100 XP

Introduction

3 minutes

Use the wealth of data in Microsoft Graph to build apps for organizations and consumers that interact with millions of users.

After completing this module, you'll be able to:

- Explain the benefits of using Microsoft Graph
- Perform operations on Microsoft Graph by using REST and SDKs
- Apply best practices to help your applications get the most out of Microsoft Graph

Next unit: Discover Microsoft Graph

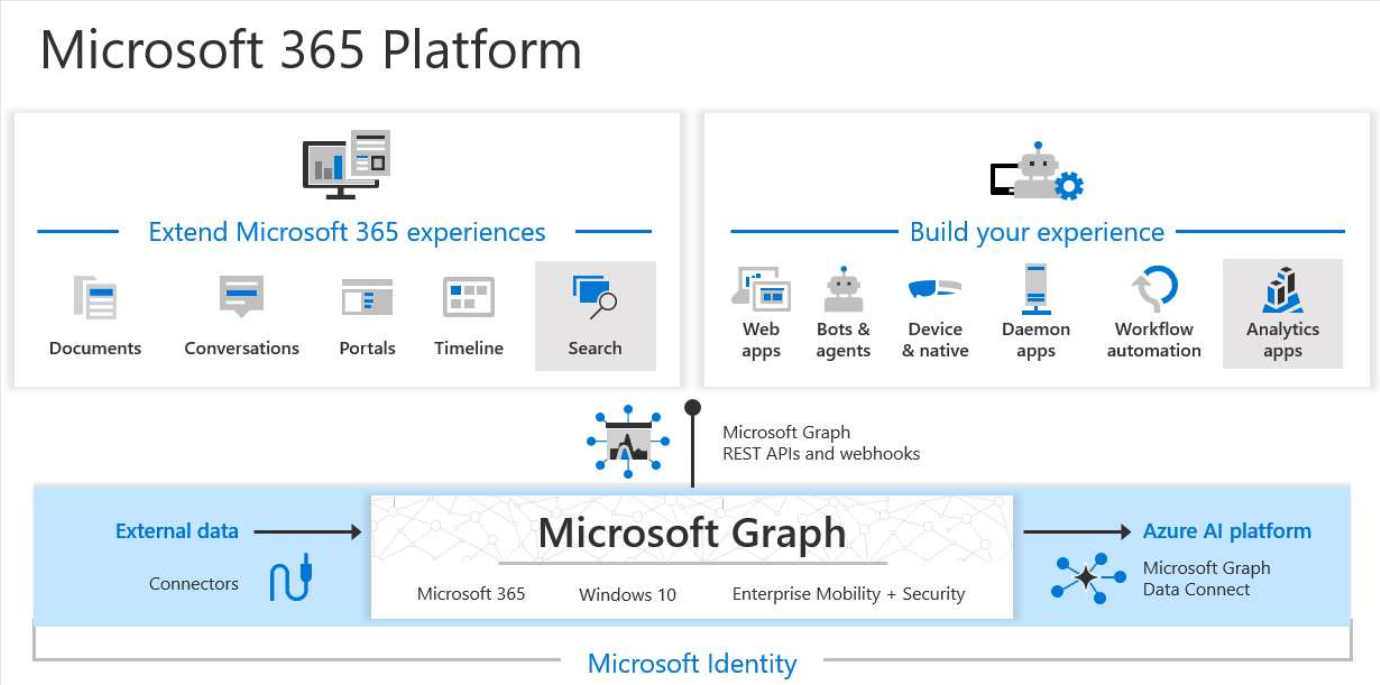
Continue

100 XP

Discover Microsoft Graph

3 minutes

Microsoft Graph is the gateway to data and intelligence in Microsoft 365. It provides a unified programmability model that you can use to access the tremendous amount of data in Microsoft 365, Windows 10, and Enterprise Mobility + Security.



In the Microsoft 365 platform, three main components facilitate the access and flow of data:

- The Microsoft Graph API offers a single endpoint, `https://graph.microsoft.com`. You can use REST APIs or SDKs to access the endpoint. Microsoft Graph also includes a powerful set of services that manage user and device identity, access, compliance, security, and help protect organizations from data leakage or loss.
- Microsoft Graph connectors work in the incoming direction, **delivering data external to the Microsoft cloud into Microsoft Graph services and applications**, to enhance Microsoft 365 experiences such as Microsoft Search. Connectors exist for many commonly used data sources such as Box, Google Drive, Jira, and Salesforce.

