Basics

Expanding objects

Much of the data we receive from cmdlets are objects that require further manipulation to get to the data we're looking for.

```
# Connect to Microsoft Graph
Connect-MgGraph -Scopes "Mail.Read"

# Define the user and folder details
$userId = "user@domain.com"
$mailFolderId = "folder-id-here"

# Get the messages in the specified folder
$messages = Get-MgUserMailFolderMessage -UserId $userId -MailFolderId $mailFolderId

# Format the output to expand sender, from, and include subject
$messages | Select-Object -Property Id,ReceivedDateTime,From,Sender,Subject | Format-List
```

Below is example output of the above script:

```
# example output without object expansion
# note the un-expanded values for From and Sender
Id                   : [REDACTED_GUID]
ReceivedDateTime     : MM/DD/YYYY 2:52:35 AM
Subject              : [REDACTED_SUBJECT]
From                 : Microsoft.Graph.PowerShell.Models.MicrosoftGraphRecipient
Sender               : Microsoft.Graph.PowerShell.Models.MicrosoftGraphRecipient
```

The actual data we're looking for related to From and Sender is in the following expansions:

```
From.EmailAddress.Name
From.EmailAddress.Address
Sender.EmailAddress.Name
Sender.EmailAddress.Address
```

Below is an example of how to accomplish that in the Select-Object statement:

```
# Connect to Microsoft Graph
Connect-MgGraph -Scopes "Mail.Read"

# Define the user and folder details
$userId = "user@domain.com"
$mailFolderId = "folder-id-here"

# Get the messages in the specified folder
$messages = Get-MgUserMailFolderMessage -UserId $userId -MailFolderId $mailFolderId

# Format the output to expand sender, from, and include subject
$messages | Select-Object -Property Subject,
             @{Name="SenderName";Expression={$_.Sender.EmailAddress.Name}},
             @{Name="SenderEmail";Expression={$_.Sender.EmailAddress.Address}},
             @{Name="FromName";Expression={$_.From.EmailAddress.Name}},
             @{Name="FromEmail";Expression={$_.From.EmailAddress.Address}} |
             Format-List
```

The output is not actual useful data, rather than the name of the object type that was returned:

```
Id                : [REDACTED_GUID]
ReceivedDateTime  : MM/DD/YYYY 2:52:35 AM
Subject           : Weekly Duo Report
SenderName        : Security Team
SenderEmail       : security@domain.com
FromName          : Security Team
FromEmail         : security@domain.com
```

Using variables in -Filter statements

Example of using a variable in a -Filter statement with the Get-ADGroup cmdlet. Note this cmdlet doesn't properly throw exceptions that can be handled by a try / catch block, so we have to use a Filter statement and check to see if anything was returned.

```
# Assign group name we're looking for to a variable
$GroupName = "Administrators"

# Get-ADGroup doesn't throw exceptions properly, so we have to work around this since we can't use try / catch
$Group = Get-ADGroup -Filter "Name -eq '$GroupName'" -Properties members

If ($Group -eq $null) {
  "# $($GroupName) - group not found!!!"
} else {
  "# $($GroupName)"
}
```

Revision #4
Created 8 November 2023 14:51:43 by bluecrow76
Updated 26 November 2024 17:10:53 by bluecrow76