

Microsoft
Official
Course



MS-600T00

Building Applications and
Solutions with Microsoft
365 Core Services

MS-600T00

**Building Applications and
Solutions with Microsoft 365
Core Services**

II Disclaimer

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

The names of manufacturers, products, or URLs are provided for informational purposes only and Microsoft makes no representations and warranties, either expressed, implied, or statutory, regarding these manufacturers or the use of the products with any Microsoft technologies. The inclusion of a manufacturer or product does not imply endorsement of Microsoft of the manufacturer or product. Links may be provided to third party sites. Such sites are not under the control of Microsoft and Microsoft is not responsible for the contents of any linked site or any link contained in a linked site, or any changes or updates to such sites. Microsoft is not responsible for webcasting or any other form of transmission received from any linked site. Microsoft is providing these links to you only as a convenience, and the inclusion of any link does not imply endorsement of Microsoft of the site or the products contained therein.

© 2019 Microsoft Corporation. All rights reserved.

Microsoft and the trademarks listed at <http://www.microsoft.com/trademarks>¹ are trademarks of the Microsoft group of companies. All other trademarks are property of their respective owners.

¹ <http://www.microsoft.com/trademarks>

MICROSOFT LICENSE TERMS

MICROSOFT INSTRUCTOR-LED COURSEWARE

These license terms are an agreement between Microsoft Corporation (or based on where you live, one of its affiliates) and you. Please read them. They apply to your use of the content accompanying this agreement which includes the media on which you received it, if any. These license terms also apply to Trainer Content and any updates and supplements for the Licensed Content unless other terms accompany those items. If so, those terms apply.

**BY ACCESSING, DOWNLOADING OR USING THE LICENSED CONTENT, YOU ACCEPT THESE TERMS.
IF YOU DO NOT ACCEPT THEM, DO NOT ACCESS, DOWNLOAD OR USE THE LICENSED CONTENT.**

If you comply with these license terms, you have the rights below for each license you acquire.

1. DEFINITIONS.

1. "Authorized Learning Center" means a Microsoft Imagine Academy (MSIA) Program Member, Microsoft Learning Competency Member, or such other entity as Microsoft may designate from time to time.
2. "Authorized Training Session" means the instructor-led training class using Microsoft Instructor-Led Courseware conducted by a Trainer at or through an Authorized Learning Center.
3. "Classroom Device" means one (1) dedicated, secure computer that an Authorized Learning Center owns or controls that is located at an Authorized Learning Center's training facilities that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
4. "End User" means an individual who is (i) duly enrolled in and attending an Authorized Training Session or Private Training Session, (ii) an employee of an MPN Member (defined below), or (iii) a Microsoft full-time employee, a Microsoft Imagine Academy (MSIA) Program Member, or a Microsoft Learn for Educators – Validated Educator.
5. "Licensed Content" means the content accompanying this agreement which may include the Microsoft Instructor-Led Courseware or Trainer Content.
6. "Microsoft Certified Trainer" or "MCT" means an individual who is (i) engaged to teach a training session to End Users on behalf of an Authorized Learning Center or MPN Member, and (ii) currently certified as a Microsoft Certified Trainer under the Microsoft Certification Program.
7. "Microsoft Instructor-Led Courseware" means the Microsoft-branded instructor-led training course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies. A Microsoft Instructor-Led Courseware title may be branded as MOC, Microsoft Dynamics, or Microsoft Business Group courseware.
8. "Microsoft Imagine Academy (MSIA) Program Member" means an active member of the Microsoft Imagine Academy Program.
9. "Microsoft Learn for Educators – Validated Educator" means an educator who has been validated through the Microsoft Learn for Educators program as an active educator at a college, university, community college, polytechnic or K-12 institution.
10. "Microsoft Learning Competency Member" means an active member of the Microsoft Partner Network program in good standing that currently holds the Learning Competency status.
11. "MOC" means the "Official Microsoft Learning Product" instructor-led courseware known as Microsoft Official Course that educates IT professionals, developers, students at an academic institution, and other learners on Microsoft technologies.
12. "MPN Member" means an active Microsoft Partner Network program member in good standing.

13. "Personal Device" means one (1) personal computer, device, workstation or other digital electronic device that you personally own or control that meets or exceeds the hardware level specified for the particular Microsoft Instructor-Led Courseware.
 14. "Private Training Session" means the instructor-led training classes provided by MPN Members for corporate customers to teach a predefined learning objective using Microsoft Instructor-Led Courseware. These classes are not advertised or promoted to the general public and class attendance is restricted to individuals employed by or contracted by the corporate customer.
 15. "Trainer" means (i) an academically accredited educator engaged by a Microsoft Imagine Academy Program Member to teach an Authorized Training Session, (ii) an academically accredited educator validated as a Microsoft Learn for Educators – Validated Educator, and/or (iii) a MCT.
 16. "Trainer Content" means the trainer version of the Microsoft Instructor-Led Courseware and additional supplemental content designated solely for Trainers' use to teach a training session using the Microsoft Instructor-Led Courseware. Trainer Content may include Microsoft PowerPoint presentations, trainer preparation guide, train the trainer materials, Microsoft One Note packs, classroom setup guide and Pre-release course feedback form. To clarify, Trainer Content does not include any software, virtual hard disks or virtual machines.
2. **USE RIGHTS.** The Licensed Content is licensed, not sold. The Licensed Content is licensed on a **one copy per user basis**, such that you must acquire a license for each individual that accesses or uses the Licensed Content.
- 2.1 Below are five separate sets of use rights. Only one set of rights apply to you.
 1. **If you are a Microsoft Imagine Academy (MSIA) Program Member:**
 1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
 2. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User who is enrolled in the Authorized Training Session, and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. provide one (1) Trainer with the unique redemption code and instructions on how they can access one (1) Trainer Content.
 3. For each license you acquire, you must comply with the following:
 1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 2. you will ensure each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 3. you will ensure that each End User provided with the hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End

User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each Trainer teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified Trainers who have in-depth knowledge of and experience with the Microsoft technology that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Authorized Training Sessions,
6. you will only deliver a maximum of 15 hours of training per week for each Authorized Training Session that uses a MOC title, and
7. you acknowledge that Trainers that are not MCTs will not have access to all of the trainer resources for the Microsoft Instructor-Led Courseware.

2. If you are a Microsoft Learning Competency Member:

1. Each license acquire may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or MCT, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Authorized Training Session and only immediately prior to the commencement of the Authorized Training Session that is the subject matter of the Microsoft Instructor-Led Courseware provided, **or**
 2. provide one (1) End User attending the Authorized Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) MCT with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
 1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 2. you will ensure that each End User attending an Authorized Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Authorized Training Session,
 3. you will ensure that each End User provided with a hard-copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,

4. you will ensure that each MCT teaching an Authorized Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Authorized Training Session,
5. you will only use qualified MCTs who also hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Authorized Training Sessions using MOC,
6. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
7. you will only provide access to the Trainer Content to MCTs.

3. If you are a MPN Member:

1. Each license acquired on behalf of yourself may only be used to review one (1) copy of the Microsoft Instructor-Led Courseware in the form provided to you. If the Microsoft Instructor-Led Courseware is in digital format, you may install one (1) copy on up to three (3) Personal Devices. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.
2. For each license you acquire on behalf of an End User or Trainer, you may either:
 1. distribute one (1) hard copy version of the Microsoft Instructor-Led Courseware to one (1) End User attending the Private Training Session, and only immediately prior to the commencement of the Private Training Session that is the subject matter of the Microsoft Instructor-Led Courseware being provided, **or**
 2. provide one (1) End User who is attending the Private Training Session with the unique redemption code and instructions on how they can access one (1) digital version of the Microsoft Instructor-Led Courseware, **or**
 3. you will provide one (1) Trainer who is teaching the Private Training Session with the unique redemption code and instructions on how they can access one (1) Trainer Content.
3. For each license you acquire, you must comply with the following:
 1. you will only provide access to the Licensed Content to those individuals who have acquired a valid license to the Licensed Content,
 2. you will ensure that each End User attending an Private Training Session has their own valid licensed copy of the Microsoft Instructor-Led Courseware that is the subject of the Private Training Session,
 3. you will ensure that each End User provided with a hard copy version of the Microsoft Instructor-Led Courseware will be presented with a copy of this agreement and each End User will agree that their use of the Microsoft Instructor-Led Courseware will be subject to the terms in this agreement prior to providing them with the Microsoft Instructor-Led Courseware. Each individual will be required to denote their acceptance of this agreement in a manner that is enforceable under local law prior to their accessing the Microsoft Instructor-Led Courseware,
 4. you will ensure that each Trainer teaching an Private Training Session has their own valid licensed copy of the Trainer Content that is the subject of the Private Training Session,

5. you will only use qualified Trainers who hold the applicable Microsoft Certification credential that is the subject of the Microsoft Instructor-Led Courseware being taught for all your Private Training Sessions,
6. you will only use qualified MCTs who hold the applicable Microsoft Certification credential that is the subject of the MOC title being taught for all your Private Training Sessions using MOC,
7. you will only provide access to the Microsoft Instructor-Led Courseware to End Users, and
8. you will only provide access to the Trainer Content to Trainers.

4. If you are an End User:

For each license you acquire, you may use the Microsoft Instructor-Led Courseware solely for your personal training use. If the Microsoft Instructor-Led Courseware is in digital format, you may access the Microsoft Instructor-Led Courseware online using the unique redemption code provided to you by the training provider and install and use one (1) copy of the Microsoft Instructor-Led Courseware on up to three (3) Personal Devices. You may also print one (1) copy of the Microsoft Instructor-Led Courseware. You may not install the Microsoft Instructor-Led Courseware on a device you do not own or control.

5. If you are a Trainer.

1. For each license you acquire, you may install and use one (1) copy of the Trainer Content in the form provided to you on one (1) Personal Device solely to prepare and deliver an Authorized Training Session or Private Training Session, and install one (1) additional copy on another Personal Device as a backup copy, which may be used only to reinstall the Trainer Content. You may not install or use a copy of the Trainer Content on a device you do not own or control. You may also print one (1) copy of the Trainer Content solely to prepare for and deliver an Authorized Training Session or Private Training Session.

2. If you are an MCT, you may customize the written portions of the Trainer Content that are logically associated with instruction of a training session in accordance with the most recent version of the MCT agreement.
3. If you elect to exercise the foregoing rights, you agree to comply with the following: (i) customizations may only be used for teaching Authorized Training Sessions and Private Training Sessions, and (ii) all customizations will comply with this agreement. For clarity, any use of "customize" refers only to changing the order of slides and content, and/or not using all the slides or content, it does not mean changing or modifying any slide or content.

- 2.2 **Separation of Components.** The Licensed Content is licensed as a single unit and you may not separate their components and install them on different devices.
- 2.3 **Redistribution of Licensed Content.** Except as expressly provided in the use rights above, you may not distribute any Licensed Content or any portion thereof (including any permitted modifications) to any third parties without the express written permission of Microsoft.
- 2.4 **Third Party Notices.** The Licensed Content may include third party code that Microsoft, not the third party, licenses to you under this agreement. Notices, if any, for the third party code are included for your information only.
- 2.5 **Additional Terms.** Some Licensed Content may contain components with additional terms, conditions, and licenses regarding its use. Any non-conflicting terms in those conditions and licenses also apply to your use of that respective component and supplements the terms described in this agreement.

3. **LICENSED CONTENT BASED ON PRE-RELEASE TECHNOLOGY.** If the Licensed Content's subject matter is based on a pre-release version of Microsoft technology ("Pre-release"), then in addition to the other provisions in this agreement, these terms also apply:
 1. **Pre-Release Licensed Content.** This Licensed Content subject matter is on the Pre-release version of the Microsoft technology. The technology may not work the way a final version of the technology will and we may change the technology for the final version. We also may not release a final version. Licensed Content based on the final version of the technology may not contain the same information as the Licensed Content based on the Pre-release version. Microsoft is under no obligation to provide you with any further content, including any Licensed Content based on the final version of the technology.
 2. **Feedback.** If you agree to give feedback about the Licensed Content to Microsoft, either directly or through its third party designee, you give to Microsoft without charge, the right to use, share and commercialize your feedback in any way and for any purpose. You also give to third parties, without charge, any patent rights needed for their products, technologies and services to use or interface with any specific parts of a Microsoft technology, Microsoft product, or service that includes the feedback. You will not give feedback that is subject to a license that requires Microsoft to license its technology, technologies, or products to third parties because we include your feedback in them. These rights survive this agreement.
 3. **Pre-release Term.** If you are an Microsoft Imagine Academy Program Member, Microsoft Learning Competency Member, MPN Member, Microsoft Learn for Educators – Validated Educator, or Trainer, you will cease using all copies of the Licensed Content on the Pre-release technology upon (i) the date which Microsoft informs you is the end date for using the Licensed Content on the Pre-release technology, or (ii) sixty (60) days after the commercial release of the technology that is the subject of the Licensed Content, whichever is earliest ("Pre-release term"). Upon expiration or termination of the Pre-release term, you will irretrievably delete and destroy all copies of the Licensed Content in your possession or under your control.
4. **SCOPE OF LICENSE.** The Licensed Content is licensed, not sold. This agreement only gives you some rights to use the Licensed Content. Microsoft reserves all other rights. Unless applicable law gives you more rights despite this limitation, you may use the Licensed Content only as expressly permitted in this agreement. In doing so, you must comply with any technical limitations in the Licensed Content that only allows you to use it in certain ways. Except as expressly permitted in this agreement, you may not:
 - access or allow any individual to access the Licensed Content if they have not acquired a valid license for the Licensed Content,
 - alter, remove or obscure any copyright or other protective notices (including watermarks), branding or identifications contained in the Licensed Content,
 - modify or create a derivative work of any Licensed Content,
 - publicly display, or make the Licensed Content available for others to access or use,
 - copy, print, install, sell, publish, transmit, lend, adapt, reuse, link to or post, make available or distribute the Licensed Content to any third party,
 - work around any technical limitations in the Licensed Content, or
 - reverse engineer, decompile, remove or otherwise thwart any protections or disassemble the Licensed Content except and only to the extent that applicable law expressly permits, despite this limitation.
5. **RESERVATION OF RIGHTS AND OWNERSHIP.** Microsoft reserves all rights not expressly granted to you in this agreement. The Licensed Content is protected by copyright and other intellectual property

laws and treaties. Microsoft or its suppliers own the title, copyright, and other intellectual property rights in the Licensed Content.

6. **EXPORT RESTRICTIONS.** The Licensed Content is subject to United States export laws and regulations. You must comply with all domestic and international export laws and regulations that apply to the Licensed Content. These laws include restrictions on destinations, end users and end use. For additional information, see www.microsoft.com/exporting.
7. **SUPPORT SERVICES.** Because the Licensed Content is provided "as is", we are not obligated to provide support services for it.
8. **TERMINATION.** Without prejudice to any other rights, Microsoft may terminate this agreement if you fail to comply with the terms and conditions of this agreement. Upon termination of this agreement for any reason, you will immediately stop all use of and delete and destroy all copies of the Licensed Content in your possession or under your control.
9. **LINKS TO THIRD PARTY SITES.** You may link to third party sites through the use of the Licensed Content. The third party sites are not under the control of Microsoft, and Microsoft is not responsible for the contents of any third party sites, any links contained in third party sites, or any changes or updates to third party sites. Microsoft is not responsible for webcasting or any other form of transmission received from any third party sites. Microsoft is providing these links to third party sites to you only as a convenience, and the inclusion of any link does not imply an endorsement by Microsoft of the third party site.
10. **ENTIRE AGREEMENT.** This agreement, and any additional terms for the Trainer Content, updates and supplements are the entire agreement for the Licensed Content, updates and supplements.
11. **APPLICABLE LAW.**
 1. United States. If you acquired the Licensed Content in the United States, Washington state law governs the interpretation of this agreement and applies to claims for breach of it, regardless of conflict of laws principles. The laws of the state where you live govern all other claims, including claims under state consumer protection laws, unfair competition laws, and in tort.
 2. Outside the United States. If you acquired the Licensed Content in any other country, the laws of that country apply.
12. **LEGAL EFFECT.** This agreement describes certain legal rights. You may have other rights under the laws of your country. You may also have rights with respect to the party from whom you acquired the Licensed Content. This agreement does not change your rights under the laws of your country if the laws of your country do not permit it to do so.
13. **DISCLAIMER OF WARRANTY. THE LICENSED CONTENT IS LICENSED "AS-IS" AND "AS AVAILABLE." YOU BEAR THE RISK OF USING IT. MICROSOFT AND ITS RESPECTIVE AFFILIATES GIVES NO EXPRESS WARRANTIES, GUARANTEES, OR CONDITIONS. YOU MAY HAVE ADDITIONAL CONSUMER RIGHTS UNDER YOUR LOCAL LAWS WHICH THIS AGREEMENT CANNOT CHANGE. TO THE EXTENT PERMITTED UNDER YOUR LOCAL LAWS, MICROSOFT AND ITS RESPECTIVE AFFILIATES EXCLUDES ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT.**
14. **LIMITATION ON AND EXCLUSION OF REMEDIES AND DAMAGES. YOU CAN RECOVER FROM MICROSOFT, ITS RESPECTIVE AFFILIATES AND ITS SUPPLIERS ONLY DIRECT DAMAGES UP TO US\$5.00. YOU CANNOT RECOVER ANY OTHER DAMAGES, INCLUDING CONSEQUENTIAL, LOST PROFITS, SPECIAL, INDIRECT OR INCIDENTAL DAMAGES.**

This limitation applies to

- anything related to the Licensed Content, services, content (including code) on third party Internet sites or third-party programs; and
- claims for breach of contract, breach of warranty, guarantee or condition, strict liability, negligence, or other tort to the extent permitted by applicable law.

It also applies even if Microsoft knew or should have known about the possibility of the damages. The above limitation or exclusion may not apply to you because your country may not allow the exclusion or limitation of incidental, consequential, or other damages.

Please note: As this Licensed Content is distributed in Quebec, Canada, some of the clauses in this agreement are provided below in French.

Remarque : Ce le contenu sous licence étant distribué au Québec, Canada, certaines des clauses dans ce contrat sont fournies ci-dessous en français.

EXONÉRATION DE GARANTIE. Le contenu sous licence visé par une licence est offert « tel quel ». Toute utilisation de ce contenu sous licence est à votre seule risque et péril. Microsoft n'accorde aucune autre garantie expresse. Vous pouvez bénéficier de droits additionnels en vertu du droit local sur la protection dues consommateurs, que ce contrat ne peut modifier. La ou elles sont permises par le droit locale, les garanties implicites de qualité marchande, d'adéquation à un usage particulier et d'absence de contrefaçon sont exclues.

LIMITATION DES DOMMAGES-INTÉRÊTS ET EXCLUSION DE RESPONSABILITÉ POUR LES DOMMAGES. Vous pouvez obtenir de Microsoft et de ses fournisseurs une indemnisation en cas de dommages directs uniquement à hauteur de 5,00 \$ US. Vous ne pouvez prétendre à aucune indemnisation pour les autres dommages, y compris les dommages spéciaux, indirects ou accessoires et pertes de bénéfices.

Cette limitation concerne:

- tout ce qui est relié au le contenu sous licence, aux services ou au contenu (y compris le code) figurant sur des sites Internet tiers ou dans des programmes tiers; et.
- les réclamations au titre de violation de contrat ou de garantie, ou au titre de responsabilité stricte, de négligence ou d'une autre faute dans la limite autorisée par la loi en vigueur.

Elle s'applique également, même si Microsoft connaissait ou devrait connaître l'éventualité d'un tel dommage. Si votre pays n'autorise pas l'exclusion ou la limitation de responsabilité pour les dommages indirects, accessoires ou de quelque nature que ce soit, il se peut que la limitation ou l'exclusion ci-dessus ne s'appliquera pas à votre égard.

EFFET JURIDIQUE. Le présent contrat décrit certains droits juridiques. Vous pourriez avoir d'autres droits prévus par les lois de votre pays. Le présent contrat ne modifie pas les droits que vous confèrent les lois de votre pays si celles-ci ne le permettent pas.

Revised April 2019



Contents

■	Module 0 Welcome to Building Applications and Solutions with Microsoft 365 Core Services	1
	Welcome to Building Applications and Solutions with Microsoft 365 Core Services	1
■	Module 1 Implement Microsoft Identity	3
	Getting started with Microsoft identity	3
	Overview of authentication and authorization with Microsoft identity	23
	Implement authentication and authorization to consume an API	43
	Secure custom APIs	61
■	Module 2 Work with Microsoft Graph	79
	Overview of Microsoft Graph	79
	Optimize Data Usage with Query Parameters	86
	Optimize Network Traffic	93
	Access User Data with Microsoft Graph	105
	Access Files with Microsoft Graph	116
	Manage a Group Lifecycle on Microsoft Graph	136
■	Module 3 Extend and Customize SharePoint	153
	SharePoint Framework Web Parts	153
	SharePoint Framework Extension	172
	Package and Deploy an SPFx Solution	193
	Consumption of Microsoft Graph and Third Party APIs	218
	Web Parts as Teams Tabs	230
	Branding and Theming in SharePoint	243
■	Module 4 Extend Teams	251
	Overview of Building Apps for Microsoft Teams	251
	Webhooks in Microsoft Teams	268
	Tabs in Microsoft Teams	272
	Messaging Extensions in Microsoft Teams	280
	Conversational Bots in Microsoft Teams	294
■	Module 5 Extend Office	299
	Office Add-ins	299
	Office JS APIs	312
	Customize Office Add-ins	334
	Testing Debugging and Deployment Options for Office Add-ins	340

Actionable Messages

346

Module 0 Welcome to Building Applications and Solutions with Microsoft 365 Core Services

Welcome to Building Applications and Solutions with Microsoft 365 Core Services

About this course

Welcome to the **Building Applications and Solutions with Microsoft 365 Core Services** course!

Building applications and solutions with Microsoft 365 core services course is designed for persons who are aspiring to the Microsoft 365 Developer role. This course covers five central elements of Microsoft 365 platform – implementing Microsoft Identity, working with Microsoft Graph, extending and customizing SharePoint, extending Teams, and extending Office. In this course, students will learn how to implement Microsoft Identity and work with Microsoft Graph. Students will also gain the knowledge on UI elements (including Adaptive Cards and UI Fabric), Integration Points (including Microsoft Teams, Office Add-ins, SharePoint Framework, Actionable Messages), and determining workload platform targets.

Level: Intermediate

Audience

Students in this course are interested in Microsoft 365 development platform or in passing the Microsoft 365 Developer Associate certification exam. Students should also have 1-2 years experience as a developer. This course assumes students know how to code and have a basic understanding of REST APIs, JSON, OAuth2, OData, OpenID Connect, Microsoft identities including Azure AD and Microsoft accounts, Azure AD B2C, and permission/consent concepts.

Prerequisites

This course assumes you have already acquired the following skills and experience:

- Students should have 1-2 years experience as a developer. This course assumes students know how to code and have a basic understanding of REST APIs, JSON, OAuth2, OData, OpenID Connect, Microsoft

identities including Azure AD and Microsoft accounts, Azure AD B2C, and permission/consent concepts.

- It is recommended that students have some experience developing solutions on Microsoft Teams, Office Add-ins, or SharePoint Framework through all phases of software development.

Learning objectives

By actively participating in this course, you will learn about the following:

- Implementing Microsoft Identity
- Working with Microsoft Graph
- Determining workload platform targets
- Integration Points, including Microsoft Teams, Office Add-ins, and SharePoint Framework

Certification exam preparation

This course helps you prepare for the

Exam MS-600: Building Applications and Solutions with Microsoft 365 Core Services¹ certification exam.

MS-600 includes five study areas, as shown in the table. The percentages indicate the relative weight of each area on the exam. The higher the percentage, the more questions you are likely to see in that area.

MS-600 Study Areas	Weight
Implement Microsoft Identity	20-25%
Work with Microsoft Graph	20-25%
Extend and Customize SharePoint	20-25%
Extend Teams	15-20%
Extend Office	15-20%

✓ Note: The relative weightings are subject to change. For the latest information visit the [exam page²](#) and review the Skills measured section.

Passing the exam will earn you the [Microsoft 365 Certified: Developer Associate³](#) certification.

The modules in the course are mapped to the objectives listed in each study area on the [Skills Measured⁴](#) sheet so it will be easy for you to focus on areas of the exam you choose to revisit.

¹ <https://docs.microsoft.com/en-us/learn/certifications/exams/ms-600>

² <https://docs.microsoft.com/en-us/learn/certifications/exams/ms-600>

³ <https://docs.microsoft.com/en-us/learn/certifications/m365-developer-associate>

⁴ <https://query.prod.cms.rt.microsoft.com/cms/api/am/binary/RE43VcG>

Module 1 Implement Microsoft Identity

Getting started with Microsoft identity

Lesson introduction

Microsoft identity platform is an evolution of the Azure Active Directory (Azure AD) developer platform. It allows developers to build applications that sign in users and access resources in both external applications such as Microsoft Office 365, the Azure portal, and thousands of other SaaS applications, and internal resources, such as apps on your corporate network and intranet, along with any cloud apps developed by your own organization.

In this lesson, you will learn the basics of Microsoft identity and developer platform, including the different identity elements, features, and supported topologies. You will also learn how to register an application with the Microsoft identity platform and how applications are represented within the Microsoft identity platform.

After this lesson, you should be able to:

- Describe the Microsoft identity platform.
- Describe the supported identity topologies.
- Register an application with Microsoft identity.
- Determine the supported account types during app registration.
- Define app roles.

Why use an identity provider?

As a developer, you must examine the requirements of your app to determine whether it will require identity services, and if so, which services. These considerations include the following:

- How customers will get your app.
- Whether the app will work on behalf of a user or as the app itself.
- The resources required by the app.
- Whether the app must ask the user for permissions to use a resource.

Instead of creating apps that each maintain their own username and password information, which incurs a high administrative burden when you need to add or remove users across multiple apps, apps can delegate authentication and authorization responsibilities to a centralized **identity provider**.

- **Authentication** is the act of challenging a party for legitimate credentials, providing the basis for creation of a security principal to be used for identity and access control. In simpler terms, it's the process of proving you are who you say you are. Authentication is sometimes shortened to AuthN.
- **Authorization** is the act of granting an authenticated party permission to do something. It specifies what data you're allowed to access and what you can do with that data. Authorization is sometimes shortened to AuthZ.

The identity provider is where the users, resources, and policies intersect within your organization.

Azure Active Directory and Microsoft identity platform

Azure Active Directory (Azure AD) is a centralized identity provider in the cloud. Azure AD authenticates users and provides access tokens, which can be used to access Web APIs and other protected resources.

Delegating authentication and authorization enables scenarios such as:

- Conditional Access policies that require a user to be in a specific location
- The use of multi-factor authentication
- Single sign-on (SSO), which enables a user to sign in once and then be automatically signed in to all of the web apps that share the same centralized directory

The **Microsoft identity platform** simplifies authentication for application developers by providing identity as a service, with support for industry-standard protocols such as **OAuth 2.0** and **OpenID Connect**, as well as open-source libraries for different platforms. It allows developers to build applications that sign in all Microsoft identities and get tokens to **call Microsoft Graph**, other Microsoft APIs, or APIs that developers have built.

Overview of Microsoft identity

The following is how Microsoft identity approaches users, resources, and policies.

Users: Azure AD, B2B & B2C

Microsoft identity includes three solutions for addressing the largest number of users.

- **Azure AD – Employees and Enterprise Resources** : Azure Active Directory (Azure AD) is Microsoft's cloud-based identity and access management service, which helps your employees sign in and access resources. This includes all customers of Office 365 and Microsoft Azure use Azure Active Directory (Azure AD).
- **Azure AD B2B – Partners** : Azure AD business to business (B2B) enables an organization in Azure AD to securely share files and resources with another organization in Azure AD.
- **Azure AD B2C – Customers/citizens**: Azure AD Business to Customer (B2C) enables an organization to provide customer-facing apps that enable users to sign in with an identity they already have, such as a Microsoft Account, Gmail, Facebook, or Twitter account or create an account for the solution.

Resources: your data in Microsoft cloud

Microsoft Graph is the gateway to all your data in the Microsoft Cloud. From Office 365, Enterprise Mobility + Security to Windows 10, Microsoft Graph enables developers to build applications that provide users access to their data. Developers can access their data with Microsoft Graph by authenticating and authorizing users with Microsoft identity.

Policies

The third element of Microsoft identity is policies. Policies have become complicated with users accessing resources from different networks and devices. The threat matrix depending on how a user is accessing a resource is complex. Various combinations of different users using different devices pose different risks to the organization.

Microsoft identity is built to support this complex and dynamic threat matrix. One policy can be applied when a user is on a trusted corporate network using their company issued laptop, while another policy can apply to the same user when they access the same resource from home on their personal tablet.

By relying on Microsoft identity to handle these complex policies, you don't have to implement the complex logic in your custom applications.

Overview of Microsoft identity platform

Microsoft identity platform (v. 2.0) is an evolution of the Azure Active Directory v1.0 developer platform. It allows developers to build applications that sign in users and access resources in both external applications, such as Microsoft Office 365, the Azure portal, and thousands of other SaaS applications, and internal resources, such as apps on your corporate network and intranet, along with any cloud apps developed by your own organization. The Microsoft identity platform consists of:

- **OAuth 2.0 and OpenID Connect standard-compliant authentication service:** Enables developers to authenticate any Microsoft identity, including:
 - Work or school accounts (provisioned through Azure AD)
 - Personal Microsoft accounts (such as Skype, Xbox, and Outlook.com)
 - Social or local accounts (via Azure AD B2C)
- **Open-source libraries:** Microsoft Authentication Libraries (MSAL) and support for other standards-compliant libraries.
- **Application management portal:** A registration and configuration experience built in the Azure portal, along with all your other Azure management capabilities.
- **Application configuration API and PowerShell:** Allows programmatic configuration of your applications through REST API (Microsoft Graph and Azure Active Directory Graph 1.6) and PowerShell, so you can automate your DevOps tasks.
- **Developer content:** Conceptual and reference documentation, quickstart samples, code samples, tutorials, and how-to guides.

For developers, Microsoft identity platform offers seamless integration into innovations in the identity and security space, such as passwordless authentication, step-up authentication, and Conditional Access. You don't need to implement such functionality yourself: applications integrated with the Microsoft identity platform natively take advantage of such innovations.

With Microsoft identity platform, you can write code once and reach any user. You can build an app once and have it work across many platforms or build an app that functions as a client, as well as a resource application (API).

Microsoft identity platform experience

With the unified Microsoft identity platform (v2.0), you can write code once and authenticate any Microsoft identity into your application.

The Microsoft identity platform provides the following development experience:

App registration experience: The Azure portal App registrations experience is the one portal experience for managing all applications you've integrated with Microsoft identity platform. The Azure portal app registration should be used instead of the Application Registration Portal.

Client SDK: You can use the Microsoft Authentication Library (MSAL) to build applications that authenticate all Microsoft identities. MSAL is Microsoft's latest identity platform solution and is preferred to ADAL.

Endpoint: Microsoft identity platform (v2.0) endpoint is now OIDC certified. It works with the Microsoft Authentication Libraries (MSAL) or any other standards-compliant library. It implements human readable scopes, in accordance with industry standards.

Supported account types and topologies

The Microsoft identity platform can sign in users with the following types of accounts:

- work or school accounts (Azure AD)
- personal accounts (Microsoft Accounts, such as Skype, Xbox, Outlook.com)
- social or local accounts

You can decide and configure which account types your application will support.

Single-tenant vs. multi-tenant applications

Azure Active Directory (Azure AD) organizes objects like users and apps into groups called **tenants**. Tenants allow an administrator to set policies on the users within the organization and the apps that the organization owns to meet their security and operational policies.

Who can sign in to your app?

If you're writing a line-of-business (LOB) application, you can sign in users in your own organization. Such an application can be referred to as a **single-tenant** application as it can only sign in users in the tenant it was registered in (known as its home tenant)

You can also write **multi-tenant** applications that sign in users from more than 1 organization. Such applications can sign in users:

- *In any Azure AD directory:* All users and guests with a work or school account from Microsoft can use your application or API. This includes schools and businesses that use Office 365. You should use this option if your target audience is business or educational customers.
- *In any Azure AD directory + personal Microsoft accounts:* All users with a work or school, or personal Microsoft account can use your application or API. It includes schools and businesses that use Office

365 as well as personal accounts that are used to sign in to services like Xbox and Skype. Use this option to target the widest set of Microsoft accounts.

- If you're writing a business-to-consumer application, you can also sign in users with their social identities, by using **Azure Active Directory B2C (Azure AD B2C)**.

Topologies

The topologies supported by Microsoft identity are described further in this section.

Consumer

Consumers refer to Microsoft Accounts (MSAs) that are created by users for personal use. Microsoft Accounts allow users to sign in to all consumer-oriented Microsoft products and cloud services such as Outlook, OneDrive, MSN, or Xbox LIVE.

The Microsoft identity platform enables developers to create consumer-facing applications that allow users to sign in using either a Microsoft account (personal) or a work or school account (organizational).

While there are many similarities between Microsoft Accounts and Azure AD accounts, Microsoft Accounts are primarily for personal or consumer services. Some small business will also use consumer accounts and consumer services like email and OneDrive. Some similarities between the topologies include account and password management, support for multi-factor authentication, and passwordless authentication. Unlike Microsoft Accounts, Azure AD accounts have additional security-related features that are available in enterprise scenarios.

As an app developer, you can use Microsoft Account as a standards-based approach for adding SSO to your app, allowing it to work with a user's pre-existing credentials. Microsoft identity platform also provides APIs that can help you build personalized app experiences using the user's personal Office 365 resources.

Enterprise

Enterprises refer to organizational accounts in Azure Active Directory (Azure AD), Microsoft's cloud-based identity and access management service. Microsoft identity platform helps your employees sign in and access internal resources such as apps in your corporate network or intranet and cloud apps that are developed by your organization. In addition to internal resources, your employees can access external resources such as Microsoft Office 365 and many other software-as-a-service (SaaS) applications.

As an app developer, you can use Microsoft identity platform as a standards-based approach for adding SSO to your app, allowing it to work with a user's pre-existing credentials. Microsoft identity platform also provides APIs that can help you build personalized app experiences using existing organizational data.

Business-to-business

Microsoft identity platform business-to-business (B2B) collaboration lets you securely share your company's applications and services with guest users from any other organization, while maintaining control over your own corporate data. You can work safely and securely with external partners, large, or small, even if they don't have Azure AD or an IT department. A simple invitation and redemption process enables partners to use their own credentials to access the company's resources.

Developers can use Microsoft identity platform business-to-business APIs to customize the invitation process or write applications like self-service sign-up portals. With Microsoft identity platform B2B, the

partner uses their own identity management solution, so there is no external administrative overhead for your organization and Azure AD isn't required.

Business-to-consumer

Microsoft identity platform business-to-consumer (B2C) is primarily for businesses and developers that create customer-facing apps. With Microsoft identity platform B2C, developers can use Microsoft identity platform as the full-featured identity system for their application, while letting customers sign in with an identity they already have established (like Facebook or Gmail).

Microsoft identity platform B2C is a customer identity access management (CIAM) solution capable of supporting millions of users and billions of authentications per day. It takes care of the scaling and safety of the authentication platform, performs monitoring, and automatically handling threats like denial-of-service, password spray, and brute force attacks. Microsoft identity platform B2C uses standards-based authentication protocols, including OpenID Connect and OAuth 2.0.

With Microsoft identity platform B2C, users will authenticate with their preferred identity provider that has been configured by your IT administrator. The user signs in using a customized sign-in page hosted by Microsoft identity platform B2C. Microsoft identity platform B2C is a white-label authentication solution. This means you can customize the entire user experience with your brand so that it blends seamlessly with your web and mobile applications.

Register an application

Developers that leverage the Microsoft identity platform within custom apps begin with registering an application in Azure AD. It doesn't matter if the app runs on a mobile device, in the cloud, or what cloud provider it runs. The app can even be a PowerShell script. The Azure AD application is the control point for how you access the identity and the information it protects. For an identity provider to know that a user has access to a particular app, both the user and the application must be registered with the identity provider.

When you register your application with Azure AD, you are providing an identity configuration for your application that allows it to integrate with Azure AD. Registering the app also allows you to:

- customize the branding of your application in the sign-in dialog. This is important because this is the first experience a user will have with your app.
- decide which user account types to support
- request scope permissions. For example, you can request the "user.read" scope, which grants permission to read the profile of the signed-in user.
- define scopes that define access to your Web API. Typically, when an app wants to access your API, it will need to request permissions to the scopes you define.
- share a secret with Azure AD that proves the app's identity to Azure AD. This is relevant in the case where the app is a confidential client application. A confidential client application is an application that can hold credentials securely. They require a trusted backend server to store the credentials.

Once registered, the application will be given a GUID that the app shares with Azure AD when it requests tokens. If the app is a confidential client application, it will also share the secret or the public key, depending on whether certificates or secrets were used.

App registration process

When you register an app, the process will automatically fill some of the fields in your registration and also provide you with code snippets to add to your web application's web.config file. These snippets are how Microsoft identity will recognize your app.

The following steps show you how to register a new application using the App registrations experience in the Azure portal so that your app can be integrated with the Microsoft identity platform.

1. Log in to the Azure portal (portal.azure.com).

If your account gives you access to more than one tenant, select your account in the top right corner, and set your portal session to the Azure AD tenant that you want.

2. From the left navigation menu, select **Azure Active Directory**. On the **Active Directory** page, select **App registrations**, and then select **New registration**.
3. When the **Register an application** page appears, enter your application's registration information:
 - **Name** - Enter a meaningful application name that will be displayed to users of the app.
 - **Supported account types** - Select which accounts you would like your application to support.

[Dashboard](#) > [App registrations](#) >

Register an application

* Name

The user-facing display name for this application (this can be changed later).

ContosoApp_1 

Supported account types

Who can use this application or access this API?

- Accounts in this organizational directory only (Contoso Enterprises - Single tenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant)
- Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)
- Personal Microsoft accounts only

[Help me choose...](#)

Redirect URI (optional)

We'll return the authentication response to this URI after successfully authenticating the user. Providing this now is optional and it can be changed later, but a value is required for most authentication scenarios.

Web



<https://ContosoApp1/auth> 

[By proceeding, you agree to the Microsoft Platform Policies](#) 

Register

The following table summarizes the options for supported account types:

Supported account types	Description
Accounts in this organizational directory only	Select this option if you're building a line-of-business (LOB) application. This option is not available if you're not registering the application in a directory. This option maps to Azure AD single-tenant, and is the default option unless you're registering the app outside of a directory.
Accounts in any organizational directory	Select this option if you would like to target all business and educational customers. This option maps to an Azure AD only multi-tenant.
Accounts in any organizational directory and personal Microsoft accounts	Select this option to target the widest set of customers. This option maps to Azure AD multi-tenant and personal Microsoft accounts. This is the default option if you are registering an app outside of a directory.

