Full credit is given to the above companies including the OS that this PDF file was generated!

PowerShell Get-Help on command 'Update-TypeData'

PS C:\Users\wahid> Get-Help Update-TypeData

NAME

Update-TypeData

SYNOPSIS

Updates the extended type data in the session.

SYNTAX

Update-TypeData [[-AppendPath] <System.String[]>] [-PrependPath <System.String[]>] [-Confirm] [-WhatIf] [<CommonParameters>]

Update-TypeData [-DefaultDisplayProperty <System.String>] [-DefaultDisplayPropertySet <System.String[]>] [-DefaultKeyPropertySet <System.String[]>] [-Force] [-InheritPropertySerializationSet <System.Nullable`1[System.Boolean]>] [-MemberName <System.String>] [-MemberType {NoteProperty | AliasProperty | ScriptProperty | CodeProperty | ScriptMethod | CodeMethod}] [-PropertySerializationSet <System.String[]>] [-SecondValue <System.Object>] [-SerializationDepth <System.Int32>] [-SerializationMethod <System.String>] [-StringSerializationSource <System.String>] [-TargetTypeForDeserialization <System.Type>] [-TypeAdapter <System.Type>] [-TypeConverter <System.Type>] -TypeName <System.String>

[-Value <System.Object>] [-Confirm] [-WhatIf] [<CommonParameters>]

Update-TypeData [-TypeData]
<System.Management.Automation.Runspaces.TypeData[]> [-Force] [-Confirm]
[-WhatIf] [<CommonParameters>]

DESCRIPTION

The `Update-TypeData` cmdlet updates the extended type data in the session by reloading the `Types.ps1xml` files into memory and adding new extended type data.

By default, PowerShell loads extended type data as it is needed. Without parameters, `Update-TypeData` reloads all of the `Types.ps1xml` files that it has loaded in the session, including any type files that you added. You can use the parameters of `Update-TypeData` to add new type files and add and replace extended type data.

The `Update-TypeData` cmdlet can be used to preload all type data. This feature is particularly useful when you are developing types and want to load those new types for testing purposes.

Beginning in Windows PowerShell 3.0, you can use `Update-TypeData` to add and replace extended type data in the session without using a `Types.ps1xml` file. Type data that is added dynamically, that is, without a file, is added only to the current session. To add the type data to all sessions, add an `Update-TypeData` command to your PowerShell profile. For more information, see about_Profiles (../Microsoft.PowerShell.Core/About/about_Profiles.md).

Also, beginning in Windows PowerShell 3.0, you can use the `Get-TypeData` cmdlet to get the extended types in the current session and the `Remove-TypeData` cmdlet to delete extended types from the current session.

Exceptions that occur in properties, or from adding properties to an `Update-TypeData` command, do not report errors. This is to suppress exceptions that would occur in many common types during formatting and outputting. If you are getting .NET properties, you can work around the suppression of exceptions by using method syntax instead, as shown in the following example:

`"hello".get_Length()`

Note that method syntax can only be used with .NET properties. Properties that are added by running the `Update-TypeData` cmdlet cannot use method syntax.

For more information about the `Types.ps1xml` files in PowerShell, see about_Types.ps1xml (../Microsoft.PowerShell.Core/About/about_Types.ps1xml.md).

PARAMETERS

-AppendPath <System.String[]>

Specifies the path to optional `.ps1xml` files. The specified files are loaded in the order that they are listed after the built-in files are loaded. You can also pipe an AppendPath value to `Update-TypeData`.

-DefaultDisplayProperty <System.String>

Specifies the property of the type that is displayed by the `Format-Wide` cmdlet when no other properties are specified.

Type the name of a standard or extended property of the type. The value of this parameter can be the name of a type that is added in the same command.

This value is effective only when there are no wide views defined for the type in a `Format.ps1xml` file.

This parameter was introduced in Windows PowerShell 3.0.

