Module 2

Objects

Create custom Objects

Extend existing Objects

Format Output

1

What is an Object?

Object definition and its elements

- **An object is an instance of particular datatype or class.**
  - Datatypes: Integer (int), Double, bool, etc.
  - Classes: program-code-template for creating objects
  - Instance: a specific realization of an object based on datatype or class
- **Objects consist of**
  - Variables
  - Methods
  - Events

2

How Objects are used in PS?

Working with existing objects in PS

- **Most cmdlets will return an object or object list**
  - get-service, get-process, get-item, get-aduser, …
- **Get-Member**
  - Object type
  - Members (Properties, Methods, …)
- ***-object cmdlets**
  - where-object: filtering object lists
  - select-object: selecting/reducing object or object lists
    - -first; -last; -unique; -skip
    - -property

```
get-process | select-object -property processname,handles
```

    - -expandproperty

```
get-process notepad.exe | select-object -expandproperty modules
```

  - sort-object: sorting object lists
  - group-object: group objects lists by a property

IT SERVICES & TRAINING

3

How Objects are used in PS?

Create a new object based on an existing data type or class

- **Variables are objects**

```
[int]$a = 123
$a.Gettype()
```

- **.Net Object**

```
Add-Type -AssemblyName System.Speech
$obj = New-Object -TypeName System.Speech.Synthesis.SpeechSynthesizer
$obj.Speak('there is a new mail.')
```

- **COM Object**

```
$ie = New-Object -ComObject InternetExplorer.Application
$ie.Navigate("http://www.microsoft.com")
$ie.visible = $true
```

- **Check**
  - get-member
  - *.gettype()

IT SERVICES & TRAINING

4

Create custom objects

How to create custom objects with properties and methods

- **Procedure 1 – detailed, comprehensive and PS-like**
  - Create a new object of type *psobject*
  - Add properties and methods

```
$myobj = New-Object -TypeName Psobject
Add-Member -InputObject $myobj `
           -MemberType 'NoteProperty' `
           -Name Greeting `
           -value "Hello World!"
```

  - Some Membertypes
    - *NoteProperty*
      - This kind of property can be set to any kind of value without validation.
    - *AliasProperty*
      - A second name for an other property.
    - *Scriptmethod*
      - PS-Script code which return a value or list.
    - *ScriptProperty*
      - Consists of two parts. First part is the get-part. This is a PS-Script code which returns a value. The second part is the set-part. This is also a PS-Script code but to set a noteproperty. A Scriptproperty could contain a validation or is able to set more than one noteproperty at the same time.

DIST IT SERVICES & TRAINING

Create custom objects

How to create custom objects with properties and methods

- **Procedure 2 – quick & simple**
  - Similar to extend an existing data type
  - Create a variable and pipe it to select-object

```
$myObj = 123 | select-object -property Name,Age
$myObj.Name = "Trainer"
$myObj.Age = 29
```

  - Gettype() returns the type PSCustomObject
  - Get-Member shows the initial type but extended by 'Selected.*'
    - Selected.System.Int32 e. g.

DIST IT SERVICES & TRAINING

Create custom objects

How to create custom objects with properties and methods

- **Procedure 3 – only for developers**
  - First create a class in c#
  - and then create a new object based on the new class

```
Add-Type -Language CSharp `
@"   public class Car
     {
         public string Color {get;set;}
         public int HorsePower {get;set;}
     } "@

$myCar = New-Object -TypeName Car
$myCar.Color = "Red"
$myCar.HorsePower = 140
$myCar
```

Creating custom objects

New keyword Class in PowerShell 5.0

- **Procedure 4 – requires PowerShell 5.0 and above**
  - Class Classname { … }
  - Within declare Properties and Methods
  - Characteristics:
    - All Members are public, but in a *.type.ps1xml hidden member possible
    - New-Object –typename Classname doesn't work
      - Use instead: [Classname]::new()
    - Classes can not be used outside the script/module
    - Methods different syntax from functions
      - If return is used a type of the return-value must be declared.
    - Method overloads are possible