- **Redirect URI (optional)** - Select the type of app you're building, **Web** or **Public client (mobile & desktop)**, and then enter the redirect URI (or reply URL) for your application.
 - For web applications, provide the base URL of your app. For example, <https://localhost:31544> might be the URL for a web app running on your local machine. Users would use this URL to sign in to a web client application.
 - For public client applications, provide the URI used by Azure AD to return token responses. Enter a value specific to your application, such as `myapp://auth`.

4. When finished, select **Register**.

Azure AD assigns a unique application (client) ID to your app, and you're taken to your application's **Overview** page. To add additional capabilities to your application, you can select other configuration options including branding, certificates and secrets, API permissions, and more.

Dashboard > App registrations >

ContosoApp_1

[Delete](#) [Endpoints](#) [Preview features](#)

[Search \(Ctrl+ /\)](#)

Overview [Manage](#)

- [Quickstart](#)
- [Integration assistant | Preview](#)
- [Branding](#)
- [Authentication](#)
- [Certificates & secrets](#)
- [Token configuration](#)
- [API permissions](#)
- [Expose an API](#)
- [Owners](#)
- [Roles and administrators | Preview](#)
- [Manifest](#)

Support + Troubleshooting

- [Troubleshooting](#)
- [New support request](#)

Essentials

Display name ContosoApp_1	Supported account types My organization only
Application (client) ID 0f8022a9-a81a-4e8e-8fdc-beef8825936e	Redirect URIs 1 web, 0 spa, 0 public client
Directory (tenant) ID 0534985e-032e-430a-9b95-60f5277b96f4	Application ID URI Add an Application ID URI
Object ID adf5dca8-6ecf-4fdb-bf79-867998dc8197	Managed application in local directory ContosoApp_1

Call APIs

Build more powerful apps with rich user and business data from Microsoft services and your own company's data sources.

[View API permissions](#)

Documentation

- [Microsoft identity platform](#)
- [Authentication scenarios](#)
- [Authentication libraries](#)
- [Code samples](#)
- [Microsoft Graph](#)
- [Glossary](#)
- [Help and Support](#)

Sign in users in 5 minutes

Use our SDKs to sign in users and call APIs in a few steps

[View all quickstart guides](#)

The application manifest

After you've done your initial registration, you decide that you've selected the wrong account type. How do you change this property for your application?

Some application properties can be added or changed after registration by using the app registration UI. Other properties, once created, must be changed in the application manifest. For example, if you need to change your account type, the property is no longer available to change in the UI. You'll have to make the change in the manifest.

Azure AD app manifest

The **application manifest** contains a definition of all the attributes of an application object in the Microsoft identity platform. It also serves as a mechanism for updating the application object. The manifest is where you can record additional metadata for your application. The metadata includes things such as a link to a logo image, Terms of Use, and a privacy statement, as well as defining group membership, role definitions, and permissions.

You can configure an app's attributes through the Azure portal or programmatically using REST API or PowerShell. However, there are some scenarios where you'll need to edit the app manifest to configure an app's attribute. These scenarios include:

- If you registered the app as Azure AD multi-tenant and personal Microsoft accounts, you can't change the supported Microsoft accounts in the UI. Instead, you must use the application manifest editor to change the supported account type.
- If you need to define permissions and roles that your app supports, you must modify the application manifest.

Configure the app manifest

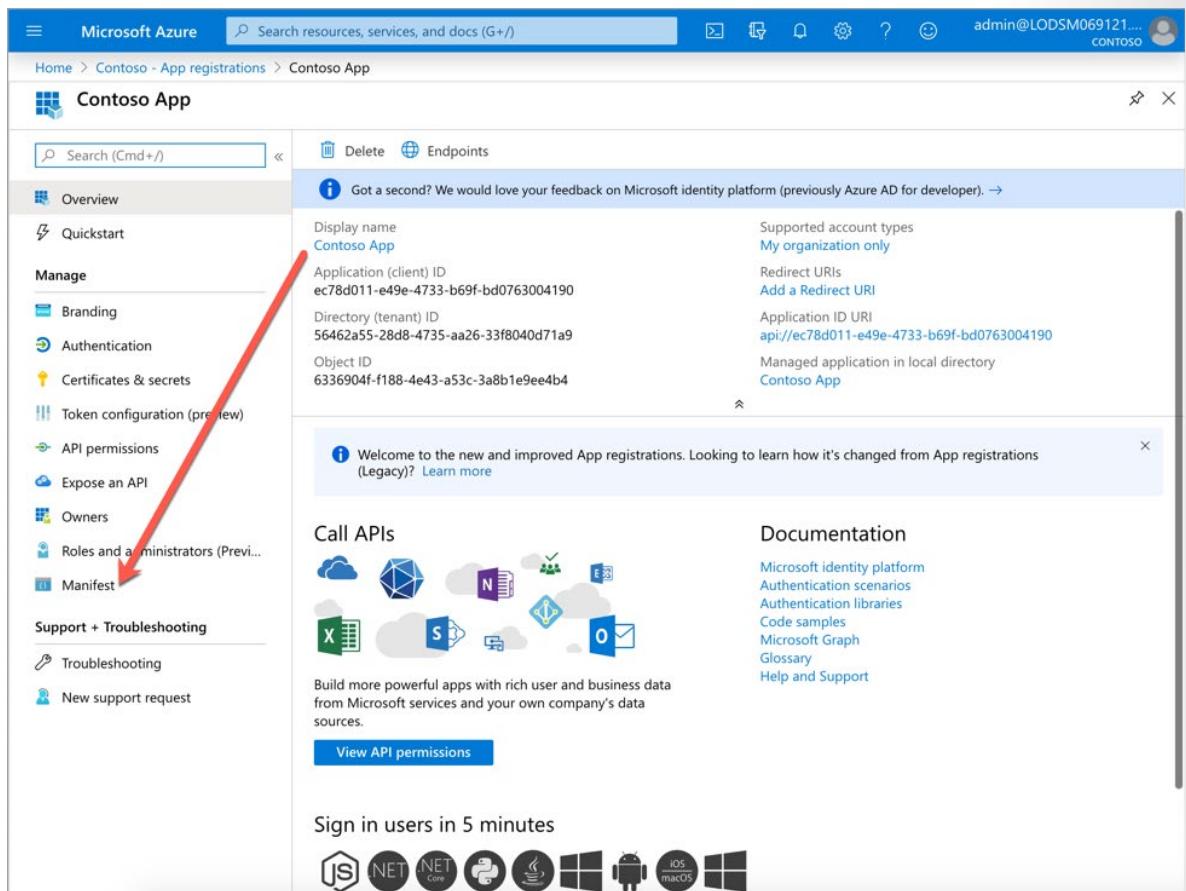
The manifest gets created automatically by the Microsoft identity platform during the app registration process (when you click Register). It defines your application entity object and follows the schema defined by Microsoft Graph.

1. To configure the app manifest, you would need to go to **App registrations** in **Azure Active Directory** and select the app you want to configure.

The screenshot shows the Azure portal interface for managing app registrations. The top navigation bar includes 'Microsoft Azure', a search bar, and user information ('admin@LODSM069121... CONTOSO'). Below the bar, the URL 'Home > Contoso - App registrations' is visible. The main content area is titled 'Contoso - App registrations' under 'Azure Active Directory'. A red arrow points from the 'App registrations' menu item in the left sidebar to the 'Contoso App' entry in the list of applications. The list table has columns for 'Display name', 'Application (client) ID', 'Created On', and 'Certificates & secrets'. The 'Contoso App' entry has a red box around its 'Display name' column. The left sidebar also lists other management options like 'Users', 'Groups', 'Organizational relationships', etc.

Display name	Application (client) ID	Created On	Certificates & secrets
CA Contoso App	ec78d011-e49e-4733-b69f...	12/16/2019	-
WE WebApp-OpenIDConnect-DotNet	9012fa9f-86b6-4801-9ea7...	12/16/2019	-
JA JavaScript-SPA-App	b3ff53fb-e176-4ce9-8947...	12/16/2019	-
GR GraphQuickstartApp	0c292494-708f-4661-9481...	12/16/2019	-

2. From the app's overview page, you would then select **Manifest** from the left navigation menu.



The screenshot shows the Microsoft Azure App registrations portal for a 'Contoso App'. The left sidebar has a red arrow pointing to the 'Manifest' link under the 'Manage' section. The main content area displays the application's manifest with fields like Display name (Contoso App), Application (client) ID (e78d011-e49e-4733-b69f-bd0763004190), and Redirect URIs. It also includes sections for 'Call APIs' (with icons for various services like Cloud, Graph, SharePoint, etc.) and 'Documentation' (links to Microsoft identity platform, Authentication scenarios, etc.). A welcome message at the top right says 'Welcome to the new and improved App registrations. Looking to learn how it's changed from App registrations (Legacy)? Learn more'.

3. The portal provides a manifest editor to make changes online easy. You can also download the manifest, edit locally, and upload again.

The screenshot shows the Microsoft Azure portal interface. In the top navigation bar, there's a search bar with the placeholder "Search resources, services, and docs (G+)" and a user account icon for "admin@LODSM069121... CONTOSO". Below the navigation bar, the breadcrumb trail shows "Home > Contoso - App registrations > Contoso App - Manifest". The main content area has a title "Contoso App - Manifest" and a sub-header "Contoso App - Manifest". On the left, a sidebar menu includes "Overview", "Quickstart", "Manage" (with "Branding", "Authentication", "Certificates & secrets", "Token configuration (preview)", "API permissions", "Expose an API", "Owners", "Roles and administrators (Preview)", and "Manifest" selected), and "Support + Troubleshooting" (with "Troubleshooting" and "New support request"). The right pane contains a JSON editor with the following code:

```
1 {
2     "id": "6336904f-f188-4e43-a53c-3a8b1e9ee4b4",
3     "acceptMappedClaims": null,
4     "accessTokenAcceptedVersion": null,
5     "addIns": [],
6     "allowPublicClient": null,
7     "appId": "ec78d011-e49e-4733-b69f-bd0763004190",
8     "appRoles": [],
9     "oauth2AllowUrlPathMatching": false,
10    "createdDateTime": "2019-12-16T06:20:46Z",
11    "groupMembershipClaims": null,
12    "identifierUris": [
13        "api://ec78d011-e49e-4733-b69f-bd0763004190"
14    ],
15    "informationalUrls": {
16        "termsOfService": null,
17        "support": null,
18        "privacy": null,
19        "marketing": null
20    },
21    "keyCredentials": [],
22    "knownClientApplications": [],
23    "logoUrl": null,
24    "logoutUrl": null,
25    "name": "Contoso App",
26    "oauth2AllowIdTokenImplicitFlow": false,
27    "oauth2AllowImplicitFlow": false,
28    "oauth2Permissions": [],
29    "oauth2RequirePostResponse": false,
30    "optionalClaims": null,
31    "orgRestrictions": [],
32    "parentalControlSettings": {
33        "countriesBlockedForMinors": [],
34        "legalAgeGroupRule": "Allow"
35    }
}
```

Overview of application model

Sometimes, the meaning of the term "application" can be misunderstood when used in the context of Microsoft identity platform. An application that has been integrated with Azure AD has implications that go beyond the software aspect. "Application" is frequently used as a conceptual term, referring to not only the application software, but also its Azure AD registration and role in authentication/authorization "conversations" at runtime.

By definition, an application can function in these roles:

- Client role (consuming a resource)
- Resource server role (exposing APIs to clients)
- Both client role and resource server role

When you register an app in the Azure portal, an **application object** and a **service principal object** get created automatically.

Application object

The application object:

- refers to the application properties stored in the manifest.
- defines the required application permissions.

The Application object is where developers specify how the app works. This includes parameters for authentication, secrets, or certificates, what permissions the app will use, any APIs the app exposes.

Azure AD applications are defined by exactly one application object. The application object resides in the Azure AD tenant where it was registered.

Service principal object

For multi-tenant applications, a copy of the application object is used in non-home tenants to define application permissions. This is referred to as the **service principle object**.

IT professionals and administrators use the service principal to determine how the application operates within their tenant. The IT Pro can limit the app to specific users and/or groups, review permissions and grant consent, assign users and/or groups to roles, or configure app provisioning.

For a multi-tenant application, when an application is given permission to access resources in a tenant, a service principal object is created in that tenant

Application and service principal relationship

Consider the application object as the global representation of your application for use across all tenants, and the service principal as the local representation for use in a specific tenant.

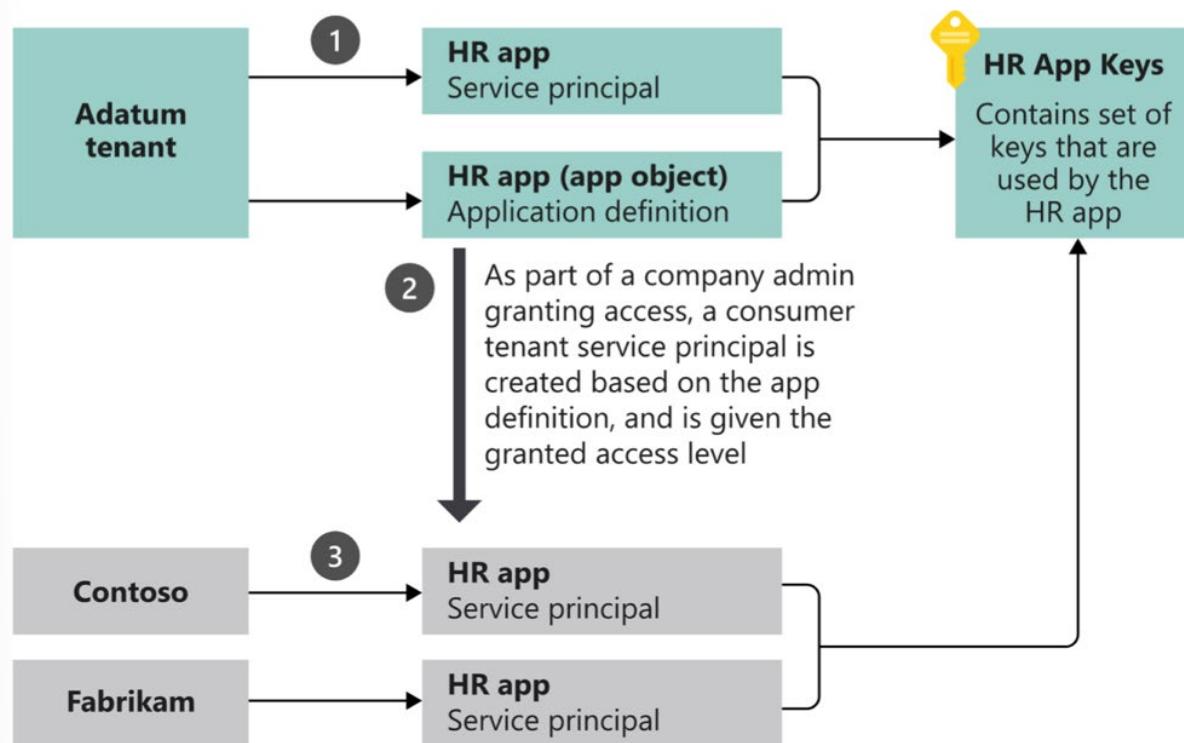
The application object serves as the template from which common and default properties are derived for use in creating corresponding service principal objects. An application object therefore has a 1:1 relationship with the software application, and a 1:many relationships with its corresponding service principal object(s).

A service principal must be created in each tenant where the application is used, enabling it to establish an identity for sign-in and/or access to resources being secured by the tenant. A single-tenant application has only one service principal (in its home tenant), created and consented for use during application registration. A multi-tenant Web application/API also has a service principal created in each tenant where a user from that tenant has consented to its use.

Example scenario

Consider a sample scenario with a multi-tenant application called **HR app**. There are three Azure AD tenants in this example scenario:

- **Adatum** - The tenant used by the company that developed the HR app
- **Contoso** - The tenant used by the Contoso organization, which is a consumer of the HR app
- **Fabrikam** - The tenant used by the Fabrikam organization, which also consumes the HR app



In this example scenario, there are 3 high-level steps related to the creation of the application object and each service principal:

Step	Description
1	Both the application object and a service principal are created in the application's home tenant (the Adatum tenant).
2	When Contoso and Fabrikam administrators complete consent, a service principal object is created in their company's Azure AD tenant and assigned the permissions that the administrator granted. Also note that the HR app could be configured/designed to allow consent by users for individual use.
3	The consumer tenants of the HR application (Contoso and Fabrikam) each have their own service principal object. Each represents their use of an instance of the application at runtime, governed by the permissions consented by the respective administrator.

Define app roles

Role-based access control (RBAC) is a popular mechanism to enforce authorization in applications. When using RBAC, an administrator grants permissions to roles, and not to individual users or groups. The administrator can then assign roles to different users and groups to control who has access to what content and functionality.

Using RBAC with Application Roles and Role Claims, developers can securely enforce authorization in their apps with little effort on their part.

Another approach is to use Azure AD Groups and Group Claims, as shown in **WebApp-Group-Claims-DotNet¹**. Azure AD Groups and Application Roles are by no means mutually exclusive; they can be used in tandem to provide even finer grained access control.

Roles ultimately get translated into permissions during authorization. But the first question is how to assign and manage roles. There are three options:

- Azure AD app roles
- Azure AD security groups
- Application role manager

Declare roles for an application

These application roles are defined in the Azure portal in the application's registration **manifest**. When a user signs into the application, Azure AD emits a `roles` claim for each role that the user has been granted individually to the user and from their group membership. Assignment of users and groups to roles can be done through the portal's UI, or programmatically using Microsoft Graph.

Declare app roles using Azure portal

1. From your application registration page in the Azure portal, select **Manifest**.
2. Edit the app manifest by locating the `appRoles` setting and adding all your Application Roles.

¹ <https://github.com/Azure-Samples/WebApp-GroupClaims-DotNet>

The screenshot shows the Azure portal interface for managing app registrations. The left sidebar has a 'Manifest' item highlighted with a red box. The main area displays a JSON manifest for a user group role. A red arrow points to the 'appRoles' field, which is currently empty: '[]'. The manifest includes fields like 'id', 'acceptMappedClaims', 'accessTokenAcceptedVersion', 'addIns', 'allowPublicClient', 'appId', 'oauth2AllowUrlPathMatching', 'createdDateTime', 'groupMembershipClaims', 'identifierUris', 'informationUrls', 'keyCredentials', 'knownClientApplications', 'logoutUrl', 'name', 'oauth2AllowIdTokenImplicitFlow', and 'oauth2AllowImplicitFlow'.

3. Save the manifest.

Note: Each app role definition in this manifest must have a differentalid GUID within the context of the manifest for the `id` property. The `value` property of each app role definition should exactly match the strings that are used in the code in the application. The `value` property can't contain spaces. If it does, you'll receive an error when you save the manifest.

Examples

For example, the following JSON defines two new roles: `ProductAdministrators` and `ProductViewers`.

```
"appRoles": [
  {
    "allowedMemberTypes": [ "User" ],
    "description": "Administrator role for Product Catalog web application.",
    "displayName": "ProductAdministrators",
    "id": "98ce9517-557f-4ac5-b827-f18d948ee552",
    "isEnabled": true,
    "lang": null,
    "origin": "Application",
    "value": "ProductAdministrators"
  },
  {
    "allowedMemberTypes": [ "User" ],
    "description": "Viewer role for Product Catalog web application",
    "displayName": "ProductViewers",
    "id": "a234b5c6-d7e8-4f9b-a0c9-1234567890ab",
    "isEnabled": true,
    "lang": null,
    "origin": "Application",
    "value": "ProductViewers"
  }
]
```

```
        "displayName": "ProductViewers",
        "id": "7465fed6-02cc-467c-87c2-fa6e0bf4f929",
        "isEnabled": true,
        "lang": null,
        "origin": "Application",
        "value": "ProductViewers"
    }
]
```

NOTE: The `displayName` cannot contain spaces and the `id` must be a unique GUID.

You can define app roles to target users, applications, or both. When available to applications, app roles appear as application permissions in the **Required Permissions** blade. The following example shows an app role targeted towards an Application.

```
"appId": "8763f1c4-f988-489c-a51e-158e9ef97d6a",
"appRoles": [
    {
        "allowedMemberTypes": [
            "Application"
        ],
        "displayName": "ConsumerApps",
        "id": "47fbb575-859a-4941-89c9-0f7a6c30beac",
        "isEnabled": true,
        "description": "Consumer apps have access to the consumer data.",
        "value": "Consumer"
    }
],
"availableToOtherTenants": false,
```

The number of roles defined affects the limits that the application manifest has.

Assign users and groups to roles

Once you've added app roles in your application, you can assign users and groups to these roles.

1. In the **Azure Active Directory** pane, select **Enterprise applications** from the **Azure Active Directory** left-hand navigation menu.
2. Select **All applications** to view a list of all your applications.
If you do not see the application you want show up here, use the various filters at the top of the **All applications** list to restrict the list or scroll down the list to locate your application.
3. Select the application in which you want to assign users or security group to roles.
4. Select the **Users and groups** pane in the application's left-hand navigation menu.
5. At the top of the **Users and groups** list, select the **Add user** button to open the **Add Assignment** pane.
6. Select the **Users and groups** selector from the **Add Assignment** pane.

A list of users and security groups will be shown along with a textbox to search and locate a certain user or group. This screen allows you to select multiple users and groups in one go.

7. Once you are done selecting the users and groups, press the **Select** button on bottom to move to the next part.
8. Choose the **Select Role** selector from the **Add assignment** pane. All the roles declared earlier in the app manifest will show up.
9. Choose a role and press the **Select** button.
10. Press the **Assign** button on the bottom to finish the assignments of users and groups to the app.
11. Confirm that the users and groups you added are showing up in the updated **Users and groups** list.

Get role claims

When a user signs in, the application receives the user's assigned role(s) in a claim with type **http://schemas.microsoft.com/ws/2008/06/identity/claims/role**.

A user can have multiple roles, or no role. In your authorization code, don't assume the user has exactly one role claim. Instead, write code that checks whether a particular claim value is present:

```
if (context.User.HasClaim(ClaimTypes.Role, "Admin")) { ... }
```

Advantages of this approach

- It uses a simple programming model.
- Roles are specific to the application. The role claims for one application are not sent to another application.
- If the customer removes the application from their AD tenant, the roles go away.
- The application doesn't need any extra Active Directory permissions, other than reading the user's profile.

Drawbacks

- Customers without Azure AD Premium cannot assign security groups to roles. For these customers, all user assignments must be done by an AD administrator.
- If you have a backend web API, which is separate from the web app, then role assignments for the web app don't apply to the web API.

Roles using Azure AD security groups

In this approach, roles are represented as AD security groups. The application assigns permissions to users based on their security group memberships.

Advantages

- For customers who do not have Azure AD Premium, this approach enables the customer to use security groups to manage role assignments.

Disadvantages

- Complexity. Because every tenant sends different group claims, the app must keep track of which security groups correspond to which application roles, for each tenant.
- If the customer removes the application from their AD tenant, the security groups are left in their AD directory.

Roles using an application role manager

With this approach, application roles are not stored in Azure AD at all. Instead, the application stores the role assignments for each user in its own DB — for example, using the **RoleManager** class in ASP.NET Identity.

Advantages

- The app has full control over the roles and user assignments.

Drawbacks

- More complex, harder to maintain.
- Cannot use AD security groups to manage role assignments.
- Stores user information in the application database, where it can get out of sync with the tenant's AD directory as users are added or removed.

Lesson review questions

1. What type of accounts does the Microsoft identity platform support?
2. Microsoft Identity supports multiple topology options. Which of the following options does not support using Microsoft Accounts for user sign-in?
 - (A) Consumer
 - (B) Azure AD Business to Business (B2B)
 - (C) Azure AD Business to Customer (B2C)
3. What is the key difference between an application and a service principal?

Correct/suggested answers:

1.
 - Applications can sign-in users with personal accounts, organizational accounts and/or social accounts depending on the application type.
 - Single-tenant applications can sign in users within the organization.
 - Multi-tenant applications can sign in users with a personal Microsoft account and/or a work or school account.
2. (B): Azure AD Business to Business (B2B). Azure AD B2B expects both organizations in a B2B relationship to have their own identity platform.

3. Application objects exist in the directory where the application is created, where service principal objects exist in each Azure AD tenant where the application is used.
 - Application objects are templates that exist in every directory where an application is used. They're used to create service principals that can be customized in each directory.
 - Service principals exist in the directory where the application is created, where application objects exist in each Azure AD tenant where the application is used.

Overview of authentication and authorization with Microsoft identity

Lesson introduction

The Microsoft identity platform (v. 2.0) endpoint supports authentication for a variety of modern app architectures, all of them based on industry-standard protocols OAuth 2.0 or OpenID Connect. Using the authentication libraries, applications authenticate identities and acquire tokens to access protected APIs.

The Microsoft identity platform implements the OAuth 2.0 authorization protocol. This protocol is a method that a third-party app can use to access web-hosted resources on behalf of a user. The web-hosted resources can define a set of permissions that you can use to implement functionality in smaller chunks. Developers can leverage one of two types of permissions supported by the Microsoft identity platform depending on the app scenario.

In this lesson, you'll learn about security tokens and in what scenarios applications will request and receive them for authentication and authorization using Microsoft identity platform. You'll also learn about the different types of permissions supported in Azure AD applications, Microsoft identity's consent framework, and how applications can obtain permissions from users and admins for to access protected APIs.

After this lesson, you should be able to:

- Describe the role of ID tokens and access tokens in authentication and authorization.
- Compare and contrast delegated permissions and application permissions.
- Describe how effective permissions are determined for delegated and application permissions.
- Determine admin consent requirements for an app's requested permissions.
- Configure delegated permissions.
- Configure application permissions.

Security tokens and application types

The Microsoft identity platform (v2.0) endpoint supports authentication for different kinds of modern application architectures. All of the architectures are based on the industry-standard protocols **OAuth 2.0** and **OpenID Connect**. By using the authentication libraries for the Microsoft identity platform, applications authenticate identities and acquire tokens to access protected APIs.

Security tokens

As a developer, when you build an application, you will ultimately request tokens from Microsoft identity to identify a user and authorize them to use an application with specific permissions. There are two different tokens that are commonly used in the Microsoft identity platform.

- ID tokens
- Access tokens

Microsoft identity platform implements security tokens as **JSON Web Tokens (JWTs)** that contain claims. JWTs are made up of three components:

- **Header** - Provides information about how to validate the token including information about the type of token and how it was signed.
- **Payload** - Contains all of the important data about the user or app that is attempting to call your service.
- **Signature** - Is the raw material used to validate the token.

ID tokens

The first type of token is an **ID token** that is made available via Microsoft identity's support of the OpenID Connect protocol. OpenID Connect extends the OAuth 2.0 authorization protocol to use as an authentication protocol. OpenID Connect enables you to implement single sign-on using OAuth in your applications.

An ID token is a security token that allows the client to verify the *identity* of the user. The ID token also gets basic profile information about the user. Because OpenID Connect extends OAuth 2.0, apps can securely acquire access tokens, which can be used to access resources that are secured by an authorization server.

The claims included within an ID token can be used for the user experience within your application, as keys in a database, and providing access to the client application.

Obtaining an ID token

When your web app needs to authenticate the user, it can direct the user to the Microsoft identity platform /authorize endpoint using an Open ID Connect library.

At this point, the user is prompted to enter their credentials, enter a multi-factor, use a FIDO2 compatible device, or perhaps use a passwordless experience using Microsoft Authenticator or Windows Hello to complete the authentication. All of these options are handled by the platform without any additional code from the developers. The Microsoft identity platform endpoint verifies that the user has consented to the permissions indicated in the scope parameter. If the user hasn't consented to any of those permissions, the Microsoft identity platform endpoint prompts the user to consent to the required permissions. (Permissions and consent are covered later in this lesson. Authentication flows are covered in detail in the next lesson.)

After the user authenticates and grants consent, the Microsoft identity platform endpoint returns a response to your app at the indicated redirect URI.

Access tokens

Many web apps need to not only sign the user in, but also to access a web service on behalf of the user by using OAuth. Applications request **access tokens** from Microsoft identity to use when calling a web service.

Access tokens enable clients to securely call APIs protected by Azure AD. Applications should never inspect or validate an access token. Access tokens should be treated as opaque strings and should only be passed to the API. The contents of the access token are intended for use by the protected resource.

When your client requests an access token, Microsoft identity platform also returns some metadata about the access token for your app's consumption. This information includes the expiry time of the access token and the scopes for which it's valid. This data allows your app to do intelligent caching of access tokens without having to parse the access token itself.

If your application is a resource (web API) that clients can request access to, access tokens provide helpful information for use in authentication and authorization, such as the user, client, issuer, permissions, and more.

Obtaining an access token

To acquire an access token, your app can make a request to the Microsoft identity platform token endpoint, like this:

```
POST common/oauth2/v2.0/token HTTP/1.1
Host: https://login.microsoftonline.com
Content-Type: application/json

{
    "grant_type": "authorization_code",
    "client_id": "6731de76-14a6-49ae-97bc-6eba6914391e",
    "scope": "https://outlook.office.com/mail.read https://outlook.office.com/mail.send",
    "code": "AwABAAAAvPM1KaPlrEqdFSBzjqfTGBcmLdgfSTLEMPGYuNHSUYBrq..."
    "redirect_uri": "https://localhost/myapp",
    "client_secret": "zc53fwe80980293klaj9823" // NOTE: Only required for web apps
}
```

You can use the resulting access token in HTTP requests to the resource. It reliably indicates to the resource that your app has the proper permission to perform a specific task.

Depending on how your client is built, it can use one (or several) of the authentication flows supported by Microsoft identity platform and Microsoft Authentication Library. These flows can produce a variety of tokens (ID tokens, refresh tokens, access tokens) as well as authorization codes, and require different tokens to make them work. These authentication flows vary based on application type and will be covered in the next lesson.

Application categories

Tokens can be acquired by several types of applications, including:

- Web apps
- Mobile apps
- Desktop apps
- Web APIs

Tokens can also be acquired by apps running on devices that don't have a browser or are running on the Internet of Things (IoT).

Applications can be categorized as follows:

- **Protected resources vs. client applications:** In some scenarios you will use Microsoft identity to protect resources like web apps or web APIs. Such APIs must validate the access tokens used to access the resource. Alternatively, client applications need to access tokens to call protected web APIs.
- **With users or without users:** Some scenarios involve a signed-in user while others, like daemon applications, don't involve a user.

- **Single-page, public client, and confidential client applications:** These are three large categories of application types. Each is used with different libraries and objects.
 - **Single-page applications** (SPAs) are web apps in which tokens are acquired by a JavaScript or TypeScript app running in the browser. MSAL.js is the only Microsoft authentication library that supports single-page applications.
 - **Public client applications:** Apps in this category, like the following types, always sign in users:
 - Desktop apps that call web APIs on behalf of signed-in users
 - Mobile apps
 - Apps running on devices that don't have a browser, like those running on IoT
 - **Confidential client applications:** Apps in this category include:
 - Web apps that call a web API
 - Web APIs that call a web API
 - Daemon apps, even when implemented as a console service like a Linux daemon or a Windows service
- **Sign-in audience:** The available authentication flows differ depending on the sign-in audience. Some flows are available only for work or school accounts. And some are available both for work or school accounts and for personal Microsoft accounts. The allowed audience depends on the authentication flows.

Overview of authorization model

Applications that integrate with Microsoft identity platform follow an authorization model that gives users and administrators control over how data can be accessed. The implementation of the authorization model has been updated on the Microsoft identity platform endpoint, and it changes how an app must interact with the Microsoft identity platform. This section covers the basic concepts of this authorization model, including scopes, permissions, and consent.

Scopes/permissions

The Microsoft identity platform implements the OAuth 2.0 authorization protocol. OAuth 2.0 is a method through which a third-party app can access web-hosted resources on behalf of a user. Any web-hosted resource that integrates with the Microsoft identity platform has a resource identifier, or *Application ID URI*. For example, some of Microsoft's web-hosted resources include:

- Microsoft Graph: <https://graph.microsoft.com>
- Office 365 Mail API: <https://outlook.office.com>
- Azure Active Directory Graph: <https://graph.windows.net>
- Azure Key Vault: <https://vault.azure.net>

Note: Microsoft Graph is the recommended API to use for surfacing Active Directory objects.

The same is true for any third-party resources that have integrated with the Microsoft identity platform. Any of these resources can also define a set of permissions that can be used to divide the functionality of

that resource into smaller chunks. As an example, Microsoft Graph has defined permissions to do the following tasks, among others:

- Read a user's calendar
- Write to a user's calendar
- Send mail as a user

By defining these types of permissions, the resource has fine-grained control over its data and how API functionality is exposed. A third-party app can request these permissions from users and administrators, who must approve the request before the app can access data or act on a user's behalf. By chunking the resource's functionality into smaller permission sets, third-party apps can be built to request only the specific permissions that they need to perform their function. Users and administrators can know exactly what data the app has access to, and they can be more confident that it isn't behaving with malicious intent.

Note: Developers should always abide by the concept of least privilege, asking for only the permissions they need for their applications to function.

In OAuth 2.0, these types of permissions are called **scopes**. They are also often referred to as **permissions**. A permission is represented in the Microsoft identity platform as a string value. Continuing with the Microsoft Graph example, the string value for each permission is:

- Read a user's calendar by using Calendars.Read
- Write to a user's calendar by using Calendars.ReadWrite
- Send mail as a user using by Mail.Send

An app most commonly requests these permissions by specifying the scopes in requests to the Microsoft identity platform authorize endpoint. However, certain high-privilege permissions can only be granted through administrator consent and requested/granted using the administrator consent endpoint.

The Microsoft identity platform implementation of OpenID Connect has a few well-defined scopes that are also hosted on the Microsoft Graph: openid, email, profile, and offline_access. The address and phone OpenID Connect scopes are not supported.

Note: At this time, the offline_access ("Maintain access to data you have given it access to") and user.read ("Sign you in and read your profile") permissions are automatically included in the initial consent to an application. These permissions are generally required for proper app functionality - offline_access gives the app access to refresh tokens, critical for native and web apps, while user.read gives access to the sub claim, allowing the client or app to correctly identify the user over time and access rudimentary user information.

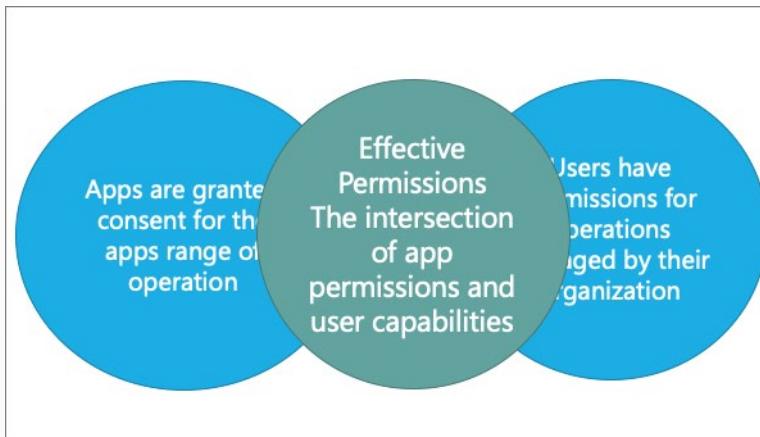
Permission types

Microsoft identity platform supports two types of permissions: **delegated permissions** and **application permissions**.

- **Delegated permissions** are used by apps that have a signed-in user present. For these apps, either the user or an administrator consents to the permissions that the app requests, and the app is delegated permission to act as the signed-in user when making calls to the target resource. Some delegated permissions can be consented to by non-administrative users, but some higher-privileged permissions require administrator consent.
- **Application permissions** are used by apps that run without a signed-in user present; for example, apps that run as background services or daemons. Application permissions can only be consented by an administrator.

Effective permissions

Effective permissions are the permissions that your app will have when making requests to the target resource. It's important to understand the difference between the delegated and application permissions that your app is granted and its effective permissions when making calls to the target resource.



- For delegated permissions, the *effective permissions* of your app will be the least privileged intersection of the delegated permissions the app has been granted (via consent) and the privileges of the currently signed-in user. Your app can never have more privileges than the signed-in user. Within organizations, the privileges of the signed-in user may be determined by policy or by membership in one or more administrator roles.
 - For example, assume your app has been granted the *User.ReadWrite.All* delegated permission. This permission nominally grants your app permission to read and update the profile of every user in an organization. If the signed-in user is a global administrator, your app will be able to update the profile of every user in the organization. However, if the signed-in user isn't in an administrator role, your app will be able to update only the profile of the signed-in user. It will not be able to update the profiles of other users in the organization because the user that it has permission to act on behalf of does not have those privileges.
- For application permissions, the *effective permissions* of your app will be the full level of privileges implied by the permission. For example, an app that has the *User.ReadWrite.All* application permission can update the profile of every user in the organization.

Admin-restricted permissions

Some high-privilege permissions in the Microsoft ecosystem can be set to *admin-restricted*. Examples of these kinds of permissions include the following:

- Read all user's full profiles by using **User.Read.All**
- Write data to an organization's directory by using **Directory.ReadWrite.All**
- Read all groups in an organization's directory by using **Groups.Read.All**

Although a consumer user might grant an application access to this kind of data, organizational users are restricted from granting access to the same set of sensitive company data. If your application requests access to one of these permissions from an organizational user, the user receives an error message that says they're not authorized to consent to your app's permissions.

If your app requires access to admin-restricted scopes for organizations, you should request them directly from a company administrator, and also by using the admin consent endpoint, described in the next topic.

If the application is requesting high-privilege delegated permissions and an administrator grants these permissions via the admin consent endpoint, consent is granted for all users in the tenant.

Alternatively, if the application is requesting application permissions and an administrator grants these permissions via the admin consent endpoint, this grant isn't done on behalf of any specific user. Instead, the client application is granted permissions *directly*. These types of permissions are only used by daemon services and other non-interactive applications that run in the background.

Consent experiences

Consent is the process of a user granting authorization to an application to access protected resources on their behalf. An admin or user can be asked for consent to allow access to their organization/individual data.

The actual user experience of granting consent will differ depending on policies set on the user's tenant, the user's scope of authority (or role), and the type of permissions being requested by the client application. This means that application developers and tenant admins have some control over the consent experience.

