceMethodParameterMetadata" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

<xsd:key name="UniqueParameterNamesWithinMethod">

<xsd:selector xpath=".//cps:Parameter" />

<xsd:field xpath="@ParameterName" />

</xsd:key>

<xsd:unique name="UniquePositionWithinMethodParameters">

<xsd:selector xpath=".//cps:CmdletParameterMetadata" />

<xsd:field xpath="@Position" />

</xsd:unique>

</xsd:element>

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CommonMethodMetadata">

<xsd:sequence>

<xsd:element name="ReturnValue" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Type" type="cps:TypeMetadata" />

<xsd:element name="CmdletOutputMetadata" type="cps:CmdletOutputMetadata" minOccurs="0" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="MethodName" type="cps:NonEmptyString" use="required">

<xsd:annotation>

<xsd:documentation>

MethodName attribute specified the name of the method that the cmdlet invocations are mapped to.

Some method names are recognized and handled in a special way.

"cim:CreateInstance" is mapped to the WMI's static, intrinsic CreateInstance method. Names of method parameters have to map to names of properties.

"cim:ModifyInstance" is mapped to the WMI's instance, intrinsic ModifyInstance method. Names of method parameters have to map to names of properties.

"cim:DeleteInstance" is mapped to the WMI's instance, intrinsic DeleteInstance method. All method parameters are ignored.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<xsd:complexType name="StaticMethodParameterMetadata">

<xsd:complexContent>

<xsd:extension base="cps:CommonMethodParameterMetadata">

<xsd:sequence>

<xsd:element name="CmdletParameterMetadata" type="cps:CmdletParameterMetadataForStaticMethodParameter" minOccurs="0" />

<xsd:element name="CmdletOutputMetadata" type="cps:CmdletOutputMetadata" minOccurs="0" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="InstanceMethodParameterMetadata">

<xsd:complexContent>

<xsd:extension base="cps:CommonMethodParameterMetadata">

<xsd:sequence>

<xsd:element name="CmdletParameterMetadata" type="cps:CmdletParameterMetadataForInstanceMethodParameter" minOccurs="0" />

<xsd:element name="CmdletOutputMetadata" type="cps:CmdletOutputMetadata" minOccurs="0" />

</xsd:sequence>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CommonMethodParameterMetadata">

<xsd:sequence>

<xsd:element name="Type" type="cps:TypeMetadata" />

</xsd:sequence>

<xsd:attribute name="ParameterName" type="cps:NonEmptyString" use="required" />

<xsd:attribute name="DefaultValue" type="xsd:string" use="optional" />

</xsd:complexType>

<!-- ############################################################ -->

<!-- # PROPERTIES METADATA ###################################### -->

<xsd:complexType name="GetCmdletParameters">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

GetCmdletParameters element defines cmdlet parameters used to select object instances.

</xsd:documentation>

</xsd:annotation>

<xsd:sequence>

<xsd:element name="QueryableProperties" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Property" type="cps:PropertyMetadata" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

<xsd:key name="PropertyNameUnderQueryableProperties">

<xsd:selector xpath=".//cps:Property" />

<xsd:field xpath="@PropertyName" />

</xsd:key>

</xsd:element>

<xsd:element name="QueryableAssociations" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Association" type="cps:Association" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

<xsd:element name="QueryOptions" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Option" type="cps:QueryOption" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="DefaultCmdletParameterSet" type="cps:ParameterSetName" use="optional" />

</xsd:complexType>

<xsd:complexType name="QueryOption">

<xsd:sequence>

<xsd:element name="Type" type="cps:TypeMetadata" />

<xsd:element name="CmdletParameterMetadata" type="cps:CmdletParameterMetadataForGetCmdletParameter" minOccurs="0" />

</xsd:sequence>

<xsd:attribute name="OptionName" type="cps:NonEmptyString" use="required" />

</xsd:complexType>

<xsd:complexType name="Association">

<xsd:sequence>

<xsd:element name="AssociatedInstance">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="Type" type="cps:TypeMetadata" />

<xsd:element name="CmdletParameterMetadata" type="cps:CmdletParameterMetadataForGetCmdletFilteringParameter" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="Association" type="cps:NonEmptyString" use="required">

<xsd:annotation>

<xsd:documentation>

Association attribute specifies the name of the association between the cmdlet argument and the instances the cmdlet acts against.

Association attribute is equivalent to the associationClassName parameter of EnumerateAssociatedInstances method of Microsoft.Management.Infrastructure.CimSession class.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="SourceRole" type="cps:NonEmptyString" use="required">

<xsd:annotation>

<xsd:documentation>

SourceRole attribute specifies the role of the cmdlet argument (in the association between the cmdlet argument and the instances the cmdlet acts against).

