

dushyantgill

exploring identity technologies and publishing the tools I build

[AUTHN AUTHZ.COM](#) [CONTACT](#) [TOOLS](#)

AADGraphPowerShell: a PowerShell client for Windows Azure AD Graph API

Graph API provides programmatic access to Windows Azure Active Directory (AD) through RESTful endpoints. Cloud services use the Graph API to query information about AAD users that isn't present in the token (like group membership data, manager employee relationship, user's address book properties like telephone number, office location, job title etc.). It is analogous to the LDAP APIs supported by the on-premises Active Directory technologies.

I have often felt the need of having a generic client that I can quickly tweak, to CRUD against the Graph API – I haven't found such a client yet – and the only alternative for me is to write a small C# program. I have been a fan of PowerShell for bringing the power of .Net programming to scripting. So, over these holidays I decided to write a PowerShell client (some generic functions and Cmdlets) that I can use to perform simple CRUD operations on Window Azure AD Graph API. PowerShell scripts are easy to distribute – and the script modules feature + advanced functions feature introduced in PowerShell v2 make it pretty simple to write and distribute scripts functions that behave pretty much like regular cmdlets. So I opted for a script module instead of a .net assembly to package the cmdlets. I do however need to find a way to split the module into multiple files (the module is already ~500 lines long).

By the way, the Official [Windows Azure AD PowerShell](#) client doesn't yet expose all the entities/types that are exposed via the Graph API. When it does start to work with all the Graph API entities, I will not need this module anymore.

Using the Module

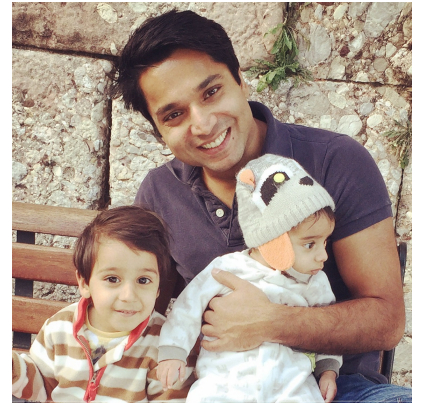
1. Get the AADGraph.psm1 file from here: <https://github.com/dushyantgill/AADGraphPowerShell/blob/master/AADGraph.psm1>. If you find bugs (you will) – either report them using the comments section or even feel free to fix it in the github project.

2. Run PowerShell as the Administrator. Run the following command to enable script execution:

```
Set-ExecutionPolicy RemoteSigned -Force
```

Dushyant

is an engineer working at



[Microsoft](#), living in the greyish-green northwest of US with his wife, sons and dog. Grew up in the Indian Airforce as the country was [transforming](#) from a quasi-communist nation to the lively jumble it is now. In 7th grade learnt [Pascal](#) from dad, has been a fan of computer programming since. Can be reached at [@dushyantgill](#) or via the contact page of this website.

This is a personal blog,

nothing here represents the opinions of my employer.

Categories

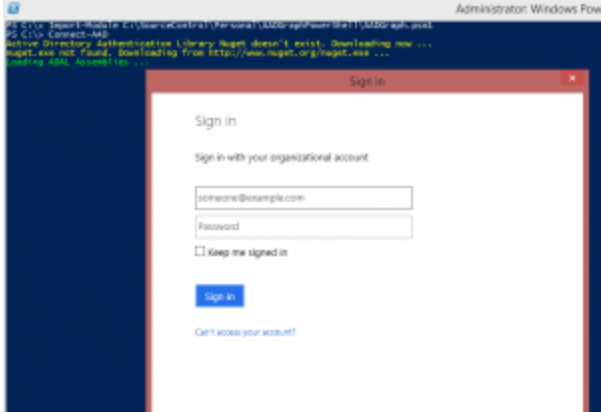
[authNauthZ.com](#)
[Authorization](#)

3. Import the Module. Replace the Path below with the location where you have downloaded and saved the AADGraph.psm1 file.

Import-Module C:\SourceControl\Personal\AADGraphPowerShell\AADGraph

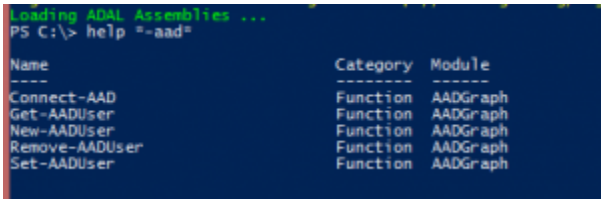
4. Run Connect-AAD and authenticate. When run for the first time this will download nuget.exe from <http://www.nuget.org> and install the ADAL.net nuget. It will then load the ADAL.net assemblies as authenticate the user.