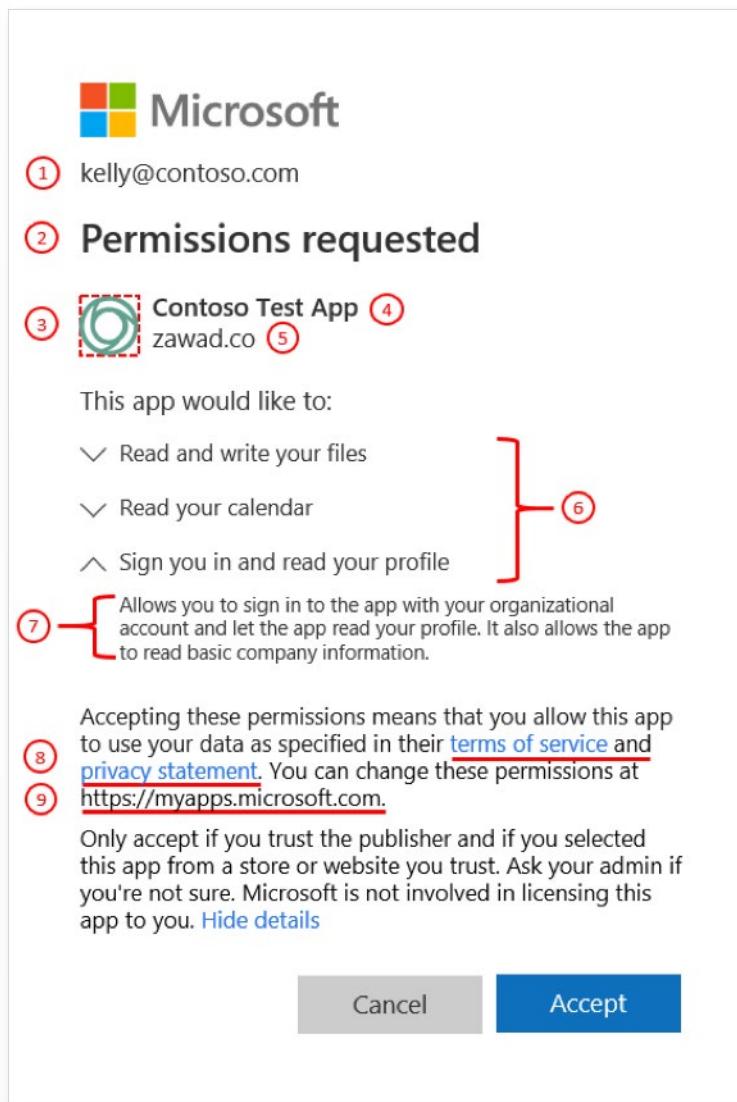
Admins have the flexibility of setting and disabling policies on a tenant or app to control the consent experience in their tenant. Application developers can dictate what types of permissions are being requested and whether they want to guide users through the user consent flow or the admin consent flow.

- **User consent flow** is when an application developer directs users to the authorization endpoint with the intent to record consent for only the current user. When the user approves the permission request, consent is recorded, and the user doesn't have to consent again on subsequent sign-ins to the application.
- **Admin consent flow** is when an application developer directs users to the admin consent endpoint with the intent to record consent for the entire tenant. To ensure the admin consent flow works properly, application developers must list all permissions in the `RequiredResourceAccess` property in the application manifest.

Building blocks of the consent prompt

The consent prompt is designed to ensure that users have enough information to determine whether they trust the client application to access protected resources on their behalf. Understanding the building blocks will help users granting consent make more informed decisions and it will help developers build better user experiences.

The following diagram and table provide information about the building blocks of the consent prompt.



#	Component	Purpose
1	User identifier	This identifier represents the user that the client application is requesting to access protected resources on behalf of.
2	Title	The title changes based on whether the users are going through the user or admin consent flow. In user consent flow, the title will be "Permissions requested," while in the admin consent flow the title will have an additional line "Accept for your organization."

#	Component	Purpose
3	App logo	This image should help users have a visual cue of whether this app is the app they intended to access. This image is provided by application developers, and the ownership of this image isn't validated.
4	App name	This value should inform users which application is requesting access to their data. Note that this name is provided by the developers and the ownership of this app name isn't validated.
5	Publisher domain	This value should provide users with a domain they may be able to evaluate for trustworthiness. This domain is provided by the developers, and the ownership of this publisher domain is validated.
6	Permissions	This list contains the permissions being requested by the client application. Users should always evaluate the types of permissions being requested to understand what data the client application will be authorized to access on their behalf if they accept. As an application developer, it is best to request access to the permissions with the least privilege.
7	Permission description	This value is provided by the service exposing the permissions. To see the permission descriptions, you must toggle the chevron next to the permission.

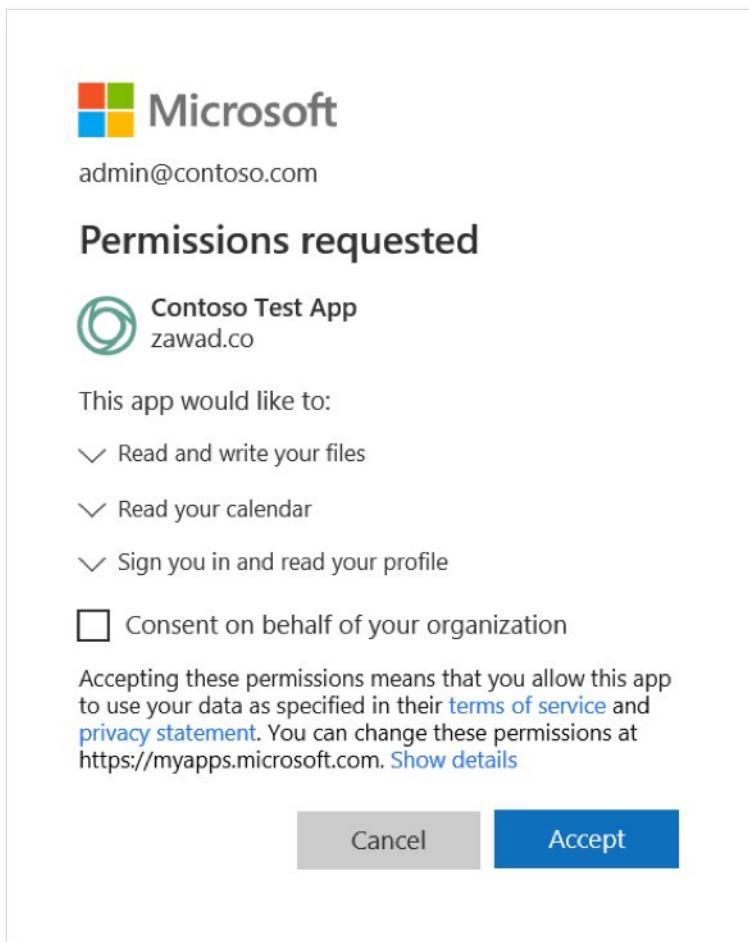
#	Component	Purpose
8	App terms	These terms contain links to the terms of service and privacy statement of the application. The publisher is responsible for outlining their rules in their terms of service. Additionally, the publisher is responsible for disclosing the way they use and share user data in their privacy statement. If the publisher doesn't provide links to these values for multi-tenant applications, there will be a bold warning on the consent prompt.
9	https://myapps.microsoft.com	This is the link where users can review and remove any non-Microsoft applications that currently have access to their data.

Common consent scenarios

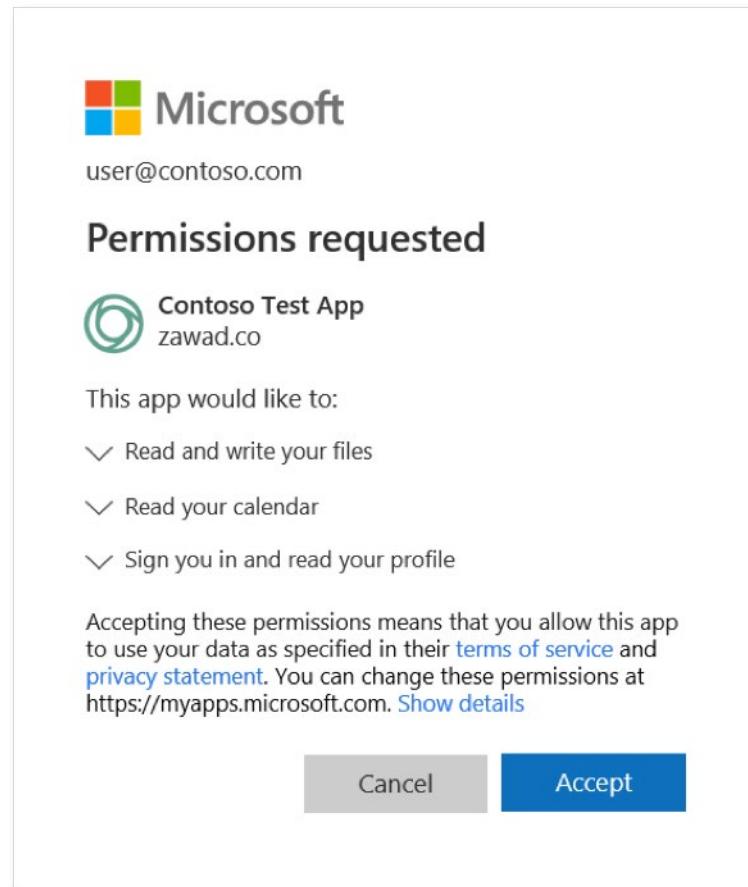
Here are the consent experiences that a user may see in the common consent scenarios:

1. Individuals accessing an app that directs them to the user consent flow while requiring a permission set that is within their scope of authority.
 - Admins will see an additional control on the traditional consent prompt that will allow them consent on behalf of the entire tenant. The control will be defaulted to off, so only when admins explicitly check the box will consent be granted on behalf of the entire tenant. As of today, this check box will only show for the Global Admin role, so Cloud Admin and App Admin will not see this checkbox.

Only static permissions, defined in the Azure AD admin center, are displayed in the admin consent experience.



- Users will see the traditional consent prompt. It lists all permissions that are being requested. This includes both the statically defined permissions as well as dynamically requested.

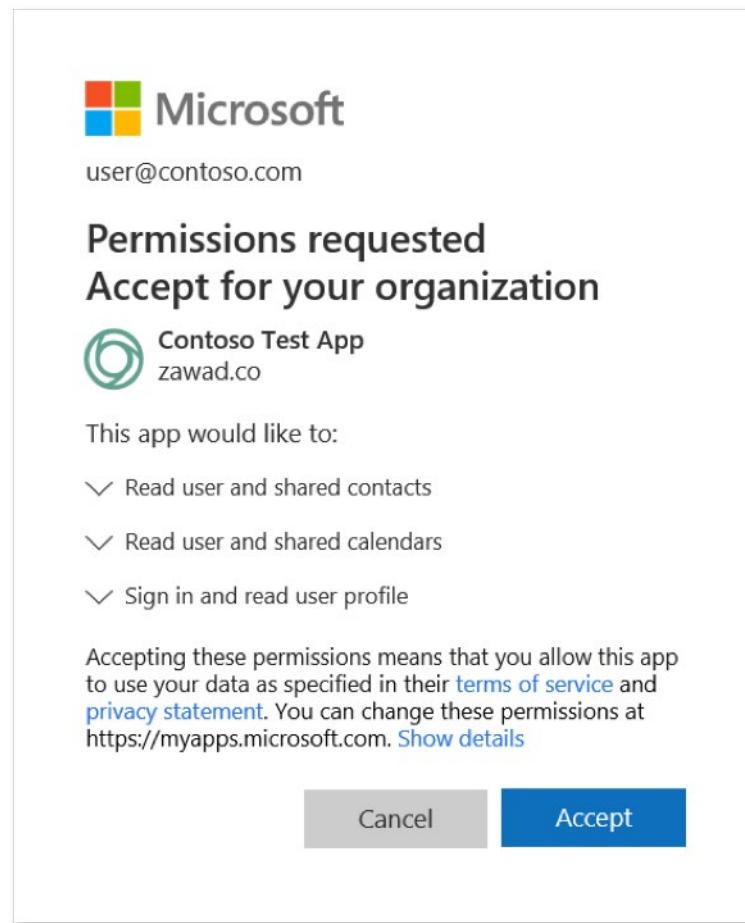


2. Individuals accessing an app that requires at least one permission that is outside their scope of authority.
 - Admins will see the same prompt as shown in the first scenario.
 - Users will be blocked from granting consent to the application, and they will be told to ask their admin for access to the app.

The screenshot shows a Microsoft admin consent dialog box. At the top left is the Microsoft logo and the email address user@contoso.com. Below that is the heading "Need admin approval". Underneath is a circular icon for "Contoso Test App" followed by the text "Contoso Test App" and the URL "zawad.co". A descriptive message states: "Contoso Test App needs permission to access resources in your organization that only an admin can grant. Please ask an admin to grant permission to this app before you can use it." At the bottom are two blue links: "Have an admin account? Sign in with that account" and "Return to the application without granting consent".

3. Individuals who navigate or are directed to the admin consent flow.

- Admin users will see the admin consent prompt. The title and the permission descriptions changed on this prompt; the changes highlight the fact that accepting this prompt will grant the app access to the requested data on behalf of the entire tenant.



- Non-admin users will see the same screen as in the second scenario.

Admin consent options

Some permissions require consent from an administrator before they can be granted within a tenant. Admins can grant consent within the app registration UI or using the admin consent endpoint.

Grant admin consent in the app registration UI

The app registration UI displays the permissions and admin consent granted to your app. It has the following sections.

Configured permissions

This section shows the permissions that have been explicitly configured on the application object (the permissions that are part of the app's required resource access list). You may add or remove permissions from this table. As an admin, you can also grant/revoke admin consent for all or only part of the selected permissions.

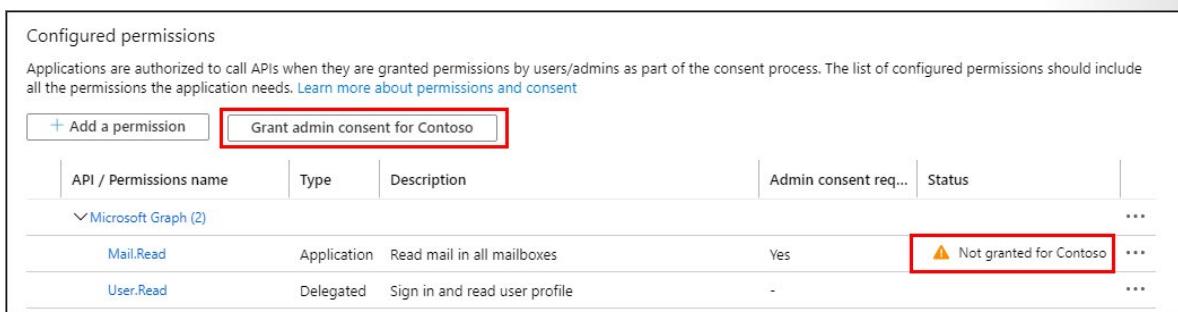
Other permissions granted

If your application is registered in a tenant, you may see an additional section titled “Other permissions granted for Tenant.” This section shows permissions that have been granted for the tenant but have not been explicitly configured on the application object (for example, permissions that were dynamically requested and consented). This section only appears if there is at least one permission that applies.

You may add a set of an API’s permissions or individual permissions that appear in this section to the **Configured permissions** section. As an admin, you can also revoke admin consent for individual APIs or permissions in this section.

Admin consent button

If your application is registered in a tenant, you will see a **Grant admin consent for Tenant** button. It will be disabled if you are not an admin, or if no permissions have been configured for the application. This button allows an admin to easily grant admin consent to the permissions configured for the application. Selecting the admin consent button launches a new window with a consent prompt showing all the configured permissions.



The screenshot shows the 'Configured permissions' section in the Azure portal. At the top, there is a note: 'Applications are authorized to call APIs when they are granted permissions by users/admins as part of the consent process. The list of configured permissions should include all the permissions the application needs.' Below this, there is a 'Grant admin consent for Contoso' button, which is highlighted with a red box. The table below lists permissions:

API / Permissions name	Type	Description	Admin consent req...	Status
Mail.Read	Application	Read mail in all mailboxes	Yes	⚠ Not granted for Contoso
User.Read	Delegated	Sign in and read user profile	-	...

If you have permissions that have been granted but not configured, when selecting the admin consent button, you will be prompted to decide how to handle these permissions. You may add them to configured permissions or you may remove them.

The consent prompt provides the option to **Accept** or **Cancel**. If you select **Accept**, admin consent is granted. If you select **Cancel**, admin consent is not granted, and you will see an error stating that consent has been declined.

The admin consent endpoint

When you’re ready to request permissions from your organization’s admin, you can also redirect the user to the Microsoft identity platform *admin consent endpoint*:

```
// Line breaks are for legibility only.  
GET https://login.microsoftonline.com/{tenant}/v2.0/adminconsent?  
client_id=6731de76-14a6-49ae-97bc-6eba6914391e  
&state=12345  
&redirect_uri=http://localhost/myapp/permissions  
&scope=  
https://graph.microsoft.com/calendars.read  
https://graph.microsoft.com/mail.send
```

The following table outlines the required and recommended parameters when using the admin consent endpoint.

Parameter	Condition	Description
tenant	Required	The directory tenant that you want to request permission from. Can be provided in GUID or friendly name format OR generically referenced with organizations, as seen in the example. Do not use 'common', because personal accounts cannot provide admin consent except in the context of a tenant. To ensure best compatibility with personal accounts that manage tenants, use the tenant ID when possible.
client_id	Required	The Application (client) ID that the Azure portal – App registrations experience assigned to your app.
redirect_uri	Required	The redirect URI where you want the response to be sent for your app to handle. It must exactly match one of the redirect URIs that you registered in the app registration portal.
state	Recommended	A value included in the request that will also be returned in the token response. It can be a string of any content you want. Use the state to encode information about the user's state in the app before the authentication request occurred, such as the page or view they were on.
scope	Required	Defines the set of permissions being requested by the application. This can be either static (using /.default) or dynamic scopes. This can include the OIDC scopes (openid, profile, email).

At this point, Azure AD requires a tenant administrator to sign in to complete the request. The administrator is asked to approve all the permissions that you have requested in the scope parameter. If you've used a static (/.default) value, it will function like the v1.0 admin consent endpoint and request consent for all scopes found in the required permissions for the app.

Recommended: Sign the user in to your app

Typically, when you build an application that uses the admin consent endpoint, the app needs a page or view in which the admin can approve the app's permissions. This page can be part of the app's sign-up flow, part of the app's settings, or it can be a dedicated "connect" flow. In many cases, it makes sense for the app to show this "connect" view only after a user has signed in with a work or school Microsoft account.

When you sign the user in to your app, you can identify the organization to which the admin belongs before asking them to approve the necessary permissions. Although not strictly necessary, it can help you create a more intuitive experience for your organizational users.

Configure permissions to consume an API

There are two ways to define the permissions an app needs that drives what permissions are listed in the user consent dialog. These two approaches are called *static* and *dynamic* consent. This topic describes how to configure delegated permissions statically and dynamically.

Static permissions

With **static consent**, the permissions requested by the app are defined in the app's registration within the Azure AD admin center.

The static consent approach is how the Microsoft identity v1.0 authorize endpoint worked. All permissions an application used needed to be defined ahead of time. When users signed in, they would have to consent to all permission requests.

This approach also enables administrators to consent on behalf of all users in the organization.

In general, it's best practice to ensure that the permissions statically defined for a given application are a superset of the permissions that it will be requesting dynamically/incrementally.

Configure scopes for static permission

Static permissions are configured from the app registration in the Azure portal. To configure permissions for an app, from the app's registration:

1. Select **API Permissions** from the left-hand navigation, and then select **Add a permission**.

The screenshot shows the Azure portal interface for managing API permissions. The left sidebar lists several sections: Overview, Quickstart, Integration assistant (preview), Manage, Branding, Authentication, Certificates & secrets, Token configuration, API permissions (which is highlighted with a red box), and Expose an API. The main content area is titled 'Identity Exercise 02 | API permissions'. It features a search bar, a refresh button, and a 'Configured permissions' section. This section includes a note about granting permissions by users/admins and a link to learn more about permissions and consent. Below this is a button labeled '+ Add a permission' (also highlighted with a red box) and a 'Grant admin consent for Contoso' button. A table lists the configured permissions, showing one entry for Microsoft Graph (1) with the permission 'User.Read' (Type: Delegated, Description: Sign in and read user profile). The table includes columns for API / Permissions name, Type, Description, Admin consent req..., and Status.

API / Permissions name	Type	Description	Admin consent req...	Status
User.Read	Delegated	Sign in and read user profile	-	...

2. On the **Request API Permissions** page, select the resource and permissions your app is requesting. This can include permissions from Microsoft APIs, APIs your organization uses, or your APIs. For example, to request permission to read users' mail using Microsoft Graph:
- Select **Microsoft APIs, Microsoft Graph**, and then select **Delegated permissions** to access the API on behalf of the signed-in user or **Application permissions** if the application will run without a signed-in user.
 - In the search box in the **Select permissions** section, enter Mail.R, select the permission Mail.Read permission, and then select **Add permissions**.

The screenshot shows the 'Request API permissions' page for Microsoft Graph. It highlights the 'Delegated permissions' section and the 'Mail.Read' permission under the 'Mail' section. The 'Add permissions' button at the bottom is also highlighted.

The added permission will now be displayed under **Configured permissions** in the **API permissions** section of the app registration.

The screenshot shows the 'Identity Exercise 02 | API permissions' page in the Azure portal. It displays the 'Configured permissions' table with the following data:

API / Permissions name	Type	Description	Admin consent req...	Status
Mail.Read	Delegated	Read user mail	-	...
User.Read	Delegated	Sign in and read user profile	-	...

Dynamic consent

Permissions can also be requested *incrementally* as needed through **dynamic consent**. This was introduced in the Microsoft identity v2.0 endpoint. With this option, permissions don't need to be specified in the app registration in the Azure AD admin center.

Each time the app requests an access token, it specifies the permissions it needs in the `scope` property. If the user hasn't already consented to those permissions, they'll go through the consent experience for the additional dynamically requested permissions.

Dynamic consent enables developers to only ask users for the permissions the app needs at that time. For example, at sign-in the app can request minimal profile information, but later request additional permissions to do more advanced tasks.

The `/.default` scope

For developers migrating apps from the Microsoft identity v1.0 endpoint to the v2.0 endpoint, Microsoft introduced the `/.default` scope. This is a built-in scope for every application that refers to the static list of permissions configured on the application registration.

A scope value of `[app_id]/.default` requests an access token with the static scopes that the application has defined in the Azure portal.

Note: Application permissions can only be requested through the use of `/.default` - so if your app needs application permissions, make sure they're listed in the app registration portal.

Note: Clients can't combine static (`/.default`) and dynamic consent in a single request. Thus, `scope=https://graph.microsoft.com/.default+mail.read` will result in an error due to the combination of scope types.

`/.default` and consent

The `/.default` scope triggers the v1.0 endpoint behavior for `prompt=consent` as well. It requests consent for all permissions registered by the application, regardless of the resource. If included as part of the request, the `/.default` scope returns a token that contains the scopes for the resource requested.

`/.default` when the user has already given consent

Because `/.default` is functionally identical to the resource-centric v1.0 endpoint's behavior, it brings with it the consent behavior of the v1.0 endpoint as well. Namely, `/.default` only triggers a consent prompt if no permission has been granted between the client and the resource by the user. If any such consent exists, then a token will be returned containing all scopes granted by the user for that resource. However, if no permission has been granted, or the `prompt=consent` parameter has been provided, a consent prompt will be shown for all scopes registered by the client application.

Example 1: The user, or tenant admin, has granted permissions

The user (or a tenant administrator) has granted the client the Microsoft Graph permissions `mail.read` and `user.read`. If the client makes a request for `scope=https://graph.microsoft.com/.default`, then no consent prompt will be shown, regardless of the contents of the client application's registered permissions for Microsoft Graph. A token would be returned containing the scopes `mail.read` and `user.read`.

Example 2: The user hasn't granted permissions between the client and the resource

No consent for the user exists between the client and Microsoft Graph. The client has registered for the **user.read** and **contacts.read** permissions, as well as the Azure Key Vault scope **https://vault.azure.net/user_impersonation**. When the client requests a token for **scope=https://graph.microsoft.com/.default**, the user will see a consent screen for the **user.read**, **contacts.read**, and the Key Vault **user_impersonation** scopes. The token returned will have just the **user.read** and **contacts.read** scopes in it.

Example 3: The user has consented and the client requests additional scopes

The user has already consented to **mail.read** for the client. The client has registered for the **contacts.read** scope in its registration. When the client makes a request for a token using **scope=https://graph.microsoft.com/.default** and requests consent through **prompt=consent**, the user will see a consent screen for only and all the permissions registered by the application. **contacts.read** will be present in the consent screen, but **mail.read** will not. The token returned will be for Microsoft Graph and will contain **mail.read** and **contacts.read**.

Using permissions

After the user consents to permissions for your app, your app can acquire access tokens that represent your app's permission to access a resource in some capacity. An access token can be used only for a single resource, but encoded inside the access token is every permission that your app has been granted for that resource.

Lesson review questions

1. What is the definition of incremental consent?
2. What is the advantage of setting permissions on a granular level?
3. Are application permissions granted to an application during registration?

Correct/suggested answers:

1. Incremental consent means that the user can be queried after sign-in to give consent to access additional resources such as a calendar.
2. By chunking functionality into smaller permission sets, you can limit access, which improves overall security of the data.
3. No. The Azure AD tenant administrator MUST explicitly grant the permissions to the application. This must be done per tenant and must be performed every time the application permissions are changed in the application registration portal.

Implement authentication and authorization to consume an API

Lesson introduction

The Microsoft Identity platform enables developers to build many different types of applications to satisfy diverse business requirements and different scenarios. By supporting multiple OAuth 2.0 standard authentication protocols, developers can create different types of applications that meet business needs including single page applications, web apps, mobile or native apps, and services or daemon apps.

In this lesson, you'll learn how you can implement different OAuth 2.0 protocol grant types (flows) in popular application types.

After this lesson, you should be able to:

- Describe the considerations for app registration based on app type and auth flow.
- Configure Microsoft Authentication Library (MSAL JS) for endpoint and token cache.
- Implement authentication using the MSAL.js login method.
- Configure incremental consent scopes in an app that consumes an API.
- Call MSAL.js using the AquireTokenSilent/AquireToken pattern to consume an API.
- Configure client credentials using a certificate for a daemon app.
- Acquire an access token for Microsoft Graph using an application permission and client credential certificate.
- Acquire an access token using a client secret.

Overview of authentication flows

Microsoft Authentication Library

While Microsoft identity is built on open standards like OAuth2 and Open ID Connect, fully implementing these protocols is time consuming and complicated. It is also required that any implementation of these standards be kept up-to-date with the latest recommendations to ensure the implementation is as secure as possible. Therefore, it is important for developers, whenever possible, to use a well-maintained Open ID Connect library. Microsoft specifically recommends the use of **Microsoft Authentication Library (MSAL)**.

Microsoft Authentication Library enables developers to acquire tokens from the Microsoft identity platform endpoint in order to access secured Web APIs. These Web APIs can be the Microsoft Graph, other Microsoft APIS, third-party Web APIs, or your own Web API. MSAL is available for .NET, JavaScript, Android, and iOS, which support many different application architectures and platforms.

MSAL gives you many ways to get tokens, with a consistent API for a number of platforms. Using MSAL provides the following benefits:

- No need to directly use the OAuth libraries or code against the protocol in your application.
- Acquires tokens on behalf of a user or on behalf of an application (when applicable to the platform).
- Maintains a token cache and refreshes tokens for you when they are close to expiring. You don't need to handle token expiration on your own.

- Helps you specify which audience you want your application to sign in (your org, several orgs, work, and school and Microsoft personal accounts, social identities with Microsoft identity platform B2C, users in sovereign, and national clouds).
- Helps you set up your application from configuration files.
- Helps you troubleshoot your app by exposing actionable exceptions, logging, and telemetry.

Summary of authentication flows

The following table summarizes the different authentication flows provided by Microsoft Authentication Library and which application scenarios they can be used in.

Flow	Description	Used in
Interactive (https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-authentication-flows)	Gets the token through an interactive process that prompts the user for credentials through a browser or pop-up window.	Desktop apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-overview), mobile apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-mobile-overview)
Implicit grant (https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-authentication-flows)	Allows the app to get tokens without performing a back-end server credential exchange. This allows the app to sign in the user, maintain session, and get tokens to other web APIs, all within the client JavaScript code.	Single-page applications (SPA) (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-spa-overview)
Authorization code (https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-authentication-flows)	Used in apps that are installed on a device to gain access to protected resources, such as web APIs. This allows you to add sign-in and API access to your mobile and desktop apps.	Desktop apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-overview), mobile apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-mobile-overview), web apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-web-app-call-api-overview)
On-behalf-of (https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-authentication-flows)	An application invokes a service or web API, which in turn needs to call another service or web API. The idea is to propagate the delegated user identity and permissions through the request chain.	Web APIs (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-web-api-call-api-overview)

Flow	Description	Used in
Client credentials (https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-authentication-flows)	Allows you to access web-hosted resources by using the identity of an application. Commonly used for server-to-server interactions that must run in the background, without immediate interaction with a user.	Daemon apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-daemon-overview)
Device code (https://docs.microsoft.com/en-us/azure/active-directory/develop/msal-authentication-flows)	Allows users to sign in to input-constrained devices such as a smart TV, IoT device, or printer.	Desktop/mobile apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-acquire-token)
Integrated Windows Authentication (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-acquire-token)	Allows applications on domain or Azure Active Directory (Azure AD) joined computers to acquire a token silently (without any UI interaction from the user).	Desktop/mobile apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-acquire-token)
Username/password (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-acquire-token)	Allows an application to sign in the user by directly handling their password. This flow isn't recommended.	Desktop/mobile apps (https://docs.microsoft.com/en-us/azure/active-directory/develop/scenario-desktop-acquire-token)

Single-page applications

Web standards and modern browsers have advanced considerably in recent years to the point that developers can create sophisticated client-side applications. Many modern web applications are built as client-side single-page apps (SPAs) written using JavaScript or a SPA framework such as Angular, Vue, or React. These applications run in a web browser and have different authentication characteristics than traditional server-side web applications.

SPAs and other JavaScript apps that run primarily in a browser have unique challenges when it comes to authentication:

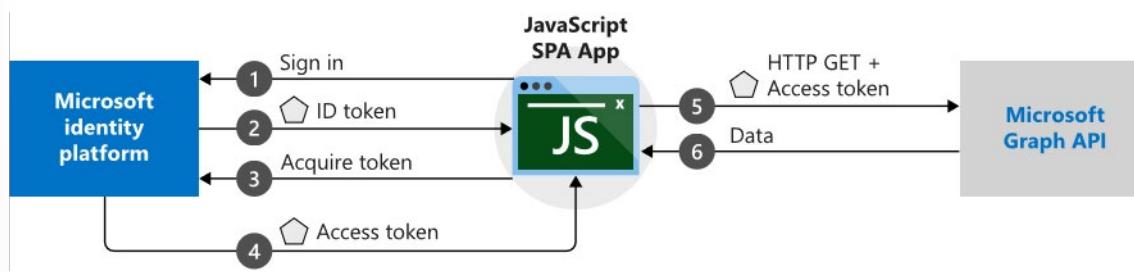
- Security characteristics of these apps are different from traditional server-based web applications
- Many authorization servers and identity providers don't support CORS requests
- Full-page browser redirects away from the app become invasive to the user experience

The Microsoft identity platform enables SPAs to sign in users and get tokens to access backend services or web APIs using the **OAuth 2.0 implicit grant flow**.

Note: The implicit grant flow is not supported in MSAL.js 2.0+. Current OAuth 2.0 best practices recommend using the authorization code flow rather than the implicit flow for SPAs. Having limited-lifetime refresh tokens also helps your application adapt to modern browser cookie privacy limitations, like Safari ITP. Applications that use MSAL.js 1.3 or earlier do not support the auth code flow. The difference in application registration steps for each flow is highlighted in the Azure AD App registration section below.

The implicit grant flow

This topic will provide an overview of how to implement authentication and authorization to consume an API (Microsoft Graph) in a single-page application using the OAuth 2.0 implicit grant flow.



The implicit grant flow works as follows for a single-page application:

1. The single-page app redirects users to the Microsoft identity platform authorize endpoint for sign-in. The user enters credentials and consents to permissions.
2. The Microsoft identity platform returns an ID token upon successful authentication.
3. The SPA requests an access token in order to access the Microsoft Graph API.
4. Microsoft identity returns an access token.
5. The SPA sends an HTTP request to Microsoft Graph, along with the acquired access token in the Authorization header.
6. Microsoft Graph returns secure data to the SPA.

Implementing this flow involves the following high-level steps:

1. Register the Azure AD application
2. MSAL.js code configuration
3. Implement sign-in
4. Acquire access token
5. Call API

Step 1: Azure AD app registration

In order for a SPA to use Microsoft identity to enable users to authenticate and obtain access tokens for use with services such as Microsoft Graph, you must register a new app with Azure AD. You can do this using the **Azure AD admin center**² (following the app registration steps introduced in Lesson 01).

By default, Azure AD app registrations enable the authorization code flow. To take advantage of this flow, your application must use MSAL.js 2.0 or later. For apps using MSAL.js 1.0, there are a few things you need to do to enable the implicit grant flow for your SPA after registering it with Azure AD.

1. In the Azure portal, select the app registration for your SPA. (Note, you may leave the **Redirect URI** field blank during initial app registration before completing the following steps)
2. Under **Manage**, select **Authentication**, and then select **Add a platform**.
3. Under **Web applications**, select **Single-page application** tile.
4. Under **Redirect URIs**, enter a redirect URI. Ensure the redirect URI points to the URL of the SPA. This URL must match the redirect URL provided by the SPA when the authentication process is initiated.

² <https://aad.portal.azure.com>

5. Next, you will need to configure your registration to support the implicit grant flow so that the authorization endpoint can issue ID and access tokens to your appEnable the **Implicit flow**:

- Locate the **Implicit grant** section and select the **ID token** option. If the app is also going to request an access token to use in authenticating requests to other Microsoft identity-protected endpoints, you must also select the **Access tokens** option.
- **Note:** for apps using the auth code flow, you would not select either of these checkboxes.

6. Select **Configure** to finish adding the redirect URI.

You've now completed the registration of your single-page application (SPA) and configured a redirect URI to which the client will be redirected and any security tokens will be sent. By selecting one or both of **ID tokens** and **Access tokens**, you've enabled the implicit grant flow.

Step 2: MSAL JS and code configuration

The easiest way to use Microsoft identity for authentication and to obtain access tokens to authorize requests to secured endpoints in SPAs is to use the Microsoft Authentication Library for JavaScript (MSAL.js).

In an MSAL library, the application registration information is passed as configuration during the library initialization. After adding a script reference to the page, add the following code to obtain an instance of the application:

```
const msalConfig = {
  auth: {
    clientId: '{{AZUREAD_APP_ID}}',
    authority: 'https://login.microsoftonline.com/{{AZUREAD_DIRECTORY_ID}}',
    redirectURI: 'https://localhost:3007'
  },
  cache: {
    cacheLocation: "localStorage",
    storeAuthStateInCookie: true
  }
};
const msalApplication = new Msal.UserAgentApplication(msalConfig);
```

The app ID and directory ID tokens in the code are placeholders for the values from the Azure AD app registration Overview page.

Step 3: Sign in

With the application configured, the next step is to implement user sign-in. Before you can get tokens to access APIs in your application, you need an authenticated user context. You can sign users in to to your application using MSAL.js in two ways:

1. Pop-up window, by using the `loginPopup` method
2. Redirect, by using the `loginRedirect` method

You can also optionally pass the scopes of the APIs for which you need the user to consent at the time of sign-in.

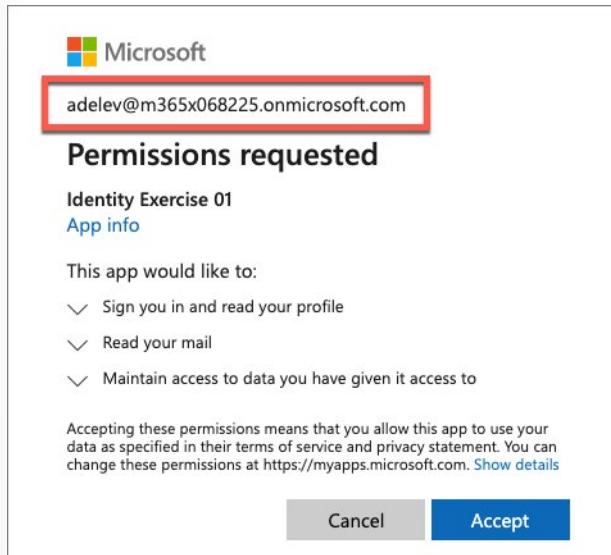
Implement login using a pop-up window:

Do this by calling the `loginPopup()` method and pass in an object with the requested permissions defined:

```
var loginRequest = {
    scopes: ["user.read", "mail.read"]
};

msalApplication.loginPopup(loginRequest)
    .then(function (loginResponse) {
        let idToken = loginResponse.idToken;
    })
    .catch(function (error) {
        console.log(error);
    });
});
```

This code will open a popup that loads the Azure AD sign-in experience. If the user hasn't signed in previously and granted the app the specified permissions, after a successful sign-in, Azure AD will present the user with the consent experience.



After completing the sign-in experience, MSAL.js will close the sign-in popup and execute the success path of the `loginPopup()` method.

Note : The preferred experience when using MSAL when signing-in and obtaining access tokens is to use a popup window. However, some browsers have known issues with popup windows. In these cases, you can choose to use MSAL's equivalent methods that use browser redirects.

Implement login using redirect:

The redirect methods don't return a promise because of the move away from the main app. To process and access the returned tokens, you need to register success and error callbacks before you call the redirect methods.

```
const config = {
    auth: {
        clientId: 'your_app_id',
        redirectUri: "your_app_redirect_uri", //defaults to application
        start page
    }
};
```

```
        postLogoutRedirectUri: "your_app_logout_redirect_uri"
    }
}

const loginRequest = {
    scopes: ["User.ReadWrite"]
}

const myMsal = new userAgentApplication(config);

function authCallback(error, response) {
    //handle redirect response
}

myMsal.handleRedirectCallback(authCallback);

myMsal.loginRedirect(loginRequest);
```

Step 4: Acquire access token

After the user has signed in, the next step is to obtain an access token to use in the authorization header of the request to the protected API.

