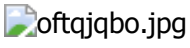
Do not click nested Typetext icons. Doing so will produce the wrong output.



http\://localhost should be input as http://localhost

At the end of this lab, you can skip the **Clean Up** exercise directing you to remove the resources from your Subscription or Resource Group(s). The clean up is handled automatically, after ending your lab.

Lab: Authenticating to and querying Microsoft Graph by using MSAL and .NET SDKs

---

Student lab manual

Lab scenario

As a new employee at your company, you signed in to your Microsoft 365 applications for the first time and discovered that your profile information isn't accurate. You also noticed that the name and profile picture when you sign in aren't correct. Rather than change these values manually, you have decided that this is a good opportunity to learn the Microsoft identity platform and how you can use different libraries such as the Microsoft Authentication Library (MSAL) and the Microsoft Graph SDK to change these values in a programmatic manner.

Objectives

After you complete this lab, you will be able to:

- Create a new application registration in Azure Active Directory (Azure AD).
- Use the MSAL.NET library to implement the interactive authentication flow.
- Obtain a token from the Microsoft identity platform by using the MSAL.NET library.
- Query Microsoft Graph by using the Microsoft Graph SDK and the device code flow.

Lab setup

- Estimated time: **45 minutes**

Instructions

Before you start

Sign in to the lab virtual machine

Ensure that you're signed in to your Windows 10 virtual machine by using the following credentials:

- Username: **Admin**
- Password: **Pa55w.rd**

Review the installed applications

Find the taskbar on your Windows 10 desktop. The taskbar contains the icons for the applications that you'll use in this lab:

- Microsoft Edge
- Visual Studio Code

Exercise 1: Create an Azure AD application registration

Task 1: Open the Azure portal

1. Sign in to the Azure portal (https://portal.azure.com).
2. If this is your first time signing in to the Azure portal, you'll notice a dialog box offering a tour of the portal. Select **Get Started** to skip the tour.

Task 2: Create an application registration

1. Create a new Azure AD application registration with the following details:
    - Name: **graphapp**
    - Supported account types: **Accounts in this organizational directory only (Default Directory only - Single tenant)**
    - Redirect URI: **Public client/native (mobile & desktop) http\://localhost**

    Note: Wait for Azure to finish creating the registration before you move forward with the lab. You'll receive a notification when the vault is created.

Task 3: Enable the default client type

1. Find the **Authentication** section of the **graphapp** application registration blade, and then enable the **Default client type** option.
2. Save your changes.

Task 4: Record unique identifiers

1. Browse to **Overview** of the **graphapp** application registration blade.
2. Find and record the value of the **Application (client) ID** text box. You'll use this value later in the lab.
3. Find and record the value of the **Directory (tenant) ID** text box. You'll use this value later in the lab.

Review

In this exercise, you created a new application registration and recorded important values that you'll need later in the lab.

Exercise 2: Obtain a token by using the MSAL.NET library

Task 1: Create a .NET project

1. Using Visual Studio Code, open the **Allfiles (F):\Allfiles\Labs\06\Starter\GraphClient** folder.
2. Using a terminal, create a new .NET project named **GraphClient** in the current folder:

```
dotnet new console --name GraphClient --output .
```

Note: The **dotnet new** command will create a new **console** project in a folder with the same name as the project.

3. Using the same terminal, import version 4.7.1 of **Microsoft.Identity.Client** from NuGet:

```
dotnet add package Microsoft.Identity.Client --version 4.7.1
```

Note: The **dotnet add package** command will add the **Microsoft.Identity.Client** package from NuGet. For more information, go to Microsoft.Identity.Client.

4. Using the same terminal, build the .NET web application:

```
dotnet build
```

5. Close the current terminal.

Task 2: Modify the Program class

1. Open the **Program.cs** file in Visual Studio Code.
2. Delete all existing code in the **Program.cs** file.
3. Add the following **using** directives for libraries that will be referenced by the application:

```
using Microsoft.Identity.Client;
using System;
using System.Collections.Generic;
using System.Threading.Tasks;
```

4. Create a new **Program** class with two constant string properties named **_clientId** and **_tenantId**, and then create an asynchronous **Main** entry point method:

```
public class Program
{
    private const string _clientId = "<app-reg-client-id>";
    private const string _tenantId = "<aad-tenant-id>";

    public static async Task Main(string[] args)
    {
    }

}
```

5. Update the **_clientId** string constant by setting its value to the **Application (client) ID** that you recorded earlier in this lab.
6. Update the **_tenantId** string constant by setting its value to the **Directory (tenant) ID** that you recorded earlier in this lab.