Connect-AAD



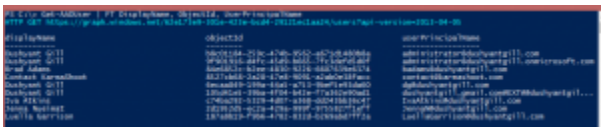
5. List the Cmdlets that are exported by the AADGraph module

Get-Help *-AAD*

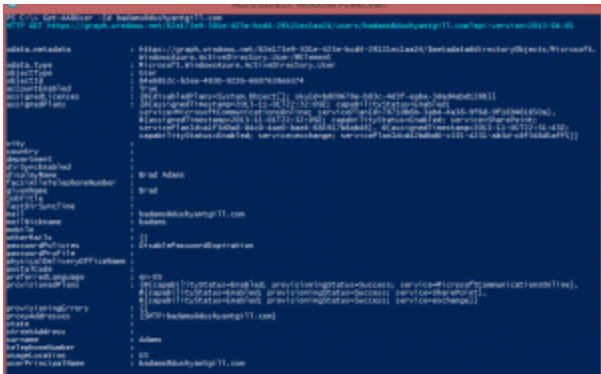


6. Play around ...

Get-AADUser | FT DisplayName, ObjectId, UserPrincipalName



Get-AADUser -Id badams@dushyantgill.com



Set-AADUser -Id badams@dushyantgill.com -jobTitle "Program Manager"

Azure

Azure AD

Azure AD Graph API

Azure AD Integrated Applications

Azure Resource Manager

PowerShell

Roles-Based Access Control

SAML

Single Sign-On

Uncategorized

Archive

October 2015 (1)

May 2015 (1)

April 2015 (1)

February 2015 (2)

January 2015 (2)

December 2014 (2)

April 2014 (3)

March 2014 (1)

January 2014 (2)

December 2013 (4)

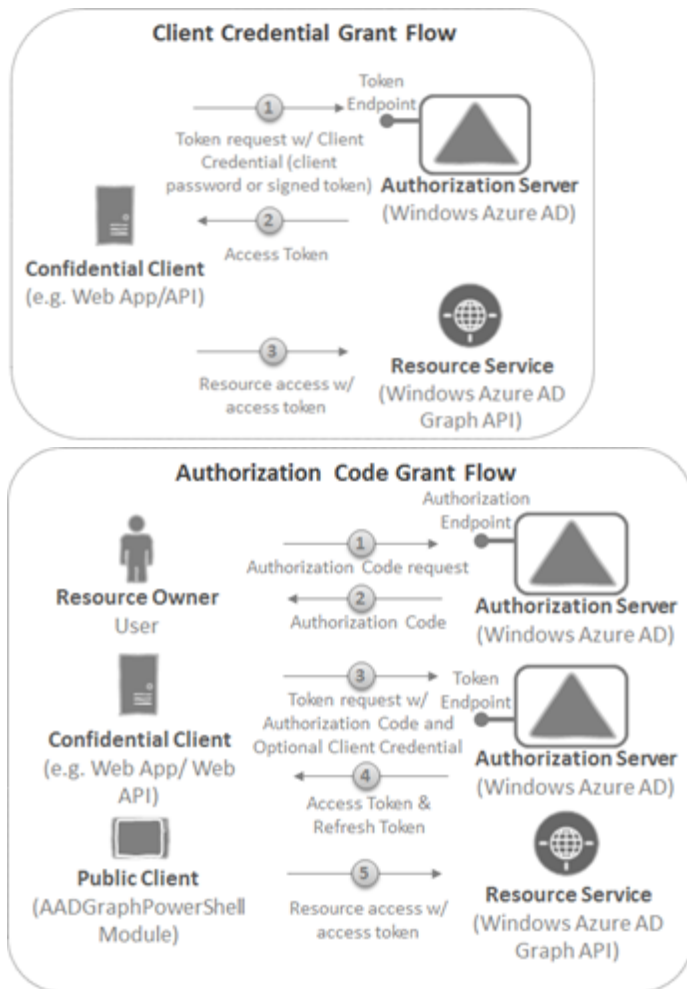
```
New-AADUser -accountEnabled $true -displayName "Test User 1" -mail
```

Remove-AADUser 0c8fa2d9-c5c4-46ef-998a-56e36574c9b8

Alright, now let's take a closer look at this PowerShell module.

Authentication

An application can be allowed to access Windows Azure AD Graph API in a couple of ways: 1) High-privileged direct access: where the application authenticates with Windows Azure AD using either client credentials or a signed assertion and receives an App only access token to access the Graph API 2) Delegated access: where the application accesses the Graph API on behalf of a Windows Azure AD user. Public clients (aka native client applications) the ones that run on the users device, like this PowerShell module can only access Windows Azure AD Graph API using the second method i.e. Authorization Code Grant Flow.