There are two methods on the MSAL.js API that you can use for this task:

- `acquireTokenSilent()`: If the user has already signed in and an interactive sign-in is not required, this method will return a response that includes the access token:

```
var accessTokenRequest = { scopes: ["user.read", "mail.read"] };

msalApplication.acquireTokenSilent(accessTokenRequest)
.then(function(accessTokenResponse) {
    let accessToken = accessTokenResponse.accessToken;
});
```

When this method is called, the library first checks the cache in browser storage to see if a valid token exists and returns it. When no valid token is in the cache, it sends a silent token request to Azure Active Directory (Azure AD) from a hidden iframe. This method also allows the library to renew tokens.

- `acquireTokenPopup()`: if the `acquireTokenSilent()` fails and/or an interactive sign-in is required, you can use this method. It will combine the request to sign-in and obtain an access token in one step:

```
var accessTokenRequest = { scopes: ["user.read", "mail.read"] };

msalApplication.acquireTokenPopup(accessTokenRequest)
.then(function(accessTokenResponse) {
    let accessToken = accessTokenResponse.accessToken;
});
```

The `scopes` array contains the permissions the app is requesting. This is an example of dynamically/incrementally requested permissions.

Step 5: Calling APIs - Microsoft Graph

Once your SPA has an access token, you can use it to call a secured endpoint, such as Microsoft Graph. Use the acquired access token as a bearer in an HTTP request to call the API:

```
getMessagesFromMSGraph (   
    'https://graph.microsoft.com/v1.0/me/messages?$top=10&$select=subject',   
    tokenResponse.accessToken   
    onSuccessGraphCallback);   
  
function getMessagesFromMSGraph(endpoint, accessToken, callback) {   
    var xmlhttp = new XMLHttpRequest();   
    xmlhttp.onreadystatechange = function () {   
        if (this.readyState == 4 && this.status == 200)   
            callback(JSON.parse(this.responseText));   
    }   
    xmlhttp.open("GET", endpoint, true);   
    xmlhttp.setRequestHeader('Authorization', 'Bearer ' + accessToken);   
    xmlhttp.send();   
}
```

Web apps that sign in users and call APIs

Developers can leverage Microsoft identity to add authentication to web apps to enable users to sign in. Adding authentication enables your web apps to access limited profile information. Once the signed in user grants consent, the web app can obtain a token from Azure AD on behalf of the signed in user and use it to request data from web APIs, such as Microsoft Graph.

The following shows you how to create server-side web apps that allow users to sign in and grant the app permissions to act on the user's behalf. Once the user has authenticated and granted the app consent to act on their behalf, the web application will use data returned from Microsoft Graph by using the **OAuth 2.0 auth code grant flow**.

The authorization code grant flow

The OAuth 2.0 authorization code grant flow is common when websites or custom applications leverage Azure AD as a federated authentication provider. When the application needs a user to sign in, or needs an access token to act on their behalf, it redirects the user over to the authorization endpoint to authenticate. The user signs in using their email and password and in turn Azure AD redirects the user upon a successful sign-in back to a specific URL in the app.

When Azure AD redirects the user back to the web app, it includes an **authorization code**. The authorization code is an encoded string that only Azure AD can read. The web app takes this authorization code, which is valid for a short time, and includes it in a request to the Azure AD token endpoint.

In addition to the authorization code, the request to the token endpoint includes a `grant_type` parameter that tells the endpoint it's exchanging the authorization code to obtain the access token on your behalf.

A benefit of this grant flow is that the web app never sees your username and password. All authentication happens over on Azure AD and the application just gets the authorization code that's a result of the sign-in process. This aspect to the auth code grant flow makes it very secure.

The following provides an overview of how to create server-side web apps that implement authentication and authorization to access Microsoft Graph using the authorization code grant flow.

Step 1: Azure AD app registration

When registering the app in Azure AD, ensure the redirect URI of the app points to the callback URL of the web app. This URL must match the redirect URL provided by the app when the authentication process is started. The authorization code will be sent to this endpoint, which means you need to configure any authentication libraries and/or middleware to listen on this endpoint to receive the authorization code.

A sign-out URL should also be specified so the authentication libraries and/or middleware deletes any cached tokens or other data that are only needed for signed in users.

Identity WebApp - Authentication

Redirect URIs

Type Redirect URI

Web	https://localhost:3007
Web	https://localhost:3007/signin-oidc
Web	e.g. https://myapp.com/auth

Suggested Redirect URIs for public clients (mobile, desktop)

If you are using the Microsoft Authentication Library (MSAL) or the Active Directory Authentication Library (ADAL) to build applications for desktop or mobile devices, you may select from the suggested Redirect URIs below or enter a custom redirect URI above. For more information, refer to the library documentation.

- msal442a8177-15ee-4bbb-b877-07e2378b5bcc://auth (MSAL only)
- https://login.microsoftonline.com/common/oauth2/nativeclient
- https://login.live.com/oauth20_desktop.srf (LiveSDK)

Advanced settings

Logout URL: https://localhost:3007/signout-oidc

Register a client secret

The web app will also need a client secret to sign in with Azure AD to exchange the authorization code for an access token. A **client secret**, also referred to as an application password, is a secret string that the application uses to prove its identity when requesting a token.

You can register your application secrets either through the interactive experience in the Azure portal or by using command-line tools (like PowerShell). From the portal, the management of client credentials happens on the **Certificates & secrets** page for an application:

The screenshot shows the Microsoft Azure portal interface. On the left, there's a sidebar with various service icons like App Services, Function Apps, SQL databases, etc. The main area shows a navigation path: Home > App registrations > ContosoApp_1 - Certificates & secrets. The title bar says 'ContosoApp_1 - Certificates & secrets'. The left sidebar under 'Manage' has 'Certificates & secrets' selected. The right panel has two main sections: 'Certificates' and 'Client secrets'. The 'Certificates' section includes a 'Upload certificate' button and a table with columns THUMBPRINT, START DATE, and EXPIRES. The 'Client secrets' section includes a 'New client secret' button and a table with columns DESCRIPTION, EX..., and VALUE.

- The application secret (also named the client secret) is generated by Azure AD during the registration of the confidential client application. This generation happens when you select 'New clientsecret'. At that point, you must copy the secret string to the clipboard for use in your app, before you select Save. This string won't be presented any longer.
- During application registration, you use the Upload certificate button to upload the certificate. Azure AD supports only certificates that are directly registered on the application and don't follow certificate chains.

Alternatively, you can register your application with Azure AD by using command-line tools.

There are three things you'll need to make note of from the Azure AD app registration:

- **Tenant ID:** ID of your Azure AD directory
- **Client ID:** unique auto-generated ID of the app (*this is also referred to as the application ID*)
- **Client secret:** secret you created during app registration

NOTE:

While a client secret is used here for purposes of simplicity, Microsoft recommends production apps use **certificates** with the OAuth 2.0 authorization code grant flow.

Step 2: MSAL .NET and code configuration

With the app registered in Azure AD, the next step is to configure the web app. For an ASP.NET Core web application, most of these settings are saved in the **appsettings.json** file.

Open this file and set the three values you collected from registering the Azure AD app. You may have to create a key for the client secret.

```
{  
    "AzureAd": {  
        "Instance": "https://login.microsoftonline.com/",  
        "Domain": "qualified.domain.name",  
        "TenantId": "22222222-2222-2222-2222-222222222222",  
        "ClientId": "11111111-1111-1111-1111111111111111",  
        "ClientSecret": "",  
        "CallbackPath": "/signin-oidc"  
    }  
}
```

Next, you need to modify the web app startup process to configure the support for signing-in with Azure AD and obtaining an ID token.

In an ASP.NET Core web application, add the following to the `ConfigureServices()` method.

```
List<string> scopes = new List<string>();  
scopes.Add("offline_access");  
scopes.Add("user.read");  
  
var appSettings = new AzureADOptions();  
Configuration.Bind("AzureAd", appSettings);  
  
var application = ConfidentialClientApplicationBuilder.Create(appSettings.ClientId)  
    .WithAuthority(appSettings.Instance + appSettings.TenantId + "/v2.0/")  
    .WithRedirectUri("https://localhost:5001" + appSettings.CallbackPath)  
    .WithClientSecret(appSettings.ClientSecret)  
    .Build();
```

Then add the following code to the end of the `ConfigureServices()` method. This will complete the Azure AD configuration by doing the following steps:

- Set the authority for Azure AD to be the OAuth 2.0 Azure AD endpoint
- Request both an authorization code and ID token when the user signs in
- Associate the user's name property in ASP.NET to the **preferred_username** claim returned in the ID token
- Create a callback when the authorization code is received to add all claims from the ID token to the current signed-in user's claims

```
services.Configure<OpenIdConnectOptions>(AzureADDefaults.OpenIdScheme, async  
options => {  
    // configure authority to use v2 endpoint  
    options.Authority = options.Authority + "/v2.0/";  
  
    // asking Azure AD for id_token (to establish identity) and authorization code (to get access/refresh tokens for calling services)  
    options.ResponseType = OpenIdConnectResponseType.CodeIdToken;
```

```
// add the permission scopes you want the application to use
options.Scope.Add("offline_access");
options.Scope.Add("user.read");

// validate the token issuer
options.TokenValidationParameters.NameClaimType = "preferred_username";

// wire up event to do second part of code authorization flow (exchanging
// authorization code for token)
var handler = options.Events.OnAuthorizationCodeReceived;
options.Events.OnAuthorizationCodeReceived = async context => {
    // handle the auth code returned post signin
    context.HandleCodeRedemption();
    if (!context.HttpContext.User.Claims.Any()) {
        (context.HttpContext.User.Identity as ClaimsIdentity).AddClaims(-
            context.Principal.Claims);
    }

    // get token
    var token = await application.AcquireTokenByAuthorizationCode(op-
        tions.Scope, context.ProtocolMessage.Code).ExecuteAsync();

    context.HandleCodeRedemption(null, token.IdToken);
    await handler(context).ConfigureAwait(false);
};

});
```

Step 3: Signing-in and acquiring tokens

The sign-in process is handled by redirecting the user to the Azure AD sign-in page. This is set by default for you when you create a new ASP.NET Core web application.

The sign-in process uses the values from the **appsettings.json** file and the configuration defined in the previous setup to create the Azure AD URL to send the user to. This URL specifies things like the Azure AD application's ID, the tenant ID, and the scopes required by the web application.

When a user signs-in to Azure AD and is redirected back to the web application, the web app can use the MSAL.NET library to obtain an access token:

```
List<string> scopes = new List<string>();
IAccount account = {};
var accessToken = await application.AcquireTokenSilent(scopes, account) .
ExecuteAsync();
```

The account is retrieved from the currently signed-in user. The token can then be used to call a secured endpoint.

Step 4: Calling APIs (MS Graph)

The last part of the process is to use the access code in a request to a secured endpoint.

One way to do this is to create an instance of the Microsoft Graph client and add it as a singleton to ASP.NET Core's dependency injection system:

```
// add this to the Startup.cs in the ConfigureServices() method after creating an
// instance of the confidential client (application)
var graphServiceClient = new GraphServiceClient(new DelegateAuthenticationProvider(async (request) => {
    var graphUserAccount = new Helpers.GraphUserAccount(request.Properties["User"] as System.Security.Claims.ClaimsPrincipal);
    var accessToken = await application.AcquireTokenSilent(scopes, graphUserAccount).ExecuteAsync();
    request.Headers.Authorization = new System.Net.Http.Headers.AuthenticationHeaderValue("bearer", accessToken.AccessToken);
}));
services.AddSingleton<GraphServiceClient>(graphServiceClient);
```

The instance of the service can then be used from a controller within the web application:

```
[Authorize]
public class UserController : Controller
{
    private readonly GraphServiceClient _graphServiceClient;

    public UserController(GraphServiceClient graphServiceClient)
    {
        _graphServiceClient = graphServiceClient;
    }

    public async Task<IActionResult> Index()
    {
        var request = this._graphServiceClient.Me.Request().GetHttpRequestMessage();
        request.Properties["User"] = HttpContext.User;
        var response = await this._graphServiceClient.HttpProvider.SendAsync(request);
        var handler = new ResponseHandler(new Serializer());
        var user = await handler.HandleResponse<User>(response);

        return View(user);
    }
}
```

Daemon and non-interactive apps

Many apps expect user interaction to authenticate and obtain tokens to call web APIs on behalf of users. However, there are also apps that need to run without user interaction such as those that process batch jobs, run services, or daemon processes. Developers can create applications that obtain tokens without user interaction using the **OAuth 2.0 client credentials flow**.

Here are some examples of use cases for daemon apps:

- web applications that are used to provision or administer users, or do batch processes in a directory
- desktop applications (such as windows services on Windows, or daemons processes on Linux) that do batch jobs, or an operating system service running in the background
- web APIs that need to manipulate directories, not specific users

Using the OAuth 2.0 client credentials flow, your application can acquire a token to call a web API on behalf of itself (not on behalf of a user).

This scenario is typically used when an app needs an access token but doesn't want to work under the context of user+app permissions. Instead, non-interactive apps must use **application permissions**. In these scenarios, the app has its own credentials that are used to authenticate and obtain an access token from a token issuer.

Another common case where non-daemon applications use the client credentials flow is when they need elevated privileges that a user doesn't have. For example, even when apps act on behalf of users, they need to access a web API or a resource with their identity and not the user's identity, such as accessing secrets in Azure KeyVault or an Azure SQL database for a cache.

Applications that acquire a token for their own identities:

- Are **confidential client applications**. These apps, given that they access resources independently of a user, need to prove their identity. They're also rather sensitive apps, which they need to be approved by the Azure Active Directory (Azure AD) tenant admins.
- Have registered a **secret** (application password or certificate) with Azure AD. This secret is passed-in during the call to Azure AD to acquire a token.

OAuth 2.0 client credentials flow

During the Azure AD application registration process, you create an SSL certificate public/private key pair.

The public certificate is registered with our Azure AD app. Within the service or daemon app, the private certificate is used to sign a JWT that is sent to the token endpoint.

Administrators also can use a client secret instead of a certificate, but this is not recommended for production.

The following example shows you how to create apps that obtain tokens without user interaction to either act on behalf of a user or do tasks with the app's identity.

Step 1: Azure AD app registration

In order for a service or daemon app to use Microsoft identity to enable users to authenticate and obtain access tokens for use with services such as Microsoft Graph, you must register a new app with Azure AD. This can be done using the Azure AD admin center <https://aad.portal.azure.com>.

Unlike for other types of applications, you don't need to specify a redirect URI or sign out URI.

In addition, you don't need to enable the ID token or access token for the implicit grant flow; these don't apply to the client credentials grant flow.

Daemon applications can only request application permissions to access APIs (not delegated permissions). Because of this, their supported account type during registration can't be an account in any organizational directory or any personal Microsoft account (for example, Skype, Xbox, Outlook.com). There's no tenant admin to grant consent to a daemon application for

a personal Microsoft account. You'll need to choose accounts in my organization or accounts in any organization.

Configure and grant application permissions

The permissions for a daemon application must be defined ahead of time, or statically. Daemon applications require that a tenant admin pre-consent to the application calling the web API. As described in the previous lesson, the permissions can be configured and granted in the API permissions page in the Azure AD admin center.

The screenshot shows the 'Request API permissions' dialog in the Azure Active Directory admin center. It is set to 'Microsoft Graph' and 'Delegated permissions'. A note says: 'Your application runs as a background service or daemon without a signed-in user.' Under 'Select permissions', 'Mail.R' is selected. In the 'Mail (1)' section, 'Mail.Read' is checked. The 'Add permissions' button is highlighted with a red box.

Register client secret or certificate

As for any confidential client application, you need to register a **client secret or certificate**. As described earlier in this lesson, you can register your application secrets either through the interactive experience in the Azure portal or by using command-line tools (like PowerShell). After generating a client secret in Azure AD, be sure to copy the secret string to the clipboard for use in your app, before you select Save. This string won't be presented any longer.

Step 2: MSAL .NET and code configuration

The configuration of an app that leverages the client credentials flow is similar to a web app or console app. First create an instance of the confidential client using the MSAL library:

```
string tenantId = config["tenantId"];
string clientId = config["applicationId"];
string clientSecret = config["applicationSecret"];
string authority = $"https://login.microsoftonline.com/{config["tenantId"]}/"
```

```
v2.0";  
  
var cca = ConfidentialClientApplicationBuilder.Create(clientId)  
    .WithAuthority(authority)  
    .WithClientSecret(clientSecret)  
    .Build();
```

This code uses the **client secret** option with the client credentials flow.

The following example demonstrates how to do the same thing using a **certificate** that has been set in the Azure AD app registration:

```
string tenantId = config["tenantId"];  
string clientId = config["applicationId"];  
X509Certificate2 certificate = ReadCertificate(config.CertificateName);  
string authority = $"https://login.microsoftonline.com/{config["tenantId"]}/  
v2.0";  
  
var cca = ConfidentialClientApplicationBuilder.Create(clientId)  
    .WithAuthority(authority)  
    .WithCertificate(certificate)  
    .Build();
```

Step 3: Signing in and acquiring tokens

Once the confidential client is created, use the `AcquireTokenForClient()` method (or its equivalent depending on the platform) to obtain an access token for the app:

```
List<string> scopes = new List<string>();  
scopes.Add("REPLACE_WITH_APP_ID/.default");  
  
result = await cca.AcquireTokenForClient(_scopes).ExecuteAsync();  
return result.AccessToken;
```

Note the scopes specified when requesting an access token. In other scenarios, the code specified the exact permission (also known as scopes) the application needed.

For the client credentials flow, you should request the `/ .default` scope, which tells Azure AD to use the application level permissions declared statically during the application registration.

In MSAL.NET, `AcquireTokenForClient` uses the application token cache. (All the other `AcquireTokenXX` methods use the user token cache.) Don't call `AcquireTokenSilent` before you call `AcquireTokenForClient`, because `AcquireTokenSilent` uses the user token cache. `AcquireTokenForClient` checks the application token cache itself and updates it.

Step 4: Calling APIs (MS Graph)

Finally, you can use the configured confidential client to call a secured endpoint, such as Microsoft Graph. When using the Microsoft Graph .NET SDK, first create an instance of the MSAL authentication provider:

```
public class MsalAuthenticationProvider : IAuthenticationProvider  
{
```

```
private IConfidentialClientApplication _application;
private string[] _scopes;

public MsalAuthenticationProvider(IConfidentialClientApplication application, string[] scopes)
{
    _application = application;
    _scopes = scopes;
}

public async Task AuthenticateRequestAsync(HttpRequestMessage request)
{
    request.Headers.Authorization = new AuthenticationHeaderValue("bearer",
await GetTokenAsync());
}

public async Task<string> GetTokenAsync()
{
    AuthenticationResult result = null;

    try {
        result = await _application.AcquireTokenForClient(_scopes).ExecuteAsync();
    } catch (MsalServiceException) { }

    return result.AccessToken;
}

// create the confidential client application (cca)
var app = new MsalAuthenticationProvider(cca, scopes);
```

Next, use the MSAL authentication provider in creating a new instance of the GraphServiceClient from the Microsoft Graph .NET SDK:

```
private static GraphServiceClient GetAuthenticatedGraphClient(IConfigurationRoot config)
{
    var authenticationProvider = CreateAuthorizationProvider(config);
    return new GraphServiceClient(authenticationProvider);
}
```

Finally, use the GraphServiceClient to submit requests to Microsoft Graph:

```
var client = GetAuthenticatedGraphClient(config);
var requestUserEmail = client.Users[config["targetUserId"]].Messages.Request();
var results = requestUserEmail.GetAsync().Result;
```

Lesson review questions

1. Why is an application secret not required when authenticating with Azure AD in a SPA?
 - (A) Instead of using an app secret, the SPA can use a x509 certificate. One of the two options is required.
 - (B) They aren't needed because the user's password acts as the Azure AD app's secret.
 - (C) Secrets can't be reliably secured in a SPA because its inherently a client-side application. In no scenario can the secret reach the user's browser and be secured.
2. Which of the following ways isn't a valid authentication option for an app using the OAuth 2.0 client credentials flow?
 - (A) Include the client ID and certificate assigned to the app within the Azure AD management portal.
 - (B) Include the client ID as well as a user account's credentials that acts as a service account.
 - (C) Include the client ID and client password assigned to the app within the Azure AD management portal.
3. What minimal scope(s) must a daemon or service app include in the access token request to Azure AD?
 - (A) A single scope of the resource followed by .default, such as <https://graph.microsoft.com/.default>.
 - (B) Every scope added from within the Azure AD admin center for the registered app.
 - (C) Nothing - because the permissions were statically specified on the app in the Azure AD admin center, daemon apps don't need to include specific permissions the access token request.

Correct/suggested answers:

1. (C)
2. (B) - Daemon apps can authenticate using a client secret or a certificate.
3. (A) - The .default scope allows the app to request the statically defined permissions defined for the app in Azure AD admin center.

Secure custom APIs

Lesson Introduction

Developers can secure custom web APIs with Microsoft identity to ensure only approved apps can access the web APIs, provided they've been granted the necessary permissions. Once an API has been secured with Microsoft identity, you can enable users and apps to authenticate and access the web APIs through many different types of applications including desktop, mobile, web, and services or daemons. The Microsoft identity platform is a great solution for securing APIs especially when your users and their data are secured with Microsoft identity.

In this lesson, you'll learn how to configure and develop custom APIs secured with Microsoft identity.

After this lesson, you should be able to:

- validate access tokens in an API
- configure effective permissions for delegated scopes
- implement app permissions using roles
- use a delegated access token to call a Microsoft API

Register a protected web API

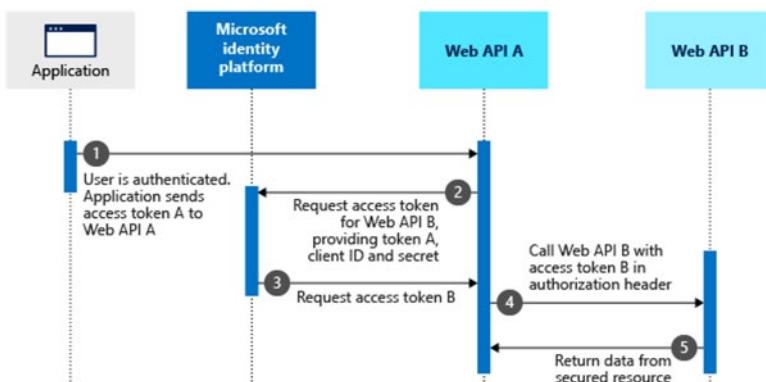
Scenarios for protected web APIs

There are two types of applications/services that might communicate with your custom API secured with Microsoft identity: an application that acts on behalf of a user, and one that acts on behalf of itself.

On-Behalf-Of flow

The first option, on behalf of, is when a user authenticates and uses an application that will then make a request, on behalf of the user, to the secured web API. These apps use the **On-Behalf-Of (OBO) flow**.

Assume that the user has been authenticated on an application using the OAuth 2.0 authorization code grant flow or another login flow. At this point, the application has an access token for API A (token A) with the user's claims and consent to access the middle-tier web API (API A). Now, API A needs to make an authenticated request to the downstream web API (API B).



In this flow:

1. The client application makes a request to API A with token A (with an aud claim of API A).

2. API A authenticates to the Microsoft identity platform token issuance endpoint and requests a token to access API B.
3. The Microsoft identity platform token issuance endpoint validates API A's credentials along with token A and issues the access token for API B (token B) to API A.
4. Token B is set by API A in the authorization header of the request to API B.
5. Data from the secured resource is returned by API B to API A, and from there to the client.

To allow access to your API in this scenario, you expose scopes, or delegated permissions.

Daemon apps

In the second option, the application acts as a service or daemon where there's no user interaction. In this case, the app acts as itself. Daemon apps authenticate with Microsoft identity, obtain an access token, and include the token in the authorization header value of each HTTP request to the web API. If your API is called by a daemon app, you need to expose application permissions.

In either case, registering a protected web API with the Microsoft identity platform follows many of the common steps for registering an app as described in lesson 01, with some specific considerations for protected APIs.

These considerations include:

- which token version will the API accept?
- the redirect URI field
- exposing scopes and app roles

The following sections describe these considerations further.

Accepted token version

The Microsoft identity platform endpoint can issue v1.0 tokens and v2.0 tokens.

The accepted token version depends on the **Supported account types** value you choose when you create your application.

If the value of Supported account types is **Accounts in any organizational directory and personal Microsoft accounts (e.g. Skype, Xbox, Outlook.com)**, the accepted token version is **v2.0**. Otherwise, the accepted token version is **v1.0**.

After you create the application, you can determine or change the accepted token version by following these steps:

1. In the Azure portal, select your app and then select Manifest.
2. Find the property accessTokenAcceptedVersion in the manifest. The property's default value is 2. The value specifies to Azure Active Directory (Azure AD) which token version the web API accepts.
 - If the value is 2, the web API accepts v2.0 tokens.
 - If the value is null, the web API accepts v1.0 tokens.
3. If you changed the token version, select Save.

Note: The web API specifies which token version it accepts. When a client requests a token for your web API from the Microsoft identity platform (v2.0) endpoint, the client gets a token that indicates which token version the web API accepts.

No redirect URI

Web APIs don't need to register a redirect URI because no user is interactively signed in.

Application ID URI and scopes

Scopes usually have the form resourceURI/scopedName. For Microsoft Graph, the scopes have shortcuts. For example, User.Read is a shortcut for <https://graph.microsoft.com/user.read>.

During app registration for a web API, you need to define these parameters:

- The resource URI
- One or more scopes
- One or more app roles

Note: By default, the application registration portal recommends that you use the resource URI `api://{clientId}`. This URI is unique but not human readable. If you change the URI, make sure the new value is unique.

To client applications, scopes show up as delegated permissions and app roles show up as application permissions for your web API.

Scopes also appear on the consent window that's presented to users of your app. So you need to provide the corresponding strings that describe the scope:

- As seen by a user.
- As seen by a tenant admin, who can grant admin consent.

Exposing delegated permissions (scopes)

Once the app is registered, you can then define the delegated permissions, or scopes, your API exposes.

You then will write the business logic within your web API to determine what is allowed or prohibited based on the scopes and user identity listed in the access token received in the HTTP request.

Each scope has name and description properties for the user consent and admin consent. As a reminder, user consent is when a user grants an app permission to act on their behalf while admin consent is when a global tenant administrator grants the permission on behalf of all users or an application.

To expose scopes:

1. Select **Expose an API** in the application registration.
2. Select **Add a scope**.
3. If prompted, accept the proposed application ID URI (`api://{clientId}`) by selecting **Save and Continue**.
4. You will need to specify the following values:
 - **Scope name** (example: `access_as_user`)
 - **Who can consent?** (Admins and users or Admins only)
 - **Admin consent display name** (example: "Access TodoListService as a user")
 - **Admin consent description** (Example: "Accesses the TodoListService web API as a user")
 - **User consent display name** (Example: "Access TodoListService as a user")

- **User consent description** (Example: Accesses the TodoListService web API as a user)
 - **State:** Enabled or Disabled
5. Select **Add scope**

Exposing application permissions (app roles)

If your web API is called by a daemon app:

If your API is called by a daemon app, there are specific considerations when registering with the Microsoft identity platform:

- You declare and expose only application permissions because daemon apps don't interact with users. Delegated permissions wouldn't make sense.
- Tenant admins can require Azure AD to issue web API tokens only to applications that have registered to access one of the API's application permissions.

To expose application permissions, you need to edit the manifest, as follows:

1. In the application registration for your application, select **Manifest**.
2. To edit the manifest, find the `appRoles` setting and add application roles. An example is provided in the following sample JSON block.
3. Leave `allowedMemberTypes` set to "Application" only.
4. Make sure `id` is a unique GUID.
5. Make sure `displayName` and `value` don't contain spaces.
6. Save the manifest.

The following sample shows the contents of `appRoles`, where the value of `id` can be any unique GUID.

```
"appRoles": [
    {
        "allowedMemberTypes": [ "Application" ],
        "description": "Accesses the TodoListService-Cert as an application.",
        "displayName": "access_as_application",
        "id": "ccf784a6-fd0c-45f2-9c08-2f9d162a0628",
        "isEnabled": true,
        "lang": null,
        "origin": "Application",
        "value": "access_as_application"
    }
],
```

Ensure that Azure AD issues tokens for your web API to only allowed clients

The web API checks for the app role. This role is a software developer's way to expose application permissions. You can also configure Azure AD to issue API tokens only to apps that the tenant admin approves for API access.

To add this increased security:

1. Go to the app **Overview** page for your app registration.
2. Under **Managed application in local directory**, select the link with the name of your app. The label for this selection might be truncated. For example, you might see Managed application in ...

Note: When you select this link, you go to the Enterprise Application Overview page. This page is associated with the service principal for your application in the tenant where you created it. You can go to the app registration page by using the back button of your browser.

3. Select the **Properties** page in the **Manage** section of the Enterprise application pages.
4. If you want Azure AD to allow access to your web API from only certain clients, set **User assignment required?** to **Yes**.
5. Select **Save**.

Note: If you set **User assignment required?** to **Yes**, Azure AD checks the app role assignments of a client when it requests a web API access token. If the client isn't assigned to any app roles, Azure AD will return the error message "invalid_client: AADSTS501051: Application <application name> is not assigned to a role for the <web API>".

If you keep **User assignment required?** set to **No**, Azure AD won't check the app role assignments when a client requests an access token for your web API. Any daemon client, meaning any client using the client credentials flow, can get an access token for the API just by specifying its audience. Any application can access the API without having to request permissions for it.

But as explained in the previous section, your web API can always verify that the application has the right role, which is authorized by the tenant admin. The API performs this verification by validating that the access token has a roles claim and that the value for this claim is correct. In the previous JSON sample, the value is `access_as_application`.

Validate access tokens

To configure the code for your protected web API, you need to understand:

- What defines APIs as protected.
- How to configure a bearer token.
- How to validate the token.
- What defines ASP.NET and ASP.NET Core APIs as protected?

Like web apps, the ASP.NET and ASP.NET Core web APIs are protected because their controller actions are prefixed with the `[Authorize]` attribute. The controller actions can be called only if the API is called with an authorized identity.

Consider the following questions:

- Only an app can call a web API. How does the API know the identity of the app that calls it?
- If the app calls the API on behalf of a user, what's the user's identity?

Bearer token

The bearer token that's set in the header when the app is called holds information about the app identity. It also holds information about the user unless the web app accepts service-to-service calls from a daemon app.

Here's a C# code example that shows a client calling the API after it acquires a token with Microsoft Authentication Library for .NET (MSAL.NET):

```
var scopes = new[] { $"api://.../access_as_user" };
var result = await app.AcquireToken(scopes)
    .ExecuteAsync();
```

```
httpClient = new HttpClient();
httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", result.AccessToken);

// Call the web API.
HttpResponseMessage response = await _httpClient.GetAsync(apiUri);
```

Note: A client application requests the bearer token to the Microsoft identity platform endpoint for the web API. The web API is the only application that should verify the token and view the claims it contains. Client apps should never try to inspect the claims in tokens. In the future, the web API might require that the token be encrypted. This requirement would prevent access for client apps that can view access tokens.

JwtBearer configuration

This section describes how to configure a bearer token.

Config file:

```
{
    "AzureAd": {
        "Instance": "https://login.microsoftonline.com/",
        "ClientId": "[Client_id-of-web-api-eg-2ec40e65-ba09-4853-bcde-
bcb60029e596]",
        /*
            You need specify the TenantId only if you want to accept access
            tokens from a single tenant
            (line-of-business app).
            Otherwise, you can leave them set to common.
            This can be:
            - A GUID (Tenant ID = Directory ID)
            - 'common' (any organization and personal accounts)
            - 'organizations' (any organization)
            - 'consumers' (Microsoft personal accounts)
        */
        "TenantId": "common"
    },
    "Logging": {
        "LogLevel": {
            "Default": "Warning"
        }
    },
    "AllowedHosts": "*"
}
```

Code initialization

When an app is called on a controller action that holds an `[Authorize]` attribute, ASP.NET and ASP.NET Core extract the access token from the Authorization header's bearer token. The access token is then forwarded to the `JwtBearer` middleware, which calls Microsoft IdentityModel Extensions for .NET.

In ASP.NET Core, this middleware is initialized in the `Startup.cs` file.

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
```

The middleware is added to the web API by this instruction:

```
services.AddAuthentication(AzureADDefaults.JwtBearerAuthenticationScheme)
    .AddAzureADBearer(options => Configuration.Bind("AzureAd", options));
```

Currently, the ASP.NET Core templates create Azure Active Directory (Azure AD) web APIs that sign in users within your organization or any organization. They don't sign in users with personal accounts. But you can change the templates to use the Microsoft identity platform endpoint by adding this code to `Startup.cs`:

```
services.Configure<JwtBearerOptions>(AzureADDefaults.JwtBearerAuthenticationScheme, options =>
{
    // This is a Microsoft identity platform web API.
    options.Authority += "/v2.0";

    // The web API accepts as audiences both the Client ID (options.Audience) and api://{ClientID}.
    options.TokenValidationParameters.ValidAudiences = new []
    {
        options.Audience,
        $"api://{options.Audience}"
    };

    // Instead of using the default validation (validating against a single tenant,
    // as we do in line-of-business apps),
    // we inject our own multitenant validation logic (which even accepts both v1 and v2 tokens).
    options.TokenValidationParameters.IssuerValidator = AadIssuerValidator.
        GetIssuerValidator(options.Authority).Validate;;
});
```

The preceding code snippet is extracted from the ASP.NET Core web API incremental tutorial in **Microsoft.Identity.Web/WebApiServiceCollectionExtensions.cs#L50-L63**³. The `AddProtectedWebApi` method, which does more than the snippet shows, is called from `Startup.cs`.

³ <https://github.com/Azure-Samples/active-directory-dotnet-native-aspnetcore-v2/blob/154282843da2fc2958fad151e2a11e521e358d42/Microsoft.Identity.Web/WebApiServiceCollectionExtensions.cs#L50-L63>

Token validation

In the preceding snippet, the JwtBearer middleware, like the OpenID Connect middleware in web apps, validates the token based on the value of TokenValidationParameters. The token is decrypted as needed, the claims are extracted, and the signature is verified. The middleware then validates the token by checking for this data:

- Audience: The token is targeted for the web API.
- Sub: It was issued for an app that's allowed to call the web API.
- Issuer: It was issued by a trusted security token service (STS).
- Expiry: Its lifetime is in range.
- Signature: It wasn't tampered with.

There can also be special validations. For example, it's possible to validate that signing keys, when embedded in a token, are trusted and that the token isn't being replayed. Finally, some protocols require specific validations.

Validators

The validation steps are captured in validators, which are provided by the Microsoft.IdentityModel.Extensions for .NET open-source library. The validators are defined in the library source file Microsoft.IdentityModel.Tokens/Validators.cs.

This table describes the validators:

Validator	Description
ValidateAudience	Ensures the token is for the application that validates the token for you.
ValidateIssuer	Ensures the token was issued by a trusted STS, meaning it's from someone you trust.
ValidateIssuerSigningKey	Ensures the application validating the token trusts the key that was used to sign the token. There's a special case where the key is embedded in the token. But this case doesn't usually arise.
ValidateLifetime	Ensures the token is still or already valid. The validator checks if the lifetime of the token is in the range specified by the notbefore and expires claims.
ValidateSignature	Ensures the token hasn't been tampered with.
ValidateTokenReplay	Ensures the token isn't replayed. There's a special case for some onetime-use protocols.

The validators are associated with properties of the TokenValidationParameters class. The properties are initialized from the ASP.NET and ASP.NET Core configuration.

In most cases, you don't need to change the parameters. Apps that aren't single tenants are exceptions. These web apps accept users from any organization or from personal Microsoft accounts. Issuers in this case must be validated.

Verify scopes and app roles

When you create the controllers, or endpoints, for your web API, you'll need to make sure the current request has the necessary permissions, or scopes, granted to it.

Specifically, you want to make sure that the API is called only by:

- Applications on behalf of users who have the right scopes.
- Daemon apps that have the right application roles.

Note: The code snippets from this article are extracted from the following samples, which are fully functional:

- **ASP.NET Core web API incremental tutorial⁴** on GitHub
- **ASP.NET web API sample⁵**

To protect an ASP.NET or ASP.NET Core web API, you must add the **[Authorize]** attribute to one of the following items:

- The controller itself if you want all controller actions to be protected
- The individual controller action for your API

```
[Authorize]
public class TodoListController : Controller
{
    ...
}
```

But this protection isn't enough. It guarantees only that ASP.NET and ASP.NET Core validate the token. Your API needs to verify that the token used to call the API is requested with the expected claims. These claims in particular need verification:

- The **scopes** if the API is called on behalf of a user.
- The **app roles** if the API can be called from a daemon app.

Verify scopes in APIs called on behalf of users

If a client app calls your API on behalf of a user, the API needs to request a bearer token that has specific scopes for the API.

```
[Authorize]
public class TodoListController : Controller
{
    /// <summary>
    /// The web API will accept only tokens 1) for users, 2) that have the
    `access_as_user` scope for
    /// this API.
    /// </summary>
    const string scopeRequiredByAPI = "access_as_user";
```

⁴ <https://github.com/Azure-Samples/active-directory-dotnet-native-aspnetcore-v2/blob/02352945c1c4abb895f0b700053506dcde7ed04a/1.%20Desktop%20app%20calls%20Web%20API/TodoListService/Controllers/TodoListController.cs#L37>

⁵ <https://github.com/Azure-Samples/ms-identity-aspnet-webapi-onbehalfof/blob/dfd0115533d5a230baff6a3259c76cf117568bd9/TodoListService/Controllers/TodoListController.cs#L48>

```
// GET: api/values
[HttpGet]
public IEnumerable<TodoItem> Get()
{
    VerifyUserHasAnyAcceptedScope(scopeRequiredByAPI);
    // Do the work and return the result.
    ...
}
...
}
```

The `VerifyUserHasAnyAcceptedScope` method does something like the following steps:

- Verify there's a claim named `http://schemas.microsoft.com/identity/claims/scope` or `scp`.
- Verify the claim has a value that contains the scope expected by the API.