Task 3: Obtain an MSAL token

1. In the **Main** method, perform the following actions:

    a. Create a new variable named *app* of type **IPublicClientApplication**.

    b. Add the following block of code to build a public client application instance by using the static **PublicClientApplicationBuilder** class, and then store it in the *app* variable:

    ```
    app = PublicClientApplicationBuilder
        .Create(_clientId)
        .WithAuthority(AzureCloudInstance.AzurePublic, _tenantId)
        .WithRedirectUri("http://localhost")
        .Build();
    ```

    c. Add the following block of code to create a new generic string **List<>** with a single value of **user.read**:

    ```
    List<string> scopes = new List<string>
    {
        "user.read"
    };
    ```

    d. Create a new variable named *result* of type **AuthenticationResult**.

    e. Add the following block of code to acquire a token interactively and store the output in the *result* variable:

    ```
    result = await app
        .AcquireTokenInteractive(scopes)
        .ExecuteAsync();
    ```

    f. Render the value of the **AuthenticationResult.AccessToken** member to the console:

    ```
    Console.WriteLine($"Token:\t{result.AccessToken}");
    ```

2. Save the **Program.cs** file.

Task 4: Test the updated application

1. Using a terminal, run the .NET console application project:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\06\Solution\GraphClient** folder.

2. The running console application will automatically open an instance of the default browser.

3. In the open browser window, sign in by using your Microsoft account.

    Note: You might have the option to select an existing Microsoft account as opposed to signing in again.

4. The browser window will automatically go to the **Permissions requested** webpage. Accept the request for permissions.

5. Close the currently open browser window.
6. In the **Visual Studio Code** window, observe the token render in the output from the currently running console application.
7. Close the current terminal.

Review

In this exercise, you acquired a token from the Microsoft identity platform by using the MSAL.NET library.

Exercise 3: Query Microsoft Graph by using the .NET SDK

Task 1: Import the Microsoft Graph SDK from NuGet

1. Using a terminal, import version 1.21.0 of **Microsoft.Graph** from NuGet:

```
dotnet add package Microsoft.Graph --version 1.21.0
```

   Note: The **dotnet add package** command will add the **Microsoft.Graph** package from NuGet. For more information, go to Microsoft.Graph.

2. Using the same terminal, import version 1.0.0-preview.2 of **Microsoft.Graph.Auth** from NuGet:

```
dotnet add package Microsoft.Graph.Auth --version 1.0.0-preview.2
```

   Note: The **dotnet add package** command will add the **Microsoft.Graph.Auth** package from NuGet. For more information, go to Microsoft.Graph.Auth.

3. Using the same terminal, build the .NET web application:

```
dotnet build
```

4. Close the current terminal.

Task 2: Modify the Program class

1. Open the **Program.cs** file in Visual Studio Code.
2. Add the following **using** directives for libraries that will be referenced by the application:

```
using Microsoft.Graph;
using Microsoft.Graph.Auth;
```

Task 3: Use the Microsoft Graph SDK to query user profile information

1. Within the **Main** method, remove the following block of unnecessary code:

```
AuthenticationResult result;

result = await app
    .AcquireTokenInteractive(scopes)
```

```
        .ExecuteAsync();

    Console.WriteLine($"Token:\t{result.AccessToken}");
```

2. In the **Main** method, perform the following actions:

    a. Create a new variable named *provider* of type **DeviceCodeProvider** that passes in the variables *app*, and *scopes* as constructor parameters:

    ```
    DeviceCodeProvider provider = new DeviceCodeProvider(app, scopes);
    ```

    b. Create a new variable named *client* of type **GraphServiceClient** that passes in the variable *provider* as a constructor parameter:

    ```
    GraphServiceClient client = new GraphServiceClient(provider);
    ```

    c. Add the following block of code to use the **GraphServiceClient** instance to asynchronously get the response of issuing an HTTP request to the relative **/Me** directory of the REST API, and then store the result in a new variable named *myProfile* of type **User**:

    ```
    User myProfile = await client.Me
        .Request()
        .GetAsync();
    ```

    d. Render the value of the **User.DisplayName** and **User.Id** members to the console:

    ```
    Console.WriteLine($"Name:\t{myProfile.DisplayName}");
    Console.WriteLine($"AAD Id:\t{myProfile.Id}");
    ```

3. Save the **Program.cs** file.

Task 4: Test the updated application

1. Using a terminal, run the .NET console application project:

    ```
    dotnet run
    ```

    Note: If there are any build errors, review the **Program.cs** file in the **Allfiles (F):\Allfiles\Labs\06\Solution\GraphClient** folder.

2. Observe the message in the output from the currently running console application. Record the value of the code in the message. You'll use this value later in the lab.
3. Go to https://microsoft.com/devicelogin, and then enter the code value that you copied earlier in the lab.
4. Sign in by using your Microsoft account.

    Note: You might have the option to select an existing Microsoft account as opposed to signing in again.

5. Close the currently open browser window.
6. In the **Visual Studio Code** window, observe the output from the Microsoft Graph request in the currently running console application.

7. Close the current terminal.

Review

In this exercise, you queried Microsoft Graph by using the SDK and MSAL-based authentication.

Exercise 4: Clean up your subscription

Task 1: Delete the application registration in Azure AD

1. Access the **graphapp** Azure AD application registration that you created earlier in this lab.
2. Delete the application registration.

Task 2: Close the active applications

1. Close the currently running Microsoft Edge application.
2. Close the currently running Visual Studio Code application.

Review

In this exercise, you cleaned up your subscription by removing the application registration used in this lab.

Congratulations!

You have successfully completed this exercise. Click **Next** to advance to the next exercise.