To make it easy for native client applications to execute the auth flows, my team provides [SDK libraries](#) called [ADAL](#) (Active Directory Authentication Library). This PowerShell module employs the [.Net ADAL library](#).

LOADING ADAL NUGET (LOAD-ACTIVEDIRECTORYAUTHENTICATIONLIBRARY)

```
function Load-ActiveDirectoryAuthenticationLibrary(){
    $mydocuments = [environment]::getfolderpath("mydocuments")
    if(-not (Test-Path ($mydocuments+"\Nugets"))){New-Item -Path ($
    $adalPackageDirectories = (Get-ChildItem -Path ($mydocuments+"\N
    if($adalPackageDirectories.Length -eq 0){
        Write-Host "Active Directory Authentication Library Nug
        if(-not(Test-Path ($mydocuments + "\Nugets\nuget.exe"))
        {
            Write-Host "nuget.exe not found. Downloading from http://www
            $wc = New-Object System.Net.WebClient
            $wc.DownloadFile("http://www.nuget.org/nuget.exe",$mydocumen
        }
        $nugetDownloadExpression = $mydocuments + "\Nugets\nuget.exe i
        Invoke-Expression $nugetDownloadExpression
    }
    $adalPackageDirectories = (Get-ChildItem -Path ($mydocuments+"\N
    $ADAL_Assembly = (Get-ChildItem "Microsoft.IdentityModel.Clients
    $ADAL_WindowsForms_Assembly = (Get-ChildItem "Microsoft.Identity
    if($ADAL_Assembly.Length -gt 0 -and $ADAL_WindowsForms_Assembly.
        Write-Host "Loading ADAL Assemblies ..." -ForegroundColor Gree
        [System.Reflection.Assembly]::LoadFrom($ADAL_Assembly.FullName
        [System.Reflection.Assembly]::LoadFrom($ADAL_WindowsForms_Asse
        return $true
    }
    }
    else{
        Write-Host "Fixing Active Directory Authentication Library pac
        $adalPackageDirectories | Remove-Item -Recurse -Force | Out-Nu
```

```

Write-Host "Not able to load ADAL assembly. Delete the Nugets
return $false
}
}

```

The Load-ActiveDirectoryAuthenticationLibrary function first checks the presence of ADAL.Net libraries in the Nugets folder in user's My Documents folder. If it doesn't find the library it downloads nuget.exe from <http://www.nuget.org> and installs the Microsoft.IdentityModel.Clients.ActiveDirectory nuget. It then Loads the two assemblies that make up the ADAL.net SDK i.e. Microsoft.IdentityModel.Clients.ActiveDirectory.dll and Microsoft.IdentityModel.Clients.ActiveDirectory.WindowsForms.dll.

ACQUIRING ACCESS TOKEN (GET-AUTHENTICATIONRESULT)

```

function Get-AuthenticationResult(){
    $clientId = "1950a258-227b-4e31-a9cf-717495945fc2"
    $redirectUri = "urn:ietf:wg:oauth:2.0:oob"
    $resourceClientId = "00000002-0000-0000-c000-000000000000"
    $resourceAppIdURI = "https://graph.windows.net"
    $authority = "https://login.windows.net/common"

    $authContext = New-Object "Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext"
    $authResult = $authContext.AcquireToken($resourceAppIdURI, $clientId)
    return $authResult
}

```

The Get-AuthenticationResult function calls the AcquireToken method provided by ADAL.net SDK to authenticate the user and returns the authentication results that contains the access token to access the Graph API. In this example, the AuthenticationContext is instantiated using the "well-known" client id and redirect URI of the Windows Azure PowerShell client. The resourceAppIdURI is the App ID URI of Windows Azure AD Graph API, and the authority URL is the Windows Azure AD OAuth Authorize endpoint. This function is so simple because ADAL is doing all the heavy work of executing the Authorization Code Grant Flow, authenticating the user and getting the access token.