```
/// <summary>
/// When applied to a <see cref="HttpContext"/>, verifies that the user
/// authenticated in the
/// web API has any of the accepted scopes.
/// If the authenticated user doesn't have any of these <paramref
/// name="acceptedScopes"/>, the
/// method throws an HTTP Unauthorized error with a message noting
/// which scopes are expected in the token.
/// </summary>
/// <param name="acceptedScopes">Scopes accepted by this API</param>
/// <exception cref="HttpRequestException"/> with a <see cref="HttpRe-
/// sponse.StatusCode"/> set to
/// <see cref=" HttpStatusCode.Unauthorized"/>
public static void VerifyUserHasAnyAcceptedScope(this HttpContext
context,
params string[] ac-
ceptedScopes)
{
    if (acceptedScopes == null)
    {
        throw new ArgumentNullException(nameof(acceptedScopes));
    }
    Claim scopeClaim = HttpContext?.User
        ?.FindFirst("http://schemas.micro-
soft.com/identity/claims/scope");
    if (scopeClaim == null || !scopeClaim.Value.Split(' ').Inter-
sect(acceptedScopes).Any())
    {
        context.Response.StatusCode = (int) HttpStatusCode.Unauthorized;
        string message = $"The 'scope' claim does not contain scopes
'{string.Join(", ", acceptedScopes)}' or was not found";
        throw new HttpRequestException(message);
    }
}
```

The preceding sample code is for ASP.NET Core. For ASP.NET, just replace `HttpContext.User` with `ClaimsPrincipal.Current`, and replace the claim type "http://schemas.microsoft.com/identity/claims/scope" with "scp". Also see the code snippet later in this article.

Verify app roles in APIs called by daemon apps

If your web API is called by a daemon app, that app should require an application permission to your web API. As shown in the previous topic of this lesson, your API exposes such permissions. One example is the `access_as_application` app role.

You now need to have your API verify that the token it receives contains the roles claim and that this claim has the expected value. The verification code is similar to the code that verifies delegated permissions, except that your controller action tests for roles instead of scopes:

```
[Authorize]
public class TodoListController : ApiController
{
    public IEnumerable<TodoItem> Get()
    {
        ValidateAppRole("access_as_application");
        ...
    }
}
```

The `ValidateAppRole` method can be something like this:

```
private void ValidateAppRole(string appRole)
{
    //
    // The `role` claim tells you what permissions the client application
    has in the service.
    // In this case, we look for a `role` value of `access_as_application`.
    //
    Claim roleClaim = ClaimsPrincipal.Current.FindFirst("roles");
    if (roleClaim == null || !roleClaim.Value.Split(' ').Contains(appRole))
    {
        throw new HttpResponseException(new HttpResponseMessage
        {
            StatusCode = HttpStatusCode.Unauthorized,
            ReasonPhrase = $"The 'roles' claim does not contain '{appRole}'"
        });
    }
}
```

This time, the code snippet is for ASP.NET. For ASP.NET Core, just replace `ClaimsPrincipal.Current` with `HttpContext.User`, and replace the "roles" claim name with "http://schemas.microsoft.com/ws/2008/06/identity/claims/role". Also see the code snippet earlier in this article.

Accepting app-only tokens if the web API should be called only by daemon apps

Users can also use roles claims in user assignment patterns, as shown in the next topic. If the roles are assignable to both, checking roles will let apps sign in as users and users to sign in as apps. We recommend that you declare different roles for users and apps to prevent this confusion.

If you want only daemon apps to call your web API, add the condition that the token is an app-only token when you validate the app role.

```
string oid = ClaimsPrincipal.Current.FindFirst("oid")?.Value;
string sub = ClaimsPrincipal.Current.FindFirst("sub")?.Value;
bool isAppOnlyToken = oid == sub;
```

Checking for the inverse condition allows only apps that sign in a user to call your API.

Call a Microsoft API

When a web application calls a protected web API on behalf of a user, the API can use that delegated access token to call another, downstream API.

Overview

A web, desktop, mobile, or single-page application client (not represented in the accompanying diagram) calls a protected web API and provides a JSON Web Token (JWT) bearer token in its "Authorization" HTTP header.

The protected web API validates the token and uses the Microsoft Authentication Library (MSAL) `AcquireTokenOnBehalfOf` method to request another token from Azure Active Directory (Azure AD) so that the protected web API can call a second web API, or downstream web API, on behalf of the user.

The protected web API can also call `AcquireTokenSilent` later to request tokens for other downstream APIs on behalf of the same user. `AcquireTokenSilent` refreshes the token when needed.

The app registration part that's related to API permissions is typical. The app configuration involves using the OAuth 2.0 On-Behalf-Of flow to exchange the JWT bearer token against a token for a downstream API. This token is added to the token cache, where it's available in the web API's controllers, and it can then acquire a token silently to call downstream APIs.

The following sections will walk through this process in greater detail.

App registration

The general app registration considerations are identical to any other protected web API, following the guidelines described earlier in this lesson in the "Register a protected web API" section.

However, because the web app now calls web APIs, it becomes a confidential client application. Extra registration information is required because the app needs to share secrets (client credentials) with the Microsoft identity platform.

Register secrets or certificates

As for any confidential client application, you need to register a secret or certificate. As described earlier in this lesson, you can register your application secrets either through the interactive experience in the

Azure portal or by using command-line tools (like PowerShell). After generating a client secret in Azure AD, be sure to copy the secret string to the clipboard for use in your app, before you select Save. This string won't be presented any longer.

Request permissions

Web apps call APIs on behalf of users for whom the bearer token was received. The web apps need to request the appropriate delegated permissions.

Code configuration

On top of the code configuration for any protected web APIs, as described earlier in this lesson, you need to subscribe to the validation of the bearer token that you receive when your API is called:

```
/// <summary>
/// Protects the web API with the Microsoft identity platform, or Azure
/// Active Directory (Azure AD) developer platform
/// This supposes that the configuration files have a section named "AzureAD"
/// </summary>
/// <param name="services">The service collection to which to add authenti-
/// cation</param>
/// <param name="configuration">Configuration</param>
/// <returns></returns>
public static IServiceCollection AddProtectedApiCallsWebApis(this IService-
Collection services,
    IConfiguration
    configuration,
    IEnumera-
    ble<string> scopes)
{
    services.AddTokenAcquisition();
    services.Configure<JwtBearerOptions>(AzureADDefaults.JwtBearerAuthenti-
    cationScheme, options =>
    {
        // When an access token for our own web API is validated, we add it
        // to the MSAL.NET cache so that it can be used from the control-
        lers.
        options.Events = new JwtBearerEvents();

        options.Events.OnTokenValidated = async context =>
        {
            context.Success();

            // Adds the token to the cache and handles the incremental
            consent
            // and claim challenges
            AddAccountToCacheFromJwt(context, scopes);
            await Task.FromResult(0);
        };
    });
    return services;
}
```

```
}
```

On-Behalf-Of flow

The `AddAccountToCacheFromJwt()` method needs to:

- Instantiate a Microsoft Authentication Library (MSAL) confidential client application.
- Call the `AcquireTokenOnBehalf` method. This call exchanges the bearer token that was acquired by the client for the web API against a bearer token for the same user, but it has the API call a downstream API.

Instantiate a confidential client application:

This flow is available only in the confidential client flow, so that the protected web API provides client credentials (client secret or certificate) to the `ConfidentialClientApplicationBuilder` class via either the `WithClientSecret` or `WithCertificate` method.

List of `IConfidentialClientApplication` methods

```
IConfidentialClientApplication app;

#if !VariationWithCertificateCredentials
app = ConfidentialClientApplicationBuilder.Create(config.ClientId)
    .WithClientSecret(config.ClientSecret)
    .Build();
#else
// Building the client credentials from a certificate
X509Certificate2 certificate = ReadCertificate(config.CertificateName);
app = ConfidentialClientApplicationBuilder.Create(config.ClientId)
    .WithCertificate(certificate)
    .Build();
#endif
```

Note: instead of proving their identity via a client secret or a certificate, confidential client applications can also prove their identity by using client assertions.

How to call On-Behalf-Of

You make the On-Behalf-Of (OBO) call by calling the `AcquireTokenOnBehalf` method on the `IConfidentialClientApplication` interface.

The `UserAssertion` class is built from the bearer token that's received by the web API from its own clients. There are two constructors:

- One that takes a JSON Web Token (JWT) bearer token
- One that takes any kind of user assertion, another kind of security token, whose type is then specified in an additional parameter named `assertionType`

In practice, the OBO flow is often used to acquire a token for a downstream API and store it in the MSAL.NET user token cache. You do this so that other parts of the web API can later call on the overrides of `AcquireTokenOnSilent` to call the downstream APIs. This call has the effect of refreshing the tokens, if needed.

```
private void AddAccountToCacheFromJwt(IEnumerable<string> scopes, JwtSecurityToken jwtToken, ClaimsPrincipal principal, HttpContext httpContext)
```

```
{  
    try  
    {  
        UserAssertion userAssertion;  
        IEnumerable<string> requestedScopes;  
        if (jwtToken != null)  
        {  
            userAssertion = new UserAssertion(jwtToken.RawData, "urn:ietf:params:oauth:grant-type:jwt-bearer");  
            requestedScopes = scopes ?? jwtToken.Audiences.Select(a =>  
$"{a}/.default");  
        }  
        else  
        {  
            throw new ArgumentOutOfRangeException("tokenValidationContext.SecurityToken should be a JWT Token");  
        }  
  
        // Create the application  
        var application = BuildConfidentialClientApplication(HttpContext, principal);  
  
        // .Result to make sure that the cache is filled in before the controller tries to get access tokens  
        var result = application.AcquireTokenOnBehalfOf(requestedScopes.  
Except(scopesRequestedByMsalNet),  
userAssertion)  
            .ExecuteAsync()  
            .GetAwaiter().GetResult();  
    }  
    catch (MsalException ex)  
    {  
        Debug.WriteLine(ex.Message);  
        throw;  
    }  
}
```

Acquire a token

After you've built a client application object, use it to acquire a token that you can use to call a web API.

Here's an example of code that's called in the actions of the API controllers. It calls a downstream API named todolist.

```
private async Task GetTodoList(bool isAppStarting)  
{  
    ...  
    //  
    // Get an access token to call the To Do service.  
    //  
    AuthenticationResult result = null;
```

```
try
{
    app = BuildConfidentialClient(HttpContext, HttpContext.User);
    result = await app.AcquireTokenSilent(Scopes, account)
        .ExecuteAsync()
        .ConfigureAwait(false);
}
...
}
```

`BuildConfidentialClient()` is similar to the scenario in A web API that calls web APIs: App configuration. `BuildConfidentialClient()` instantiates `IConfidentialClientApplication` with a cache that contains information for only one account. The account is provided by the `GetAccountIdentifier` method.

The `GetAccountIdentifier` method uses the claims that are associated with the identity of the user for whom the web API received the JSON Web Token (JWT):

```
public static string GetMsalAccountId(this ClaimsPrincipal claimsPrincipal)
{
    string userObjectId = GetObjectId(claimsPrincipal);
    string tenantId = GetTenantId(claimsPrincipal);

    if (!string.IsNullOrWhiteSpace(userObjectId)
        && !string.IsNullOrWhiteSpace(tenantId))
    {
        return $"{userObjectId}.{tenantId}";
    }

    return null;
}
```

Call an API

After you have a token, you can call a protected web API. You do this from the controller of your web API.

Controller code:

The following code continues the example code that's shown in A web API that calls web APIs: Acquire a token for the app. The code is called in the actions of the API controllers. It calls a downstream API named `todolist`.

After you've acquired the token, use it as a bearer token to call the downstream API.

(ASP.NET Core)

```
private async Task GetTodoList(bool isAppStarting)
{
    ...
    // Get an access token to call the To Do service.
    //
    AuthenticationResult result = null;
    try
    {
```

```
app = BuildConfidentialClient(HttpContext, HttpContext.User);
result = await app.AcquireTokenSilent(Scopes, account)
    .ExecuteAsync()
    .ConfigureAwait(false);
}

...

// After the token has been returned by Microsoft Authentication Library
// (MSAL), add it to the HTTP authorization header before making the call to
// access the To Do list service.
_httpClient.DefaultRequestHeaders.Authorization = new AuthenticationHeaderValue("Bearer", result.AccessToken);

// Call the To Do list service.
HttpResponseMessage response = await _httpClient.GetAsync(TodoListBaseAd-
dress + "/api/todolist");
...
}
```

Lesson review questions

1. Which of the following parameters does not need to be defined when registering a protected web API with Microsoft identity?
 - (A) A resource URI
 - (B) One or more scopes
 - (C) A redirect URI
 - (D) One or more app roles
2. Which authentication flow can web APIs use to call downstream APIs using delegated access tokens?
 - (A) Implicit grant flow
 - (B) Client credentials flow
 - (C) On-behalf-of flow

Correct/suggested answers:

1. (C) - Web APIs don't need to register a redirect URI because no user is interactively signed in.
2. (C)

Module 2 Work with Microsoft Graph

Overview of Microsoft Graph

Lesson introduction

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 Office 365, Windows 10, and Enterprise Mobility + Security. You can use the wealth of data in Microsoft Graph to build apps for organizations and consumers that interact with millions of users.

After this lesson, you should be able to:

- Describe Microsoft Graph.
- Describe benefits of using the Microsoft Graph native SDKs.
- Describe how to work with the Microsoft Graph API.

Overview of Microsoft Graph

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

Microsoft Graph and the Microsoft 365 platform

Microsoft Graph powers 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>¹, to provide access to rich, people-centric data and insights exposed as resources of Microsoft 365 services. You can use REST APIs or SDKs to access the endpoint and build apps that support scenarios spanning across productivity, collaboration, education, security, identity, access, device management, and much more.

¹ <https://graph.microsoft.com>

- **Microsoft Graph data connect²** provides a set of tools to streamline secure and scalable delivery of Microsoft Graph data to popular Azure data stores. This cached data serves as data sources for Azure development tools that you can use to build intelligent applications.
- **Microsoft Graph connectors** (private preview) works in the incoming direction, delivering external data into Microsoft Graph services and applications and enhancing custom experiences.

With the ability to access Microsoft Graph data and other datasets to derive insights and analytics, you can extend Microsoft 365 experiences, or build unique, intelligent applications.

What's in Microsoft Graph?

Microsoft Graph exposes REST APIs and client libraries that allow you to access data from the following Microsoft 365 services:

- Office 365 services: Delve, Excel, Microsoft Bookings, Microsoft Teams, OneDrive, OneNote, Outlook/Exchange, Planner, and SharePoint
- Enterprise Mobility and Security services: Advanced Threat Analytics, Advanced Threat Protection, Azure Active Directory (Azure AD), Identity Manager, and Intune
- Windows 10 services: activities, devices, notifications
- Dynamics 365 Business Central

What can you do with Microsoft Graph?

Use Microsoft Graph to build experiences around the user's unique context to help them be more productive. For example, you could leverage Microsoft Graph to build an app that...

- Looks at a user's next meeting and helps them prepare for it by providing profile information for attendees, including their job titles and managers, as well as information about the latest documents they're working on and people they're collaborating with.
- Scans a user's calendar and suggests the best times for the next team meeting.
- Fetches the latest sales projection chart from an Excel file in a user's OneDrive and lets them update the forecast in real time, all from their phone.
- Subscribes to changes in users' calendars, sends them an alert when they are spending too much time in meetings, and provides recommendations for the ones they can miss or delegate based on how relevant the attendees are to them.
- Helps users sort out personal and work information on their phone; for example, by categorizing pictures that should go to their personal OneDrive and business receipts that should go to your OneDrive for Business.
- Analyzes at-scale Office 365 data so that decision makers can unlock valuable insights into time allocation and collaboration patterns that improve business productivity.
- Brings custom business data into Microsoft Graph, indexing it to make it searchable along with data from Microsoft 365 services.

Consider the first scenario about researching meeting attendees as an example. With the Microsoft Graph API, you can:

1. Get the email addresses of the **meeting event³** attendees.

² <https://docs.microsoft.com/en-us/graph/overview#access-microsoft-graph-data-at-scale-using-microsoft-graph-data-connect>

³ <https://docs.microsoft.com/en-us/graph/api/resources/event?view=graph-rest-1.0>

2. Look them up individually as a **user**⁴ in Azure Active Directory to **get their profile information**⁵.
3. You can then navigate to other resources using relationships:
 - Connect to their manager through a manager relationship.
 - Get valuable insights and intelligence including the popular files trending around the user.
 - Get the most relevant people to the user.
 - Extend the scenario to get to the user's groups through a **memberOf**⁶ relationship.
 - Reach other members in each group.
 - Tap into other scenarios enabled by groups, such as **education**⁷ and **teamwork**⁸.

Microsoft Graph continues to open up the Microsoft 365 platform for developers, and always only with the appropriate permissions.

Access the Microsoft Graph

As a developer, you can create all sorts of applications that will communicate with Microsoft Graph. To support as many developers and platforms as possible, Microsoft Graph has two options for you to choose from when integrating Microsoft Graph into your applications.

Use the Microsoft Graph REST API

At its core, Microsoft Graph is a REST API. This means that you can use any platform, any framework, and any programming language you're most comfortable with. The only requirement is that you issue common HTTP requests and process HTTP responses. All modern platforms, frameworks, and languages have these capabilities.

Use Microsoft Graph native SDKs

Microsoft Graph also provides multiple native SDKs for developers who want to use a rich programming model within their applications. The SDKs are designed to simplify building high-quality, efficient, and resilient applications that access Microsoft Graph and abstract away the tasks related to constructing and processing REST requests over HTTP.

Microsoft Graph native SDKs are available to be included in your projects via GitHub and popular platform package managers.

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.

⁴ <https://docs.microsoft.com/en-us/graph/api/resources/user?view=graph-rest-1.0>

⁵ <https://docs.microsoft.com/en-us/graph/api/user-get?view=graph-rest-1.0>

⁶ <https://docs.microsoft.com/en-us/graph/api/user-list-memberof?view=graph-rest-1.0>

⁷ <https://docs.microsoft.com/en-us/graph/education-concept-overview>

⁸ <https://docs.microsoft.com/en-us/graph/teams-concept-overview>

SDKs are currently available for the following languages and platforms:

- Android
- Angular
- ASP.NET
- iOS
- JavaScript
- Node.js
- Java
- PHP
- Python
- Ruby

Authentication options

Microsoft Graph supports two styles of authentication: **Azure AD accounts**, also known as work or school accounts, and **Microsoft accounts**. Azure AD accounts are typically used to access content and resources within Office 365 and Microsoft 365. Microsoft accounts are used to access consumer services such as OneDrive Consumer, Outlook.com, and other related services.

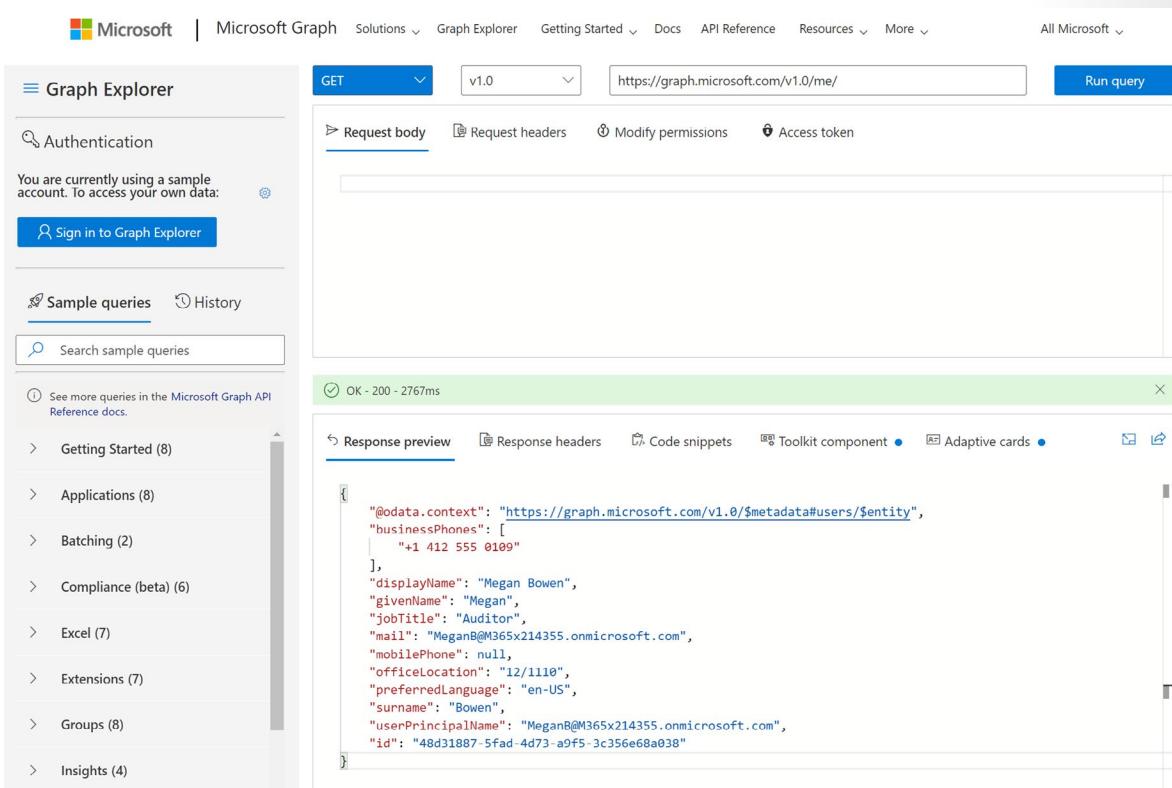
Because Microsoft Graph supports both styles of authentication, the same API and endpoints can be used to create applications that will expose business data in Microsoft 365 or consumer data within Microsoft consumer services. Native support for different authentication options makes it easy for you to learn a single API and means that you can configure your application to support both business and consumer data that is driven strictly by the user and the type of account they sign in with.

Work with the Microsoft Graph API

Microsoft Graph Explorer

The Microsoft Graph Explorer is a tool that lets developers make requests and see responses against the Microsoft Graph: **Microsoft Graph Explorer**⁹. You can use this tool to quickly navigate and test API endpoints.

⁹ <https://developer.microsoft.com/en-us/graph/graph-explorer>



Use the Microsoft Graph API

To read from or write to a resource such as a user or an email message, you construct a RESTful API request as follows:

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

- **{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**.

The key to working with Microsoft Graph is becoming familiar with the language and structure of requests. In this module, you will utilize various types of requests to Microsoft Graph through the Microsoft Graph Explorer.

HTTP method

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 you might introduce breaking changes to our beta APIs, it is recommended 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**.

Throughout this module, you will be exposed to the use of different types of resources, and you will learn to distinguish between entities, relationships, and actions.

Query parameters

Query parameters can be OData system query options or other strings that a method accepts to customize its response. These allow you to limit the number of results returned in a response, filter results, or search for data. An overview of query parameters will be provided later in this module.

Lesson review questions

1. What is the Microsoft Graph API?
2. What protocol is used to make a request to Microsoft Graph?

Correct/suggested answers:

1. It's a way to programmatically access the data from Office 365, Windows 10, and Enterprise Mobility + Security.
2. HTTPS

Optimize Data Usage with Query Parameters

Lesson introduction

Microsoft Graph supports optional query parameters that you can use to specify and control the amount of data returned in a response. While OData is supported and should be used to make queries, the behavior may change between resources. Support for the exact query parameters varies from one API operation to another, and depending on the API, can differ between the v1.0 and beta endpoints.

In this lesson, you'll look at supported query parameters and use Microsoft Graph Explorer to practice running queries.

After this lesson, you should be able to:

- Use the **\$select** query parameter.
- Order results using the **\$orderby** query parameter.
- Set the page size of results using the **\$skip** and **\$top** query parameters.
- Expand and retrieve resources using the **\$expand** query parameter.
- Retrieve the total count of matching resources using the **\$count** query parameter.
- Search for resources using the **\$search** query parameter.
- Filter results with the **\$filter** query parameter.

Introduction to query parameters

Microsoft Graph APIs implement many of the OData protocol's query parameters. Query parameters help you achieve multiple tasks, such as limiting the amount of data returned by requests to Microsoft Graph, searching for information and filtering results. For example, you can use query parameters to control how many fields are returned for each record and how many records are returned for a request.

Query parameters are added to the query string portion of a request URL and have a \$ prefix. Each parameter has a different syntax for usage. For example, the **\$count** query parameter, when set to true, will tell Microsoft Graph to report the total number of items in the collection regardless of how many items are returned by the request.

Note:

OData 4.0 supports system query options only in GET operations.

Microsoft Graph uses a subset of the OData v4 query language, and not all query parameters are supported on all resources. A Microsoft Graph API operation might support one or more of the following OData system query options:

Name	Description	Example
\$count	Retrieves the total count of matching resources.	<code>/me/messages?\$top=2&\$count=true</code>
\$expand	Retrieves related resources.	<code>/groups?\$expand=members</code>
\$filter	Filters results (rows).	<code>/users?\$filter=startswith(givenName,'J')</code>
\$format	Returns the results in the specified media format.	<code>/users?\$format=json</code>

Name	Description	Example
\$orderby	Orders results.	/users?\$orderby=displayName desc
\$search	Returns results based on search criteria. Currently supported on messages and person collections.	/me/messages?\$search=pizza
\$select	Filters properties (columns).	/users?\$select=givenName,surname
\$skip	Indexes into a result set. Also used by some APIs to implement paging and can be used together with \$top to manually page results.	/me/messages?\$skip=11
\$top	Sets the page size of results.	/users?\$top=2

You can see examples of these parameters here: [Use query parameters to customize responses¹⁰](#).

Other query parameters

A Microsoft Graph API operation might also support the following non-OData system query option.

Name	Description	Example
\$skipToken	Retrieves the next page of results from result sets that span multiple pages. (Some APIs use \$skip instead.)	/users?\$skipto-

The **\$select** query parameter

\$select allows you to only fetch the data required for each record, reducing the volume of network traffic generated by an application.

You can also use the **\$select** query parameter to return a set of properties that are different than the default set for an individual resource or a collection of resources. With **\$select**, you can specify a subset or a superset of the default properties.

Providing a comma-separated list of fields as the value to **\$select** controls the fields returned for a given query.

For example, when retrieving the messages of the signed-in user, you can specify that only the **from** and **subject** properties be returned:

```
GET https://graph.microsoft.com/v1.0/me/messages?$select=from,subject
```

In general, you should use **\$select** to limit the properties returned by a query to those needed by your app. This is especially true of queries that might potentially return a large result set. Limiting the properties returned in each row reduces network load and help improve your app's performance.

Note:

In v1.0, some Azure Active Directory (Azure AD) resources that derive from **directoryObject**, such as **user**

¹⁰ <https://docs.microsoft.com/en-us/graph/query-parameters>

and **group**, return a limited, default subset of properties on reads. For these resources, you must use **\$select** to return properties outside of the default set.

The **\$orderby** query parameter

Applications often need to fetch results in a specific order. Using the **\$orderby** parameter and performing the ordering on the server is more efficient than fetching all results and ordering on the client.

Use the **\$orderby** query parameter to specify the sort order of the items returned from Microsoft Graph. Provide a comma-separated list of fields to order, and, optionally, add an ordering argument. By default data is returned in ascending order, but you can add the **desc** keyword after the property to sort in descending order.

For example, the following request returns the users in the organization ordered by their display name:

```
GET https://graph.microsoft.com/v1.0/users?$orderby=displayName
```

You can also sort by complex type entities. The following request gets messages and sorts them by the address field of the from property, which is of the complex type **emailAddress**:

```
GET https://graph.microsoft.com/v1.0/me/messages?$orderby=from/emailAddress/address
```

Sorting can also use complex object properties. To sort the results in ascending or descending order, append either **asc** or **desc** to the field name, separated by a space; for example: **?\$orderby=name%20desc**.

With some APIs, you can order results on multiple properties. For example, the following request orders the messages in the user's Inbox, first by the name of the person who sent it in descending order (Z to A), and then by subject in ascending order (default):

```
GET https://graph.microsoft.com/v1.0/me/mailFolders/Inbox/messages?$orderby=from/emailAddress/name desc,subject
```

The **\$skip** and **\$top** query parameters

The **\$skip** and **\$top** query parameters allow you to set the number of results returned from a request.

Use **\$top** to control the number of results and **\$skip** to control where in a given result set the data should be selected from.

The **\$top** query parameter

Controlling the number of results returned in a given request allows you to page through large data sets. Use the **\$top** query parameter to specify the page size of the result set, or how many results are returned in a response.

For example, the following request returns the first five messages in the user's mailbox:

```
GET https://graph.microsoft.com/v1.0/me/messages?$top=5
```

Request paged results

The number of results specified with the **\$top** parameter defines the page size of the result set. When a result set spans multiple pages, Microsoft Graph returns an `@odata.nextLink` property in the response that contains a URL to the next page of results.

For example, the following URL requests the first 5 of all users in the organization.

[https://graph.microsoft.com/v1.0/users?\\$.top=5](https://graph.microsoft.com/v1.0/users?$.top=5)

If the result contains more than 5 users, Microsoft Graph will return an `@odata:nextLink` property similar to the following along with the first page of users.

"@odata.nextLink": "https://graph.microsoft.com/v1.0/users?\$top=5&\$skiptoken=X%274453707 ... 6633B900000000000000000000000000%27"

You can retrieve the next page of results by sending the URL value of the `@odata:nextLink` property to Microsoft Graph:

Microsoft Graph will continue to return a reference to the next page of data in the `@odata:nextLink` property with each response until all pages of the result have been read.

The `$skip` query parameter

Use the `$skip` query parameter to set the number of items to skip at the start of a collection.

For example, the following request returns events for the user sorted by date created, starting with the 21st event in the collection:

```
GET https://graph.microsoft.com/v1.0/me/events?$orderby=createdDateTime&$skip=20
```

The \$expand query parameter

The **\$expand** query parameter fetches additional resources related to the requested resource. You can use this query to fetch a number of related resources in a single request by providing a comma-separated list of properties to be expanded and included in the results for the request.

Many Microsoft Graph resources expose both declared properties of the resource and its relationships with other resources. These relationships are also called reference properties or navigation properties, and they can reference either a single resource or a collection of resources. For example, the mail folders, manager, and direct reports of a user are all exposed as relationships.

Normally, you can query either the properties of a resource or one of its relationships in a single request, but not both. You can use the **\$expand** query string parameter to include the expanded resource or collection referenced by a single relationship (navigation property) in your results.

The following example gets root drive information along with the top-level child items in a drive:

```
GET https://graph.microsoft.com/v1.0/me/drive/root?$expand=children
```

Adding the \$select query parameter

With some resource collections, you can also specify the properties to be returned in the expanded resources by adding a **\$select** parameter.

The following example performs the same query as the previous example but uses a **\$select** statement to limit the properties returned for the expanded child items to the id and name properties:

```
GET https://graph.microsoft.com/v1.0/me/drive/root?$expand=children($select=id,name)
```

Note:

Combining **\$select** with **\$expand** helps reduce the amount of data being transferred between your application and the Microsoft Graph.

Not all relationships and resources support the **\$expand** query parameter. Also, not all resources or relationships support using **\$select** on expanded items.

For example, you can expand the **directReports**, **manager**, and **memberOf** relationships on a user, but you cannot expand its events, messages, or photo relationships.

The \$count query parameter

There are many scenarios in which an application needs to know how many results could be returned without fetching all results. Adding **\$count=true** as a query string parameter includes a count of the total number of items in a collection alongside the page of data values returned from Microsoft Graph.

For example, the following request returns both the contact collection of the current user and the number of items in the contact collection in the **@odata.count** property:

```
GET https://graph.microsoft.com/v1.0/me/contacts?$count=true
```

Note:

\$count is not supported for collections of resources that derive from **directoryObject**, such as collections of users or groups.

The \$search query parameter

Search is a powerful means of finding content and is less restrictive than **\$filter** expressions. **\$search** is a means of doing full-text searches on some resources.

Use the **\$search** query parameter to restrict the results of a request to match a search criterion. Search also supports Keyword Query Language (KQL) structured queries that allow you to target searches to specific fields.

Note:

You can currently search only message and person collections. A **\$search** request returns up to 250 results. You cannot use **\$filter** or **\$orderby** in a search request.

Use \$search on message collections

You can search messages based on a value in specific message properties. The results of the search are sorted by the date and time that the message was sent.

If you do a search on messages and specify only a value without specific message properties, the search is carried out on the default search properties of from, subject, and body.

The following example returns all messages in the signed-in user's Inbox that contains *pizza* in any of the three default search properties:

```
GET https://graph.microsoft.com/v1.0/me/messages?$search="pizza"
```

Use \$search on person collections

You can use the Microsoft Graph People API to retrieve the people who are most relevant to a user. Relevance is determined by the user's communication and collaboration patterns and business relationships. The People API supports the **\$search** query parameter. Searches on people occur on both the **displayName** and **emailAddress** properties of the person resource.

The following request does a search for a person named *Irene McGowen* in the **displayName** and **emailAddress** properties in each person in the people collection of the signed-in user. Because a person named *Irene McGowan* is relevant to the signed-in user, the information for *Irene McGowan* is returned.

```
GET https://graph.microsoft.com/v1.0/me/people/?$search="Irene McGowen"
```

The \$filter query parameter

The **\$filter** query parameter helps you find resources that match a specified query. **\$filter** allows you to only fetch the required records from Microsoft Graph. Use the **\$filter** query parameter to retrieve just a subset of a collection.

For example, to find users whose display name starts with the letter 'J', use **startswith**:

```
GET https://graph.microsoft.com/v1.0/users?$filter=startswith(displayName, 'J')
```

Support for **\$filter** operators varies across Microsoft Graph APIs. The following logical operators are generally supported:

- equals (**eq**)
- not equals (**ne**)
- greater than (**gt**)
- greater than or equals (**ge**)
- less than (**lt**), less than or equals (**le**)
- and (**and**)
- or (**or**)
- not (**not**)

Note: Many operations are supported for filtering. The **startswith** function and any lambda, while supported, may not actually do any filtering when supplied on an API that doesn't support them, or may result in an error in some cases.

Examples

The following request gets all unread mail in the signed-in user's inbox, using the **equals** logical operator:

```
GET https://graph.microsoft.com/v1.0/me/mailFolders/inbox/messages?$filter=isRead eq false
```

The following request gets all the signed-in user's events that start after 7/1/2017, using the **greater than or equals** logical operator:

```
https://graph.microsoft.com/v1.0/me/events?$filter=startDateTime ge '2017-07-01T08:00'
```

Lesson review questions

1. What are optional query parameters used for?
2. Which query parameter is used to control page size?
3. When might you use the \$expand query parameter?

Correct/suggested answers:

1. To control the amount of data returned in a response, filter data returned in a response, or search for data.
2. The \$top query parameter limits the number of results returned per page.
3. This query can be used to fetch a number of related resources in a single request.

Optimize Network Traffic

Lesson introduction

Microsoft Graph provides a unified programmability model that developers can use to build apps for organizations and consumers that interact with the data of millions of users.

In this lesson, you'll learn how Microsoft has implemented throttling to Microsoft Graph to limit the overuse of Microsoft Graph resources. You'll learn how to avoid throttled requests, as well as how to properly handle scenarios when Microsoft Graph throttles high user traffic in a graceful way, using techniques like change notifications (webhooks), batching, and delta query. You'll also learn how to handle throttling when it occurs.

After this lesson, you should be able to:

- Monitor for changes using change notifications.
- Use **\$batch** to combine multiple requests.
- Get changes using a delta query.
- Implement error 429 handler.

Receive change notifications

Change notifications allow you to build applications that respond to changes in data within Microsoft Graph. If your application needs to know about changes to data, you can get a webhook notification whenever data of interest has changed. This is more efficient than simply polling on a regular basis.

You register a URL for Microsoft Graph to send change notifications for a specified resource or set of resources. When a change notification is received:

- The ID is used to fetch additional information about the changed item.
- Any additional steps are performed as dictated by the behavior of the client application.
- An **HTTP 202 – Accepted** is sent.

The Microsoft Graph API uses a webhook mechanism to deliver notifications to clients. A client is a web service that configures its own URL to receive notifications. Client apps use notifications to update their state upon changes.

After Microsoft Graph accepts the subscription request, it pushes notifications to the URL specified in the subscription. The app then acts according to its business logic. For example, it fetches more data, updates its cache and views, and so on.

Supported resources

Using the Microsoft Graph API, an app can subscribe to changes on the following resources:

- Outlook message
- Outlook event
- Outlook personal contact
- User
- Group
- Office 365 group conversation

- Content within the hierarchy of any folder **driveItem** on a user's personal OneDrive
- Content within the hierarchy of the root folder **driveItem** on OneDrive for Business
- Security alert

You can create a subscription to:

- A specific Outlook folder such as the Inbox: **me/mailFolders('inbox')/messages**
- A top-level resource: **me/messages**, **me/contacts**, **me/events**, **users**, or **groups**
- A specific resource instance: **users/{id}**, **groups/{id}**, **groups/{id}/conversations**
- Any folder in a user's personal OneDrive: **/drives/{id}/root** /**drives/{id}/root/subfolder**
- The root folder of a SharePoint/OneDrive for Business drive: **/drive/root**
- A new Security API alert: **/security/alerts?\$filter=status eq 'New'**, **/security/alerts?\$filter=vendorInformation/provider eq 'ASC'**

Manage subscriptions

Clients can create subscriptions, renew subscriptions, and delete subscriptions.

Create a subscription

Creating a subscription is the first step to start receiving notifications for a resource. The subscription process is as follows:

1. The client sends a subscription (**POST**) request for a specific resource.
2. Microsoft Graph verifies the request.
 - If the request is valid, Microsoft Graph sends a validation token to the notification URL.
 - If the request is invalid, Microsoft Graph sends an error response with code and details.
3. The client sends the validation token back to Microsoft Graph.
4. Microsoft Graph sends a response back to the client.

The client must store the subscription ID to correlate notifications with the subscription.

Subscription request example

```
POST https://graph.microsoft.com/v1.0/subscriptions
Content-Type: application/json
{
    "changeType": "created,updated",
    "notificationUrl": "https://webhook.azurewebsites.net/notificationClient",
    "resource": "/me/mailfolders('inbox')/messages",
    "expirationDateTime": "2016-03-20T11:00:00.000000Z",
    "clientState": "SecretClientState"
}
```

The **changeType**, **notificationUrl**, **resource**, and **expirationDateTime** properties are required.

The **resource** property specifies the resource that will be monitored for changes. For example, you can create a subscription to a specific mail folder: **me/mailfolders('inbox')/messages** or on behalf of a user given by an administrator consent: **users/john.doe@onmicrosoft.com/mailfolders('inbox')/messages**.

Although **clientState** is not required, you must include it to comply with Microsoft's recommended notification handling process. Setting this property allows you to confirm that notifications you receive originate from the Microsoft Graph service. For this reason, the value of the property should remain secret and known only to your application and the Microsoft Graph service.

If successful, Microsoft Graph returns a **201 Created** code and a subscription object in the body.

Notification endpoint validation

Microsoft Graph validates the notification endpoint provided in the **notificationUrl** property of the subscription request before creating the subscription. The validation process occurs as follows:

1. Microsoft Graph sends a **POST** request to the notification URL:

```
POST https://[notificationUrl]?validationToken={opaqueTokenCreatedByMicrosoftGraph}
```

Note:

Because the **validationToken** is a query parameter, it must be properly decoded by the client, per HTTP coding practices. If the client does not decode the token, and instead uses the encoded value in the next step (response), validation will fail. Also, the client should treat the token value as opaque because the token format may change in the future without notice.

2. The client must provide a response with the following characteristics within 10 seconds:
 - A **200 (OK)** status code.
 - The content type must be `text/plain`.
 - The body must include the validation token provided by Microsoft Graph.
 - The client should discard the validation token after providing it in the response.

Renew a subscription

The client can renew a subscription with a specific expiration date of up to three days from the time of request. The **expirationDateTime** property is required.

Subscription renewal example

```
PATCH https://graph.microsoft.com/v1.0/subscriptions/{id}
Content-Type: application/json
{
    "expirationDateTime": "2016-03-22T11:00:00.000000Z"
}
```

If successful, Microsoft Graph returns a **200 OK** code and a subscription object in the body. The subscription object includes the new **expirationDateTime** value.

Because Microsoft Graph requires relatively short-lived subscriptions, renewal of subscriptions must be managed by client applications.

Delete a subscription

The client can stop receiving notifications by deleting the subscription using its ID.