Create custom objects

Example

```
Class myClass
{
    [String]$name
    [DateTime]$Birth

    SetDayOfBirth($value)
    {
       If ($value -is [DateTime]
       {
          $this.Birth = $value
       }
       else
       {
          Write-host “Please provide a DateTime as Argument.”
       }
    }
    [DateTime]GetDayOfBirth()
    {
       $this.$Birth
    }

    Age([ValidateSet("Year", "Days", "Exact")][String]$Unit = "Exact")
    {
       switch ($Unit)
       {
          "Year"
          { [math]::Round(((Get-date) - $this.Birth).Days/365 , 1)
            break }
          "Days"
          { ((Get-date) - $this.Birth).Days; break }
          "Exact"
          { ((Get-date) - $this.Birth); break }
       }
    }
}
```

IT SERVICES
& TRAINING

Extend existing Objects

How to add additional properties to exiting objects.

- **To extend objects create a ‘custom type extension’. It is handled by PowerShell’s Extension Type System (ETS).**
- **Requirements**
  - Existing objects
  - .Net or custom objects
  - Xml-file: *.types.ps1xml
  - Update-TypeData
- **Type extensions / Members**
  - AliasProperty
  - NoteProperty
  - ScriptProperty
  - ScriptMethod
  - PropertySet

```
<?xml version="1.0" encoding="utf-8" ?>
<Types>
   <Type>
        <Name>ObjectName</Name>
        <Members>
            …
        </Members>
   </Type>
</Types>
```

IT SERVICES
& TRAINING

Extend existing Objects - AliasProperty

How to extend an object with an AliasProperty

- *.ps1xml

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
     <AliasProperty>
       <Name>Byte</Name>
       <ReferencedMemberName>Length</ReferencedMemberName>
     </AliasProperty>
     <AliasProperty>
       <Name>Filetype</Name>
       <ReferencedMemberName>Extension</ReferencedMemberName>
     </AliasProperty>
   </Members>
</Type>
```

- Usage

```
Update-TypeData -AppendPath Name.types.ps1xlm -Confirm
Get-Childitem c:\windows\win.ini | Get-Member
(Get-Childitem c:\windows\win.ini).Byte
(Get-Childitem c:\windows\win.ini).FileType
```

DIST IT SERVICES & TRAINING

Extend existing Objects - NoteProperty

How to extend an object with a NoteProperty

- *.ps1xml

```
<Type>
   <Name>System.IO.DirectoryInfo</Name>
     <Members>
       <NoteProperty>
           <Name>Status</Name>
          <Value>Success</Value>
      </NoteProperty>
     </Members>
</Type>
```

- Usage

```
Update-TypeData -AppendPath Name.types.ps1xlm -Confirm
Get-Item c:\windows | Get-Member
(Get-Item c:\windows).Status
```

- **Note**
  - In this example the value of ‘Status’ of *each* directory is ‘Success’. It cannot be changed, it’s read-only.

DIST IT SERVICES & TRAINING

Extend existing Objects - ScriptProperty

How to extend an object with an ScriptProperty

- *.ps1xml

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
     <ScriptProperty>
       <Name>Encrypted</Name>
       <GetScriptBlock>
         $corFileName = $this.FullName -replace "\\","\\"
         (Get-CimInstance -Query ('Select Encrypted ↵
           From Cim_DataFile Where Name = "' + $corFileName ↵
           + '"')).Encrypted
       </GetScriptBlock>
     </ScriptProperty>
   </Members>
</Type>
```

- Usage

```
Update-TypeData -AppendPath Name.types.ps1xlm -Confirm
Get-Childitem c:\windows\win.ini | Get-Member
(Get-Childitem c:\windows\win.ini).Encrypted
```

DIST IT SERVICES & TRAINING

Extend existing Objects - ScriptMethod

How to extend an object with an ScriptMethod

- *.ps1xml

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
       <ScriptMethod>
         <Name>AppendID</Name>
         <Script>
           param ( [String]$ID )
           $this.BaseName + $id + $this.Extension
         </Script>
       </ScriptMethod>
   </Members>
</Type>
```

- Usage

```
Update-TypeData -AppendPath Name.types.ps1xlm -Confirm
Get-Childitem c:\windows\win.ini | Get-Member
(Get-Childitem c:\windows\win.ini).AppendID(“123”)
```

DIST IT SERVICES & TRAINING

Extend existing Objects - PropertySet

How to extend an object with an PropertySet

- *.ps1xml

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
     <PropertySet>
       <Name>myPropSet</Name>
       <ReferencedProperties>
         <Name>Fullname</Name>
         <Name>BaseName</Name>
       </ReferencedProperties>
     </PropertySet>
   </Members>
</Type>
```

