Microsoft

Objects, Variables, Data Types

Module 3

Learnings covered in this Unit

- What are objects and why do they matter?
- Discover objects and benefit from object members
- Create variables or use built-in (Automatic) variables
- Learn basic types
- Master strings (Expandable strings, escape character, literal string)

Object Models

What is an object?

STRUCTURED DATA

COMBINES SIMILAR INFORMATION AND CAPABILITIES INTO ONE ENTITY

A COLLECTION OF PARTS AND HOW TO USE THEM

How Would You Model a TV?

Properties (Information)	Methods (Actions)
Is it on?	Toggle Power
Current Channel	Channel Up
Current Volume	Channel Down
Screen Size	Volume Up
Brand	Volume Down
Input	Change Input
Screen Type	Set Channel(<int>)

To change the channel to a particular one we have to pass in data (the channel number).

Understanding Instances

Type [Microsoft.TV]	
Members	
Properties	Methods
DisplayType	VolumeUp()
Input	VolumeDown()
Size	ChannelUp()
ModelNumber	TogglePower()
…	…

$MyTv1	
Property	Value
DisplayType	LCD
Input	VGA
Size	42
ModelNumber	PTV-42732
…	…

$MyTv2	
Property	Value
DisplayType	LED
Input	HDMI1
Size	80
ModelNumber	LEDTV-80432
…	…

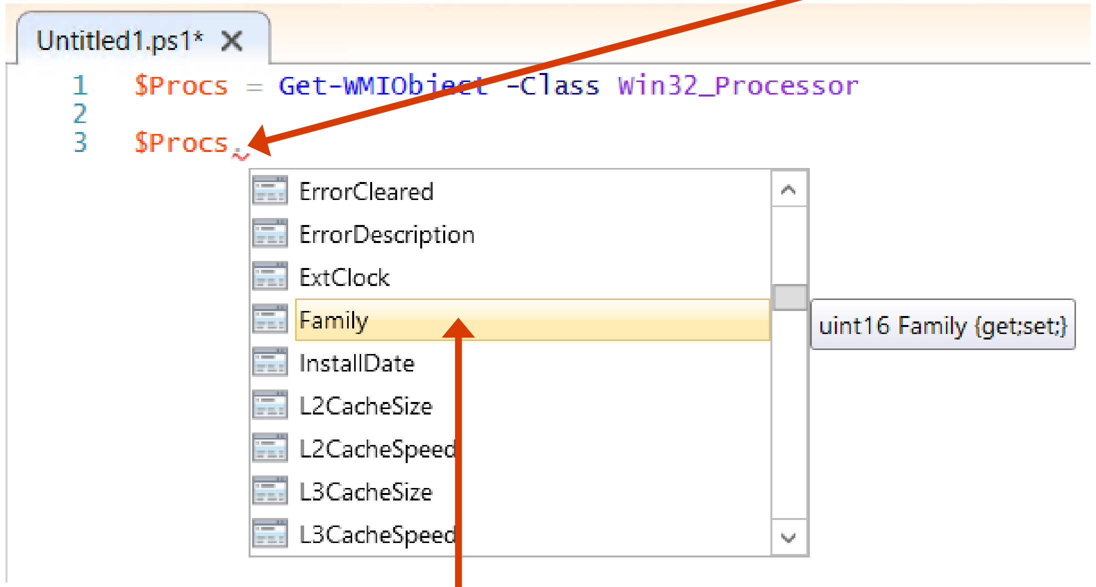
Object-Based Shell

- Everything is represented as an OBJECT
- An OBJECT is an INSTANCE of a TYPE
- OBJECTS have data fields (PROPERTIES) and procedures (METHODS)
- A TYPE represents a construct that defines a template of MEMBERS
- PROPERTIES and METHODS are collectively known as MEMBERS