```
DELETE https://graph.microsoft.com/v1.0/subscriptions/{id}
```

If successful, Microsoft Graph returns a 204 No Content code.

Notifications

The client starts receiving notifications after creating the subscription. Microsoft Graph sends a POST request to the notification URL when the resource changes. Notifications are sent only for the changes of the type specified in the subscription; for example, **created**.

Note:

When using multiple subscriptions that monitor the same resource type and use the same notification URL, a **POST** can be sent that contains multiple notifications with different subscription IDs. There is no guarantee that all notifications in the **POST** will belong to a single subscription.

Notification properties

The notification object has the following properties.

Property	Type	Description
subscriptionId	string	The ID of the subscription that generated the notification.
subscriptionExpirationDate-Time	dateTime	The expiration time for the subscription.
clientState	string	The clientState property specified in the subscription request (if any).
changeType	string	The event type that caused the notification. For example, created on mail receive, or updated on marking a message read.
resource	string	The URI of the resource relative to https://graph.microsoft.com (https://graph.microsoft.com).
resourceData	object	The content of this property depends on the type of resource being subscribed to.

For example, for Outlook resources, **resourceData** contains the following fields.

Property	Type	Description
@odata.type	string	The OData entity type in Microsoft Graph that describes the represented object.
@odata.id	string	The OData identifier of the object.

Property	Type	Description
@odata.etag	string	The HTTP entity tag that represents the version of the object.
id	string	The identifier of the object.

Note:

The id value provided in **resourceData** is valid at the time the notification was generated. Some actions, such as moving a message to another folder, may result in the id no longer being valid when the notification is processed.

Notification example

When the user receives an email, Microsoft Graph sends a notification like the following:

```
{
  "value": [
    {
      "subscriptionId": "<subscription_guid>",
      "subscriptionExpirationDateTime": "2016-03-19T22:11:09.952Z",
      "clientState": "secretClientValue",
      "changeType": "created",
      "resource": "users/{user_guid}@<tenant_guid>/messages/{long_id_string}",
      "resourceData": {
        "@odata.type": "#Microsoft.Graph.Message",
        "@odata.id": "Users/{user_guid}@<tenant_guid>/Messages/{long_id_string}",
        "@odata.etag": "W/\\"CQAAABYAAADkrWGo7bouTKlsgTZMr9KwAAU-WRHf\\\"",
        "id": "<long_id_string>"
      }
    }
  ]
}
```

Note that the value field is an array of objects. When there are many queued notifications, Microsoft Graph may send multiple items in a single request. Notifications from different subscriptions can be included in the same notification request.

Process the notification

Each notification received by your app should be processed. The following are the minimum tasks that your app must perform to process a notification:

- Send a **202 - Accepted** status code in your response to Microsoft Graph. If Microsoft Graph doesn't receive a **2xx** class code, it will try publishing the notification several times, for a period of about 4 hours; after that, the notification will be dropped and won't be delivered.

Note:

Send a **202 - Accepted** status code as soon as you receive the notification, even before validating its authenticity. You are simply acknowledging the receipt of the notification and preventing unnecessary

retries. The current timeout is 30 seconds, but it might be reduced in the future to optimize service performance.

- Validate the **clientState** property. It must match the value originally submitted with the subscription creation request.

Note:

If this isn't true, you should not consider this a valid notification. It is possible that the notification has not originated from Microsoft Graph and may have been sent by a rogue actor. You should also investigate where the notification comes from and take appropriate action.

- Update your application based on your business logic.
- Repeat for other notifications in the request.

Once registered successfully, a subscription sends notifications to the URL provided when creating the subscription. These must be processed in a timely fashion and send an HTTP 202 response to acknowledge successful receipt of the notifications.

Perform batch requests

\$batch provides a means of combining multiple Microsoft Graph requests into a single HTTP request, allowing you to reduce the number of HTTP calls an application needs to make.

Creating a **\$batch** request requires a JSON array of requests that you then **POST** to **\$batch**. You use **dependsOn** to specify dependencies between requests in a batch.

JSON batching allows you to optimize your application by combining multiple requests into a single JSON object. For example, a client might want to compose a view of unrelated data such as:

- An image stored in OneDrive
- A list of Planner tasks
- The calendar for a group

Combining these three individual requests into a single batch request can save the application significant network latency. Using batching to reduce the impact of network latency can improve the efficiency of your client application.

JSON batch request

First you construct the JSON batch request for the previous example. In this scenario, the individual requests are not interdependent in any way and therefore can be placed into the batch request in any order:

```
POST https://graph.microsoft.com/v1.0/$batch
Accept: application/json
Content-Type: application/json

{
  "requests": [
    {
      "id": "1",
      "method": "GET",
      "url": "/me/drive/root:{file}:content"
    },
  ]
```

```
{  
    "id": "2",  
    "method": "GET",  
    "url": "/me/planner/tasks"  
},  
{  
    "id": "3",  
    "method": "GET",  
    "url": "/groups/{id}/events"  
},  
{  
    "id": "4",  
    "url": "/me",  
    "method": "PATCH",  
    "body": {  
        "city" : "Redmond"  
    },  
    "headers": {  
        "Content-Type": "application/json"  
    }  
}  
]  
}
```

Responses to the batched requests might appear in a different order. The id property can be used to correlate individual requests and responses:

```
200 OK  
Content-Type: application/json  
  
{  
    "responses": [  
        {  
            "id": "1",  
            "status": 302,  
            "headers": {  
                "location": "https://b0mpua-by3301.files.1drv.com/  
y23vmagahszhxzlcvhasdhasghasodfi"  
            }  
        },  
        {  
            "id": "3",  
            "status": 401,  
            "body": {  
                "error": {  
                    "code": "Forbidden",  
                    "message": "..."  
                }  
            }  
        },  
        {  
            "id": "2",  
            "status": 200,  
            "body": {  
                "id": "1",  
                "method": "GET",  
                "url": "/me/planner/tasks"  
            }  
        }  
    ]  
}
```

```
        "id": "2",
        "status": 200,
        "body": {
            "@odata.context": "https://graph.microsoft.com/v1.0/$meta-
data#Collection(microsoft.graph.plannerTask)",
            "value": []
        }
    },
{
    "id": "4",
    "status": 204,
    "body": null
}
]
```

Sequencing requests with the dependsOn property

Individual requests can be executed in a specified order by using the **dependsOn** property. This property is an array of strings that reference the id of a different individual request. For this reason, the values for id must be unique. For example, in the following request, the client is specifying that requests 1 and 3 should be run first, then request 2, then request 4.

If an individual request fails, any request that depends on that request fails with status code **424 (Failed Dependency)**.

Note:

Using the **dependsOn** property can allow client applications to be more responsive by eliminating multiple round trips to Microsoft Graph.

```
{
    "requests": [
        {
            "id": "1",
            "method": "GET",
            "url": "..."
        },
        {
            "id": "2",
            "dependsOn": [ "1" ],
            "method": "GET",
            "url": "..."
        },
        {
            "id": "3",
            "method": "GET",
            "url": "..."
        },
        {
            "id": "4",
            "dependsOn": [ "2" ],
            "method": "GET",
            "url": "..."
        }
    ]
}
```

```
        "url": "..."  
    }  
]  
}
```

Get changes using a delta query

Using delta query to get changes is a means of polling Microsoft Graph for changes to some sets of data. This provides an efficient mechanism for querying changes in data.

Follow this high-level process:

1. Set up a delta query.
2. Follow the **nextLink** until there are no results returned.
3. Store the last value of the **nextLink** returned.
4. Return to step 1 using the stored **nextLink** value.

Delta query enables applications to discover newly created, updated, or deleted entities without performing a full read of the target resource with every request. Microsoft Graph applications can use delta query to efficiently synchronize changes with a local data store.

The typical call pattern is as follows:

1. The application begins by calling a **GET** request with the delta function on the desired resource.
2. Microsoft Graph sends a response containing the requested resource and a state token.
 - If a **nextLink** URL is returned, there may be additional pages of data to be retrieved in the session. The application continues making requests using the **nextLink** URL to retrieve all pages of data until a **deltaLink** URL is returned in the response.
 - If a **deltaLink** URL is returned, there is no more data about the existing state of the resource to be returned. For future requests, the application uses the **deltaLink** URL to learn about changes to the resource.
3. When the application needs to learn about changes to the resource, it makes a new request using the **deltaLink** URL received in step 2. This request may be made immediately after completing step 2 or when the application checks for changes.
4. Microsoft Graph returns a response describing changes to the resource since the previous request, and either a **nextLink** URL or a **deltaLink** URL.

Optional query parameters

If a client uses a query parameter, it must be specified in the initial request. Microsoft Graph automatically encodes the specified parameter into the **nextLink** or **deltaLink** provided in the response. The calling application only needs to specify the query parameters once upfront. Microsoft Graph adds the specified parameters automatically for all subsequent requests.

Note the following regarding optional query parameters:

- **\$orderby** is not supported for delta queries. Do not assume a specific sequence of the responses returned from a delta query. Assume that the same item can show up anywhere in the **nextLink** sequence and handle that in your merge logic.

- **\$top** is not supported for delta queries, and the number of objects in each page can vary depending on the resource type and the type of changes made to the resource.

For users and groups, there are restrictions on using some query parameters:

- If a **\$select** query parameter is used, the parameter indicates that the client prefers to only track changes on the properties or relationships specified in the **\$select** statement. If a change occurs to a property that is not selected, the resource for which that property changed does not appear in the delta response after a subsequent request.
- **\$expand** is only supported for the manager and members navigational property for users and groups, respectively.
- Scoping filters allow you to track changes to one or more specific users or groups by object ID. For example, the following request returns changes for the groups matching the IDs specified in the query filter.
- Once a delta query is established, the client need not track the query parameters used to create it to make subsequent calls with those parameters.

Note:

There are some limitations on the query parameters available, particularly **\$orderby**, **\$select**, and **\$expand**.

State tokens

A delta query **GET** response always includes a URL specified in a **nextLink** or **deltaLink** response header. **The nextLink URL includes a skipToken, and a deltaLink URL includes a deltaToken.** These tokens are opaque to the client.

The following details are important:

- Each token reflects the state and represents a snapshot of the resource in that round of change tracking.
- The state tokens also encode and include other query parameters (such as **\$select**) specified in the initial delta query request. Therefore, it's not required to repeat them in subsequent delta query requests.
- When carrying out delta query, you can copy and apply the **nextLink** or **deltaLink** URL to the next delta function call without having to inspect the contents of the URL, including its state token.
- Clients must store the **deltaLink** or **nextLink** in order to receive only the changed data between usages of a delta query.

Handle throttling

In many scenarios, you may create applications that make a lot of calls or several expensive calls to Microsoft Graph. This can cause throttling. When this occurs, it is important that the application stop applying load to Microsoft Graph until it is ready to receive traffic again.

Throttling limits the number of concurrent calls to a service to prevent overuse of resources. Microsoft Graph is designed to handle a high volume of requests. If an overwhelming number of requests occurs, throttling helps maintain optimal performance and reliability of the Microsoft Graph service.

Implementing error 429 is a strategy for handling periods where an application exceeds throttling limits. When receiving an HTTP Status code of 429, the client application should stop sending requests for the period specified in the Retry-After header.

What happens when throttling occurs?

Throttling limits vary based on the scenario. For example, if you are performing a large volume of writes, the possibility for throttling is higher than if you are only performing reads.

When a throttling threshold is exceeded, Microsoft Graph limits any further requests from that client for a period of time. When throttling occurs, Microsoft Graph returns HTTP status code 429 (Too many requests), and the requests fail.

A suggested wait time is returned in the response header of the failed request. Throttling behavior can depend on the type and number of requests.

For example, if you have a high volume of requests, all requests types are throttled. Threshold limits vary based on the request type. Therefore, you could encounter a scenario where writes are throttled but reads are still permitted.

Detect and handle throttling

The most common causes of throttling of clients include:

- A large number of requests across all applications in a tenant.
- A large number of requests from a particular application across all tenants.

The following are best practices for handling throttling:

- Reduce the number of operations per request.
- Reduce the frequency of calls.
- Avoid immediate retries, because all requests accrue against your usage limits.

When you implement error handling, use the HTTP error code 429 to detect throttling. The failed response includes the **Retry-After** response header.

Backing off requests using the **Retry-After** delay is the fastest way to recover from throttling because Microsoft Graph continues to log resource usage while a client is being throttled. Do the following:

- Wait the number of seconds specified in the **Retry-After** header.
- Retry the request.

If the request fails again with a 429-error code, you are still being throttled. Continue to use the recommended **Retry-After** delay and retry the request until it succeeds.

Client applications that fail to respond to HTTP 429 responses are likely to be continually throttled, while implementing a retry handler that uses the **Retry-After** response header is the fastest way to recover from having a client application exceeding throttling limits.

Not all resources are guaranteed to provide a **Retry-After** value. The following resources currently provide a **Retry-After** header:

- User
- Photo
- Mail
- Calendar (users and groups)
- Contact
- Attachment

- Group conversations
- People and social
- Drive (OneDrive)

Lesson review questions

1. What is Microsoft Graph throttling?
2. What happens when throttling occurs?
3. What are best practices for handling throttling?

Correct/suggested answers:

1. Throttling limits the number of concurrent calls to a service to prevent overuse of resources. Microsoft Graph is designed to handle a high volume of requests. If an overwhelming number of requests occurs, throttling helps maintain optimal performance and reliability of the Microsoft Graph service.
2. When a throttling threshold is exceeded, Microsoft Graph limits any further requests from that client for a period of time. When throttling occurs, Microsoft Graph returns HTTP status code 429 (Too many requests), and the requests fail. A suggested wait time is returned in the response header of the failed request. Throttling behavior can depend on the type and number of requests. For example, if you have a high volume of requests, all request types are throttled.
3. Best practices:
 - Reduce the number of operations per request.
 - Reduce the frequency of calls.
 - Avoid immediate retries, because all requests accrue against your usage limits.

Access User Data with Microsoft Graph

Lesson introduction

In this lesson, you will focus on user management capabilities exposed by Microsoft Graph, through its administration API for users in Azure Active Directory (Azure AD).

You will work with the Microsoft Graph Explorer to discover and practice accessing user data with Microsoft Graph.

After this lesson, you should be able to:

- Get the signed in user's profile.
- Get the user object based on the user's unique identifier.
- Get a list of users in the organization.
- Get the user's profile photo.
- Get the user's manager profile.

Work with users in Microsoft Graph

Users are the representation of an Azure AD work or school user account or a Microsoft account in Microsoft Graph. The **user** resource in Microsoft Graph is a hub from which you can access the relationships and resources that are relevant to your users, allowing you to develop user-centric applications.

Microsoft Graph enables developers full control over the lifecycle of users in Microsoft 365 including creating, updating, and deleting users in addition to listing users in the organization. Developers can use Microsoft Graph to access the relationships, documents, contacts, and preferences that are contextually relevant to the signed-in user. The **user** resource provides a straightforward way for you to access and manipulate user resources without having to perform additional calls, look up specific authentication information, and directly issue queries against other Microsoft Graph resources. Following are common scenarios to leverage Microsoft Graph user resource:

Manage your organization

You can create new users in your organization or update the resources and relationships for existing users. You can use Microsoft Graph to perform the following user management tasks:

- Create or delete users in your Azure AD organization.
- List a user's group memberships and determine whether a user is a member of a group.
- List the users who report to a user and assign managers to a user.
- Upload or retrieve a photo for the user.

Work with calendars and tasks

You can view, query, and update user calendar and calendar groups associated with a user, including:

- List and create events on a users calendar
- View tasks assigned to a user
- Find free meeting times for a set of users

- Get a list of reminders set on a user's calendar

Administer mail and handle contact

You can configure user mail settings and contact lists and send mail on a user's behalf, including

- List mail messages and send new mail
- Create and list user contacts and organize contacts in folders
- Retrieve and update mailbox folders and settings

Enrich your app with user insights

Maximize relevance in your application by promoting recently used or trending documents and contacts associated with a user. You can use Microsoft Graph to:

- Return documents recently viewed and modified by a user
- Return documents and sites trending around a user's activity
- List documents shared with a user through email or OneDrive for Business

Common user properties

You can access users through Microsoft Graph in two ways:

- By their ID: /users/{id} | userPrincipalName
- By using the alias for the signed-in user: /me (which is the same as /users/{signed-in user's id})

The following represent the default set of properties that are returned when getting a user or listing users. These are a subset of all available properties. To get more user properties, use the \$select query parameter.

Property	Description
id	The unique identifier for the user.
businessPhones	The user's phone numbers.
displayName	The name displayed in the address book for the user.
givenName	The first name of the user.
jobTitle	The user's job title.
mail	The user's email address.
mobilePhone	The user's cellphone number.
officeLocation	The user's physical office location.
preferredLanguage	The user's language of preference.
surname	The last name of the user.
userPrincipalName	The user's principal name.

Get the signed-in user's profile

Most applications need to know some information about the currently signed-in user. Meaning, your application needs to be able to retrieve properties and relationships of a user object.

In Microsoft Graph, using `/me` is a shortcut to the profile of the user to which the authentication token belongs.

Permissions

One of the following permissions is required to call the API.

HTTP request for the signed-in user

Getting information about the currently signed-in user with the `/me` token is quick and efficient:

```
GET /me
```

The following example illustrates the default request and response.

Request

Using HTTP:

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

Using C#:

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

Using JavaScript:

```
const options = {
    authProvider,
};

const client = Client.init(options);
let res = await client.api('/me')
    .get();
```

HTTP response

```
HTTP/1.1 200 OK
Content-type: application/json
Content-length: 491
{
    "businessPhones": [
        "businessPhones-value"
    ],
    "displayName": "displayName-value",
    "givenName": "givenName-value",
    "jobTitle": "jobTitle-value",
    "mail": "mail-value",
    "mobilePhone": "mobilePhone-value",
```

```
    "officeLocation": "officeLocation-value",
    "preferredLanguage": "preferredLanguage-value",
    "surname": "surname-value",
    "userPrincipalName": "userPrincipalName-value",
    "id": "id-value"
}
```

Get the user object based on the user's unique identifier

You can also get the user object or fetch information about users based on the user's unique identifier. You can use the user's **id** or **userPrincipalName**.

Note:

You can get the user information for the signed-in user by replacing `/users/{id | userPrincipalName}` with `/me`.

The user principal name is usually in the form **userName@domain**; for example, **meganb@contoso.onmicrosoft.com**.

You may also come up with the Id for the user when navigating from other associated objects, such as groups.

Permissions

One of the following permissions is required to call the API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	User.Read, User.ReadWrite, User.ReadBasic.All, User.Read.All, User.ReadWrite.All, Directory.Read.All, Directory.ReadWrite.All, Directory.AccessAsUser.All
Delegated (personal Microsoft account)	User.Read, User.ReadWrite
Application	User.Read.All, User.ReadWrite.All, Directory.Read.All, Directory.ReadWrite.All

HTTP request for a specific user

```
GET /users/{id | userPrincipalName}
```

By default, only a limited set of properties are returned (**businessPhones, displayName, givenName, id, jobTitle, mail, mobilePhone, officeLocation, preferredLanguage, surname, userPrincipalName**). To return an alternative property set, you must specify the desired set of user properties using the OData \$select query parameter.

The following example illustrates the request and response to return **displayName, givenName**, and **postalCode** by using the OData \$select query parameter.

HTTP request

```
GET https://graph.microsoft.com/v1.0/users/{id | userPrincipalName)?$select=displayName,givenName,postalCode
```

HTTP response

```
HTTP/1.1 200 OK
Content-type: application/json
Content-length: 491
{
    "displayName": "displayName-value",
    "givenName": "givenName-value",
    "postalCode": "postalCode-value"
}
```

Get a list of users in the organization

The **/users** request retrieves a list of user objects in the organization. For example, in a scenario where the app performs operations on resources that other users have shared with the signed-in user. Outlook resources like mail, calendars, and contacts. Accessing or reading mail in the mailbox of the signed-in user, as well as mail in mailboxes that other users in the organization have shared with the signed-in user.

Permissions

One of the following permissions is required to call this API. Personal Microsoft accounts are not supported because they are viewed as individual accounts that do not have a reporting structure or associated organization.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	User.ReadBasic.All, User.Read.All, User.ReadWrite.All, Directory.Read.All, Directory.ReadWrite.All, Directory.AccessAsUser.All
Delegated (personal Microsoft account)	Not supported
Application	User.Read.All, User.ReadWrite.All, Directory.Read.All, Directory.ReadWrite.All

HTTP request for a list of user objects

```
GET /users
```

The following example illustrates the default request and response.

Request

Using HTTP:

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

Using C#:

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var users = await graphClient.Users
    .Request()
    .GetAsync();
```

Using JavaScript:

```
const options = {
    authProvider,
};

const client = Client.init(options);
let res = await client.api('/users')
    .get();
```

HTTP response

```
HTTP/1.1 200 OK
Content-type: application/json
Content-length: 608
{
    "value": [
        {
            "businessPhones": [
                "businessPhones-value"
            ],
            "displayName": "displayName-value",
            "givenName": "givenName-value",
            "jobTitle": "jobTitle-value",
            "mail": "mail-value",
            "mobilePhone": "mobilePhone-value",
            "officeLocation": "officeLocation-value",
            "preferredLanguage": "preferredLanguage-value",
            "surname": "surname-value",
            "userPrincipalName": "userPrincipalName-value",
            "id": "id-value"
        }
    ]
}
```

Get the user's profile photo

Profile photos can be set on user accounts, groups, and contacts in Office 365. When an application needs to fetch image data, several sizes of a user photo may exist. Sometimes an application needs to use one that is smaller or larger than the default. Developers can use Microsoft Graph to view, download, and manage profile photo for these three resource types.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	For user resource: User.Read, User.ReadBasic.All, User.Read.All, User.ReadWrite, User.ReadWrite.All
	For group resource: Group.Read.All, Group.ReadWrite.All
	For contact resource: Contacts.Read, Contacts.ReadWrite
Delegated (personal Microsoft account)	Not supported
Application	For user resource: User.Read.All, User.ReadWrite.All
	For group resource: Group.Read.All, Group.ReadWrite.All
	For contact resource: Contacts.Read, Contacts.ReadWrite

HTTP request for the specified profile photo

```
GET /me/photo/$value
GET /users/{id | userPrincipalName}/photo/$value
GET /groups/{id}/photo/$value
GET /me/contacts/{id}/photo/$value
GET /users/{id | userPrincipalName}/contacts/{id}/photo/$value
GET /me/contactfolders/{contactFolderId}/contacts/{id}/photo/$value
GET /users/{id | userPrincipalName}/contactfolders/{contactFolderId}/
contacts/{id}/photo/$value
```

HTTP request for the metadata of specified profile photo (profilePhoto properties)

```
GET /me/photo
GET /me/photos
GET /users/{id | userPrincipalName}/photo
GET /groups/{id}/photo
GET /me/contacts/{id}/photo
GET /users/{id | userPrincipalName}/contacts/{id}/photo
GET /me/contactfolders/{contactFolderId}/contacts/{id}/photo
GET /users/{id | userPrincipalName}/contactfolders/{contactFolderId}/
contacts/{id}/photo
```

The following example illustrates the request and response to get the photo for the signed-in user in the largest available size

HTTP request

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

Response

- Contains the binary data of the requested photo.
- The HTTP response code is 200.
- Note that the response is a binary blob of data.

The following example illustrates the request and response to get the 48x48 photo for the signed-in user

HTTP request

```
GET https://graph.microsoft.com/v1.0/me/photos/48x48/$value  
Content-Type: image/jpg
```

Response

- Contains the binary data of the requested 48x48 photo.
- The HTTP response code is 200.

Note:

The supported sizes of HD photos on Office 365 are as follows: 48x48, 64x64, 96x96, 120x120, 240x240, 360x360, 432x432, 504x504, and 648x648.

The following example illustrates the request and response to get the metadata of the user photo of the signed-in user

HTTP request

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

Response

The following response data shows the photo metadata:

```
HTTP/1.1 200 OK  
Content-type: application/json  
{  
    "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#Me/  
photo/$entity",  
    "@odata.id": "https://graph.microsoft.com/v1.0/users('ddfc489-628b-  
7d04-b48b-20075df800e5@1717622f-1d94-c0d4-9d74-f907ad6677b4')/photo",  
    "@odata.mediaContentType": "image/jpeg",  
    "@odata.mediaEtag": "\"BA09D118\"",  
    "id": "240x240",  
    "width": 240,  
    "height": 240
```

```
}
```

Get the user's manager profile

Many applications, particularly those that provide business processes, need to gain approvals from a user's manager, and this API provides a means of discovering who that is. Running this query returns the user or contact assigned as the user's manager.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	User.ReadBasic.All, User.Read.All, User.Read-Write.All, Directory.Read.All, Directory.Read-Write.All, Directory.AccessAsUser.All
Delegated (personal Microsoft account)	Not supported

HTTP request to get user's manager

```
GET /me/manager  
GET /users/{id | userPrincipalName}/manager
```

The following example illustrates the default request and response.

Request

Using HTTP:

```
GET https://graph.microsoft.com/v1.0/users/{id|userPrincipalName}/manager
```

Using C#:

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );  
var directoryObject = await graphClient.Users["{id|userPrincipalName}"].  
Manager  
.Request()  
.GetAsync();
```

Using JavaScript:

```
const options = {  
    authProvider,  
};  
const client = Client.init(options);  
let res = await client.api('/users/{id|userPrincipalName}/manager')  
.get();
```

HTTP response

The following is an example of the response.***

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "objectType": "User",
    "id": "111048d2-2761-4347-b978-07354283363b",
    "accountEnabled": true,
    "department": "Sales & Marketing",
    "displayName": "Sara Davis",
    "givenName": "Sara",
    "jobTitle": "Finance VP",
    "mail": "SaraD@contoso.onmicrosoft.com",
    "mailNickname": "SaraD",
    "state": "CA",
    "streetAddress": "9256 Towne Center Dr., Suite 400",
    "surname": "Davis",
    "usageLocation": "US",
    "userPrincipalName": "SaraD@contoso.onmicrosoft.com"
}
```

Lesson review questions

1. How do you get the signed-in user's profile? What information do you get when passing the /users token? Name 5 major properties and their purpose.
2. What permission type and what minimum permission is required to get a user manager's profile?

correct/suggested answer:

1. HTTP request: GET /me

- Major properties include:
 - businessPhones
 - displayName
 - givenName
 - id
 - jobTitle
 - mail
 - mobilePhone
 - officeLocation
 - preferredLanguage
 - surname
 - userPrincipalName

2. Suggested answer

- Permission type: **Delegated (work or school account)**
- Minimum permission required: **User.ReadBasic.All**

Access Files with Microsoft Graph

Lesson introduction

You can use Microsoft Graph to build a variety of experiences with Office 365 files across OneDrive, OneDrive for Business, and SharePoint document libraries. Scenarios can range from simply storing user documents to complex file-sharing scenarios.

In this lesson, you'll work with files through Microsoft Graph, including getting a list of files for a signed-in user, downloading a file, getting a list of trending files, and uploading large files.

After this lesson, you should be able to:

- Get the list of files in the signed-in user's OneDrive.
- Download a file from the signed-in user's OneDrive using a unique file ID.
- Download a file from a SharePoint site using the relative path to the file.
- Get the list of files trending around the signed-in user.
- Upload a large file to OneDrive.
- Get a user object from an owner list in a group and retrieve that user's files.

Work with files in Microsoft Graph

Files in Office 365 are stored in drives. Users can store files in a personal drive—their OneDrive—or in a shared drive powered by a SharePoint document library. OneDrive's flexibility lets users collaborate however it works best for them. Users can share links to files, copy, or move files to team drives, or even attach OneDrive files to mail messages in Outlook.

Microsoft Graph exposes two resource types for working with files:

- **Drive** - Represents a logical container of files.
- **DriveItem** - Represents an item within a drive, like a document, photo, video, or folder.

Drive and **DriveItem** resources expose data in three different ways:

- *Properties* (like **id** and **name**) expose simple values (strings, numbers, Booleans).
- *Facets* (like **file** and **photo**) expose complex values. The presence of **file** or **folder** facets indicates behaviors and properties of a **DriveItem**.
- *References* (like **children** and **thumbnails**) point to collections of other resources.

Drive resource

The drive resource is the top level object representing a user's OneDrive or a document library in SharePoint. A drive resource can be addressed either by the drive's unique ID or by the default drive for a User, Group, or organization. Here are common tasks that developers can work with drive resource:

- Get Drive metadata of another Drive
- Get root folder for user's default Drive
- List children under the Drive
- List changes for all Items in the Drive
- Search for Items in the Drive

- Access special folder

Here is a JSON representation of a Drive resource. For the descriptions of the properties, please refer to the [link¹¹](#).

```
{  
    "id": "string",  
    "createdBy": { "@odata.type": "microsoft.graph.identitySet" },  
    "createdDateTime": "string (timestamp)",  
    "description": "string",  
    "driveType": "personal | business | documentLibrary",  
    "items": [ { "@odata.type": "microsoft.graph.driveItem" } ],  
    "lastModifiedBy": { "@odata.type": "microsoft.graph.identitySet" },  
    "lastModifiedDateTime": "string (timestamp)",  
    "name": "string",  
    "owner": { "@odata.type": "microsoft.graph.identitySet" },  
    "quota": { "@odata.type": "microsoft.graph.quota" },  
    "root": { "@odata.type": "microsoft.graph.driveItem" },  
    "sharepointIds": { "@odata.type": "microsoft.graph.sharepointIds" },  
    "special": [ { "@odata.type": "microsoft.graph.driveItem" } ],  
    "system": { "@odata.type": "microsoft.graph.systemFacet" },  
    "webUrl": "url"  
}
```

DriveItem resource

All file system objects in OneDrive and SharePoint are returned as driveitem resources. There are two primary ways of addressing a **driveitem** resource:

- By the **driveitem** unique identifier using `drive/items/{item-id}`
- By file system path using `/drive/root:/path/to/file`

Driveitem resources have facets modeled as properties that provide data about the driveitem's identities and capabilities. Items with the **folder** facet act as containers of items and therefore have a `children` reference pointing to a collection of **driveitems** under the folder.

Most of the interaction with files occurs through interaction with **DriveItem** resources. Developers can use following methods with DriveItem resources:

- Get item
- Create item
- Update item
- Delete item
- Move item
- Copy item
- Search items
- Preview item
- List children

¹¹ <https://docs.microsoft.com/en-us/graph/api/resources/drive?view=graph-rest-1.0>

- List versions
- Add permissions
- List permissions
- Delete permission
- Upload content
- Download content
- Download specific file format
- List thumbnails
- List changes in a drive
- Create sharing link
- Get analytics
- Get WebSocket channel
- Get activities by interval

Here is a JSON representation of a driveitem resource. For the descriptions of the properties, please refer to the [link¹²](#).

```
{  
    "audio": { "@odata.type": "microsoft.graph.audio" },  
    "content": { "@odata.type": "Edm.Stream" },  
    "cTag": "string (etag)",  
    "deleted": { "@odata.type": "microsoft.graph.deleted" },  
    "description": "string",  
    "file": { "@odata.type": "microsoft.graph.file" },  
    "fileSystemInfo": { "@odata.type": "microsoft.graph.fileSystemInfo" },  
    "folder": { "@odata.type": "microsoft.graph.folder" },  
    "image": { "@odata.type": "microsoft.graph.image" },  
    "location": { "@odata.type": "microsoft.graph.geoCoordinates" },  
    "package": { "@odata.type": "microsoft.graph.package" },  
    "photo": { "@odata.type": "microsoft.graph.photo" },  
    "publication": { "@odata.type": "microsoft.graph.publicationFacet" },  
    "remoteItem": { "@odata.type": "microsoft.graph.remoteItem" },  
    "root": { "@odata.type": "microsoft.graph.root" },  
    "searchResult": { "@odata.type": "microsoft.graph.searchResult" },  
    "shared": { "@odata.type": "microsoft.graph.shared" },  
    "sharepointIds": { "@odata.type": "microsoft.graph.sharepointIds" },  
    "size": 1024,  
    "specialFolder": { "@odata.type": "microsoft.graph.specialFolder" },  
    "video": { "@odata.type": "microsoft.graph.video" },  
    "webDavUrl": "string",  
    /* relationships */  
    "activities": [{ "@odata.type": "microsoft.graph.itemActivity" }],  
    "analytics": { "@odata.type": "microsoft.graph.itemAnalytics" },  
    "children": [{ "@odata.type": "microsoft.graph.driveItem" }],  
    "createdByUser": { "@odata.type": "microsoft.graph.user" },  
}
```

¹² <https://docs.microsoft.com/en-us/graph/api/resources/driveitem?view=graph-rest-1.0>

```

"lastModifiedByUser": { "@odata.type": "microsoft.graph.user" },
"permissions": [ { "@odata.type": "microsoft.graph.permission" } ],
"subscriptions": [ { "@odata.type": "microsoft.graph.subscription" } ],
"thumbnails": [ { "@odata.type": "microsoft.graph.thumbnailSet" } ],
"versions": [ { "@odata.type": "microsoft.graph.driveItemVersion" } ],
/* inherited from baseItem */
"id": "string (identifier)",
"createdBy": { "@odata.type": "microsoft.graph.identitySet" },
"createdDateTime": "String (timestamp)",
"eTag": "string",
"lastModifiedBy": { "@odata.type": "microsoft.graph.identitySet" },
"lastModifiedDateTime": "String (timestamp)",
"name": "string",
"parentReference": { "@odata.type": "microsoft.graph.itemReference" },
"webUrl": "string",
/* instance annotations */
"@microsoft.graph.conflictBehavior": "string",
"@microsoft.graph.downloadUrl": "url",
"@microsoft.graph.sourceUrl": "url"
}
}

```

Get information of a Drive

A Drive is the top-level container for a file system, such as OneDrive or SharePoint document libraries. Developers can use Microsoft Graph to retrieve the **properties** and **relationships** of a Drive resource.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Files.Read , Files.ReadWrite , Files.Read.All , Files.ReadWrite.All , Sites.Read.All , Sites.ReadWrite.All
Delegated (personal Microsoft account)	Files.Read , Files.ReadWrite , Files.Read.All , Files.ReadWrite.All
Application	Files.Read.All , Files.ReadWrite.All , Sites.Read.All , Sites.ReadWrite.All

HTTP request

Developers can retrieve information of a drive based on different scenarios. Developers can use the **\$select** query parameter to shape the response.

Scenarios	HTTP request
Get current user's OneDrive	GET /me/drive
Get a user's OneDrive	GET /users/{idOrUserPrincipalName}/drive
Get the document library associated with a group	GET /groups/{groupId}/drive
Get the document library for a site	GET /sites/{siteId}/drive

Scenarios	HTTP request
Get a drive by ID	GET /drives/{drive-id}

Example: Get current user's OneDrive

OneDrive is the files hub in Office 365. With OneDrive, users can access these files no matter where they are stored, and with Microsoft Graph, you can use a single API to work with those files.

OneDrive users will always have at least one drive available, their default drive. Users without a OneDrive license may not have a default drive available. If a user's OneDrive is not provisioned but the user has a license to use OneDrive, this request automatically provisions the user's drive when using delegated authentication.

Following is an example to retrieve the properties and relationships of a signed in user's drive.

HTTP request

```
GET /me/drive
```

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var drive = await graphClient.Me.Drive
    .Request()
    .GetAsync();

const options = {
    authProvider,
};
const client = Client.init(options);
let res = await client.api('/me/drive')
    .get();
```

HTTP response

The method returns a **Drive resource** for the matching drive in the response body.

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "id": "b!t18F8ybsHUq1z3LTz8xvZqP8zaSWjkFNhsME-Fepo75dTf9vQKfeRb1BZ-
josQrd7",
    "driveType": "business",
    "owner": {
        "user": {
            "id": "eafeelb77-fb3b-4f65-99d6-274c11914d12",
            "displayName": "Ryan Gregg"
        }
    },
    "quota": {
        "deleted": 256938,
        "remaining": 1099447353539,
```

```
        "state": "normal",
        "total": 1099511627776
    }
}
```

Get a list of items in a Drive

DriveItems with a non-null **folder** or **package** facet can have one or more child DriveItems.

Developers can use Microsoft Graph to return a collection of DriveItems in the **children** relationship of a DriveItem.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Files.Read , Files.ReadWrite , Files.Read.All , Files.ReadWrite.All , Sites.Read.All , Sites.ReadWrite.All
Delegated (personal Microsoft account)	Files.Read , Files.ReadWrite , Files.Read.All , Files.ReadWrite.All
Application	Files.Read.All , Files.ReadWrite.All , Sites.Read.All , Sites.ReadWrite.All

HTTP request

Developers can list items in a drive based on different scenarios. Developers can use the **\$expand**, **\$select**, **\$skipToken**, **\$top** and **\$orderby** query parameter to customize the response.

```
GET /me/drive/items/{item-id}/children
GET /drives/{drive-id}/items/{item-id}/children
GET /drives/{drive-id}/root:{path-relative-to-root}:/children
GET /groups/{group-id}/drive/items/{item-id}/children
GET /sites/{site-id}/drive/items/{item-id}/children
GET /users/{user-id}/drive/items/{item-id}/children
```

Example: List items in the root of the current user's OneDrive

To retrieve files in the root of the drive, use the **root** relationship on the drive, then access the children relationship.