CRUD Functions

My goal is to have CRUD cmdlets for all Graph API entities (for instance New-AADUser, Set-AADApplication etc.). These cmdlets will simply invoke the appropriate RESTful method on the Graph API for the specific entity:

- Get-AAD* cmdlets will invoke an HTTP GET on either https://graph.windows.net/{tenant}/{entity_type} or https://graph.windows.net/{tenant}/{entity_type}/{entity_id}
- New-AAD* cmdlets will invoke POST on https://graph.windows.net/{tenant}/{entity_type}
- Set-AAD* cmdlets will invoke PATCH on https://graph.windows.net/{tenant}/{entity_type}/{entity_id}
- Remove-AAD* cmdlets will invoke DELETE on https://graph.windows.net/{tenant}/{entity_type}/{entity_id}

There is common functionality in these CRUD functions that I decided to encapsulate in the following generic CRUD functions:

GET-AADOBJECT

```
function Get-AADObject([string]$type) {
    $objects = $null
    if($authenticationResult -ne $null){
        $header = $authenticationResult.CreateAuthorizationHeader()
        $uri = [string]::Format("https://graph.windows.net/{0}/{1}?api
        Write-Host HTTP GET $uri -ForegroundColor Cyan
        $result = Invoke-RestMethod -Method Get -Uri $uri -Headers @{"
        if ($result -ne $null) {
            $objects = $result.Value
        }
    }
    else{
        Write-Host "Not connected to an AAD tenant. First run Connect-
    }
    return $objects
}
```

The Get-AADObject function receives as input the entity type e.g. “users” or “applications”. It then constructs the entity URI and employs the Invoke-RestMethod base PowerShell cmdlet to invoke an HTTP GET on Graph API. The function retrieves the access token from the global authentication result variable and adds an Authorization header to the REST method call. Invoke-RestMethod cmdlets does the heavy lifting of creating an HTTPClient invoking the API and converting the returned JSON into objects that the Get-AADObject emits on the PowerShell pipeline.

GET-AADOBJECTBYID

```
function Get-AADObjectById([string]$type, [string]$id) {
    $object = $null
    if($global:authenticationResult -ne $null){
        $header = $authenticationResult.CreateAuthorizationHeader()
        $uri = [string]::Format("https://graph.windows.net/{0}/{1}/{2}
        Write-Host HTTP GET $uri -ForegroundColor Cyan
        $object = Invoke-RestMethod -Method Get -Uri $uri -Headers @{"
    }
    else{
        Write-Host "Not connected to an AAD tenant. First run Connect-
    }
    return $object
}
```

The Get-AADObjectById is similar to Get-AADObject except that it invokes HTTP GET on a specific Graph API object. I could have combined the two into a single cmdlet (I might in the future).

NEW-AADOBJECT

```
function New-AADObject([string]$type, [object]$object) {
    $newObject = $null
    if($global:authenticationResult -ne $null) {
        $header = $authenticationResult.CreateAuthorizationHeader()
        $uri = [string]::Format("https://graph.windows.net/{0}/{1}?api
        Write-Host HTTP POST $uri -ForegroundColor Cyan
        $enc = New-Object "System.Text.ASCIIEncoding"
```



```

$body = ConvertTo-Json -InputObject $object
Write-Host $body -ForegroundColor Cyan
$byteArray = $enc.GetBytes($body)
$contentLength = $byteArray.Length
$headers = @{"Authorization"=$header;"Content-Type"="applicati
$newObject = Invoke-RestMethod -Method Post -Uri $uri -Headers
}
else{
    Write-Host "Not connected to an AAD tenant. First run Connect-
}
return $newObject
}

```

New-AADObject invokes the POST method, with the new object data passed as Json text in the body of the POST message. The function receives the new object data in the form of the \$object parameter and employs the ConvertTo-Json base PowerShell cmdlet to serialize the object into a Json string. It then adds the Content-Length header (in addition to Authorization and Content-Type headers) and invokes the POST method. On successful completion of a POST method Graph API returns the newly created object back; the function emits this object on the pipeline.

SET-AADOBJECT

```

function Set-AADObject([string]$type, [string]$id, [object]$object
$updatedObject = $null
if($global:authenticationResult -ne $null) {
    $header = $authenticationResult.CreateAuthorizationHeader()
    $uri = [string]::Format("https://graph.windows.net/{0}/{1}/{2}
    Write-Host HTTP PATCH $uri -ForegroundColor Cyan
    $enc = New-Object "System.Text.ASCIIEncoding"
    $body = ConvertTo-Json -InputObject $object
    Write-Host $body -ForegroundColor Cyan
    $byteArray = $enc.GetBytes($body)
    $contentLength = $byteArray.Length
    $headers = @{"Authorization"=$header;"Content-Type"="applicati
    $newObject = Invoke-RestMethod -Method Patch -Uri $uri -Header
}
else{
    Write-Host "Not connected to an AAD tenant. First run Connect-
}
return $updatedObject
}

```

Set-AADObject is similar to the New-AADObject except that it invokes the PATCH method on a specific object.