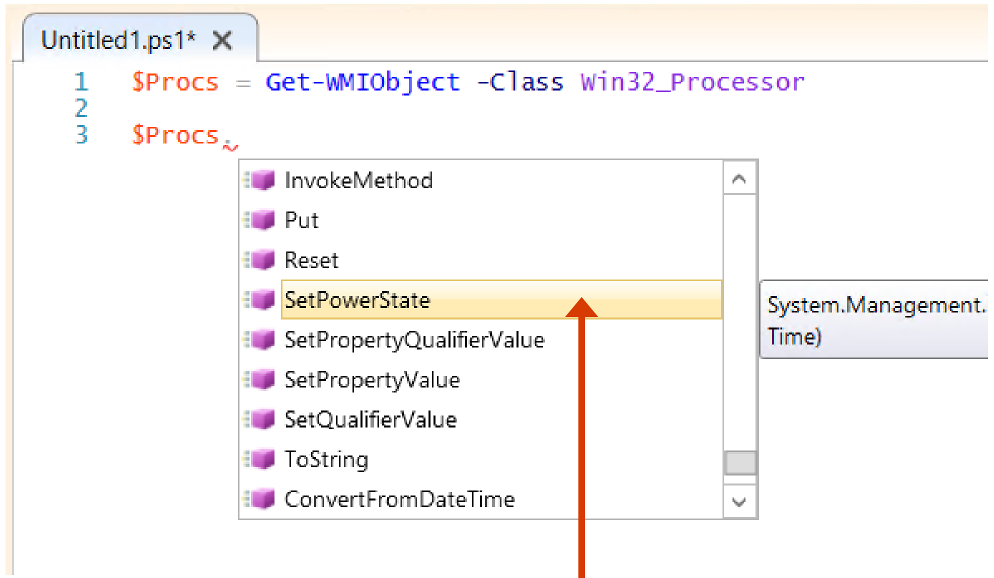
Accessing Members – ISE

ISE IntelliSense

Type "." to access members



```
$Procs = Get-WMIObject -Class Win32_Processor

$Procs.
```



```
$Procs = Get-WMIObject -Class Win32_Processor

$Procs.
```

