Cmdlet Overview

Article • 09/17/2021

A cmdlet is a lightweight command that is used in the PowerShell environment. The PowerShell runtime invokes these cmdlets within the context of automation scripts that are provided at the command line. The PowerShell runtime also invokes them programmatically through PowerShell APIs.

Cmdlets

Cmdlets perform an action and typically return a Microsoft .NET object to the next command in the pipeline. A cmdlet is a single command that participates in the pipeline semantics of PowerShell. This includes binary (C#) cmdlets, advanced script functions, CDXML, and Workflows.

This SDK documentation describes how to create binary cmdlets written in C#. For information about script-based cmdlets, see:

- about_Functions_Advanced
- about_Functions_CmdletBindingAttribute
- about_Functions_Advanced_Methods

To create a binary cmdlet, you must implement a cmdlet class that derives from one of two specialized cmdlet base classes. The derived class must:

- Declare an attribute that identifies the derived class as a cmdlet.
- Define public properties that are decorated with attributes that identify the public properties as cmdlet parameters.
- Override one or more of the input processing methods to process records.

You can load the assembly that contains the class directly by using the Import-Module cmdlet, or you can create a host application that loads the assembly by using the System.Management.Automation.Runspaces.Initialsessionstate API. Both methods provide programmatic and command-line access to the functionality of the cmdlet.

Cmdlet Terms

The following terms are used frequently in the PowerShell cmdlet documentation:

Cmdlet attribute

A .NET attribute that is used to declare a cmdlet class as a cmdlet. Although PowerShell uses several other attributes that are optional, the Cmdlet attribute is required. For more information about this attribute, see Cmdlet Attribute Declaration.

Cmdlet parameter

The public properties that define the parameters that are available to the user or to the application that is running the cmdlet. Cmdlets can have required, named, positional, and *switch* parameters. Switch parameters allow you to define parameters that are evaluated only if the parameters are specified in the call. For more information about the different types of parameters, see Cmdlet Parameters.

Parameter set

A group of parameters that can be used in the same command to perform a specific action. A cmdlet can have multiple parameter sets, but each parameter set must have at least one parameter that is unique. Good cmdlet design strongly suggests that the unique parameter also be a required parameter. For more information about parameter sets, see Cmdlet Parameter Sets.

Dynamic parameter

A parameter that is added to the cmdlet at runtime. Typically, the dynamic parameters are added to the cmdlet when another parameter is set to a specific value. For more information about dynamic parameters, see Cmdlet Dynamic Parameters.

Input processing methods

The System.Management.Automation.Cmdlet class provides the following virtual methods that are used to process records. All the derived cmdlet classes must override one or more of the first three methods:

- System.Management.Automation.Cmdlet.BeginProcessing: Used to provide

optional one-time, pre-processing functionality for the cmdlet.
- System.Management.Automation.Cmdlet.ProcessRecord: Used to provide record-by-record processing functionality for the cmdlet. The System.Management.Automation.Cmdlet.ProcessRecord method might be called any number of times, or not at all, depending on the input of the cmdlet.
- System.Management.Automation.Cmdlet.EndProcessing: Used to provide optional one-time, post-processing functionality for the cmdlet.
- System.Management.Automation.Cmdlet.StopProcessing: Used to stop processing when the user stops the cmdlet asynchronously (for example, by pressing `CTRL`+`C`).

For more information about these methods, see Cmdlet Input Processing Methods.

When you implement a cmdlet, you must override at least one of these input processing methods. Typically, the **ProcessRecord()** is the method that you override because it is called for every record that the cmdlet processes. In contrast, the **BeginProcessing()** method and the **EndProcessing()** method are called one time to perform pre-processing or post-processing of the records. For more information about these methods, see Input Processing Methods.