-DefaultDisplayPropertySet <System.String[]>

Specifies one or more properties of the type. These properties are displayed by the `Format-List`, `Format-Table`, and `Format-Custom` cmdlets when no other properties are specified.

Type the names of standard or extended properties of the type. The value of this parameter can be the names of types that are added in the same command.

This value is effective only when there are no list, table, or custom views, respectively, defined for the type in a `Format.ps1xml` file.

This parameter was introduced in Windows PowerShell 3.0.

-DefaultKeyPropertySet <System.String[]>

Specifies one or more properties of the type. These properties are used by the `Group-Object` and `Sort-Object` cmdlets when no other properties are specified.

Type the names of standard or extended properties of the type. The value of this parameter can be the names of types that are added in the same command.

This parameter was introduced in Windows PowerShell 3.0.

-Force <System.Management.Automation.SwitchParameter>

Indicates that the cmdlet uses the specified type data, even if type data has already been specified for that type.

This parameter was introduced in Windows PowerShell 3.0.

-InheritPropertySerializationSet <System.Nullable`1[System.Boolean]>

Indicates whether the set of properties that are serialized is inherited.
The default value is `$Null`. The acceptable values for this parameter are:

- `$True`. The property set is inherited.

- `$False`. The property set is not inherited.

- `$Null`. Inheritance is not defined.

This parameter is valid only when the value of the SerializationMethod
parameter is `SpecificProperties`. When the value of this parameter is
`$False`, the PropertySerializationSet parameter is required.
This parameter was introduced in Windows PowerShell 3.0.

-MemberName <System.String>
Specifies the name of a property or method.

Use this parameter with the TypeName , MemberType , Value and SecondValue
parameters to add or change a property or method of a type.

This parameter was introduced in Windows PowerShell 3.0.

-MemberType <System.Management.Automation.PSMemberTypes>
Specifies the type of the member to add or change.

Use this parameter with the TypeName , MemberType , Value and SecondValue
parameters to add or change a property or method of a type. The acceptable
values for this parameter are:

- AliasProperty

- CodeMethod

- CodeProperty

- Noteproperty

- ScriptMethod

- ScriptProperty

For information about these values, see PSMemberTypes Enumeration (/dotnet/api/system.management.automation.psmembertypes).
This parameter was introduced in Windows PowerShell 3.0.

-PrependPath <System.String[]>
Specifies the path to the optional `.ps1xml` files. The specified files are loaded in the order that they are listed before the built-in files are loaded.

-PropertySerializationSet <System.String[]>
Specifies the names of properties that are serialized. Use this parameter when the value of the SerializationMethod parameter is SpecificProperties .

-SecondValue <System.Object>
Specifies additional values for AliasProperty , ScriptProperty , CodeProperty , or CodeMethod members.

Use this parameter with the TypeName , MemberType , Value , and SecondValue parameters to add or change a property or method of a type.

When the value of the MemberType parameter is `AliasProperty`, the value

of the SecondValue parameter must be a data type. PowerShell converts (that is, casts) the value of the alias property to the specified type. For example, if you add an alias property that provides an alternate name for a string property, you can also specify a SecondValue of System.Int32 to convert the aliased string value to an integer.

When the value of the MemberType parameter is `ScriptProperty`, you can use the SecondValue parameter to specify an additional script block. The script block in the value of the Value parameter gets the value of a variable. The script block in the value of the SecondValue parameter set the value of the variable.

This parameter was introduced in Windows PowerShell 3.0.

-SerializationDepth <System.Int32>

Specifies how many levels of type objects are serialized as strings. The default value `1` serializes the object and its properties. A value of `0` serializes the object, but not its properties. A value of `2` serializes the object, its properties, and any objects in property values.

This parameter was introduced in Windows PowerShell 3.0.

