Add a Parent OU Path to Your Active Directory Objects

mattmcnabb.github.io/add-a-parent-ou-path-to-your-active-directory-objects

One question I have seen quite often in PowerShell forums is “How do I find the parent OU of a user/computer?” Turns out this isn’t immediately available for your objects. I typically use a string replace to infer the OU path from the distinguishedname of the object like this:

```
$OU = Get-ADComputer MattsPC |
Select-Object @{
n='ParentContainer'
e={$_.distinguishedname -replace "CN=$($_.cn),"}
}
```

view raw 1.ps1 hosted with ❤ by GitHub
This works just fine, but what if you need this data outside of your scripts? It might be useful for this to always be included in your active directory queries.

PowerShell’s Extensible Type System

PowerShell’s type system is nothing if not flexible. Known as ETS, the Extensible Type System, it allows you to modify the data types that are output on the fly and create new structures that fit your needs better than what is provided. Prior to PowerShell version 3.0 this was only possible using XML files, but now we can extend our data types using native PowerShell cmdlets.

There are 3 cmdlets included with PowerShell version 3.0 and up that work with the ETS: `Get-TypeData`, `Update-TypeData` and `Remove-TypeData`. For our purposes we are going to focus on `Update-TypeData` which will allow us to add new properties and methods into our PowerShell objects.

How it Works

First we need to define what it is that we want to add to our objects. In this case we want a `Path` property to be available whenever we use cmdlets like `Get-ADUser` or `Get-ADComputer`. Next we need to define what data types are involved. We can use the `GetType` method to see what types are returned by a cmdlet:

```
PS> (Get-ADUser Matt).GetType()

IsPublic IsSerial Name      BaseType
-------- -------- ----      --------
True     False    ADUser    Microsoft.ActiveDirectory.Management.ADAccount
```

Notice the `Name` field is `ADUser`. While this is a valid type this would only cover users and we want to add a Path parameter to both users and computers. In the output above we can see that the `ADUser` type is derived from the `ADAccount` base type. Let's see if computers are the same:

```
PS> (get-adcomputer MattsPC).gettype()

IsPublic IsSerial Name             BaseType
-------- -------- ----             --------
True     False    ADComputer       Microsoft.ActiveDirectory.Management.ADAccount
```

Looks like a winner! You can see that computer objects are derived from the same .NET base type.

Now let's take a look at how we can use Update-TypeData to add our `Path` parameter:

```
$Splat = @{
TypeName = 'Microsoft.ActiveDirectory.Management.ADAccount'
MemberType = 'ScriptProperty'
MemberName = 'Path'
Value = {$this.DistinguishedName -replace "CN=$($this.Name),"}
}
Update-TypeData @Splat
```

view raw 2.ps1 hosted with ❤ by GitHub
Note: I used splatting here to pass parameters to the cmdlet. This was done to improve readability of the code on this page and you can provide the parameters to explicitly to the cmdlet if you prefer. If you aren't familiar with splatting, read more about it here.

Now when you run `Get-ADUser` or `Get-ADComputer`, the output will include your `Path` property:

```
PS> Get-ADUser Matt

DistinguishedName : CN=Matthew McNabb,OU=Admins,DC=domain,DC=com
Enabled           : True
GivenName         : Matt
Name              : Matt McNabb
ObjectClass       : user
ObjectGUID        : 964ff8c6-7872-41ec-b46e-9008344e1182
SamAccountName    : matt
SID               : S-1-5-21-1606980848-362388127-725345643-23774
Surname           : McNabb
UserPrincipalName : matt@domain.com
Path              : OU=Admins,DC=domain,DC=com
```

Making it Stick

The only problem with this approach is that the new type data is dynamic - it only exists in the current PowerShell session. The help for `Update-TypeData` states this:

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.

When it says “into memory,” you can interpret that to mean “temporary.” So how do we make this work for us whenever we use PowerShell to get information on our Active Directory objects? You’ll have to add to add the command into your PowerShell profile. You can find more info on setting up your profile here. Another approach to this would be to add the `Update-TypeData` cmdlet to a custom module so that any time the module is imported your object data will be updated with your custom property.

Hopefully this has been a useful introduction to PowerShell’s Extensible Type System. I’m sure you can think of many more uses for this trick that will help you get your work done more easily!