REMOVE-AADOBJECT

```

function Remove-AADObject([string]$type, [string]$id) {
    $deleteResult = $null
    if($global:authenticationResult -ne $null) {
        $header = $authenticationResult.CreateAuthorizationHeader()
        $uri = [string]::Format("https://graph.windows.net/{0}/{1}/{2}
        Write-Host HTTP DELETE $uri -ForegroundColor Cyan
        $headers = @{"Authorization"=$header;"Content-Type"="applicati
        $deleteResult = Invoke-RestMethod -Method Delete -Uri $uri -He
    }
    else{
        Write-Host "Not connected to an AAD tenant. First run Connect-
    }
    return $deleteResult
}

```

Remove-AADObject simply invokes HTTP DELETE on a specific object and returns the result.

Alright! We now have the functions that will form the basic building blocks for our specialized cmdlets.

Exported Cmdlets

PowerShell allows you to export (in the form of Cmdlets) specific functions from a module. I chose to keep the above generic *-AADObject functions private and export only the specialized CRUD functions for the AAD entities (plus one more function i.e. Connect-AAD)

CONNECT-AAD

```
function Connect-AAD {
    PROCESS {
        $global:authenticationResult = $null
        if(Load-ActiveDirectoryAuthenticationLibrary)
        {
            $global:authenticationResult = Get-AuthenticationResult
        }
    }
}
```

All that Connect-AAD does is that it calls the Get-AuthenticationResult method and sets a global variable \$global:authenticationResult with the return value. Connect-AAD serves to authenticate the user as well as refresh the access token if the token expires during the use of the PowerShell session.

Specialized CRUD Cmdlets

As you can imagine, there will be many specialized CRUD cmdlets in the module, so I'll just cover the specialized cmdlets for one entity type (User).

GET-AADUSER

```
function Get-AADUser {
    [CmdletBinding()]
    param (
        [parameter(Mandatory=$false,
            ValueFromPipeline=$true,
            HelpMessage="Either the ObjectId or the UserPrincipalName of t
            [string]
            $Id
        )
    PROCESS {
        if($Id -ne "") {
            Get-AADObjectById -Type "users" -Id $Id
        }
        else {
            Get-AADObject -Type "users"
        }
    }
}
```

The Get-AADObject and Get-AADObjectById generic functions do the work of calling the RESTful Graph APIs. Get-AADUser (and other Get-AAD* cmdlets in this module) are simple cmdlets that leverage these generic functions. The function definition is

more formal, with the parameter attribute defining whether the Id parameter is mandatory, as well as providing a HelpMessage for the parameter.

NEW-AADUSER

```
function New-AADUser {
    [CmdletBinding()]
    param (
        [parameter(Mandatory=$false, ValueFromPipelineByPropertyName=$true,
            HelpMessage="Controls whether the new user account is created",
            [string]
            $accountEnabled = $true,

        [parameter(Mandatory=$true, ValueFromPipelineByPropertyName=$true,
            HelpMessage="The name displayed in the address book for the user",
            [string]
            $displayName,

        [parameter(Mandatory=$true, ValueFromPipelineByPropertyName=$true,
            HelpMessage="-----SNIP-----",
            [string]
            $usageLocation

    )
    PROCESS {
        # Mandatory properties of a new User
        $newUserPasswordProfile = "" | Select password, forceChangePasswordNextLogin
        $newUserPasswordProfile.password = $password
        $newUserPasswordProfile.forceChangePasswordNextLogin = $forceChangePasswordNextLogin

        $newUser = "" | Select accountEnabled, displayName, mailNickname
        $newUser.accountEnabled = $accountEnabled
        $newUser.displayName = $displayName
        $newUser.mailNickname = $mailNickname
        $newUser.passwordProfile = $newUserPasswordProfile
        $newUser.userPrincipalName = $userPrincipalName

        #Optional parameters/properties
        foreach($psbp in $PSBoundParameters.GetEnumerator()){
            $key = $psbp.Key
            $value = $psbp.Value
            if($key -eq "city" -or $key -eq "country" -or $key -eq "department" -or $key -eq "givenName" -or $key -eq "jobTitle" -or $key -eq "mailNickname" -or $key -eq "passwordPolicies" -or $key -eq "physicalDeliveryOfficeLocation" -or $key -eq "state" -or $key -eq "streetAddress" -or $key -eq "surname")
            {
                Add-Member -InputObject $newUser -MemberType NoteProperty $key $value
            }
        }

        New-AADObject -Type users -Object $newUser
    }
}
```

New-AADObject has many many parameters – so I snipped much of the parameter definition above. The New-AADObject cmdlet receives the displayName, mailNickName, userPrincipalName and the password as mandatory parameters. It assigns default values to the other required properties of a new user object in AAD (i.e. accountEnabled and forceChangePasswordNextLogin).

