

# Create user access token & authorization header

To authentication against the Microsoft Graph API there are two general concepts. Application permissions allow an application in Microsoft Entra ID to act as it's own entity, rather than on behalf of a specific user. Delegated permissions allow an application in Microsoft Entra ID to perform actions on behalf of a particular user.

This guide focuses on authentication as a user to create scripts in the context of the provided user. To create or renew a token in the application context there are other instructions.

## Use case

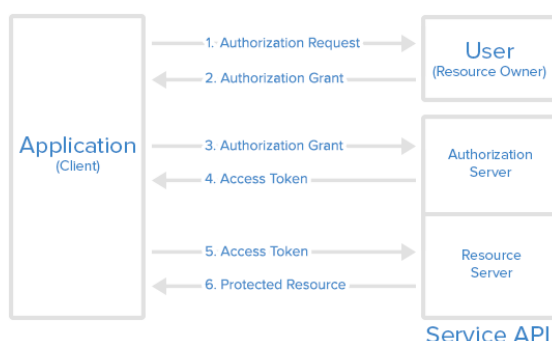
This authentication is required if you want to retrieve, update, or create resources using the Microsoft APIs (Azure Management API, Microsoft Graph API). This is for delegated-based permissions such as user triggered scripts or automations.

## Graph API authentication

To authenticate with application permission you have to use an Microsoft Entra ID App Registration. There you can specify an Client Secret as it is described here: [Get app details and gr... | LNC Docs \(lucanoahcaprez.ch\)](#)

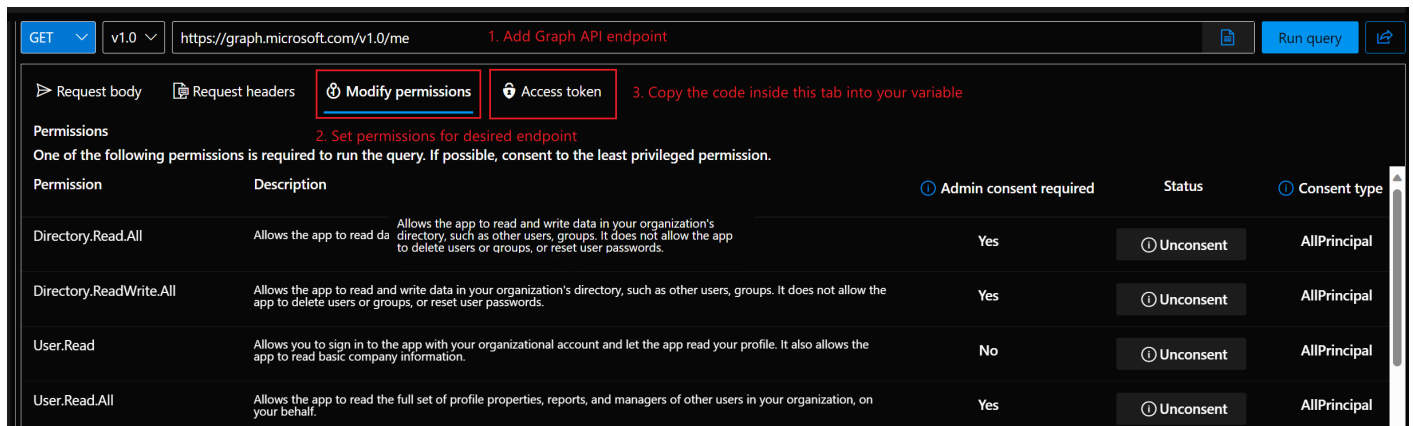
The authentication method used for Microsoft Graph API is the industry-standard OAuth 2.0. This concept uses access tokens for authenticating against API endpoints. These access tokens are then sent to the resource server in the HTTP header. Therefore we have to create the header correctly to use the resources of the Microsoft Graph API.