ShouldProcess feature

PowerShell allows you to create cmdlets that prompt the user for feedback before the cmdlet makes a change to the system. To use this feature, the cmdlet must declare that it supports the `ShouldProcess` feature when you declare the Cmdlet attribute, and the cmdlet must call the System.Management.Automation.Cmdlet.ShouldProcess and System.Management.Automation.Cmdlet.ShouldContinue methods from within an input processing method. For more information about how to support the `ShouldProcess` functionality, see Requesting Confirmation.

Transaction

A logical group of commands that are treated as a single task. The task automatically fails if any command in the group fails, and the user has the choice to accept or reject the actions performed within the transaction. To participate in a transaction, the cmdlet must declare that it supports transactions when the Cmdlet attribute is declared. Support for transactions was introduced in Windows PowerShell 2.0. For more information about transactions, see How to Support Transactions.

How Cmdlets Differ from Commands

Cmdlets differ from commands in other command-shell environments in the following ways:

- Cmdlets are instances of .NET classes; they are not stand-alone executables.
- Cmdlets can be created from as few as a dozen lines of code.
- Cmdlets do not generally do their own parsing, error presentation, or output formatting. Parsing, error presentation, and output formatting are handled by the PowerShell runtime.
- Cmdlets process input objects from the pipeline rather than from streams of text, and cmdlets typically deliver objects as output to the pipeline.
- Cmdlets are record-oriented because they process a single object at a time.

Cmdlet Base Classes

Windows PowerShell supports cmdlets that are derived from the following two base classes.

- Most cmdlets are based on .NET classes that derive from the System.Management.Automation.Cmdlet base class. Deriving from this class allows a cmdlet to use the minimum set of dependencies on the Windows PowerShell runtime. This has two benefits. The first benefit is that the cmdlet objects are smaller, and you are less likely to be affected by changes to the PowerShell runtime. The second benefit is that, if you have to, you can directly create an instance of the cmdlet object and then invoke it directly instead of invoking it through the PowerShell runtime.

- The more-complex cmdlets are based on .NET classes that derive from the System.Management.Automation.PSCmdlet base class. Deriving from this class gives you much more access to the PowerShell runtime. This access allows your cmdlet to call scripts, to access providers, and to access the current session state. (To access the current session state, you get and set session variables and preferences.) However, deriving from this class increases the size of the cmdlet object, and it means that your cmdlet is more tightly coupled to the current version of the PowerShell runtime.

In general, unless you need the extended access to the PowerShell runtime, you should derive from the System.Management.Automation.Cmdlet class. However, the PowerShell runtime has extensive logging capabilities for the execution of cmdlets. If your auditing model depends on this logging, you can prevent the execution of your cmdlet from within another cmdlet by deriving from the System.Management.Automation.PSCmdlet class.

Cmdlet Attributes

PowerShell defines several .NET attributes that are used to manage cmdlets and to specify common functionality that is provided by PowerShell and that might be required by the cmdlet. For example, attributes are used to designate a class as a cmdlet, to specify the parameters of the cmdlet, and to request the validation of input so that cmdlet developers do not have to implement that functionality in their cmdlet code. For more information about attributes, see PowerShell Attributes.

Cmdlet Names

PowerShell uses a verb-and-noun name pair to name cmdlets. For example, the `Get-Command` cmdlet included in PowerShell is used to get all the cmdlets that are registered in the command shell. The verb identifies the action that the cmdlet performs, and the noun identifies the resource on which the cmdlet performs its action.

These names are specified when the .NET class is declared as a cmdlet. For more information about how to declare a .NET class as a cmdlet, see Cmdlet Attribute Declaration.

Writing Cmdlet Code

This document provides two ways to discover how cmdlet code is written. If you prefer to see the code without much explanation, see Examples of Cmdlet Code. If you prefer more explanation about the code, see the GetProc Tutorial, StopProc Tutorial, or SelectStr Tutorial topics.

For more information about the guidelines for writing cmdlets, see Cmdlet Development

Guidelines.

See Also

PowerShell Cmdlet Concepts

Writing a PowerShell Cmdlet

PowerShell SDK