Base PowerShell has an awesome cmdlet Add-Member, that adds properties to dynamic objects on the fly. This cmdlet uses the Add-Member to create a dynamic object \$newUser and add to it the properties that have been specified on the command line for the new User. It then calls the generic function New-AADObject that does the work of transforming the \$newObject into Json string and invoking a POST method on <https://graph.windows.net/{tenant}/users>.

You'll notice that the New-AADUser cmdlet decorates all its parameters with the ValueFromPipelineByPropertyName=\$true property of the parameter attribute. This allows the cmdlet to bind the value for that parameter from a property (of the same name as the parameter) of an object from the pipeline. So if there is a CSV with data for say 1000 new user objects we will be able to do something like:

```
Import-CSV "CSVFileWith1000Users.csv" | New-AADUser
```

Ah! that is if Windows Azure AD doesn't throttle us in the middle of that – I shall leave this though here – it is worthy of another blog post.

SET-AADUSER

```
function Set-AADUser {
    [CmdletBinding()]
    param (
        [parameter(Mandatory=$true,
            ValueFromPipeline=$true,
            HelpMessage="Either the ObjectId or the UserPrincipalName of t
            [string]
            $Id,

        [parameter(Mandatory=$false,
            HelpMessage="Controls whether the user account is enabled or d
            [string]
            $accountEnabled,

        [parameter(Mandatory=$false,
            HelpMessage="The name displayed in the address book for the us
            [string]
            $displayName,

        [parameter(Mandatory=$false,
            ----SNIP-----
            [string]
            $usageLocation
    )
    PROCESS {
        $updatedUser = New-Object System.Object

        foreach($psbp in $PSBoundParameters.GetEnumerator()){
            $key = $psbp.Key
            $value = $psbp.Value
            if($key -eq "accountEnabled" -or $key -eq "displayName" -or :
            $key -eq "city" -or $key -eq "country" -or $key -eq "departm
            $key -eq "givenName" -or $key -eq "jobTitle" -or $key -eq "m
            $key -eq "passwordPolicies" -or $key -eq "physicalDeliveryOf
            $key -eq "state" -or $key -eq "streetAddress" -or $key -eq "
                Add-Member -InputObject $updatedUser -MemberType NotePrope
            }
        }
        if($PSBoundParameters.ContainsKey('password')){
            $updatedUserPasswordProfile = "" | Select password, forceCha
            $updatedUserPasswordProfile.password = $PSBoundParameters['p
            $updatedUserPasswordProfile.forceChangePasswordNextLogin = $
            $updatedUser.passwordProfile = $updatedUserPasswordProfile
        }

        Set-AADObject -Type users -Id $Id -Object $updatedUser
    }
}
```

Set-AADUser is very similar to New-AADUser, except that it does not bind parameter values from the pipeline. This cmdlet too creates a dynamic object \$updatedUser and adds properties to it using Add-Member. Like New-AADUser it handles the PasswordProfile properties in a special way. It finally calls Set-AADObject and passes

in the Id of the user to be updates and the properties that need to be updated in the form of \$updatedUser.

REMOVE-AADUSER

```
function Remove-AADUser {
    [CmdletBinding()]
    param (
        [parameter(Mandatory=$true,
            ValueFromPipeline=$true,
            HelpMessage="Either the ObjectId or the UserPrincipalName of t
            [string]
            $Id
        )
    ]
    PROCESS {
        Remove-AADObject -Type "users" -Id $id
    }
}
```

Remove-AADUser simply invokes the Remove-AADObject generic function.

So, we have here a PowerShell script module that has building block functions that you can use to invoke the Windows Azure AD Graph APIs and also use these generic functions to quickly write specialized cmdlets for AAD Graph API entities (as I have done for AADUser) ...

Enjoy!

By dushyantgill | December 27, 2013 |

Azure AD, Azure AD Graph API, PowerShell | 14 Comments |

← Authorizing access in your cloud service using Windows Azure AD groups

Team's access to Twitter account without sharing account password →

14 thoughts on “AADGraphPowerShell: a PowerShell client for Windows Azure AD Graph API”



Konsole

August 29, 2014 at 1:06 pm

Getting the following error:

Invoke-RestMethod : The 'Content-Type' header must be modified using the appropriate property or method.