SourceRole attribute is equivalent to the sourceRole parameter of EnumerateAssociatedInstances method of Microsoft.Management.Infrastructure.CimSession class.

SourceRole should be the name of a property on the class specified in the Association attribute.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="ResultRole" type="cps:NonEmptyString" use="required">

<xsd:annotation>

<xsd:documentation>

ResultRole attribute specifies the role of the cmdlet argument (in the association between the cmdlet argument and the instances the cmdlet acts against).

ResultRole attribute is equivalent to the resultRole parameter of EnumerateAssociatedInstances method of Microsoft.Management.Infrastructure.CimSession class.

ResultRole should be the name of a property on the class specified in the Association attribute.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<xsd:complexType name="PropertyMetadata">

<xsd:sequence>

<xsd:element name="Type" type="cps:TypeMetadata" />

<xsd:choice maxOccurs="unbounded">

<xsd:element name="RegularQuery" type="cps:WildcardablePropertyQuery">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

RegularQuery element defines a cmdlet parameter that limits which objects will be processed by the cmdlet

- only objects with a property value equal to the cmdlet parameter argument will be processed.

Comparison of strings and characters is always case-insensitive.

Example for &lt;RegularQuery&gt; element that is applied to an ObjectId property:

The following cmdlet invocation:

Get-MyObject -ObjectId 123,456

will be translated into the following WQL query:

SELECT \* FROM MyObject WHERE ((ObjectId = 123) OR (ObjectId = 456))

Example for &lt;RegularQuery AllowGlobbing=&quot;false&quot; &gt; element that is applied to a Name property:

The following cmdlet invocation:

Get-MyObject -LiteralName p\*,q\*

will be translated into the following WQL query:

SELECT \* FROM MyObject WHERE ((Name = &quot;p\*&quot;) OR (Name = &quot;q\*&quot;))

Example for &lt;RegularQuery AllowGlobbing=&quot;true&quot; &gt; element that is applied to a Name property:

The following cmdlet invocation:

Get-MyObject -Name p\*,q\*

will be translated into the following WQL query:

SELECT \* FROM MyObject WHERE ((Name like &quot;p%&quot;) OR (Name like &quot;q%&quot;))

</xsd:documentation>

</xsd:annotation>

</xsd:element>

<xsd:element name="ExcludeQuery" type="cps:WildcardablePropertyQuery">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

ExcludeQuery element defines a cmdlet parameter that limits which objects will be processed by the cmdlet

- only objects with a property value \*not\* equal to the cmdlet parameter argument will be processed.

Comparison of strings and characters is always case-insensitive.

Example for &lt;ExcludeQuery&gt; element that is applied to an ObjectId property:

The following cmdlet invocation:

Get-MyObject -ExcludeObjectId 123,456

will be translated into the following WQL query:

SELECT \* FROM MyObject WHERE ((NOT Name = 123) AND (NOT Name = 456))

</xsd:documentation>

</xsd:annotation>

</xsd:element>

<xsd:element name="MinValueQuery" type="cps:PropertyQuery">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

MinValueQuery element defines a cmdlet parameter that limits which objects will be processed by the cmdlet

- only objects with a property value greater than or equal to the cmdlet parameter argument will be processed.

Example for &lt;MinValueQuery&gt; element that is applied to an WorkingSet property:

The following cmdlet invocation:

Get-MyObject -MinWorkingSet 123

will be translated into the following WQL query:

SELECT \* FROM MyObject WHERE (WorkingSet &gt;= 123)

</xsd:documentation>

</xsd:annotation>

</xsd:element>

<xsd:element name="MaxValueQuery" type="cps:PropertyQuery">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

MaxValueQuery element defines a cmdlet parameter that limits which objects will be processed by the cmdlet

- only objects with a property value less than or equal to the cmdlet parameter argument will be processed.

Example for &lt;MaxValueQuery&gt; element that is applied to an WorkingSet property:

The following cmdlet invocation:

Get-MyObject -MaxWorkingSet 123

will be translated into the following WQL query:

SELECT \* FROM MyObject WHERE (WorkingSet =&lt; 123)

</xsd:documentation>

</xsd:annotation>

</xsd:element>

</xsd:choice>

</xsd:sequence>

<xsd:attribute name="PropertyName" type="cps:NonEmptyString" use="required" />

</xsd:complexType>

<xsd:complexType name="WildcardablePropertyQuery">

<xsd:complexContent>

<xsd:extension base="cps:PropertyQuery">

<xsd:attribute name="AllowGlobbing" type="xsd:boolean" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

AllowGlobbing attribute specifies if strings with globbing characters (wildcards) are supported.