- Usage

```
Update-TypeData -AppendPath Name.types.ps1xlm -Confirm
Get-Childitem c:\windows\win.ini | Get-Member
Get-Childitem c:\windows\win.ini | Select-Object myPropSet
```

IT SERVICES & TRAINING

Extending existing Objects – Code*

Difficult to extend objects by CodeProperty or CodeMethod

- CodePro
  - Both ref
  - Propert
  - Require
    - Proper
    - Metho

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
     <CodeProperty>
       <Name>customMode</Name>
       <GetCodeReference>
           <TypeName>Micosoft.PowerShell.Commands.FileSystemProvider</TypeName>
           <MethodName>Mode</MethodName>
       </GetCodeReference>
     </CodeProperty>
   </Members>
</Type>
```

- Examples
  - Microsoft.PowerShell.Commands.FileSystemProvider::Mode
  - about_t

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
     <CodeMethod>
       <Name>customMode</Name>
       <CodeReference>
           <TypeName>Micosoft.PowerShell.Commands.FileSystemProvider</TypeName>
           <MethodName>Mode</MethodName>
       </CodeReference>
     </CodeMethod>
   </Members>
</Type>
```

Extending existing Objects

You can hide some members

- **Attribute IsHidden="true"**
  - It allows you to hide some members
- **Concealable members**
  - AliasProperty
  - NoteProperty
  - ScriptProperty
  - PropertySet
- **Example**
- **Hint**
  - Gm –force shows also hidden members

```
<Type>
   <Name>System.IO.FileInfo</Name>
   <Members>
     <PropertySet IsHidden="true">
       <Name>myPropSet</Name>
       <ReferencedProperties>
         <Name>Fullname</Name>
         <Name>BaseName</Name>
       </ReferencedProperties>
     </PropertySet>
   </Members>
</Type>
```

```
Update-TypeData -AppendPath Name.tpyes.ps1xlm -Confirm
Get-Childitem c:\windows\win.ini | Get-Member -force
```

Default Display Properties

How to define the default properties for output.

- **Default display properties are defined by a**
  - MemberSet and
  - PropertySet
- **Mandatory names**
  - MemberSet/Name: PSStandardMembers
  - PropertySet/Name: DefaultDisplayPropertySet

```
<Type>
   <MemberSet>
     <Name>PSStandardMembers</Name>
     <Members>
       <PropertySet>
         <Name>DefaultDisplayPropertySet</Name>
         <ReferencedProperties>
           <Name>Fullname</Name>
           <Name>Basename</Name>
         </ReferencedProperties>
       </PropertySet>
     </Members>
   </MemberSet>
</Type>
```

Lab A

- **Type Extension for a 'Service'**
  - ScriptProperty: StartType
    - *Hint*: (Get-CimInstance -Query ('Select * From Win32_Service Where Name = "bits" ')).StartMode
  - ScriptMethod: Restart()
  - AliasProperty: Computer for MachineName
  - PropertySet: ServiceInfo - ServiceName, StartType, Status

Page ▪ 19

19

Custom Format View

- **Format View**
  - Definition for the output of objects or object lists
  - Saved in *.format.ps1xml
  - Default files in $PSHome
  - Custom files possible
- **Requirements**
  - *.format.ps1xml
  - Update format
- **Different Formats**
  - Table
  - List
  - Wide
  - Custom

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
   <ViewDefinitions>
     <View>
       …
     </View>
   </ViewDefinitions>
</Configuration>
```

```
Update-FormatData -PrependPath C:\myCustom.format.ps1xml
                  - or -
Update-FormatData -AppendPath C:\myCustom.format.ps1xml
```

Page ▪ 20

20

*.format.ps1xml

Structure of a custom view file

```
<?xml version="1.0" encoding="utf-8" ?>
<Configuration>
   <ViewDefinitions>
     <View>

       <Name>Name of the view</Name>

       <ViewSelectedBy>
         <TypeName>System.IO.FileInfo</TypeName>
       </ViewSelectedBy>