### Abstract Protocol Flow



# Get access token from Graph Explorer

The simplest variant can be used for one-off tokens or scripts that are rarely executed by the user. The code can simply be copied out under “Access token” in Microsoft Graph Explorer. It is also important that “Bearer” is added at the front (in the scripts that are provided here, the "Bearer" addition is made in the respective API call) and that the permissions for each Graph Endpoint are set under “Modify permissions”.



1. Add Graph API endpoint

2. Set permissions for desired endpoint

3. Copy the code inside this tab into your variable

Permissions

One of the following permissions is required to run the query. If possible, consent to the least privileged permission.

Permission	Description	Admin consent required	Status	Consent type
Directory.Read.All	Allows the app to read and write data in your organization's directory, such as other users, groups. It does not allow the app to delete users or groups, or reset user passwords.	Yes	Unconsent	AllPrincipal
Directory.ReadWrite.All	Allows the app to read and write data in your organization's directory, such as other users, groups. It does not allow the app to delete users or groups, or reset user passwords.	Yes	Unconsent	AllPrincipal
User.Read	Allows you to sign in to the app with your organizational account and let the app read your profile. It also allows the app to read basic company information.	No	Unconsent	AllPrincipal
User.Read.All	Allows the app to read the full set of profile properties, reports, and managers of other users in your organization, on your behalf.	Yes	Unconsent	AllPrincipal

If the permission were recently granted, refresh the page a few times or be a bit more patient ;)

## Build header via PowerShell script

The following scripts creates the header that contains the header property and the corresponding Bearer token. This function needs the arguments tenant ID, client (Application) ID, the client secret, the username and the password. This function has to be called the first time with these parameters.

**Attention:** This function returns a complete header. If you want to specify more information in the request header you have to use the function lower on this page.

```
# Function for getting Microsoft Entra ID Authentication Header
function Build-MicrosoftEntraIDUserAuthenticationHeader {
    param (
        [Parameter(Mandatory=$true)]
        [string] $TenantId,
        [string] $ClientId,
        [string] $ClientSecret,
        [string] $Username,
        [string] $Password,
        $RefreshToken
    )
}
```

```
$authenticationurl = "https://login.microsoftonline.com/$tenantid/oauth2/v2.0/token"
```

```
if($refreshtoken -and $tenantId){
```

```
    $tokenBodySource = @{
```

```
        grant_type = "refresh_token"
```

```
        scope = "https://graph.microsoft.com/.default"
```

```
        refresh_token = $refreshtoken
```

```
    }
```

```
}
```

```
elseif($tenantId -and $clientId -and $clientSecret){
```

```
    $tokenBodySource = @{
```

```
        client_id = $clientId
```

```
        scope = 'https://graph.microsoft.com/.default'
```

```
        grant_type = 'password'
```

```
        username = $username
```

```
        password = $password
```

```
        client_secret = $clientSecret
```

```
    }
```

```
}
```

```
else{
```

```
    Write-Error "Authorization not successful. Not enough information provided."
```

```
}
```

```
while ([string]::IsNullOrEmpty($AuthResponse.access_token)) {
```

```
    $AuthResponse = try {
```

```
        Invoke-RestMethod -Method POST -Uri $authenticationurl -Body $tokenBodySource
```

```
    }
```

```
    catch {
```

```
        $ErrorAuthResponse = $_.ErrorDetails.Message | ConvertFrom-Json
```

```
        if ($ErrorAuthResponse.error -ne "authorization_pending") {
```

```
            Write-Error "Authorization not successful. Error while posting body source:
```

```
$(($ErrorAuthResponse.error)"
```

```
        throw
```

```
    }
```

```
}
```

```
}
```

```
if($AuthResponse.token_type -and $AuthResponse.access_token){
```

```
    $global:MicrosoftEntraIDAccessToken = "($($AuthResponse.token_type) $($AuthResponse.access_token)"
```

```

$global:MicrosoftEntraIDHeader = @{
    "Authorization" = "$global:MicrosoftEntraIDAccessToken"
}
Write-Output "Authorization successful! Token saved in variable."
}
else{
    Write-Error "Authorization not successful. Not enough information provided."
}
}

# Authorization Header with ClientId, ClientSecret and User Credentials
$tenantId=""
$ClientId=""
$ClientSecret=""
$Username = "<yourusername>"
$Password = '<yourpassword>'
Build-MicrosoftEntraIDUserAuthenticationHeader -ClientId $ClientId -TenantId $tenantId -ClientSecret
$ClientSecret -Username $Username -Password $Password

# Authorization Header with refresh_token
$tenantId=""
$refreshtoken=""
Build-MicrosoftEntraIDUserAuthenticationHeader -TenantId $tenantId -refreshtoken $refreshtoken