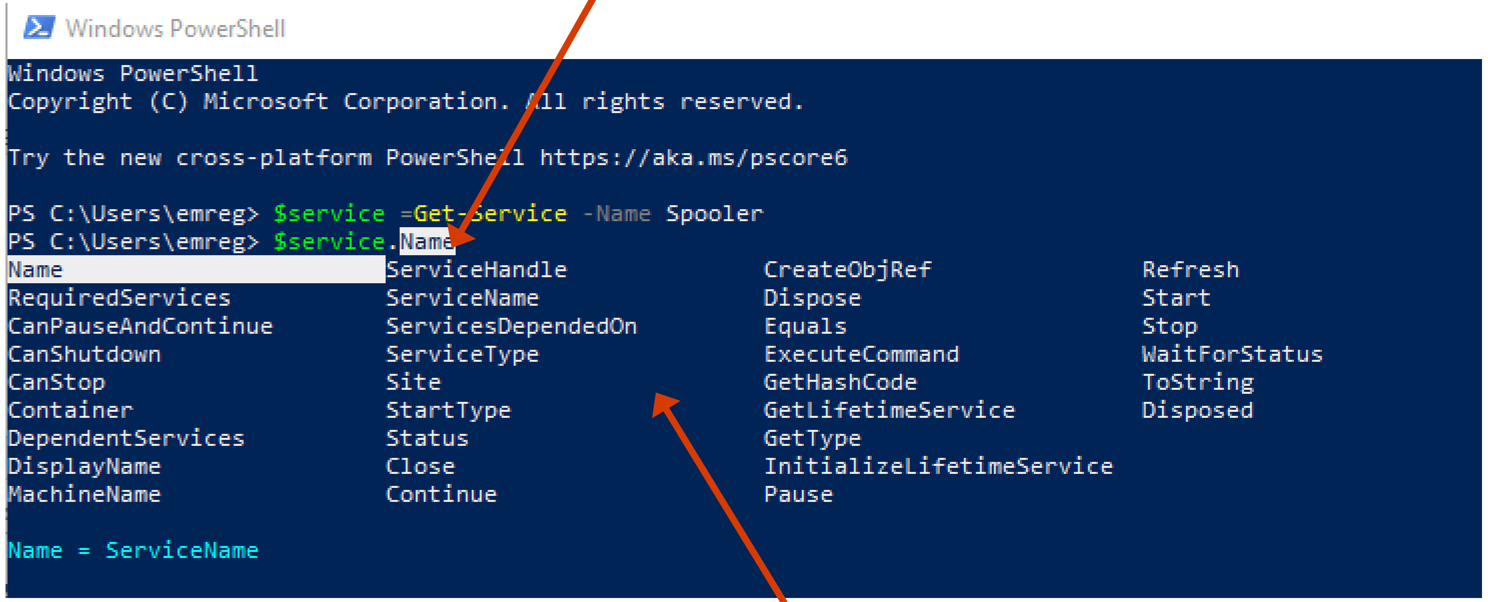
Properties

Methods

Accessing Members – Console

Console IntelliSense

Type "." then CTRL + Space



Windows PowerShell

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS C:\Users\emreg> $service =Get-Service -Name Spooler
PS C:\Users\emreg> $service.Name
Name                       ServiceHandle              CreateObjRef               Refresh
RequiredServices           ServiceName                Dispose                    Start
CanPauseAndContinue        ServicesDependedOn         Equals                     Stop
CanShutdown                ServiceType                ExecuteCommand             WaitForStatus
CanStop                    Site                       GetHashCode                ToString
Container                  StartType                  GetLifetimeService         Disposed
DependentServices          Status                     GetType
DisplayName                Close                      InitializeLifetimeService
MachineName                Continue                   Pause

Name = ServiceName
```

Properties & Methods

Demonstration

PowerShell Objects

Identify PROPERTIES and METHODS for an object

Why Should discover / identify methods and properties

Take Action

Methods are ready to use functions. You can take action immediately.

Parse Less

Properties are structured data, you don't need to parse the results in most cases.

Get-Member Overview

Discover Properties and methods of an Object

- Displays PROPERTIES and Methods
- Shows the Type of the Object
- PROPERTIES are columns of Information
- METHODS are actions that can be taken on the object

Get-Member cmdlet

Shows the **type** name, **properties** and **methods**

The object is passed to **–InputObject** parameter

```
PS>  Get-Member -InputObject "Some String"

 TypeName: System.String

Name         MemberType           Definition
----         ----------           ----------
Trim         Method               string Trim(Params char[] trimChars)....
Length       Property             int Length {get;}
```

Get-Member Property Definition

```
PS C:\> $item = Get-Item C:\Windows\System32\drivers\etc\hosts
PS C:\> Get-Member -inputobject $item -Name LastWriteTime

 TypeName: System.IO.FileInfo

Name           MemberType           Definition
----           ----------           ----------
LastWriteTime  Property             datetime LastWriteTime {get;set;}
```

Data type: **[DateTime]**

Can be **get** (received) or **set** (changed)

```
PS C:\> $file = Get-Item C:\Windows\System32\drivers\etc\hosts
PS C:\> $file.LastWriteTime = (Get-Date)
PS C:\> Get-Item C:\Windows\System32\drivers\etc\hosts
Directory: C:\Windows\System32\Drivers\etc

Mode                 LastWriteTime         Length Name
----                 -------------         ------ ----
-a----          12/23/2020   4:23 PM          894 hosts
```

Get-Member Method Definition

```
PS C:\> $file = Get-Item C:\Windows\notepad.exe
PS C:\> Get-Member -InputObject $file -Name CopyTo

 TypeName: System.IO.FileInfo

Name    MemberTyp    Definition
----    ----------   ----------
CopyTo  Method       System.IO.FileInfo CopyTo(string destFileName),
                     System.IO.FileInfo CopyTo(string destFileName, bool ov..
```

Two parameter sets

This Method **returns** a **System.IO.FileInfo**