Example of a wildcard: "foo\*" (matches all strings beginning with "foo")

If AllowGlobbing attribute is ommited then its value is based on the type of the filtered property.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="PropertyQuery">

<xsd:sequence>

<xsd:element name="CmdletParameterMetadata" type="cps:CmdletParameterMetadataForGetCmdletFilteringParameter" minOccurs="0" />

</xsd:sequence>

</xsd:complexType>

<!-- ############################################################ -->

<!-- # COMMON PROPERTIES/PARAMETERS METADATA##################### -->

<xsd:complexType name="CmdletParameterMetadataForStaticMethodParameter">

<xsd:complexContent>

<xsd:extension base="cps:CmdletParameterMetadata">

<xsd:attribute name="ValueFromPipeline" type="xsd:boolean" use="optional" />

<xsd:attribute name="ValueFromPipelineByPropertyName" type="xsd:boolean" use="optional" />

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CmdletParameterMetadataForInstanceMethodParameter">

<xsd:complexContent>

<xsd:extension base="cps:CmdletParameterMetadata">

<xsd:attribute name="ValueFromPipelineByPropertyName" type="xsd:boolean" use="optional" />

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CmdletParameterMetadataForGetCmdletFilteringParameter">

<xsd:complexContent>

<xsd:extension base="cps:CmdletParameterMetadataForGetCmdletParameter">

<xsd:attribute name="ErrorOnNoMatch" type="xsd:boolean" use="optional" />

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CmdletParameterMetadataForGetCmdletParameter">

<xsd:complexContent>

<xsd:extension base="cps:CmdletParameterMetadata">

<xsd:attribute name="ValueFromPipeline" type="xsd:boolean" use="optional" />

<xsd:attribute name="ValueFromPipelineByPropertyName" type="xsd:boolean" use="optional" />

<xsd:attribute name="CmdletParameterSets" type="cps:NonEmptyListOfParameterSetNames" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

CmdletParameterSets attribute is a whitespace-separated list of names of parameter sets,

that the cmdlet parameter should belong to.

If this parameter is ommited, then the cmdlet parameter belongs to all parameter sets.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:extension>

</xsd:complexContent>

</xsd:complexType>

<xsd:complexType name="CmdletParameterMetadata">

<xsd:sequence>

<xsd:element name="AllowEmptyCollection" minOccurs="0" />

<xsd:element name="AllowEmptyString" minOccurs="0" />

<xsd:element name="AllowNull" minOccurs="0" />

<xsd:element name="ValidateNotNull" minOccurs="0" />

<xsd:element name="ValidateNotNullOrEmpty" minOccurs="0" />

<xsd:element name="ValidateCount" minOccurs="0">

<xsd:complexType>

<xsd:attribute name="Min" type="xsd:nonNegativeInteger" use="required" />

<xsd:attribute name="Max" type="xsd:nonNegativeInteger" use="required" />

</xsd:complexType>

</xsd:element>

<xsd:element name="ValidateLength" minOccurs="0">

<xsd:complexType>

<xsd:attribute name="Min" type="xsd:nonNegativeInteger" use="required" />

<xsd:attribute name="Max" type="xsd:nonNegativeInteger" use="required" />

</xsd:complexType>

</xsd:element>

<!-- ValidatePattern omitted - CDXML does not allow DOS attacks against PS client -->

<xsd:element name="ValidateRange" minOccurs="0">

<xsd:complexType>

<xsd:attribute name="Min" type="xsd:integer" use="required" />

<xsd:attribute name="Max" type="xsd:integer" use="required" />

</xsd:complexType>

</xsd:element>

<!-- ValidateScript omitted - CDXML does not allow execution of arbitrary code in PS client -->

<xsd:element name="ValidateSet" minOccurs="0">

<xsd:complexType>

<xsd:sequence>

<xsd:element name="AllowedValue" type="xsd:string" maxOccurs="unbounded" />

</xsd:sequence>

</xsd:complexType>

</xsd:element>

</xsd:sequence>

<xsd:attribute name="IsMandatory" type="xsd:boolean" use="optional" />

<xsd:attribute name="Aliases" type="cps:NonEmptyListOfAliases" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Aliases attribute specifies a white-space separated list of aliases for the cmdlet parameter.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="PSName" type="cps:CmdletParameterName" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

PSName attribute specifies the name of a cmdlet parameter.