HTTP request

```
GET /me/drive/root/children
```

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var children = await graphClient.Me.Drive.Root.Children
```

```
.Request()  
.GetAsync();  
  
const options = {  
    authProvider,  
};  
const client = Client.init(options);  
let res = await client.api('/me/drive/root/children')  
.get();
```

HTTP response

If successful, this method returns the list of items in the children collection of the target item. The children collection will be composed of driveItem resources. If *if-none-match* request header is included and the eTag (or cTag) provided matches the current tag on the file, an HTTP 304 Not Modified response is returned.

```
HTTP/1.1 200 OK  
Content-type: application/json  
{  
    "value": [  
        {"name": "myfile.jpg", "size": 2048, "file": {} },  
        {"name": "Documents", "folder": { "childCount": 4 } },  
        {"name": "Photos", "folder": { "childCount": 203 } },  
        {"name": "my sheet(1).xlsx", "size": 197 }  
    ],  
    "@odata.nextLink": "https://..."  
}
```

Note:

If a collection exceeds the default page size (200 items), the **@odata.nextLink** property is returned in the response to indicate more items are available and provide the request URL for the next page of items.

You can control the page size through optional query string parameters, such as **\$skipToken** and **\$top**.

Get the list of trending items around the signed-in user

Developers can use Microsoft Graph to list a set of items that have been recently used by the signed in user. This collection includes items that are in the user's drive, as well as items they have access to from other drives.

Permissions

One of the following permissions is required to call the API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Files.Read, Files.ReadWrite, Files.Read.All, Files.ReadWrite.All, Sites.Read.All, Sites.ReadWrite.All
Delegated (personal Microsoft account)	Files.Read, Files.ReadWrite, Files.Read.All, Files.ReadWrite.All
Application	Files.Read.All, Files.ReadWrite.All, Sites.Read.All, Sites.ReadWrite.All

HTTP request

```
GET /me/drive/recent
```

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var recent = await graphClient.Me.Drive
    .Recent()
    .Request()
    .GetAsync();

const options = {
    authProvider,
};
const client = Client.init(options);
let res = await client.api('/me/drive/recent')
    .get();
```

HTTP response

This method returns a collection of **DriveItem** resources for items that the owner of the drive recently accessed.

```
{
    "value": [
        {
            "id": "1312abc!1231",
            "remoteItem": {
                "id": "1991210caf!192",
                "name": "March Proposal.docx",
                "file": {},
                "size": 19121,
                "parentReference": {
                    "driveId": "1991210caf",
                    "id": "1991210caf!104"
                }
            },
            "fileSystemInfo": {
                "lastAccessedDateTime": "2017-02-20T19:13:00Z"
            }
        }
    ]
}
```

```
        },
        {
            ""id"": """",
            ""name"": """",
            ""file"": {},
            ""size"": 37810,
            ""parentReference"": {
                ""driveId"": """",
                ""id"": """
            },
            ""fileSystemInfo"": {
                ""lastAccessedDateTime"": """
            }
        }
    ]
}
```

Some **driveItems** returned from the recent action will include the **remoteItem** facet, which indicates that they are items from another drive. To access the original **driveItem** object, you need to make a request using the information provided in **remoteItem** in the following format:

HTTP request

```
GET /drives/{remoteItem-driveId}/items/{remoteItem-id}
```

Download files

Developers can use Microsoft Graph to download the contents of the primary stream (file) of a DriveItem. Only driveItems with the **file** property can be downloaded.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Files.Read, Files.ReadWrite, Files.Read.All, Files.ReadWrite.All, Sites.Read.All, Sites.ReadWrite.All
Delegated (personal Microsoft account)	Files.Read, Files.ReadWrite, Files.Read.All, Files.ReadWrite.All
Application	Files.Read.All, Files.ReadWrite.All, Sites.Read.All, Sites**.ReadWrite.All**

HTTP request

If if-none-match request header is included and the eTag (or cTag) provided matches the current tag on the file, an HTTP 304 Not Modified response is returned.

```
GET /drives/{drive-id}/items/{item-id}/content
GET /groups/{group-id}/drive/items/{item-id}/content
```

```
GET /me/drive/root:{item-path}/content
GET /me/drive/items/{item-id}/content
GET /sites/{siteId}/drive/items/{item-id}/content
GET /users/{userId}/drive/items/{item-id}/content
```

Example: Download a file from the signed-in user's OneDrive using file unique id

Here is an example to download a complete file.

```
GET /me/drive/items/{item-id}/content

GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var stream = await graphClient.Me.Drive.Items["{item-id}"].Content
    .Request()
    .GetAsync();

const options = {
    authProvider,
};
const client = Client.init(options);
let res = await client.api('/me/drive/items/{item-id}/content')
    .get();
```

Response

Returns a **302Found** response redirecting to a pre-authenticated download URL for the file. This is the same URL available through the **@microsoft.graph.downloadUrl** property on the DriveItem.

To download the contents of the file your application will need to follow the **Location** header in the response. Many HTTP client libraries will automatically follow the 302 redirection and start downloading the file immediately.

Pre-authenticated download URLs are only valid for a short period of time (a few minutes) and do not require an **Authorization** header to download.

```
HTTP/1.1 302 Found
Location: https://b0mpua-by3301.files.1drv.com/y23vmagahszxz1cvhasdhasgha-
sodfi
```

Partial range downloads

To download a partial range of bytes from the file, your app can use the **Range** header as specified in RFC 2616. Note that you must append the **Range** header to the actual **@microsoft.graph.downloadUrl** URL and not to the request for **/content**.

```
GET https://b0mpua-by3301.files.1drv.com/y23vmag
Range: bytes=0-1023
```

This will return an **HTTP 206 Partial Content** response with the request range of bytes from the file. If the range cannot be generated the Range header may be ignored and an **HTTP 200** response would be returned with the full contents of the file.

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 0-1023/2048
Content-Type: application/octet-stream
<first 1024 bytes of file>
```

Upload a large file

Uploading large files (>4Mb) to a drive uses a different pattern to overcome the inherent limitations of an HTTP-based API.

It does this by creating a file upload session and uploading the file in chunks, passing information in the headers as to which part of the file is being uploaded and how big the file is. When the last section of the file is uploaded, the server returns an HTTP status code of 201.

Create an upload session to allow your app to upload files up to the maximum file size. An upload session allows your app to upload ranges of the file in sequential API requests, which allows the transfer to be resumed if a connection is dropped while the upload is in progress.

To upload a file using an upload session, there are two steps:

- Create an upload session.
- Upload bytes to the upload session.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Files.ReadWrite, Files.ReadWrite.All, Sites.ReadWrite.All
Delegated (personal Microsoft account)	Files.ReadWrite, Files.ReadWrite.All
Application	Sites.ReadWrite.All

Example: upload a large file to the signed-in user's OneDrive

Step 1: Create an upload session

To begin a large file upload, your app must first request a new upload session. This creates a temporary storage location where the bytes of the file will be saved until the complete file is uploaded. Once the last byte of the file has been uploaded, the upload session is completed and the final file is shown in the destination folder.

```
POST /drives/{driveId}/items/{itemId}/createUploadSession
POST /groups/{groupId}/drive/items/{itemId}/createUploadSession
POST /me/drive/items/{itemId}/createUploadSession
POST /sites/{siteId}/drive/items/{itemId}/createUploadSession
```

```
POST /users/{userId}/drive/items/{itemId}/createUploadSession
```

HTTP request

The HTTP request includes the path to the signed-in user's OneDrive to upload the file. No request body is required. However, you can specify an item property in the request body, providing additional data about the file being uploaded.

```
POST /me/drive/root:{item-path}/createUploadSession
Content-Type: application/json
{
    "item": {
        "@odata.type": "microsoft.graph.driveItemUploadableProperties",
        "@microsoft.graph.conflictBehavior": "rename",
        "name": "largefile.dat"
    }
}
```

HTTP response

The response to this request, if successful, provide the details for where the remainder of the requests should be sent as an **UploadSession** resource.

This resource provides details about where the byte range of the file should be uploaded and when the upload session expires.

```
{
    "uploadUrl": "https://sn3302.up.1drv.com/up/fe6987415ace7X4e1eF866337",
    "expirationDateTime": "2015-01-29T09:21:55.523Z"
}
```

Step 2: Upload bytes to the upload session

To upload the file, or a portion of the file, your app makes a PUT request to the **uploadUrl** value received in the **createUploadSession** response. You can upload the entire file, or split the file into multiple byte ranges, as long as the maximum bytes in any given request is less than 60 MiB.

The fragments of the file must be uploaded sequentially in order. Uploading fragments out of order will result in an error.

Note:

If your app splits a file into multiple byte ranges, the size of each byte range MUST be a multiple of 320 KiB (327,680 bytes). Using a fragment size that does not divide evenly by 320 KiB will result in errors committing some files.

In this example, the app is uploading the first 26 bytes of a 128-byte file.

The **Content-Length** header specifies the size of the current request.

The **Content-Range** header indicates the range of bytes in the overall file that this request represents.

The total length of the file is known before you can upload the first fragment of the file.

HTTP request

```
PUT https://sn3302.up.1drv.com/up/fe6987415ace7X4e1eF866337
Content-Length: 26
Content-Range: bytes 0-25/128
<bytes 0-25 of the file>
```

Note:

Your app must make sure that the total file size specified in the Content-Range header is the same for all requests. If a byte range declares a different file size, the request will fail.

HTTP response

When the request is complete, the server will respond with **202 Accepted** if there are more byte ranges that need to be uploaded.

```
{
  "expirationDateTime": "2015-01-29T09:21:55.523Z",
  "nextExpectedRanges": ["26-"]
}
```

Your app can use the **nextExpectedRanges** value to determine where to start the next byte range. You may see multiple ranges specified, indicating parts of the file that the server has not yet received. This is useful if you need to resume a transfer that was interrupted and your client is unsure of the state on the service.

Do not assume that **nextExpectedRanges** will return ranges of proper size for a byte range to upload. The **nextExpectedRanges** property indicates ranges of the file that have not been received and not a pattern for how your app should upload the file.

```
HTTP/1.1 202 Accepted
Content-Type: application/json
{
  "expirationDateTime": "2015-01-29T09:21:55.523Z",
  "nextExpectedRanges": [
    "12345-55232",
    "77829-99375"
  ]
}
```

The **nextExpectedRanges** property won't always list all of the missing ranges. On successful fragment writes, it will return the next range to start from (eg. "523-").

On failures when the client sent a fragment the server had already received, the server will respond with **HTTP 416 Requested Range Not Satisfiable**. You can request upload status to get a more detailed list of missing ranges.

Including the **Authorization** header when issuing the **PUT** call may result in a **HTTP 401 Unauthorized response**. The Authorization header and bearer token should only be sent when issuing the POST during the first step. It should not be included when issuing the **PUT**.

Step 3: Completing a file

When the last byte range of a file is received, the server will response with an **HTTP 201 Created** or **HTTP 200 OK**. The response body will also include the default property set for the **driveItem** representing the completed file.

HTTP request

```
PUT https://sn3302.up.1drv.com/up/fe6987415ace7X4e1eF866337
Content-Length: 21
Content-Range: bytes 101-127/128
<final bytes of the file>
```

HTTP response

```
HTTP/1.1 201 Created
Content-Type: application/json
{
    "id": "912310013A123",
    "name": "largefile.vhd",
    "size": 128,
    "file": {}
}
```

Handling upload conflicts

There could be issues while uploading the bytes. If a conflict occurs after the file is uploaded (for example, an item with the same name was created during the upload session), an error is returned when the last byte range is uploaded.

```
HTTP/1.1 409 Conflict
Content-Type: application/json
{
    "error": {
        "code": "upload_name_conflict",
        "message": "Another file exists with the same name as the uploaded session. You can redirect the upload session to use a new filename by calling PUT with the new metadata and @microsoft.graph.sourceUrl attribute."
    }
}
```

Cancel the upload session

To cancel an upload session send a **DELETE** request to the upload URL. This cleans up the temporary file holding the data previously uploaded. This should be used in scenarios where the upload is aborted, for example, if the user cancels the transfer.

Temporary files and their accompanying upload session are automatically cleaned up after the **expirationDateTime** has passed. Temporary files may not be deleted immediately after the expiration time has elapsed.

```
DELETE https://sn3302.up.1drv.com/up/fe6987415ace7X4e1eF866337
```

Resuming an in-progress upload

If an upload request is disconnected or fails before the request is completed, all bytes in that request are ignored. This can occur if the connection between your app and the service is dropped. If this occurs, your app can still resume the file transfer from the previously completed fragment.

To find out which byte ranges have been received previously, your app can request the status of an upload session.

```
GET https://sn3302.up.1drv.com/up/fe6987415ace7X4e1eF86633784148bb98a1zjcU-hf7b0mpUadahs
```

The server will respond with a list of missing byte ranges that need to be uploaded and the expiration time for the upload session.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "expirationDateTime": "2015-01-29T09:21:55.523Z",
    "nextExpectedRanges": ["12345-"]
}
```

Best practices

- Resume or retry uploads that fail due to connection interruptions or any **5xx** errors, including:
 - **500 Internal Server Error**
 - **502 Bad Gateway**
 - **503 Service Unavailable**
 - **504 Gateway Timeout**
- Use an exponential backoff strategy if any **5xx** server errors are returned when resuming or retrying upload requests.
- For other errors, you should not use an exponential backoff strategy but limit the number of retry attempts made.
- Handle **404 Not Found** errors when doing resumable uploads by starting the entire upload over. This indicates the upload session no longer exists.
- Use resumable file transfers for files larger than 10 MiB (10,485,760 bytes).
- A byte range size of 10 MiB for stable high-speed connections is optimal. For slower or less reliable connections, you may get better results from a smaller fragment size. The recommended fragment size is between 5-10 MiB.

- Use a byte range size that is a multiple of 320 KiB (327,680 bytes). Failing to use a fragment size that is a multiple of 320 KiB can result in large file transfers failing after the last byte range is uploaded.

The following code snippet shows how the .NET core SDK abstracts much of the complexity involved in performing this task.

```

/// <summary>
/// Take a file greater than 4MB and upload it to the service
/// </summary>
/// <param name="fileToUpload">The file that we want to upload</param>
/// <param name="uploadToSharePoint">Should we upload to SharePoint or
OneDrive?</param>
public async Task<DriveItem> UploadLargeFile(StorageFile fileToUpload,
bool uploadToSharePoint)
{
    Stream fileStream = (await fileToUpload.OpenReadAsync()).AsStreamFor-
Read();
    DriveItem uploadedFile = null;
    UploadSession uploadSession = null;
    // Do we want OneDrive for Business/Consumer or do we want a Share-
Point Site?
    if (uploadToSharePoint)
    {
        uploadSession = await graphClient.Sites["root"].Drive.Root.
ItemWithPath(fileToUpload.Name).CreateUploadSession().Request().PostAsync();
    }
    else
    {
        uploadSession = await graphClient.Me.Drive.Root.ItemWithPath(fi-
leToUpload.Name).CreateUploadSession().Request().PostAsync();
    }
    if(uploadSession != null)
    {
        // Chunk size must be divisible by 320KiB, our chunk size will
be slightly more than 1MB
        int maxSizeChunk = (320 * 1024) * 4;
        ChunkedUploadProvider uploadProvider = new ChunkedUploadProvid-
er(uploadSession, graphClient, fileStream, maxSizeChunk);
        var chunkRequests = uploadProvider.GetUploadChunkRequests();
        var exceptions = new List<Exception>();
        var readBuffer = new byte[maxSizeChunk];
        foreach (var request in chunkRequests)
        {
            var result = await uploadProvider.GetChunkRequestRespon-
seAsync(request, readBuffer, exceptions);
            if(result.UploadSucceeded)
            {
                uploadedFile = result.ItemResponse;
            }
        }
    }
    return (uploadedFile);
}

```

```
}
```

Get a user object from an owner list in a group and retrieve that user's files

In this final topic of this lesson, you'll learn how to traverse through Microsoft Graph to get a user object from an owner list in a group and then retrieve that user's files.

Traverse through Microsoft Graph

The following order is needed to traverse through Microsoft Graph based on this topic's scenario:

1. Enumerate owners of the groups to get a user ID.
2. Grant the appropriate permissions for the list owner's API call if necessary.
3. Get the user's drive ID.
4. List the items in the user's drive using the user ID and the drive ID returned from previous HTTP responses.

Enumerate owners of groups

You can retrieve the list of owners by using \$select and \$expand query parameters against the group return type.

HTTP request

```
GET /groups?$select=owners&$expand=owners
```

Response

A response may return a failed message due to insufficient privileges. In this case, you will be asked to modify your permissions upon receiving a failure status code 403.

Grant permissions

You will need to modify your permissions and grant at least one of the following permissions for the API to succeed.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Group.Read.All and User.ReadBasic.All , Group.Read.All and User.Read.All , Group.Read.All and User.ReadWrite.All
Delegated (personal Microsoft account)	Not supported.
Application	Group.Read.All and User.Read.All , Group.Read.All and User.ReadWrite.All

Modify Permissions

Select different **permissions** to try out Microsoft Graph API endpoints.

`files.readwrite.appprovider`

Files.ReadWrite.Selected

Financials.ReadWrite.All

Group.Read.All Admin

Group.ReadWrite.All Admin

IdentityProvider.Read.All *Preview* Admin

IdentityProvider.ReadWrite.All *Preview* Admin

IdentityRiskEvent.Read.All *Preview* Admin

IdentityRiskEvent.ReadWrite.All *Preview* Admin

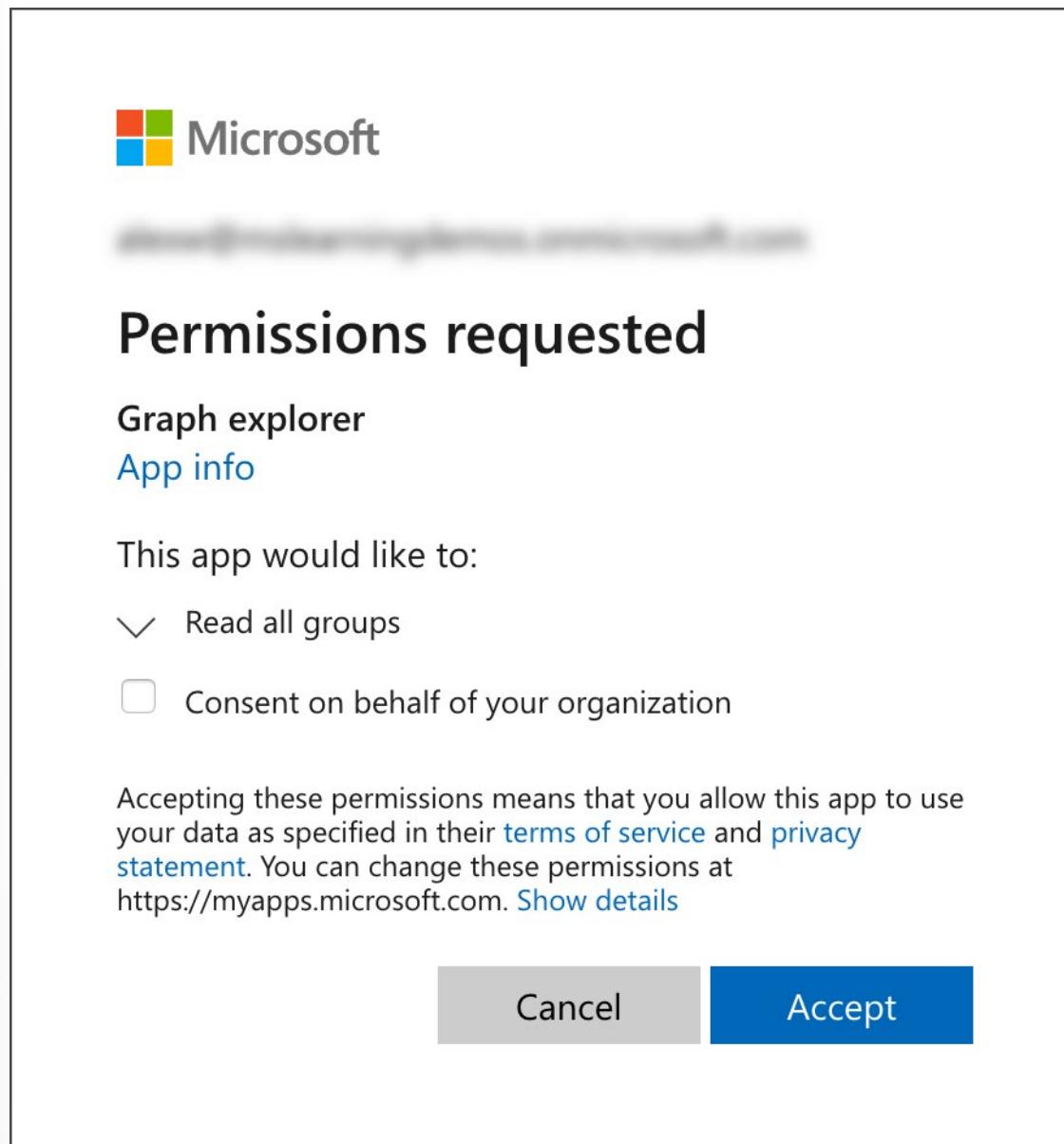
IdentityRiskyUser.Read.All *Preview* Admin

—

ⓘ To change permissions, you will need to log-in again.

ⓘ You have selected permissions that only an administrator can grant. To get access, an administrator can grant [access to your entire organization](#).

Modify Permissions **Close**



Once the appropriate permissions are granted, the HTTP request should return a successful response. Upon receiving a list of owner objects in the response, you would then identify the owner you want to pull the files for and then copy their **id** or **userPrincipalName**.

HTTP request

The following HTTP request example shows the GET request using both ways, first by **ID** and second by **userPrincipalName**.

```
GET /users/{id | userPrincipalName}/drives  
GET /users/b7c0bf77-12d2-41d8-9271-de03037c99ef/drives  
GET /users/demouser@demo.onmicrosoft.com/drives
```

Response

The response will include an id property for the user's drive. You would use this ID for the next HTTP request detailed later.

List the items in the owner's drive

To retrieve the files that are in the user's OneDrive, use the following HTTP request, replacing the tokens for the user ID and drive ID retrieved from the previous calls.

HTTP request

```
GET /users/{id | userPrincipalName}/drives/{drive-id}/root/children  
GET /users/b7c0bf77-12d2-41d8-9271-de03037c99ef/drives/b!Gf4Y0NAr506TrH8_  
AJquxUiPH-3Z7IhGka35v0mWRVyb4o1NxtdLRoVtapeRnZDZ/root/children
```

In this lesson, you learned how to work with files through Microsoft Graph, including getting a list of files for a signed-in user, downloading a file, getting a list of trending files, and uploading large files.

Lesson review questions

1. If a user's OneDrive is not provisioned but the user has a license to use OneDrive, what HTTP request will automatically provision the user's drive when using delegated authentication?
2. What are the two resource types Microsoft Graph exposes that you can use to work with files? Provide the details of what each one represents.

Correct/suggested answers:

1. GET /me/drive
2. 2 resource types:
 - **Drive:** Represents a logical container of files, like a document library or a user's OneDrive.
 - **DriveItem:** Represents an item within a drive, like a document, photo, video, or folder.

Manage a Group Lifecycle on Microsoft Graph

Lesson introduction

Office 365 groups are collections of users who share access to resources in Microsoft services or within your app. Because group membership is managed centrally, any changes to membership affect all services associated with the group.

Groups form the foundation that enables user collaboration and integration across services to support rich scenarios in task planning, teamwork, education, and more. When you integrate with Office 365 groups, your application can support millions of users as they transition across various experiences in the Office 365 suite and beyond.

For Graph, this represents an Azure Active Directory (Azure AD) group, which can be an Office 365 group, or a security group. You can use the Microsoft Graph API to create, manage, or delete groups throughout the lifecycle of collaboration.

After this lesson, you should know how to use Microsoft Graph to:

- Get the list of groups for the current user.
- Get the information on a group by ID.
- Get the list of members in a group.
- Get the list of owners in a group.
- Get the list of groups the signed-in user is a member.
- Provision a group.
- Provision a team with a group.
- Delete a group.

Work with groups in Microsoft Graph

Microsoft Graph enables developers to work with groups in Microsoft 365. Groups are collections of users and other principals that share access to resources in Microsoft services or your app. Using the Microsoft Graph, developers can view and manage groups within Microsoft 365.

Groups are collections of users and other principals who share access to resources in Microsoft services or in your app. Microsoft Graph provides APIs that you can use to create and manage different types of groups and group functionality according to your scenario. All group-related operations in Microsoft Graph require administrator consent.

The group resource within Microsoft Graph represents multiple things as there are different types of groups. The types of groups accessible from Microsoft Graph include:

- Office 365 groups
- Security groups

Note: Groups can only be created through work or school accounts. Personal Microsoft accounts don't support groups.

Type	Use case	groupType	mail-enabled	security-enabled	Can be created and managed via API?
Office 365 groups	Facilitating user collaboration with shared Microsoft online resources.	["Unified"]	true	false	Yes
Security groups	Controlling user access to in-app resources.	[]	false	true	Yes
Mail-enabled security groups	Controlling user access to in-app resources, with a shared group mailbox.	[]	true	true	No
Distribution groups	Distributing mail to the members of the group. It is recommended to use Office 365 groups due to the richer set of resources it provides.	[]	true	false	No

Office 365 groups

Office 365 groups enable people to collaborate on a project or a team. The members within a group share resources, such as Outlook conversations and a calendar, SharePoint files, a OneNote notebook, a SharePoint team site, Planner plans, and Intune device management.

Within Microsoft Graph, these groups are referred to as unified groups.

The power of Office 365 groups is in its collaborative nature, perfect for people who work together on a project or a team. They are created with resources that members of the group share, including:

- Outlook conversations
- Outlook calendar
- SharePoint files
- OneNote notebook
- SharePoint team site
- Planner plans
- Intune device management

Group in Outlook example

The following is a JSON representation of groups in Outlook.

```
{  
    "@odata.context": "https://graph.microsoft.com/v1.0/$metadata-  
#groups/$entity",  
    "id": "4c5ee71b-e6a5-4343-9e2c-4244bc7e0938",  
    "deletedDateTime": null,  
    "classification": "MBI",  
    "createdDateTime": "2016-08-23T14:46:56Z",  
    "description": "This is a group in Outlook",  
    "displayName": "OutlookGroup101",  
    "groupTypes": [  
        "Unified"  
    ],  
    "mail": "outlookgroup101@service.microsoft.com",  
    "mailEnabled": true,  
    "mailNickname": "outlookgroup101",  
    "preferredLanguage": null,  
    "proxyAddresses": [  
        "smtp:outlookgroup101@microsoft.onmicrosoft.com",  
        "SMTP:outlookgroup101@service.microsoft.com"  
    ],  
    "securityEnabled": false,  
    "theme": null,  
    "visibility": "Public"  
}
```

Security groups and mail-enabled security groups

Unlike Office 365 groups, security groups are used to control access to resources. Apps can check if a user is a member of a security group to determine if they can access specific resources within the app. Another capability of security groups is that while they can contain users like Office 365 groups, security groups can also contain other security groups. This allows admins added flexibility in determining users who can access secured resources.

Mail-enabled security groups are used in the same way that security groups are, but with the added feature of a shared mailbox for the groups. Mail-enabled security groups can't be created through the API, but other group operations work. Mail-enabled security groups are read only.

Security group example

The following is a JSON representation of a security group.

```
{  
    "@odata.type": "#microsoft.graph.group",  
    "id": "f87faa71-57a8-4c14-91f0-517f54645106",  
    "deletedDateTime": null,  
    "classification": null,  
    "createdDateTime": "2016-07-20T09:21:23Z",  
    "description": "This group is a Security Group",  
    "groupTypes": [  
        "Security"  
    ],  
    "mail": "outlookgroup101@service.microsoft.com",  
    "mailEnabled": true,  
    "mailNickname": "outlookgroup101",  
    "preferredLanguage": null,  
    "proxyAddresses": [  
        "smtp:outlookgroup101@microsoft.onmicrosoft.com",  
        "SMTP:outlookgroup101@service.microsoft.com"  
    ]  
}
```

```
        "displayName": "SecurityGroup101",
        "groupTypes": [],
        "mail": null,
        "mailEnabled": false,
        "mailNickname": "",
        "preferredLanguage": null,
        "proxyAddresses": [],
        "securityEnabled": true
    }
```

Dynamic membership

All types of groups can have dynamic membership rules that automatically add or remove members from a group based on the user's properties. This provides additional flexibility in managing group membership in that users don't have to be manually added or removed from a group. Attribute-based rules derived from user properties enable administrators to specify, for example, all users in the marketing department should have access to the group.

Developers can use Microsoft Graph to manage dynamic membership on groups through the group's properties **membershipRule** and **membershipRuleProcessingState**.

The following request creates a new Office 365 group for the marketing employees:

```
POST https://graph.microsoft.com/beta/groups
{
    "description": "Marketing department folks",
    "displayName": "Marketing department",
    "groupTypes": [
        "Unified",
        "DynamicMembership"
    ],
    "mailEnabled": true,
    "mailNickname": "marketing",
    "securityEnabled": false,
    "membershipRule": 'user.department -eq "Marketing"',
    "membershipRuleProcessingState": "on"
}
```

The following HTTP request uses the Microsoft Graph API to enable a group for dynamic membership and to set the membership criteria:

```
PATCH https://graph.microsoft.com/v1.0/groups/{id}
{
    "groupTypes": ["Unified", "DynamicMembership"],
    "membershipRule": "user.department -eq 'Marketing'",
    "membershipRuleProcessingState": "on"
}
```

Dynamic membership requires the tenant to have an Azure Active Directory Premium P1 license or greater.

Get group information

Developers can get a list of groups, or specific groups, with the Microsoft Graph API or one of the multiple Microsoft Graph SDKs.

Permissions

The specific permission required will depend on the operation you want to perform. For example, if you're creating, editing or deleting a group, one of the write permissions is required.

Granting any of the group related-operations to an app requires administrator consent.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Group.Read.All, Directory.Read.All, Group.ReadWrite.All, Directory.ReadWrite.All, Directory.AccessAsUser.All
Delegated (personal Microsoft account)	Not supported.
Application	Group.Read.All, Directory.Read.All, Group.ReadWrite.All, Directory.ReadWrite.All

List groups within an organization

Microsoft Graph can be used to get a list of all groups within an organization. This list includes all Office 365 groups and security groups.

To request a list of all groups, submit an HTTP GET request to the **/groups** endpoint:

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

The same request can be done using the Microsoft Graph .NET SDK with the following code:

```
var client = GetAuthenticatedGraphClient(...);
var requestAllGroups = client.Groups.Request();
var resultsAllGroups = requestAllGroups.GetAsync().Result;
foreach (var group in resultsAllGroups)
{
    Console.WriteLine(group.Id + ":" + group.DisplayName + "<" + group.
Mail + ">");
}
```

The list groups returned by the **/groups** endpoint include a subset of all the properties available on a group if the query parameter **\$select** isn't specified. If you want to control the specific properties returned in the request, include a **\$select** query parameter with a comma-delimited list of all the properties you want returned.

If you want a list of a specific type of group, such as all the Office 365 groups (also known as unified groups), use the **\$filter** query parameter on the **groupTypes** property:

```
GET https://graph.microsoft.com/v1.0/groups?$filter=groupTypes/
any(c:c+eq+'Unified')
```

Get a specific group

To get a specific group, include the ID of the group in the HTTP request to the **/groups** endpoint:

```
GET https://graph.microsoft.com/v1.0/groups/{ID}
```

The same request can be done using the Microsoft Graph .NET SDK with the following code:

```
var client = GetAuthenticatedGraphClient(...);
var requestGroup = client.Groups[groupID].Request();
var resultsGroup = requestGroup.GetAsync().Result;
Console.WriteLine(resultsGroup.Id + ":" + resultsGroup.DisplayName + " <" +
+ resultsGroup.Mail + ">");
```

HTTP response

The following is an example of the response. It includes only the default properties.****

```
HTTP/1.1 200 OK
Content-type: application/json
{
    "id": "b320ee12-b1cd-4cca-b648-a437be61c5cd",
    "deletedDateTime": null,
    "classification": null,
    "createdDateTime": "2018-12-22T00:51:37Z",
    "creationOptions": [],
    "description": "Self help community for library",
    "displayName": "Library Assist",
    "groupTypes": [
        "Unified"
    ],
    "mail": "library2@contoso.com",
    "mailEnabled": true,
    "mailNickname": "library",
    "onPremisesLastSyncDateTime": null,
    "onPremisesSecurityIdentifier": null,
    "onPremisesSyncEnabled": null,
    "preferredDataLocation": "CAN",
    "proxyAddresses": [
        "smtp:library7423@contoso.com",
        "SMTP:library2@contoso.com"
    ],
    "renewedDateTime": "2018-12-22T00:51:37Z",
    "resourceBehaviorOptions": [],
    "resourceProvisioningOptions": [],
    "securityEnabled": false,
    "visibility": "Public",
    "onPremisesProvisioningErrors": []
}
```

Get group owners

Non-admin users can be assigned as an owner of a group that grants them permission to modify the group. Developers can use Microsoft Graph to retrieve a list of the group's owners. This helps communicate who has access to group content, or who might need to perform administrative duties, such as renewing the group or approving a join request.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Group.Read.All and User.ReadBasic.All , Group.Read.All and User.Read.All , Group.Read.All and User.ReadWrite.All
Delegated (personal Microsoft account)	Not supported.
Application	Group.Read.All and User.Read.All , Group.Read.All and User.ReadWrite.All

Get the owners of a specific group

To get a list of the group's owners, access the **owners** property on a group:

```
GET /groups/{id}/owners
```

The same request can be done using C# with the following code:

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var owners = await graphClient.Groups["{id}"].Owners
    .Request()
    .GetAsync();
```

The same request can be done using JavaScript with the following code:

```
const options = {
    authProvider,
};

const client = Client.init(options);
let res = await client.api('/groups/{id}/owners')
    .get();
```

HTTP response

The following is an example of the response:***

```
HTTP/1.1 200 OK
Content-type: application/json
Content-length: 55
{
    "value": [
        {
            "id": "1234567890abcdef1234567890abcdef",
            "emailAddress": "owner@contoso.com",
            "userType": "Member"
        }
    ]
}
```

```
    "@odata.type": "#microsoft.graph.user"
  }
]
}
```

Get group members

Developers can use Microsoft Graph to retrieve a list of the group's members. This helps communicate who has access to group content, or who might need to perform administrative duties, such as renewing the group or approving a join request. A group can have users, contacts, and other groups as members. This operation is not transitive.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	User.ReadBasic.All, User.Read.All, Group.Read.All, Directory.Read.All
Delegated (personal Microsoft account)	Not supported.
Application	User.Read.All, Group.Read.All, Directory.Read.All

Get the members of a specific group

To get a list of the users who have been added as members to a group, access the **members** property on a group:

```
GET /groups/{id}/members
```

The same request can be done using C# with the following code:

```
GraphServiceClient graphClient = new GraphServiceClient( authProvider );
var members = await graphClient.Groups["{id}"].Members
    .Request()
    .GetAsync();
```

The same request can be done using JavaScript with the following code:

```
const options = {
  authProvider,
};
const client = Client.init(options);
let res = await client.api('/groups/{id}/members')
  .get();
```

HTTP response

The following is an example of the response:

```
HTTP/1.1 200 OK
Content-type: application/json
Content-length: 55
{
    "value": [
        {
            "id": "id-value"
        }
    ]
}
```

Get the list of groups where a user is an owner

Microsoft Graph can be used to obtain a list of all groups a user is an owner of. This is done by requesting all directory objects the user owns. A directory object is a base type for many other entity types, including Office 365 groups and security groups.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	User.Read.All, User.ReadWrite.All, Directory.Read.All, Directory.ReadWrite.All, Directory.AccessAsUser.All
Delegated (personal Microsoft account)	Not supported.
Application	User.Read.All, User.ReadWrite.All, Directory.Read.All, Directory.ReadWrite.All

Get all groups a user is an owner of

The **ownedObjects** property on a user resource provides a list of the directory objects the user is an owner of:

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

Or:

```
GET https://graph.microsoft.com/v1.0/users/{ID | userPrincipalName}/ownedObjects
```

You can examine the properties of a directory object to determine what type of a group it is. For example, an Office 365 group is indicated by the property **groupTypes:["Unified"]** on the directory object, while a security group is indicated by the **securityEnabled:true** property.

The same request can be done using the Microsoft Graph .NET SDK with the following code:

```
var client = GetAuthenticatedGraphClient(...);
var requestOwnerOf = client.Me.OwnedObjects.Request();
var resultsOwnerOf = requestOwnerOf.GetAsync().Result;
```

By attempting to cast the directory object as a specific type in the Microsoft Graph .NET SDK, you can determine what type of object is.

For example, the following code tries to cast the object to a **Microsoft.Graph.Group** object (an Office 365 group) and a **Microsoft.Graph.DirectoryRole** object (a security group).

```
foreach (var ownedObject in resultsOwnerOf)
{
    var group = ownedObject as Microsoft.Graph.Group;
    var role = ownedObject as Microsoft.Graph.DirectoryRole;
    if (group != null) {
        Console.WriteLine("Office 365 Group: " + group.Id + ": " + group.
Displayname);
    } else if (role != null) {
        Console.WriteLine(" Security Group: " + role.Id + ": " + role.
Displayname);
    } else {
        Console.WriteLine(ownedObject.ODataType + ": " + ownedObject.Id);
    }
}
```

Get the list of groups where a user is a member

Microsoft Graph can be used to obtain a list of all groups a user is a member of. This is done by requesting all groups a user is a member of using the **/memberOf** property on the user:

```
GET https://graph.microsoft.com/v1.0/me/memberOf
// or
GET https://graph.microsoft.com/v1.0/users/{ID | userPrincipalName}/member-
of
```

Like the previous example, this returns a list of directory objects. You can examine the properties of each object returned to determine what type of group it is.

The same request can be done using the Microsoft Graph .NET SDK with the following code:

```
var client = GetAuthenticatedGraphClient(...);
var requestGroupsMemberOf = client.Me.MemberOf.Request();
var resultsGroupsMemberOf = requestGroupsMemberOf.GetAsync().Result;
foreach (var groupDirectoryObject in resultsGroupsMemberOf)
{
    var group = groupDirectoryObject as Microsoft.Graph.Group;
    var role = groupDirectoryObject as Microsoft.Graph.DirectoryRole;
    if (group != null) {
        Console.WriteLine("Office 365 Group: " + group.Id + ": " + group.
Displayname);
    } else if (role != null) {
        Console.WriteLine(" Security Group: " + role.Id + ": " + role.
Displayname);
    } else {
        Console.WriteLine(groupDirectoryObject.ODataType + ": " + groupDi-
rectoryObject.Id);
```

```
    }
}
```

HTTP response

The following is an example of the response.