- Microsoft Graph Data Connect provides a set of tools to streamline secure and scalable **delivery of Microsoft Graph data to popular Azure data stores**. The cached data serves as data sources for Azure development tools that you can use to build intelligent applications.

Next unit: Query Microsoft Graph by using REST

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Query Microsoft Graph by using REST

3 minutes

Microsoft Graph is a RESTful web API that enables you to access Microsoft Cloud service resources. After you register your app and get authentication tokens for a user or service, you can make requests to the Microsoft Graph API.

The Microsoft Graph API defines most of its resources, methods, and enumerations in the OData namespace, `microsoft.graph`, in the Microsoft Graph metadata. A small number of API sets are defined in their sub-namespaces, such as the call records API which defines resources like callRecord in `microsoft.graph.callRecords`.

Unless explicitly specified in the corresponding topic, assume types, methods, and enumerations are part of the `microsoft.graph` namespace.

Call a REST API method

To read from or write to a resource such as a user or an email message, you construct a request that looks like the following:

HTTP Copy

```
{HTTP method} https://graph.microsoft.com/{version}/{resource}?{query-parameters}
```

The components of a request include:

- `{HTTP method}` - The HTTP method used on the request to Microsoft Graph.
- `{version}` - The version of the Microsoft Graph API your application is using.
- `{resource}` - The resource in Microsoft Graph that you're referencing.
- `{query-parameters}` - Optional OData query options or REST method parameters that customize the response.

After you make a request, a response is returned that includes:

- Status code - An HTTP status code that indicates success or failure.
- Response message - The data that you requested or the result of the operation. The response message can be empty for some operations.
- `nextLink` - If your request returns a lot of data, you need to page through it by using the URL returned in `@odata.nextLink`.

HTTP methods

Microsoft Graph uses the HTTP method on your request to determine what your request is doing. The API supports the following methods.

Method	Description
GET	Read data from a resource.
POST	Create a new resource, or perform an action.
PATCH	Update a resource with new values.
PUT	Replace a resource with a new one.
DELETE	Remove a resource.

- For the CRUD methods `GET` and `DELETE`, no request body is required.
- The `POST`, `PATCH`, and `PUT` methods require a request body, usually specified in JSON format, that contains additional information, such as the values for properties of the resource.

Version

Microsoft Graph currently supports two versions: `v1.0` and `beta`.

- `v1.0` includes generally available APIs. Use the v1.0 version for all production apps.
- `beta` includes APIs that are currently in preview. Because we might introduce breaking changes to our beta APIs, we recommend that you use the beta version only to test apps that are in development; do not use beta APIs in your production apps.

Resource

A resource can be an entity or complex type, commonly defined with properties. Entities differ from complex types by always including an **id** property.

Your URL will include the resource you are interacting with in the request, such as `me`, **user**, **group**, **drive**, and **site**. Often, top-level resources also include *relationships*, which you can use to access additional resources, like `me/messages` or `me/drive`. You can also interact with resources using *methods*; for example, to send an email, use `me/sendMail`.

Each resource might require different permissions to access it. You will often need a higher level of permissions to create or update a resource than to read it. For details about required permissions, see the method reference topic.

Query parameters

Query parameters can be OData system query options, or other strings that a method accepts to customize its response.

You can use optional OData system query options to include more or fewer properties than the default response, filter the response for items that match a custom query, or provide additional parameters for a method.

For example, adding the following `filter` parameter restricts the messages returned to only those with the `emailAddress` property of `jon@contoso.com`.

HTTP Copy

```
GET https://graph.microsoft.com/v1.0/me/messages?filter=emailAddress eq
'jon@contoso.com'
```

Additional resources

Below are links to some tools you can use to build and test requests using Microsoft Graph APIs.

- Graph Explorer
- Postman

Next unit: Query Microsoft Graph by using SDKs

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Query Microsoft Graph by using SDKs

3 minutes

The Microsoft Graph SDKs are designed to simplify building high-quality, efficient, and resilient applications that access Microsoft Graph. The SDKs include two components: a service library and a core library.

The service library contains models and request builders that are generated from Microsoft Graph metadata to provide a rich, strongly typed, and discoverable experience when working with the many datasets available in Microsoft Graph.