```
PS C:\> $file = Get-Item C:\Windows\notepad.exe
PS C:\> $file.CopyTo("C:\Temp\notepad.exe", $True)

 Mode                 LastWriteTime         Length Name
----                 -------------         ------- ----
-a----        7/16/2016   7:43 AM          243200 notepad.exe
```

The newly copied file is **System.IO.FileInfo**

Demonstration:
Gettype()
Get-Member

Variables

Variables Overview

✓ What are variables?

User-Defined Variables

Working with Strings

What Are Variables?

- Unit of memory
- Defined and accessed using a dollar sign prefix ($)
- Holds an object which can also be a collection of objects
- Variable names can include spaces and special characters
- Not case-sensitive

- Kinds of variables:
  - Automatic (built-in)
  - User-defined

Automatic Variables

BUILT-IN

CREATED AND MAINTANED BY POWERSHELL

STORE POWERSHELL STATE

Automatic Variables Examples

Get-Help about_Automatic_Variables

Type	Example
List of all errors	`PS C:\> $Error`
Execution status of last operation	`PS C:\> $?`
User’s home directory	`PS C:\> $HOME`
Current host application for PowerShell	`PS C:\> $Host`
NULL or empty value	`PS C:\> $null`
Full path of installation directory for PowerShell	`PS C:\> $PSHOME`
Represents TRUE in commands	`PS C:\> $true`
Represent FALSE in commands	`PS C:\> $false`

Demonstration: Automatic Variables

User-Defined Variables

EXISTS ONLY IN CURRENT SESION

CREATED AND MAINTANED BY USER

LOST WHEN SESSION IS CLOSED

Creating User Defined Variable

Assignment Operator
‘=’

-OutVariable common parameter

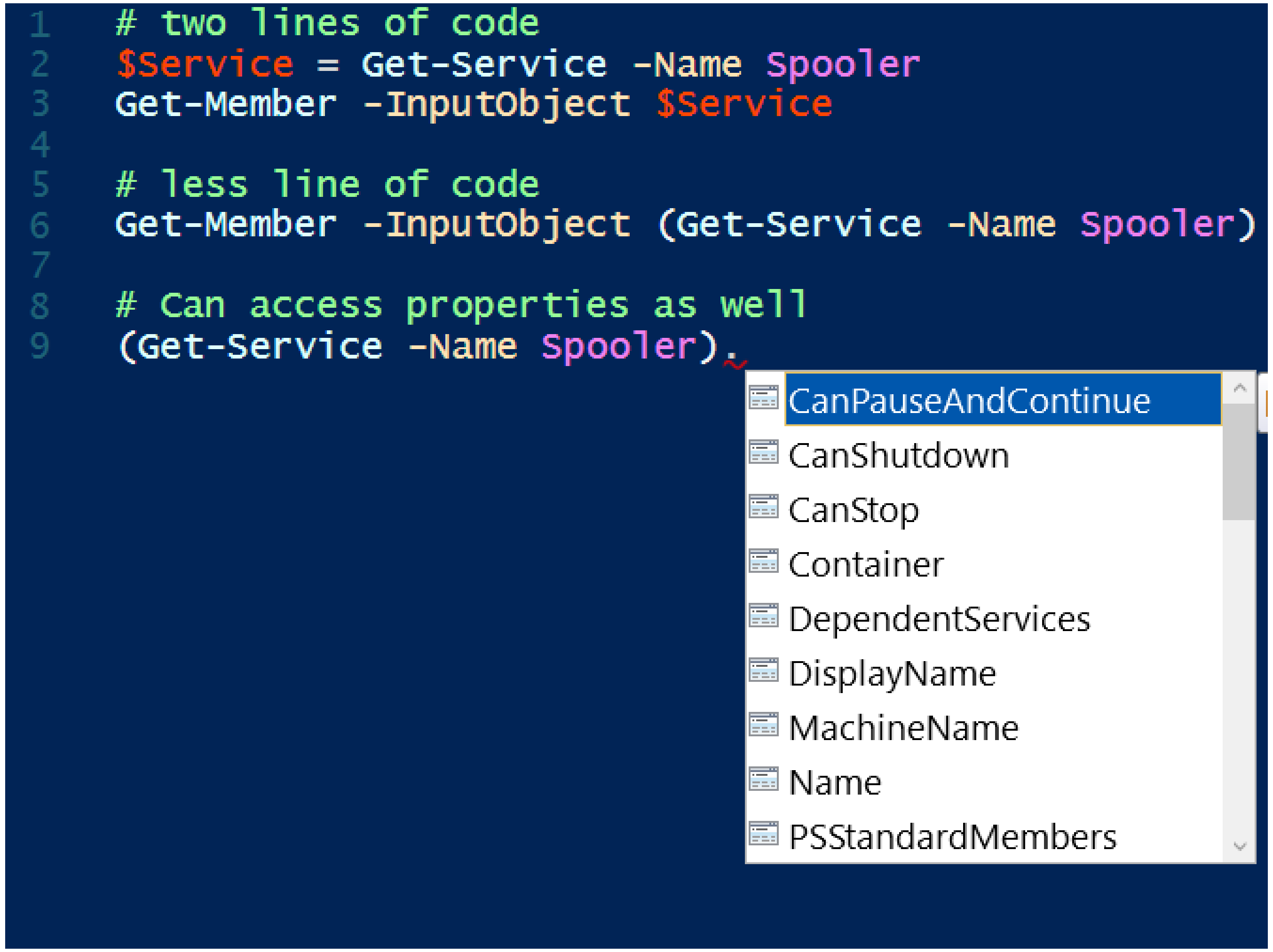
Variable **Cmdlets**