If PSName attribute is ommited then it is based on the contents of PropertyName or ParameterName or OptionName attribute (whichever one is applicable).

Example:

&lt;Property PropertyName=&quot;Name&quot;&gt;

...

&lt;!-- &quot;Name&quot; is used as the cmdlet parameter name --&gt;

&lt;RegularQuery AllowGlobbing=&quot;true&quot; /&gt;

&lt;!-- &quot;LiteralName&quot; is used as the cmdlet parameter name --&gt;

&lt;RegularQuery AllowGlobbing=&quot;false&quot;&gt;

&lt;CmdletParameterMetadata PSName=&quot;LiteralName&quot; /&gt;

&lt;/RegularQuery&gt;

&lt;/Property&gt;

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="Position" type="xsd:nonNegativeInteger" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

Position attribute specifies position of the cmdlet parameter.

If Position attribute is ommited, then the cmdlet parameter cannot be used positionally - the user always has to explicitly specify the name of the parameter.

System may change relative parameter positions to guarantee that cmdlet parameters defined by GetCmdletParameters element are always

before cmdlet parameters defined under Method element.

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<xsd:complexType name="CmdletOutputMetadata">

<xsd:sequence>

<xsd:element name="ErrorCode" minOccurs="0">

</xsd:element>

</xsd:sequence>

<xsd:attribute name="PSName" type="cps:CmdletOutputName" use="optional" />

</xsd:complexType>

<xsd:complexType name="TypeMetadata">

<xsd:attribute name="PSType" type="cps:PowerShellTypeIdentifier" use="required">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

PSType attribute specifies the name of the .NET type of the cmdlet parameter.

Example: "System.String"

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

<xsd:attribute name="ETSType" type="cps:NonEmptyString" use="optional">

<xsd:annotation>

<xsd:documentation xml:lang="en-us">

ETSType attribute specifies the Windows PowerShell type name of the type of the cmdlet parameter.

ETSType attribute is equivalent to System.Management.Automation.PSTypeNameAttribute.

Example: "Microsoft.Management.Infrastructure.CimInstance#Win32\_Process"

</xsd:documentation>

</xsd:annotation>

</xsd:attribute>

</xsd:complexType>

<!-- ############################################################ -->

<!-- # SIMPLE TYPES ############################################# -->

<xsd:simpleType name="NonEmptyListOfParameterSetNames">

<xsd:restriction>

<xsd:simpleType>

<xsd:list itemType="cps:ParameterSetName" />

</xsd:simpleType>

<xsd:minLength value="1" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="NonEmptyListOfAliases">

<xsd:restriction>

<xsd:simpleType>

<xsd:list itemType="cps:Alias" />

</xsd:simpleType>

<xsd:minLength value="1" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="VersionString">

<xsd:annotation>

<xsd:documentation>

Represents a version number that consist of two to four components: major, minor, build, and revision.

String representation of a version is "major.minor[.build[.revision]]" (optional components are shown in square brackets).

All defined components MUST be integers greater than or equal to 0.

For example, if the major number is 6, the minor number is 2, the build number is 1, and the revision number is 3, then string representation of the version would be "6.2.1.3".

</xsd:documentation>

</xsd:annotation>

<xsd:restriction base="xsd:string">

<xsd:pattern value="[12]?[0-9]{1,9}\.[12]?[0-9]{1,9}(\.[12]?[0-9]{1,9}(\.[12]?[0-9]{1,9})?)?" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="EnumValueName">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&csharpIdentifierRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="EnumTypeName">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="200" />

<xsd:pattern value="&csharpNamespaceRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="PowerShellTypeIdentifier">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="1000" />

<xsd:pattern value="&clrTypeRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="NonEmptyString">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="Verb">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&powerShellVerbRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="Noun">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&powerShellNounRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="Alias">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&powerShellAliasRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="ParameterSetName">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&powerShellParameterSetRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="CmdletParameterName">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&powerShellCmdletParameterRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="CmdletOutputName">

<xsd:restriction base="xsd:string">

<xsd:minLength value="1" />

<xsd:maxLength value="100" />

<xsd:pattern value="&powerShellCmdletParameterRegex;" />

</xsd:restriction>

</xsd:simpleType>

<xsd:simpleType name="ConfirmImpact">

<xsd:restriction base="xsd:string">

<xsd:enumeration value="None" />

<xsd:enumeration value="Low" />

<xsd:enumeration value="Medium" />

<xsd:enumeration value="High" />

</xsd:restriction>

</xsd:simpleType>

</xsd:schema>