The core library provides a set of features that enhance working with all the Microsoft Graph services. Embedded support for retry handling, secure redirects, transparent authentication, and payload compression, improve the quality of your application's interactions with Microsoft Graph, with no added complexity, while leaving you completely in control. The core library also provides support for common tasks such as paging through collections and creating batch requests.

In this unit you will learn about the available SDKs and see some code examples of some of the most common operations.

Install the Microsoft Graph .NET SDK

The Microsoft Graph .NET SDK is included in the following NuGet packages:

- Microsoft.Graph - Contains the models and request builders for accessing the `v1.0` endpoint with the fluent API. Microsoft.Graph has a dependency on Microsoft.Graph.Core.
- Microsoft.Graph.Beta - Contains the models and request builders for accessing the `beta` endpoint with the fluent API. Microsoft.Graph.Beta has a dependency on Microsoft.Graph.Core.
- Microsoft.Graph.Core - The core library for making calls to Microsoft Graph.
- Microsoft.Graph.Auth - Provides an authentication scenario-based wrapper of the Microsoft Authentication Library (MSAL) for use with the Microsoft Graph SDK. Microsoft.Graph.Auth has a dependency on Microsoft.Graph.Core.

Create a Microsoft Graph client

The Microsoft Graph client is designed to make it simple to make calls to Microsoft Graph. You can use a single client instance for the lifetime of the application. The following code examples show how to create an instance of a Microsoft Graph client. The authentication provider will handle acquiring access tokens for the application. The different application providers support different client scenarios. For details about which provider and options are appropriate for your scenario, see Choose an Authentication Provider.

C# Copy

```
// Build a client application.
IPublicClientApplication publicClientApplication = PublicClientApplicationBuilder
            .Create("INSERT-CLIENT-APP-ID")
            .Build();
// Create an authentication provider by passing in a client application and graph
scopes.
DeviceCodeProvider authProvider = new DeviceCodeProvider(publicClientApplication,
graphScopes);
// Create a new instance of GraphServiceClient with the authentication provider.
GraphServiceClient graphClient = new GraphServiceClient(authProvider);
```

Read information from Microsoft Graph

To read information from Microsoft Graph, you first need to create a request object and then run the `GET` method on the request.

C# Copy

```
// GET https://graph.microsoft.com/v1.0/me

var user = await graphClient.Me
    .Request()
    .GetAsync();
```

Retrieve a list of entities

Retrieving a list of entities is similar to retrieving a single entity except there a number of other options for configuring the request. The `$filter` query parameter can be used to reduce the

result set to only those rows that match the provided condition. The `$orderBy` query parameter will request that the server provide the list of entities sorted by the specified properties.

C# Copy

```
// GET https://graph.microsoft.com/v1.0/me/messages?$select=subject,sender&$filter=
<some condition>&orderBy=receivedDateTime

var messages = await graphClient.Me.Messages
    .Request()
    .Select(m => new {
        m.Subject,
        m.Sender
    })
    .Filter("<filter condition>")
    .OrderBy("receivedDateTime")
    .GetAsync();
```

Delete an entity

Delete requests are constructed in the same way as requests to retrieve an entity, but use a `DELETE` request instead of a `GET`.

C# Copy

```
// DELETE https://graph.microsoft.com/v1.0/me/messages/{message-id}

string messageId = "AQMkAGUy...";
var message = await graphClient.Me.Messages[messageId]
    .Request()
    .DeleteAsync();
```

Create a new entity

For SDKs that support a fluent style, new items can be added to collections with an `Add` method. For template-based SDKs, the request object exposes a `post` method.

C# Copy

```
// POST https://graph.microsoft.com/v1.0/me/calendars

var calendar = new Calendar
{
```

```
    Name = "Volunteer"
};

var newCalendar = await graphClient.Me.Calendars
    .Request()
    .AddAsync(calendar);
```

Additional resources

- Microsoft Graph REST API v1.0 reference

Next unit: Apply best practices to Microsoft Graph

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Apply best practices to Microsoft Graph

3 minutes

This unit describes best practices that you can apply to help your applications get the most out of Microsoft Graph and make your application more reliable for end users.

Authentication

To access the data in Microsoft Graph, your application will need to acquire an OAuth 2.0 access token, and present it to Microsoft Graph in either of the following:

- The HTTP *Authorization* request header, as a *Bearer* token
- The graph client constructor, when using a Microsoft Graph client library

Use the Microsoft Authentication Library API, MSAL to acquire the access token to Microsoft Graph.