-SerializationMethod <System.String>

Specifies a serialization method for the type. A serialization method determines which properties of the type are serialized and the technique that is used to serialize them. The acceptable values for this parameter are:

- `AllPublicProperties`. Serialize all public properties of the type. You can use the SerializationDepth parameter to determine whether child properties are serialized. - `String`. Serialize the type as a string. You can use the StringSerializationSource to specify a property of the type to use as the serialization result. Otherwise, the type is serialized by

using the ToString method of the object. - `SpecificProperties`. Serialize only the specified properties of this type. Use the PropertySerializationSet parameter to specify the properties of the type that are serialized. You can also use the InheritPropertySerializationSet parameter to determine whether the property set is inherited and the SerializationDepth parameter to determine whether child properties are serialized.

In PowerShell, serialization methods are stored in PSStandardMembers internal objects.

This parameter was introduced in Windows PowerShell 3.0.

-StringSerializationSource <System.String>

Specifies the name of a property of the type. The value of specified property is used as the serialization result. This parameter is valid only when the value of the SerializationMethod parameter is String.

-TargetTypeForDeserialization <System.Type>

Specifies the type to which object of this type are converted when they are deserialized.

This parameter was introduced in Windows PowerShell 3.0.

-TypeAdapter <System.Type>

Specifies the type of a type adapter, such as Microsoft.PowerShell.Cim.CimInstanceAdapter . A type adapter enables PowerShell to get the members of a type.

This parameter was introduced in Windows PowerShell 3.0.

-TypeConverter <System.Type>

Specifies a type converter to convert values between different types. If a

type converter is defined for a type, an instance of the type converter is used for the conversion.

Enter a System.Type value that is derived from the System.ComponentModel.TypeConverter or System.Management.Automation.PSTypeConverter classes.

This parameter was introduced in Windows PowerShell 3.0.

-TypeData <System.Management.Automation.Runspaces.TypeData[]>
Specifies an array of type data that this cmdlet adds to the session. Enter a variable that contains a TypeData object or a command that gets a TypeData object, such as a `Get-TypeData` command. You can also pipe a TypeData object to `Update-TypeData`.

This parameter was introduced in Windows PowerShell 3.0.

-TypeName <System.String>
Specifies the name of the type to extend.

For types in the System namespace, enter the short name. Otherwise, the full type name is required. Wildcards are not supported.

You can pipe type names to `Update-TypeData`. When you pipe an object to `Update-TypeData`, `Update-TypeData` gets the type name of the object and type data to the object type.

Use this parameter with the MemberName , MemberType , Value and SecondValue parameters to add or change a property or method of a type.

This parameter was introduced in Windows PowerShell 3.0.

-Value <System.Object>

Specifies the value of the property or method.

If you add an `AliasProperty`, `CodeProperty`, `ScriptProperty`, or `CodeMethod` member, you can use the SecondValue parameter to add additional information.

Use this parameter with the MemberName , MemberType , Value and SecondValue parameters to add or change a property or method of a type.

This parameter was introduced in Windows PowerShell 3.0.

-Confirm <System.Management.Automation.SwitchParameter>
Prompts you for confirmation before running the cmdlet.

-WhatIf <System.Management.Automation.SwitchParameter>
Shows what would happen if the cmdlet runs. The cmdlet is not run.

<CommonParameters>
This cmdlet supports the common parameters: Verbose, Debug, ErrorAction, ErrorVariable, WarningAction, WarningVariable, OutBuffer, PipelineVariable, and OutVariable. For more information, see about_CommonParameters (https:/go.microsoft.com/fwlink/?LinkID=113216).

---------- Example 1: Update extended types ----------

Update-TypeData

This command updates the extended type configuration from the `Types.ps1xml` files that have already been used in the session.
---------- Example 2: Update types multiple times ----------

Update-TypeData -PrependPath TypesA.types.ps1xml, TypesB.types.ps1xml
Update-TypeData -PrependPath TypesA.types.ps1xml

----- Example 3: Add a script property to DateTime objects -----

```
Update-TypeData -TypeName "System.DateTime" -MemberType ScriptProperty
-MemberName "Quarter" -Value {
  if ($this.Month -in @(1,2,3)) {"Q1"}
  elseif ($this.Month -in @(4,5,6)) {"Q2"}
  elseif ($this.Month -in @(7,8,9)) {"Q3"}
  else {"Q4"}
}
(Get-Date).Quarter

Q1
```

The `Update-TypeData` command uses the TypeName parameter to specify the System.DateTime type, the MemberName parameter to specify a name for the new property, the MemberType property to specify the ScriptProperty type, and the Value parameter to specify the script that determines the annual quarter.