       <GroupBy> … </GroupBy>

       <TableControl> Definition for a table </TableControl>
         - or -
       <ListControl> Definition for a list </ListControl>
         - or -
       <WideControl> Definition for a single-property-list </WideControl>
         - or -
       <CustomControl> Definition for something else </CustomControl>

     </View>
   </ViewDefinitions>
</Configuration>
```

Page ▪ 21

<TableControl>

Structure of a table

```
<TableControl>

   <TableHeaders>

     <TableColumnHeader>
       <Label>Size</Label>
       <Alignment>right</Alignment>
     </TableColumnHeader>

     <TableColumnHeader>
       <Label>Name</Label>
     </TableColumnHeader>

   </TableHeaders>

   <TableRowEntries> … </TableRowEntries>

</TableControl>
```

Page ▪ 22

<TableControl>

Structure of a table

```
<TableControl>

   <TableHeaders> … </TableHeaders>

   <TableRowEntries>

     <TableRowEntry>

       <TableColumnItems>

         <TableColumnItem>
           <PropertyName>Length</PropertyName>
         </TableColumnItem>

         <TableColumnItem>
           <ScriptBlock> any PS Scriptscode </ScriptBlock>
         </TableColumnItem>

       </TableColumnItems>

     </TableRowEntry>

   </TableRowEntries>

</TableControl>
```

Page ▪ 23

23

<ListControl>

Structure of a list

```
<ListControl>
   <ListEntries>
     <ListEntry>
       <ListItems>

         <ListItem>
           <Label>Size</Label>
           <PropertyName>Length</PropertyName>
         </ListItem>
           …
         <ListItem>
           <Label>Mode</Label>
           <PropertyName>Mode</PropertyName>
           <ItemSelectionCondition>
             <ScriptBlock>$_.basename -notlike "my*"</ScriptBlock>
           </ItemSelectionCondition>
         </ListItem>

       </ListItems>
     </ListEntry>
   </ListEntries>
</ListControl>
```

Page ▪ 24

24

<WideControl>

Structure of a list

```
<WideControl>

   <WideEntries>
     <WideEntry>
       <WideItem>
         <PropertyName>Basename</PropertyName>
       </WideItem>
     </WideEntry>
   </WideEntries>

</WideControl>
```

Page ▪ 25

25

*.format.ps1xml for custom Objects

How to use a format file with a custom object

- **Challenge**
  - Structure and schema like discussed before
  - <ViewSelectedBy><TypeName> is what?
- **Name a custom object**
  - Default type name: *System.Management.Automation.PSCustomObject*
  - Define a custom type name

```
$myObject = New-Object -TypeName PSObject
$myObject.PSTypeNames.Insert(0,”Custom.TypeName”)
```

Page ▪ 27

27

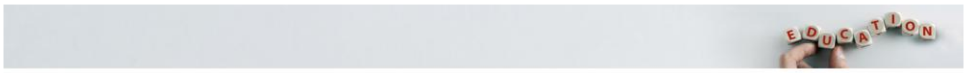
Links

- **Formatting file overview**
  - https://learn.microsoft.com/en-us/powershell/scripting/developer/format/formatting-file-overview?view=powershell-7.5
- **Formatting file concepts**
  - https://learn.microsoft.com/en-us/powershell/scripting/developer/format/formatting-file-concepts?view=powershell-7.5
- **Xml schema**
  - https://learn.microsoft.com/en-us/powershell/scripting/developer/format/configuration-element-format?view=powershell-7.5

Lab B

```
$OS = (Get-CimInstance -ClassName Win32_OperatingSystem)[0].Caption
$RAM =(Get-CimInstance -ClassName Win32_ComputerSystem)[0].TotalPhysicalMemory
$Uptime = (Get-CimInstance -ClassName Win32_OperatingSystem)[0].LastBootUptime
```

- **Exercise 1: Create a custom object**
  - Name: aPS.ComputerInfo
  - Properties: OS, RAM, Uptime
  - Methods: GetUptime(Unit Minutes or Hours)
- **Exercise 2: Create a formatting file**
  - For aPS.ComputerInfo
  - Table: RAM, OS, Uptime
- **Exercise 3: Extend object**
  - System.Diagnostics.Process
  - Aliasproperty: Computer for MachineName
  - ScriptProperty: Uptime



Do You Have Any Questions?