Consent and authorization

Apply the following best practices for consent and authorization in your app:

- **Use least privilege**. Only request permissions that are absolutely necessary, and only when you need them. For the APIs your application calls check the permissions section in the method topics. For example, see creating a user and choose the least privileged permissions.

- **Use the correct permission type based on scenarios**. If you're building an interactive application where a signed in user is present, your application should use *delegated* permissions. If, however, your application runs without a signed-in user, such as a background service or daemon, your application should use application permissions.

> **⊗ Caution**
>
> Using application permissions for interactive scenarios can put your application at compliance and security risk. Be sure to check user's privileges to ensure they don't

have undesired access to information, or are circumnavigating policies configured by an administrator.

- **Consider the end user and admin experience**. This will directly affect end user and admin experiences. For example:
  - Consider who will be consenting to your application, either end users or administrators, and configure your application to request permissions appropriately.
  - Ensure that you understand the difference between static, dynamic and incremental consent.
- **Consider multi-tenant applications**. Expect customers to have various application and consent controls in different states. For example:
  - Tenant administrators can disable the ability for end users to consent to applications. In this case, an administrator would need to consent on behalf of their users.
  - Tenant administrators can set custom authorization policies such as blocking users from reading other user's profiles, or limiting self-service group creation to a limited set of users. In this case, your application should expect to handle 403 error response when acting on behalf of a user.

Handle responses effectively

Depending on the requests you make to Microsoft Graph, your applications should be prepared to handle different types of responses. The following are some of the most important practices to follow to ensure that your application behaves reliably and predictably for your end users. For example:

- **Pagination**: When querying a resource collection, you should expect that Microsoft Graph will the return result set in multiple pages, due to server-side page size limits. Your application should **always** handle the possibility that the responses are paged in nature, and use the `@odata.nextLink` property to obtain the next paged set of results, until all pages of the result set have been read. The final page will not contain an `@odata.nextLink` property. For more details, see paging.
- **Evolvable enumerations**: Adding members to existing enumerations can break applications already using these enums. Evolvable enums is a mechanism that Microsoft Graph API uses to add new members to existing enumerations without causing a breaking change for

applications. By default, a GET operation returns only known members for properties of evolvable enum types and your application needs to handle only the known members. If you design your application to handle unknown members as well, you can opt-in to receive those members by using an HTTP `Prefer` request header.

Storing data locally

Your application should ideally make calls to Microsoft Graph to retrieve data in real time as necessary. You should only cache or store data locally if necessary for a specific scenario, and if that use case is covered by your terms of use and privacy policy, and does not violate the Microsoft APIs Terms of Use. Your application should also implement proper retention and deletion policies.

Next unit: Knowledge check

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

200 XP

Knowledge check

3 minutes

Check your knowledge

1. Which HTTP method below is used to update a resource with new values?

- ○ POST
- ◉ PATCH

  ✔ That's correct. The PATCH method does update a resource with a new value.
- ○ PUT

2. Which of the components of the Microsoft 365 platform is used to deliver data external to Azure into Microsoft Graph services and applications?

- ◉ Microsoft Graph API

  ✘ That's incorrect. The Microsoft Graph API offers a single endpoint to use with APIs and SDKs.
- ○ Microsoft Graph connectors

  ✔ That's correct. Microsoft Graph connectors work in the incoming direction. Connectors exist for many commonly used data sources such as Box, Google Drive, Jira, and Salesforce.
- ○ Microsoft Graph Data Connect

3. Which of the following Microsoft Graph .NET SDK packages provides an authentication scenario-based wrapper of the Microsoft Authentication Library?

- ○ Microsoft.Graph
- ◉ Microsoft.Graph.Core

✗ That's incorrect. This is the core library for making calls to Microsoft Graph.

○ Microsoft.Graph.Auth

✓ That's correct. The Microsoft.Graph.Auth package provides an authentication scenario-based wrapper of the Microsoft Authentication Library for use with the Microsoft Graph SDK.

Next unit: Summary

Continue >

How are we doing? ☆ ☆ ☆ ☆ ☆

100 XP

Summary

3 minutes

In this module, you learned how to:

- Explain the benefits of using Microsoft Graph
- Perform operations on Microsoft Graph by using REST and SDKs
- Apply best practices to help your applications get the most out of Microsoft Graph

Module complete:

Unlock achievement

How are we doing? ☆ ☆ ☆ ☆ ☆