Parameter name: name

At Q:\test.ps1:33 char:15

+ \$object = Invoke-RestMethod -Method Get -Uri \$uri -Headers

@{"Authorization" ...

+

~~~~~

~~~~~

```
+ CategoryInfo : NotSpecified: (:) [Invoke-RestMethod], ArgumentException
+ FullyQualifiedErrorId :
System.ArgumentException,Microsoft.PowerShell.Commands.InvokeRestMethodCommand
```

Reply ↓



Ken H

September 5, 2014 at 2:06 pm

This looks awesome. However, I get an error when I invoke connect-aad after importing the module and bringing in ADAL with nugget. What am I missing? I'm on Windows 8.1.

```
PS v4 > Connect-AAD
Multiple ambiguous overloads found for "AcquireToken"
and the argument count: "3".
At
C:\Windows\system32\WindowsPowerShell\v1.0\Modules\AADGraph\AADGraph.psm1:42 char:3
+ $authResult =
$authContext.AcquireToken($resourceAppIdURI,
$clientId, $redirect ...
+
~~~~~
~~~~~
+ CategoryInfo          : NotSpecified: (:) [],
MethodException
+ FullyQualifiedErrorId : MethodCountCouldNotFindBest
```

Reply ↓



Randy

September 5, 2014 at 6:23 pm

I get this error when I execute Connect-AAD

```
PS C:\graph> Connect-AAD $cred
Multiple ambiguous overloads found for "AcquireToken" and the argument
count: "3".
At C:\graph\AADGraph.psm1:42 char:3
+ $authResult = $authContext.AcquireToken($resourceAppIdURI, $clientId,
$redirect ...
+
~~~~~
~~~~~
+ CategoryInfo          : NotSpecified: (:) [], MethodException
+ FullyQualifiedErrorId : MethodCountCouldNotFindBest
```

Reply ↓

Simon Thorpe



October 1, 2014 at 7:52 pm

I am running 4.0 PowerShell and was able to successfully import your CMDlets. However, when I attempt to connect, I get the error below. Any idea what is wrong? I also have the Windows Azure AD Module for Windows Powershell installed. I have a suspicion that is causing the conflict. Is there a way to resolve this?

Multiple ambiguous overloads found for "AcquireToken" and the argument count: "3".

At

C:\Users\sthorpe\Documents\WindowsPowerShell\Modules\AADGraph\AADGraph.psm1:42 char:3

```
+ $authResult = $authContext.AcquireToken($resourceAppIdURI, $clientId, $redirect ...
```

+

~~~~~  
~~~~~

+ CategoryInfo : NotSpecified: (:) [], MethodException

+ FullyQualifiedErrorId : MethodCountCouldNotFindBest

Reply ↓



dushyantgill Post author

February 19, 2015 at 7:30 am

Sorry about the super late reply – the newer version of ADAL broke my code. I have fixed it to install only a specific version of ADAL. Please download, install and try again.

Reply ↓



Eric

February 19, 2015 at 6:16 am

Dusyant, excellent information, really appreciate all the info you share etc. I am running into a interesting issue when trying to install the AADGraph powershell solution.

When I run the Install-AADGraphModule.ps1 script, I get the following:

```
PS C:\Install\AAD> .\Install-AADGraphModule.ps1
```

Creating module directory under

c:\users\eric\Documents\WindowsPowerShell\Modules

Installing Active Directory Authentication Library Nuget in

c:\users\eric\Document

s\WindowsPowerShell\Modules\AADGraph\Nugets

Downloading nuget.exe from <http://www.nuget.org/nuget.exe>

Copying module files to the module directory

Loading ADAL Assemblies ...

Exception calling "LoadFrom" with "1" argument(s): "The specified path, file name, or both are too long. The fully qualified file name must be less than

260 characters, and the directory name must be less than 248 characters.”

At

c:\users\eric\Documents\WindowsPowerShell\Modules\AADGraph\AADGraph.

psm1:22

char:5

+ [System.Reflection.Assembly]::LoadFrom(\$ADAL_Assembly.FullName) |

out-null

+

~~~~~

~~~~~

+ CategoryInfo : NotSpecified: (:) [], MethodInvocationException

+ FullyQualifiedErrorId : PathTooLongException

CommandType Name ModuleName

Function Connect-AAD AADGraph

Function Get-AADLinkedObject AADGraph

Function Get-AADObject AADGraph

Function Get-AADObjectById AADGraph

Function Get-AADTenantDetail AADGraph

Function Get-AADUser AADGraph

Function New-AADObject AADGraph

Function New-AADUser AADGraph

Function Remove-AADObject AADGraph

Function Remove-AADUser AADGraph

Function Set-AADObject AADGraph

Function Set-AADObjectProperty AADGraph

Function Set-AADTenantDetail AADGraph

Function Set-AADUser AADGraph

Function Set-AADUserThumbnailPhoto AADGraph

When I then try and run the Connect-AAD option I get this error:

PS C:\Install\AAD> Connect-AAD

New-Object : Cannot find type

[Microsoft.IdentityModel.Clients.ActiveDirectory.AuthenticationContext]:

verify that the assembly containing this type is loaded.