```
PS C:\> $svcs = Get-Service

PS C:\> Get-Service -OutVariable svcs

PS C:\> New-Variable -Name svcs -Value (Get-Service)

PS C:\> $svcs

Status    Name          DisplayName
------    ----          -----------
Stopped   AeLookupSvc   Application Experience
Stopped   ALG           Application Layer Gateway Service
Running   AppIDSvc      Application Identity
Running   Appinfo       Application Information
...
```

Subexpression

Expressions within Expressions instead of user-defined variables

- Can be used as in line expressions
- Avoids using unnecessary variables
- Can be nested
- The expression within, returns object or objects



```
# two lines of code
$Service = Get-Service -Name Spooler
Get-Member -InputObject $Service

# less line of code
Get-Member -InputObject (Get-Service -Name Spooler)

# Can access properties as well
(Get-Service -Name Spooler).
```

Variable Cmdlets

Name	Example
New-Variable	`PS C:\> New-Variable zipcode -Value 98033`
Clear-Variable	`PS C:\> Clear-Variable -Name Processes`
Remove-Variable	`PS C:\> Remove-Variable -Name Smp`
Set-Variable	`PS C:\> Set-Variable -Name desc -Value "Description"`
Get-Variable	`PS C:\> Get-Variable -Name m*`

Constant Variables

- Variables can only be made constant at creation (cannot use "=")
- Cannot be deleted
- Cannot be changed

```
PS C:\> New-Variable -Name pi -Value 3.14159 -Option Constant
```

ReadOnly Variables

- Cannot mark a variable ReadOnly with "="
- Cannot be easily deleted (must use Remove-Variable with -Force)
- Cannot be changed with "=" (must use Set-Variable with -Force