The value of the Value property is a script that calculates the current annual quarter. The script block uses the `$this` automatic variable to represent the current instance of the object and the In operator to determine whether the month value appears in each integer array. For more information about the `-in` operator, see about_Comparison_Operators (../Microsoft.PowerShell.Core/about/about_Comparison_Operators.md).

The second command gets the new Quarter property of the current date.

-- Example 4: Update a type that displays in lists by default --

```
Update-TypeData -TypeName "System.DateTime" -DefaultDisplayPropertySet
"DateTime, DayOfYear, Quarter"
Get-Date | Format-List
```

Thursday, March 15, 2012 12:00:00 AM

DayOfYear : 75

Quarter   : Q1

The first command uses the `Update-TypeData` cmdlet to set the default list properties for the System.DateTime type. The command uses the TypeName parameter to specify the type and the DefaultDisplayPropertySet parameter to specify the default properties for a list. The selected properties include the new Quarter script property that was added in a previous example.

The second command uses the `Get-Date` cmdlet to get a System.DateTime object that represents the current date. The command uses a pipeline operator (`|`) to send the DateTime object to the `Format-List` cmdlet. Because the `Format-List` command does not specify the properties to display in the list, PowerShell uses the default values that were established by the `Update-TypeData` command.

-------- Example 5: Update type data for a piped object --------

```
Get-Module | Update-TypeData -MemberType ScriptProperty -MemberName
"SupportsUpdatableHelp" -Value {
  if ($this.HelpInfoUri) {$True} else {$False}
}
Get-Module -ListAvailable | Format-Table Name, SupportsUpdatableHelp
```

```
Name                          SupportsUpdatableHelp
----                          ---------------------
Microsoft.PowerShell.Diagnostics                True
Microsoft.PowerShell.Host                       True
Microsoft.PowerShell.Management                 True
Microsoft.PowerShell.Security                   True
Microsoft.PowerShell.Utility                    True
Microsoft.WSMan.Management                      True
```

PSDiagnostics False

PSScheduledJob True

PSWorkflow True

ServerManager True

TroubleshootingPack False

This example demonstrates that when you pipe an object to `Update-TypeData`, `Update-TypeData` adds extended type data for the object type.

This technique is quicker than using the `Get-Member` cmdlet or the `Get-Type` method to get the object type. However, if you pipe a collection of objects to `Update-TypeData`, it updates the type data of the first object type and then returns an error for all other objects in the collection because the member is already defined on the type.

The first command uses the `Get-Module` cmdlet to get the PSScheduledJob module. The command pipes the module object to the `Update-TypeData` cmdlet, which updates the type data for the System.Management.Automation.PSModuleInfo type and the types derived from it, such as the ModuleInfoGrouping type that `Get-Module` returns when you use the ListAvailable parameter in the command.

The `Update-TypeData` commands adds the SupportsUpdatableHelp script property to all imported modules. The value of the Value parameter is a script that returns `$True` if the HelpInfoUri property of the module is populated and `$False` otherwise.

The second command pipes the module objects from `Get-Module` to the `Format-Table` cmdlet, which displays the Name and SupportsUpdatableHelp properties of all modules in a list.

REMARKS

To see the examples, type: "get-help Update-TypeData -examples".

For more information, type: "get-help Update-TypeData -detailed".

For technical information, type: "get-help Update-TypeData -full".

For online help, type: "get-help Update-TypeData -online"