At

c:\users\eric\Documents\WindowsPowerShell\Modules\AADGraph\AADGraph.

psm1:41

char:18

+ \$authContext = New-Object

“Microsoft.IdentityModel.Clients.ActiveDirectory.Aut ...

+

~~~~~

~~~~~

~~~

+ CategoryInfo : InvalidType: (:) [New-Object], PSArgumentExcepti

on

+ FullyQualifiedErrorId :

TypeNotFound,Microsoft.PowerShell.Commands.NewOb

jectCommand

You cannot call a method on a null-valued expression.

At

c:\users\eric\Documents\WindowsPowerShell\Modules\AADGraph\AADGraph.

psm1:42

char:3

```
+ $authResult = $authContext.AcquireToken($resourceAppIdURI, $clientId,  
$redirect ...
```

+

~~~~~

~~~~~

~~~

```
+ CategoryInfo          : InvalidOperation: (:) [], RuntimeException
```

```
+ FullyQualifiedErrorId : InvokeMethodOnNull
```

Any ideas on what is causing this?

Thank you.

E.R.

Reply ↓



dushyantgill Post author

February 19, 2015 at 7:23 am

Thanks for reporting this Eric. The newer version of ADAL broke my code.

I have fixed it to install only a specific version of ADAL.

Please download, install and try again.

Reply ↓



Eric

February 19, 2015 at 4:10 pm

Excellent, all working great now and I am able to install, connect and update photo properly now. THANK YOU!

Reply ↓



dushyantgill Post author

February 19, 2015 at 4:54 pm

I'm glad. Let me know if you have any other feedback on the tools.

Reply ↓

6. Pingback: Script: – Bulk Assign Users to SaaS Application using Graph API & ADAL | James Evans - EduTech's Blog



Doug Wilkins

April 9, 2015 at 4:14 pm

having some difficulty using Connect-AAD. I can connect using cloud identities but federated identities do not seem to work. When I try to connect I get the following type of error:

```
•Activity ID: 00000000-0000-0000-0301-0080000000fd
•Relying party: Microsoft Office 365 Identity
Platform
•Error time: Thu, 09 Apr 2015 16:09:18 GMT
•Cookie: enabled
•User agent string: Mozilla/4.0 (compatible; MSIE
7.0; Windows NT 6.3; Win64; x64; Trident/7.0;
.NET4.0E; .NET4.0C; InfoPath.3; .NET CLR 3.5.30729;
.NET CLR 2.0.50727; .NET CLR 3.0.30729)
```

Reply ↓



Doug Wilkins

April 20, 2015 at 1:35 pm

Figured it out – ADAL requires that Forms Authentication be enabled on the intranet portion of the Authentication methods on ADFS before it will work on the LAN. I believe this enables Basic Authentication with TLS/SSL which ADAL can then use

Reply ↓

8. Pingback: PowerShellでGraph APIを使ってAzure Active Directoryにアクセスする方法

9. Pingback: [EWS]Using Oauth authentication with EWS in Office 365 - Developer Messaging - Site Home - MSDN Blogs

Leave a Reply

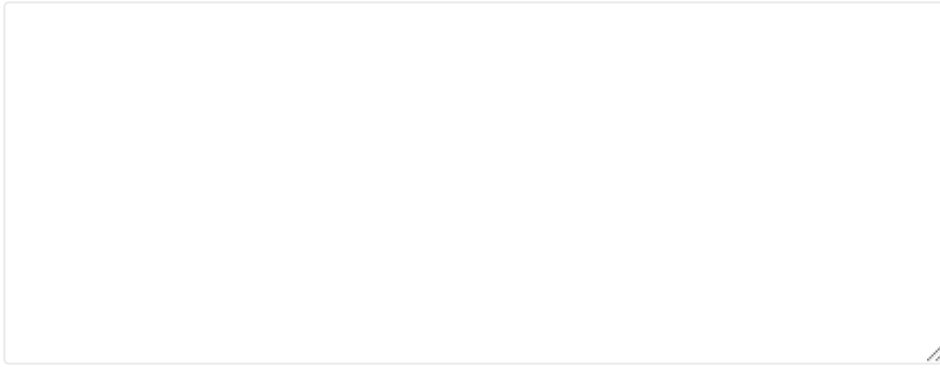
Your email address will not be published. Required fields are marked *

Name *

Email *

Website

Comment



You may use these HTML tags and attributes: ` <abbr title=""> <acronym title=""> <blockquote cite=""> <cite> <code> <del datetime=""> <i> <q cite=""> <s> <strike> `

Post Comment

☐ Notify me of follow-up comments by email.

☐ Notify me of new posts by email.

Copyright © 2017 dushyantgill | Theme by: Theme Horse | Powered by: WordPress