```

# Get Bearer Token via PowerShell script

This function allows you to use more than one header property. It only returns the access token which then has to be built into a header by itself. But the other functionalities and parameters work like the function above.

```

# Function for getting Microsoft Entra ID Access Token
function Build-MicrosoftEntraIDUserAccessHeader {
    param (
        [Parameter(Mandatory=$true)]
        [string] $TenantId,
        [string] $ClientId,
        [string] $ClientSecret,

```

```
[string] $Username,  
[string] $Password,  
$RefreshToken  
)
```

```
$authenticationurl = "https://login.microsoftonline.com/$tenantid/oauth2/v2.0/token"
```

```
if($refreshToken -and $tenantId){
```

```
    $tokenBodySource = @{  
        grant_type = "refresh_token"  
        scope = "https://graph.microsoft.com/.default"  
        refresh_token = $refreshToken  
    }  
}
```

```
}
```

```
elseif($tenantId -and $clientId -and $clientSecret){
```

```
    $tokenBodySource = @{  
        client_id = $clientId  
        scope = 'https://graph.microsoft.com/.default'  
        grant_type = 'password'  
        username = $username  
        password = $password  
        client_secret = $clientSecret  
    }  
}
```

```
}
```

```
else{
```

```
    Write-Error "Authorization not successful. Not enough information provided."
```

```
}
```

```
while ([string]::IsNullOrEmpty($AuthResponse.access_token)) {
```

```
    $AuthResponse = try {  
        Invoke-RestMethod -Method POST -Uri $authenticationurl -Body $tokenBodySource  
    }  
}
```

```
catch {
```

```
    $ErrorAuthResponse = $_.ErrorDetails.Message | ConvertFrom-Json
```

```
    if ($ErrorAuthResponse.error -ne "authorization_pending") {
```

```
        Write-Error "Authorization not successful. Error while posting body source:
```

```
$(($ErrorAuthResponse.error)"
```

```
        throw
```

```
    }
```

```
}
```

```
}

if($AuthResponse.token_type -and $AuthResponse.access_token){
    $global:MicrosoftEntraIDAccessToken = "$($AuthResponse.token_type) $($AuthResponse.access_token)"
    Write-Output "Authorization successful! Token saved in variable."
}
else{
    Write-Error "Authorization not successful. Not enough information provided."
}
}

# Authorization Header with ClientId, ClientSecret and User Credentials
$tenantId="<tenantid>"
$ClientId="<appregistrationclientapp>"
$ClientSecret="<appregistrationclientsecret>"
$Username = "<yourusername>"
$Password = '<yourpassword>'
Build-MicrosoftEntraIDUserAuthenticationHeader -ClientId $ClientId -TenantId $tenantId -ClientSecret
$ClientSecret -Username $Username -Password $Password

# Authorization Header with refresh_token
$tenantId="<tenantid>"
$refreshtoken="<yourrefreshtoken>"
Build-MicrosoftEntraIDUserAuthenticationHeader -TenantId $tenantId -refreshtoken $refreshtoken
```

---

Revision #8

Created 12 December 2022 08:47:49

Updated 21 July 2024 15:11:16