```
PS C:\> New-Variable -Name max -Value 256 -Option ReadOnly
```

Objects and Variables

Summary

Always keep in mind, Everything is OBJECT in PowerShell

Each Object Has a TYPE

Variables reference OBJECTS

Demonstration: User Defined Variables

Types

Understanding Types

Type Operations

Every object exists of a TYPE

Object types are declared when created

PowerShell will search for a best match type for you when not casted

General Types

Alias	Full Name	Description
Object	System.Object	Every type in PowerShell is derived from object
Boolean	System.Boolean	$true and $false
Char	System.Char	Stores UTF-16-encoded 16-bit Unicode code point
Int	System.Int32	-2147483648 to 2147483647
Long	System.Int64	-9223372036854775808 to 9223372036854775807
Double	System.Double	Double-precision floating-point number
Enum	System.Enum	Defines a set of named constants
Array	System.Array	One or more dimensions with 0 or more elements
DateTime	System.DateTime	Stores date and time values

What Object Type am I Using?

.GetType()

- All objects will have a “GetType” method which returns the type
- “GetType” also returns detailed type information
- The Return value is itself an object representing the type, it has a FullName property

```
PS C:\> ("").GetType().FullName
System.String

PS C:\> ("").GetType().Assembly
mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089

PS C:\> ("").GetType().Basetype
System.Object
```

Working with Strings

Literal Strings – Single Quotes

Task	**Example**
Create a variable	`PS C:\> $a = 123`
Include the variable in a literal string	`PS C:\> $b = 'As easy as $a'`

Notice that $a is not expanded	`PS C:\> $b`
	`As easy as $a`

Expandable Strings – Double Quotes

Task	**Example**
Create a variable	`PS C:\> $a = 123`
Include the variable in an expandable string	`PS C:\> $b = "As easy as $a"`

Notice that $a is expanded	`PS C:\> $b`
	`As easy as 123`

Here Strings

Simplify use of longer, more complex string assignments

- Here String can contain quotes, @ sign, etc.

```
Literal Here String

PS C:\> $lhere = @'
As
'easy'
as
$a
'@

PS C:\> $lhere
As
'easy'
as
$a
```

```
Expandable Here String

PS C:\> $ehere = @"
As
"easy"
as
$a
"@

PS C:\> $ehere
As
"easy"
as
123
```

Variable Subexpression

Within an expandable string, it might be necessary to display the results of an operation or a property of an object.

```
# Properties Not expanded
PS C:\> $a = Get-Service -Name BITS
PS C:\> $b = "$a.Name is $a.Status"
System.ServiceProcess.ServiceController.name is
System.ServiceProcess.ServiceController.status

# RIGHT WAY using Subexpression
PS C:\> $a = Get-Service -Name BITS
PS C:\> $b = "$($a.Name) is $($a.Status)"
BITS is Running

# This can also be used on any operation that you want to run in a string
PS C:\> $a = "Your Lucky Number is $(Get-Random)" # Get-Random gives you a
random number
PS C:\> $a
Your Lucky Number is 1023023027
```

Note the colorization. PowerShell is not processing the properties as part of the Expansion.

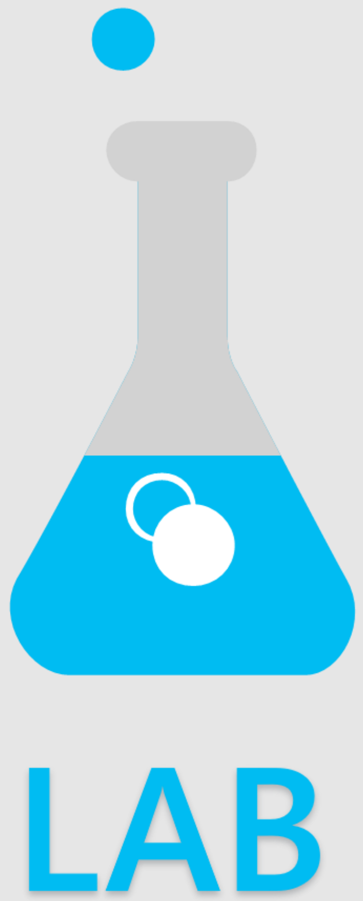
When a variable is expanded, the ToString method is called. Most objects default for ToString is to display their Type Name.

Demonstration: Strings, Here Strings, and Subexpression

Strings, Here Strings and Subexpression

Lab 2: Objects

60 minutes



Microsoft