```
{
  "value": [
    {
      "@odata.type": "#microsoft.graph.group",
      "id": "id-value",
      "createdDateTime": null,
      "description": "All users at the company",
      "displayName": "All Users",
      "groupTypes": [],
      "mailEnabled": false,
      "securityEnabled": true,
    }
  ]
}
```

Direct vs. transitive membership

Unlike the previous example, the **memberOf** property returns a collection of directory objects that the user is a *direct* member of. These are groups that the user has been explicitly added to.

Microsoft Graph can also return the list of directory objects a user is a **transitive** member of. These are the groups that user hasn't been directly added to but is a member of through a nested security group. This can happen if the user is in a security group that's been added to another group or is a member of a group through dynamic membership.

To perform a transitive membership check, use the **getMemberGroups** method on the Microsoft Graph API or the **GetMemberGroups()** method on the Microsoft Graph .NET SDK.

Note:

Office 365 groups cannot contain groups, so membership in an Office 365 group is always direct.

Create a group

You can use Microsoft Graph to manage the complete lifecycle of groups, including Office 365 groups and security groups. This includes creating, updating, and deleting. You can even use Microsoft Graph to create a Microsoft Teams team from an Office 365 group.

Groups sit at the center of modern collaboration in Microsoft 365. Many organizations need to develop applications that will provision groups for collaboration as part of their corporate governance around Microsoft 365.

When creating or updating a group, the HTTP request body contains the group object to create or update. This is typically submitted as a JSON object in string form. This object must contain the required properties for a group, but you can optionally specify any other writable property when creating and updating groups.

To create a group, submit a request to the /groups endpoint as an HTTP POST with the request header **Content-Type** set to **application/json**. The body of the request should include the JSON representation of the group in string format with the following minimal properties:

- **displayName** (string)
- **mailEnabled** (boolean)
- **mailNickname** (string)
- **securityEnabled** (boolean)

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Group.ReadWrite.All , Directory.ReadWrite.All , Directory.AccessAsUser.All
Delegated (personal Microsoft account)	Not supported.
Application	Group.ReadWrite.All , Directory.ReadWrite.All

HTTP request

To create an Office 365 group, set the **groupTypes** collection to **Unified**.

```
POST /groups
```

Example: create an Office 365 group

The following example creates an Office 365 group.

```
POST https://graph.microsoft.com/v1.0/groups
Content-type: application/json
Content-length: 244
{
    "description": "Self help community for library",
    "displayName": "Library Assist",
    "groupTypes": [
        "Unified"
    ],
    "mailEnabled": true,
    "mailNickname": "library",
    "securityEnabled": false
}
```

HTTP response

The content of the POST is what controls the group being created. The following is an example of the response:

```
{  
    "id": "b320ee12-b1cd-4cca-b648-a437be61c5cd",  
    "deletedDateTime": null,  
    "classification": null,  
    "createdDateTime": "2018-12-22T00:51:37Z",  
    "creationOptions": [],  
    "description": "Self help community for library",  
    "displayName": "Library Assist",  
    "groupTypes": [  
        "Unified"  
    ],  
    "mail": "library7423@contoso.com",  
    "mailEnabled": true,  
    "mailNickname": "library",  
    "onPremisesLastSyncDateTime": null,  
    "onPremisesSecurityIdentifier": null,  
    "onPremisesSyncEnabled": null,  
    "preferredDataLocation": "CAN",  
    "proxyAddresses": [  
        "SMTP:library7423@contoso.com"  
    ],  
    "renewedDateTime": "2018-12-22T00:51:37Z",  
    "resourceBehaviorOptions": [],  
    "resourceProvisioningOptions": [],  
    "securityEnabled": false,  
    "visibility": "Public",  
    "onPremisesProvisioningErrors": []  
}
```

Example: create an Office 365 group with owners and members

You can also configure the owners and members of a group at creation time, use the **additionalData** property. This property is a **Dictionary** type that accepts a string as the key and an array of string references to specific user endpoints.

For example, the following code will create a new Office 365 group with one owner and two members

```
private static async Task<Microsoft.Graph.Group> CreateGroupAsync(GraphServiceClient client)  
{  
    // create object to define members & owners as 'additionalData'  
    var additionalData = new Dictionary<string, object>();  
    additionalData.Add("owners@odata.bind",  
        new string[] {  
            "https://graph.microsoft.com/v1.0/users/d280a087-e05b-4c23-  
            b073-738cdb82b25e"  
        }  
    );  
    additionalData.Add("members@odata.bind",  
        new string[] {
```

```
        "https://graph.microsoft.com/v1.0/users/70c095fe-df9d-4250-  
867d-f298e237d681",  
        "https://graph.microsoft.com/v1.0/users/8c2da469-1eba-47a4-  
9322-ee0ddd24d99a"  
    }  
);  
var group = new Microsoft.Graph.Group  
{  
    AdditionalData = additionalData,  
    Description = "My first group created with the Microsoft Graph .NET  
SDK",  
    DisplayName = "My First Group",  
    GroupTypes = new List<String>() { "Unified" },  
    MailEnabled = true,  
    MailNickname = "myfirstgroup01",  
    SecurityEnabled = false  
};  
var requestNewGroup = client.Groups.Request();  
return await requestNewGroup.AddAsync(group);  
}
```

Create a Team with a group

You can use Microsoft Graph to create a new Microsoft Teams team under an existing group, provided the group has at least one owner.

Permissions

One of the following permissions is required to call this API.

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Group.ReadWrite.All
Delegated (personal Microsoft account)	Not supported.
Application	Group.ReadWrite.All

HTTP request

In order to create a team, the group must have a least one owner. If the group was created less than 15 minutes prior, it's possible for the Create team call to fail with a 404-error code due to replication delays. The recommended pattern is to retry the Create team call three times, with a 10-second delay between calls.

Note that group creation is a prerequisite to team creation and that code must implement a retry behavior.

The following is an example of the request. The **PUT** request here is updating the team settings for the group and can control many of the options available to end users of the group.

```
PUT /groups/{id}/team
```

Example: create a Microsoft Teams team from an Office 365 group

```
PUT https://graph.microsoft.com/v1.0/groups/{id}/team
Content-type: application/json
{
    "memberSettings": {
        "allowCreateUpdateChannels": true
    },
    "messagingSettings": {
        "allowUserEditMessages": true,
        "allowUserDeleteMessages": true
    },
    "funSettings": {
        "allowGiphy": true,
        "giphyContentRating": "strict"
    }
}
```

HTTP response

The following is an example of the response:

```
HTTP/1.1 201 Created
Content-type: application/json
Content-length: 401
{
    "memberSettings": {
        "allowCreateUpdateChannels": true,
        "allowDeleteChannels": true,
        "allowAddRemoveApps": true,
        "allowCreateUpdateRemoveTabs": true,
        "allowCreateUpdateRemoveConnectors": true
    },
    "guestSettings": {
        "allowCreateUpdateChannels": true,
        "allowDeleteChannels": true
    },
    "messagingSettings": {
        "allowUserEditMessages": true,
        "allowUserDeleteMessages": true,
        "allowOwnerDeleteMessages": true,
        "allowTeamMentions": true,
        "allowChannelMentions": true
    },
    "funSettings": {
        "allowGiphy": true,
        "giphyContentRating": "strict",
        "allowStickersAndMemes": true,
        "allowCustomMemes": true
    }
}
```

```
    }  
}
```

Delete a group

Groups have a lifecycle and should be removed when they are no longer needed. This topic covers how to delete a group using Microsoft Graph. When deleted, Office 365 groups are moved to a temporary container and can be restored within 30 days. After that time, they are permanently deleted.

Permissions

One of the following permissions is required to call this API:

Permission type	Permissions (from least to most privileged)
Delegated (work or school account)	Group.ReadWrite.All, Directory.AccessAsUser.All
Delegated (personal Microsoft account)	Not supported.
Application	Group.ReadWrite.All

You can send an **HTTP DELETE** request to **/groups/{id}** to delete the group with the specified ID. To delete a group, submit an HTTP DELETE request to the group endpoint. For example:

```
DELETE https://graph.microsoft.com/v1.0/groups/{ID}
```

The same request submitted using the Microsoft Graph .NET SDK:

```
var groupIdToDelete = "{ID}";  
await client.Groups[groupIdToDelete].Request().DeleteAsync();
```

Lesson review questions

1. What's the HTTP request to get a list of groups for the current user?
2. What permissions are required for the Application permission type to get the list of owners of a group?

Correct/suggested answers:

1. GET /me/memberOf
2. Required permissions:

Group.Read.All and **User.Read.All**

OR

Group.Read.All and **User.ReadWrite.All**

Module 3 Extend and Customize SharePoint

SharePoint Framework Web Parts

Lesson introduction

The SharePoint Framework (SPFx) is a page and web part model that provides full support for client-side SharePoint development, easy integration with SharePoint data, and support for open source tooling.

With the SharePoint Framework, you can use modern web technologies and tools in your preferred development environment to build productive experiences and apps that are responsive and mobile-ready.

In this lesson, you'll learn the components of SPFx web parts including rendering framework options, making your SharePoint client-side web parts configurable, using Office UI Fabric React components in your SharePoint client-side web part, and differentiating an app page and a web part.

After this lesson, you should be able to:

- Identify the appropriate tool to create an SPFx Web Part project.
- Describe rendering framework options.
- Describe properties of client-side web parts.
- Describe Office UI Fabric in client-side web parts.
- Describe when to use an app page.
- Differentiate between an app page and a web part.

Introduction to SharePoint Framework

SharePoint Framework is the development model used for the creation of custom experiences for SharePoint and Office 365.

Historically, Microsoft created web parts as full trust C# assemblies that were installed on the cloud servers. However, current development models generally involve JavaScript running in a browser making REST API calls to the SharePoint and Office 365 back-end workloads. C# assemblies don't work in this

world. A new development model was needed. The SharePoint Framework is the next evolution in SharePoint development.

The SharePoint Framework works for SharePoint Online, and for on-premises; e.g., SharePoint 2016 Feature Pack 2 and SharePoint 2019.

Use SharePoint Framework (SPFx) to create the following:

- SharePoint Framework (SPFx) web parts.
- SPFx extensions.
- Single Part App Pages.
- Microsoft Teams customizations.

Key features of the SharePoint Framework

Key features of the SharePoint Framework include the following:

- It runs in the context of the current user and connection in the browser. There are no iFrames for customization (JavaScript is embedded directly into the page).
- The controls are rendered in the normal page Document Object Model (DOM).
- The controls are responsive and accessible by design.
- It enables the developer to access the lifecycle in addition to **render**, **load**, **serialize** and **deserialize**, **configuration changes**, and more.
- It is framework-agnostic. You can use any JavaScript framework that you like: React, Handlebars, Knockout, Angular, and more.
- The toolchain is based on common open-source client development tools such as node package manager (npm), TypeScript, Yeoman, webpack, and gulp.
- Performance is reliable.
- End users can use SPFx client-side solutions that are approved by the tenant administrators (or their delegates) on all sites, including self-service team, group, or personal sites.
- SPFx web parts can be added to both classic and modern pages.

The runtime model improves on the Script Editor web part. It includes a robust client API, an HttpClient object that handles authentication to SharePoint and Office 365, contextual information, easy property definition and configuration, and more.

If you work primarily with C#, you want to learn more about client-side JavaScript development. Most of your existing JavaScript knowledge related to SharePoint is completely transferable as the data models have not changed. You'll use the same **REST services**¹ or **JavaScript Object Model (JSOM)**², depending on your requirements. If you are a C# developer, TypeScript is a nice transition into the JavaScript world. The choice of integrated development environment (IDE) is up to you. Many developers like to use the cross-platform IDE Visual Studio Code. Many developers also use products like Sublime and ATOM. Use what works best for you.

¹ <https://msdn.microsoft.com/en-us/library/office/jj860569.aspx>

² <https://msdn.microsoft.com/en-us/library/office/jj193034.aspx>

Why use the SharePoint Framework?

SharePoint was launched as an on-premises product in 2001. Over time, a large developer community extended and shaped it in many ways. For the most part, the developer community followed the same patterns and practices that the SharePoint product development team used, including web parts, SharePoint feature XML, and more. Many features were written in C#, compiled to DLLs, and deployed to on-premises farms.

That architecture worked well in environments with only one enterprise, but it didn't scale to the cloud, where multiple tenants run side-by-side. As a result, Microsoft introduced two alternative models: client-side JavaScript injection, and SharePoint Add-ins. Both of these solutions have pros and cons.

JavaScript injection

One of the most popular web parts in SharePoint Online is the Script Editor. You can paste JavaScript into the Script Editor web part and have that JavaScript execute when the page renders. It's simple and rudimentary, but effective. It runs in the same browser context as the page and is in the same DOM, so it can interact with other controls on the page. It is also relatively performant and simple to use.

There are a few downsides to this approach. First, while you can package your solution so that end users can drop the control onto the page, you can't easily provide configuration options. Also, the end user can edit the page and modify the script, which can break the web part. Another big problem is that the Script Editor web part is not marked as "Safe for Scripting." Most self-service site collections (my-sites, team sites, group sites) enable a feature known as "NoScript," which removes the Add/Customize Pages (ACP) permission in SharePoint. This means that the Script Editor web part will be blocked from executing on these sites.

SharePoint Add-in model

The current option for solutions that run in NoScript sites is the add-in/app-part model. This implementation creates an iFrame where the actual experience resides and executes. This model is external to the system and has no access to the current DOM/connection, and so it is easier for information workers to trust and deploy. End users can install add-ins on NoScript sites.

There are some downsides to this approach as well. First, it runs in an iFrame. iFrames are slower than the Script Editor web part because they require a new request to another page. The page must go through authentication and authorization, make its own calls to get SharePoint data, load various JavaScript libraries, and more. A Script Editor web part might typically take 100 milliseconds to load and render, while an app part might take two seconds or more. Additionally, the iFrame boundary makes it more difficult to create responsive designs and inherit Cascading Style Sheet (CSS) and theming information. iFrames do have stronger security, which can be useful for you (your page is inaccessible by other controls on the page) and for the end user (the control has no access to their connection to Office 365).

SharePoint Framework development tools and libraries

The SharePoint Framework includes several client-side JavaScript libraries that you can use to build your solutions. The tools and libraries that you can use to develop client-side web parts include:

TypeScript

TypeScript is a typed superset of JavaScript that compiles to plain JavaScript. SharePoint client-side development tools are built using TypeScript classes, modules, and interfaces. You can use these to build robust client-side web parts.

JavaScript frameworks

You can choose any one of a number of JavaScript frameworks to develop client-side web parts. The following are some of the most popular:

- React
- AngularJS 1.x
- Angular 2 for TypeScript 2.x
- Vue.js
- Handlebars

Because client-side web parts are components that are dropped into a SharePoint page, it is recommended that you choose a JavaScript framework that supports a similar component model. Lightweight frameworks such as React, Handlebars, and Angular 2 all support a component model and are well suited to building client-side web parts.

Node Package Manager (npm)

SharePoint client-side development tools use the npm package manager, which is similar to NuGet, to manage dependencies and other required JavaScript helpers. npm is typically included as part of Node.js setup.

Node.js

Node.js is an open-source, cross-platform runtime environment for hosting and serving JavaScript code. You can use Node.js to develop server-side web applications written in JavaScript. The Node.js ecosystem is tightly coupled with npm and task runners such as gulp to provide an efficient environment for building JavaScript-based applications. Node.js is similar to IIS Express or IIS, but includes tools to simplify client-side development.

Gulp task runner

SharePoint client-side development tools use Gulp as the build process task runner to:

- Bundle and minify JavaScript and CSS files.
- Run tools to call the bundling and minification tasks before each build.
- Compile LESS or Sass files to CSS.
- Compile TypeScript files to JavaScript.

Webpack

Webpack is a module bundler that takes your web part files and dependencies, then generates one or more JavaScript bundles so that you can load different bundles for different scenarios.

The development tool chain uses CommonJS for bundling. This enables you to define modules and where you want to use them. The tool chain also uses SystemJS, a universal module loader, to load your modules. This helps you scope your web parts by making sure that each web part is executed in its own namespace.

Source code editors

SharePoint Framework is client-side driven, allowing you to choose from a variety of HTML/JavaScript code editors, such as:

- Visual Studio Code
- Atom
- Webstorm

SharePoint Framework documentation uses Visual Studio Code in the docs and examples. Visual Studio Code is a lightweight but powerful source code editor from Microsoft that runs on your desktop and is available for Windows, Mac, and Linux. It comes with built-in support for JavaScript, TypeScript, and Node.js, and has a rich ecosystem of extensions for other languages (such as C++, C#, Python, PHP) and runtimes.

SharePoint REST APIs

The SharePoint Framework provides key integrations with SharePoint experiences and targets web development. The SharePoint REST APIs enable you to interact with SharePoint and other workloads that shape your web part functionality.

Patterns and Practices (PnP)

The **Office Dev Patterns and Practices/SharePoint Pattern and Practices (PnP)**³ initiative provides code samples, patterns, and other resources to help you convert your existing solution to the SharePoint Framework.

Yeoman generators

Yeoman helps you kickstart new projects, prescribing best practices and tools to help you stay productive. The Yeoman SharePoint generator is available as part of the framework to kickstart new client-side web part projects.

Generate SharePoint Framework Projects using Yeoman

The best way to create a SharePoint Framework project is to use the Yeoman generator which utilizes pre-built project templates. These templates are updated frequently, so you should run a command to update your Yeoman templates with the latest version.

To update, run the cmdlet below in PowerShell:

```
npm install -g @microsoft/generator-sharepoint
```

³ <https://pnp.github.io/>

The basics of creating a SharePoint Framework project using Yeoman starts out the same. You launch PowerShell from Windows 10 and then navigate to the directory in which you want to create the project. Some developers create an _git folder on their local computer where they save Yeoman projects.

Beginning the creation of a SharePoint Framework project using Yeoman is simple. Run the following commands in Powershell:

1. Create a new project directory in your favorite location, giving it a project name such as: `md field-extension`
2. Go to the project directory by changing the directory to the new project folder: `cd field-extension`
3. Create a new extension by running the Yeoman SharePoint Generator in the project folder you just changed to: `yo @microsoft/sharepoint`

SharePoint client-side web parts

SharePoint client-side web parts are controls that appear inside a SharePoint page but run locally in the browser. They're the building blocks of pages that appear on a SharePoint site.

You can build client-side web parts using modern script development tools and the SharePoint workbench (a development test surface), and you can deploy your client-side web parts to modern pages and classic web part pages in Office 365 tenants.

In addition to plain JavaScript projects, you can build web parts alongside common scripting frameworks, such as AngularJS and React. For example, you can use React along with components from Office UI Fabric React to quickly create experiences based on the same components used in Office 365.

Integrate web part properties with SharePoint

The old model for building classic web parts isolated web part properties from SharePoint, and left end users to manage their values. SharePoint Framework offers you a new set of capabilities that simplify managing web part properties' values and integrate them with SharePoint Search.

Note:

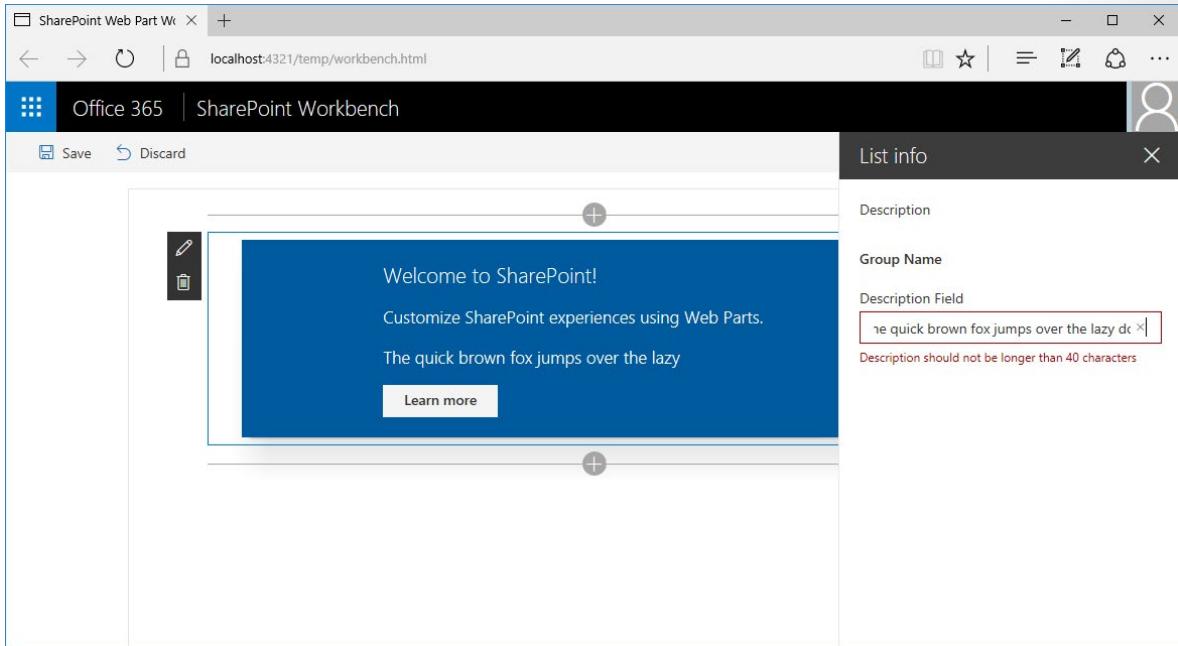
The following guide applies only to SharePoint Framework client-side web parts placed on modern SharePoint pages. Capabilities described in this article don't apply to classic web parts or SharePoint Framework client-side web parts placed on classic pages.

Client-side web part properties

When building SharePoint Framework client-side web parts, you can define properties that can be configured by users. By using properties instead of fixed values, web parts are more flexible and suitable for many different scenarios.

Before accepting values for web part properties from end users, you should always validate them. This not only allows you to ensure that your web parts are user-friendly, but also helps prevent storing invalid data in the web part's configuration.

The image below shows an example of a property validation message.



You should also consider that the SharePoint Framework doesn't support personalization and all users see the same configuration of the particular web part.

Property pane

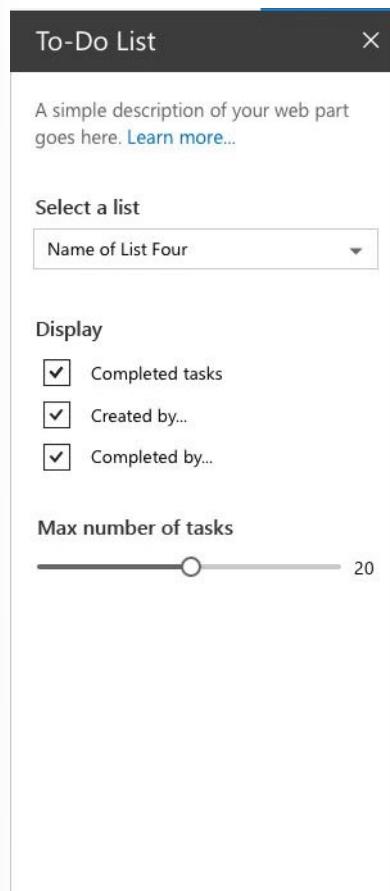
The property pane is where you can define properties to customize your web part. The property pane is client-side driven and provides a consistent design across SharePoint.

The property pane properties are defined in **propertyPaneSettings**.

A property pane has three key metadata: a page, an optional header, and at least one group.

- **Pages** provide you the flexibility to separate complex interactions and put them into one or more pages. Pages contain a header and groups.
- **Headers** allow you to define the title of the property pane.
- **Groups** allow you to define the various sections or fields for the property pane through which you want to group your field sets.

The image below shows an example of a property pane in SharePoint.



Property pane types

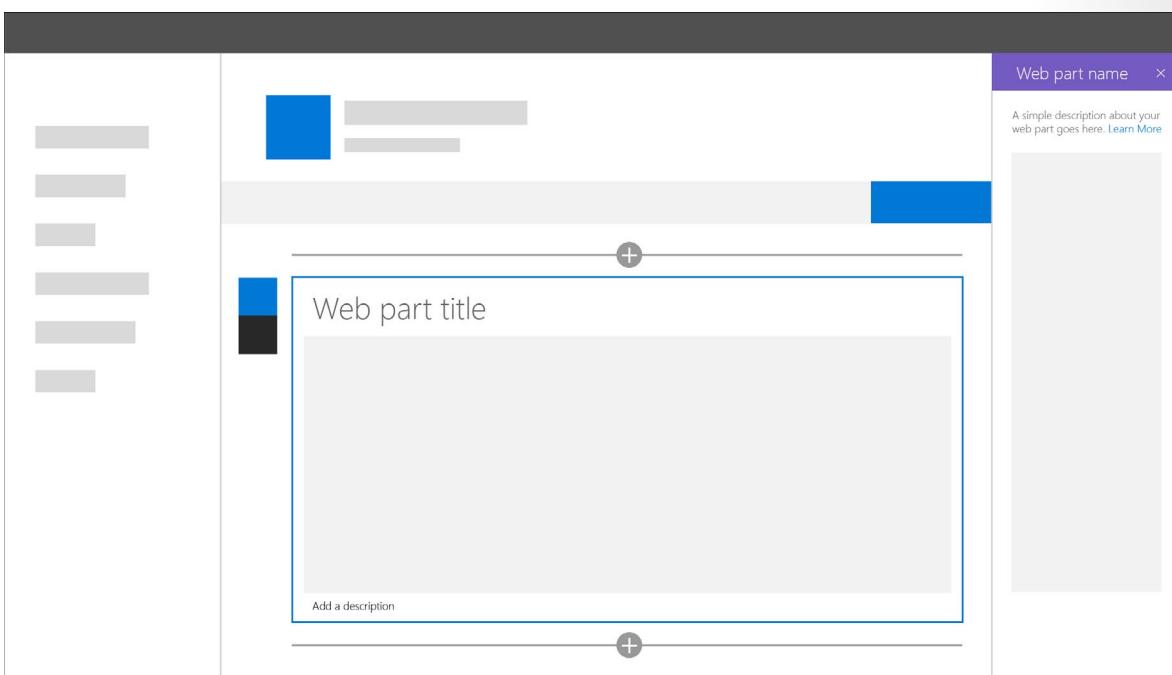
You can use three types of property panes to design and develop web parts that fit your business or customer needs.

To open a pane to configure settings for a web part, select **Edit**. Use the pane to enable and disable features, select a source, choose a layout, and set options. Edit web part content within the web part rather than in the property pane.

The property pane is 320 px and the rest of the page responsively reflows when it is opened.

Single pane

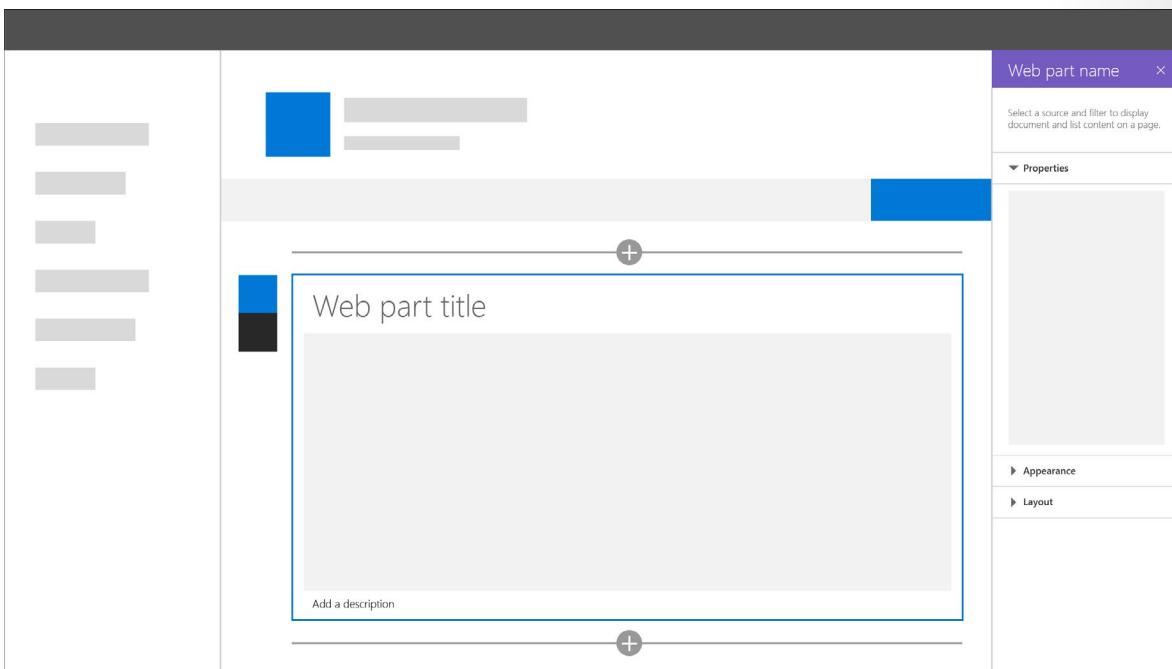
Use a single pane for simple web parts that have only a small number of properties to configure.



Accordion pane

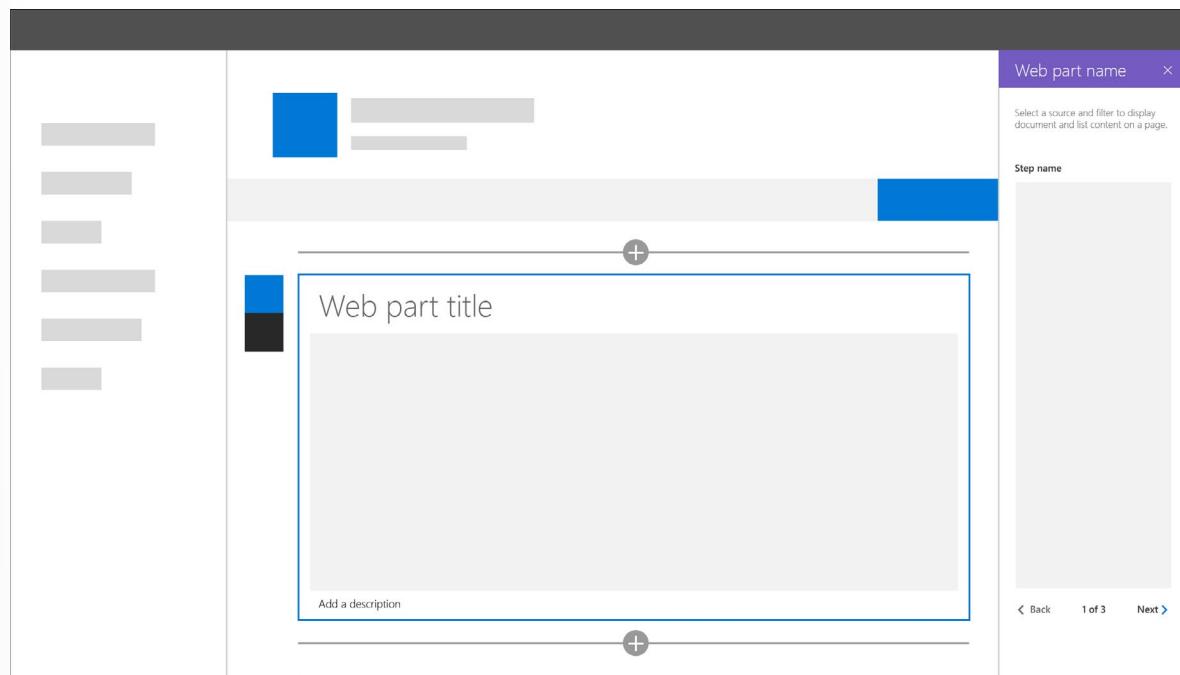
Use an accordion pane to contain a group or groups of properties with many options, and where the groups create a long scrolling list of options. For example, you might have three groups named Properties, Appearance, and Layout, each with ten components.

Use accordion panes when you need to apply categorization for a complex web part.



Steps pane

Use a steps pane to group properties in multiple steps or pages when you need the web part to be configured in a linear order.



Configure the property pane

The following code example initializes and configures the property pane in your web part. You override the **getPropertyPaneConfiguration** method and return a collection of property pane page(s).

```
protected getPropertyPaneConfiguration(): IPropertyPaneConfiguration {
    return {
        pages: [
            {
                header: {
                    description: strings.PropertyPaneDescription
                },
                groups: [
                    {
                        groupName: strings.BasicGroupName,
                        groupFields: [
                            PropertyPaneTextField('description', {
                                label: strings.DescriptionFieldLabel
                            })
                        ]
                    }
                ]
            }
        ];
    };
}
```

```
}
```

Property pane fields

The following field types are supported:

- Button
- Checkbox
- Choice group
- Dropdown
- Horizontal rule
- Label
- Link
- Slider
- Textbox
- Multi-line Textbox
- Toggle
- Custom

The field types are available as modules in **sp-client-platform**. They require an import before you can use them in your code:

```
import {  
    PropertyPaneTextField,  
    PropertyPaneCheckbox,  
    PropertyPaneLabel,  
    PropertyPaneLink,  
    PropertyPaneSlider,  
    PropertyPaneToggle,  
    PropertyPaneDropdown  
} from '@microsoft/sp-webpart-base';
```

Every field type method is defined as follows, taking **PropertyPaneTextField** as an example:

```
PropertyPaneTextField('targetProperty', {  
    //field properties are defined here  
})
```

The **targetProperty** defines the associated object for that field type and is also defined in the props interface in your web part.

To assign types to these properties, define an interface in your web part class that includes one or more target properties.

```
export interface IHelloWorldWebPartProps {  
    targetProperty: string  
}
```

This is then available in your web part by using **this.properties.targetProperty**.

```
<p class="ms-font-l ms-fontColor-white">${escape(this.properties.targetProperty)}</p>
```

Once the properties are defined, you can access them in your web part by using the **this.properties.[property-name]**.

Handle field changes

The property pane has two interaction modes:

- Reactive
- Non-reactive

In reactive mode every change triggers a change event. Reactive behavior automatically updates the web part with the new values.

While reactive mode is sufficient for many scenarios, at times you need non-reactive behavior. Non-reactive does not update the web part automatically unless the user confirms the changes.

To turn on the non-reactive mode, insert the following code in your web part:

```
protected get disableReactivePropertyChanges(): boolean {
    return true;
}
```

Office UI Fabric React components in SharePoint client-side web part

The Office UI Fabric is the official front-end framework for building experiences in Office 365 and SharePoint. Developers can customize and override the default styles, designs, and components to tailor the solution's needs.

There are two parts of the Office UI Fabric that are available for use by developers:

- **Office UI Fabric Core⁴**: A set of core styles, typography, a responsive grid, animations, icons, and other fundamental building blocks of the overall design language.
- **Office UI Fabric React⁵**: A set of React components built on top of the Fabric design language for use in React-based projects.

Using Office UI Fabric Core and Fabric React in SharePoint Framework

SharePoint provides seamless integration with Fabric that enables Microsoft to deliver a robust and consistent design language across various SharePoint experiences such as modern team sites, modern pages, and modern lists. Additionally, the Office UI Fabric is available for developers in the SharePoint Framework when building custom SharePoint solutions.

⁴ <https://developer.microsoft.com/en-us/fabric>

⁵ <https://developer.microsoft.com/en-us/fabric>

Microsoft uses Fabric Core and Fabric React in SharePoint. Microsoft regularly pushes updates to SharePoint Online that could also update the Fabric Core and Fabric React versions as well. These updates could potentially conflict with third-party customer solutions built with previous versions of Fabric Core and Fabric React, which could cause exceptions in those customizations. The primary reason for these types of breaks is the use of Global CSS styles in both Fabric libraries. Avoid such conflicts at all costs.

To achieve reliability, one of the main problems developers need to solve is Global CSS styles. This results from not using global class names in the HTML markup and instead using Fabric Core mixins and variables in the Sass declaration file. This involves importing the Fabric Core's Sass declarations into your Sass file and then consuming the variables and mixins appropriately.

Office UI Fabric Core package

The SharePoint Framework Fabric Core npm package ([@microsoft/sp-office-ui-fabric-core⁶](https://www.npmjs.com/package/@microsoft/sp-office-ui-fabric-core)) contains a subset of supported Fabric Core styles that can be safely consumed within a SharePoint Framework component.

The following core styles are supported in the package:

- Typography
- Layouts
- Colors
- Themes
- Localization

The following are not yet supported in the package:

- Animations
- Icons

Starting with the SharePoint Framework Yeoman generator version 1.3.4, the default project (web parts and extensions) templates come set up with the new [@microsoft/sp-office-ui-fabric-core](https://www.npmjs.com/package/@microsoft/sp-office-ui-fabric-core) package and consume core styles from the package instead of using global CSS styles.

Update existing projects

To use the Fabric Core package in your existing project, install the package as a dependency:

```
npm install @microsoft/sp-office-ui-fabric-core --save
```

After it's installed, you can then import the Fabric Core Sass declarations in your Sass definition file and use the mixins and variables as described in the following section.

Use Fabric Core styles

To use the Fabric Core styles, you need to import the SPFabricCore declarations into your Sass file.

Note:

Make sure you have installed the [@microsoft/sp-office-ui-fabric-core npm](https://www.npmjs.com/package/@microsoft/sp-office-ui-fabric-core) package.

⁶ <https://www.npmjs.com/package/@microsoft/sp-office-ui-fabric-core>

```
@import '~@microsoft/sp-office-ui-fabric-core/dist/sass/SPFabricCore.scss';
```

Now you can use the core styles as mixins and variables.

```
.row {  
  @include ms-Grid-row;  
  @include ms-fontColor-white;  
  background-color: $ms-color-themeDark;  
  padding: 20px;  
}
```

Office UI Fabric React components

Office UI Fabric React is the front-end framework for building experiences for Office and Office 365. It includes a robust collection of responsive, mobile-first components that make it easy for you to create web experiences by using the Office Design Language.

- **Fabric website**⁷: Contains detailed API documentation along with implementation code examples for each control.
- **API reference**⁸: Contains detailed API reference documentation.
- **Office UI Fabric React**⁹: A source code repository on GitHub.

Use Office UI Fabric React

We recommend that you use the versions of the Office UI Fabric React package included in the project in the SharePoint Framework's Yeoman generator. For instance, the SharePoint Framework v1.7.0 release uses Fabric React v5.131.0

Note:

Fabric React versions 2.x or older are not supported in SharePoint Framework.

After the Fabric React package is installed, you can import the required components from the Fabric React bundle.

```
//import Button component from Fabric React Button bundle  
import { Button } from 'office-ui-fabric-react/lib/Button';  
//use it in your component  
render() {  
  ...  
  <div>  
    <Button>click me</Button>  
  </div>  
  ...  
}
```

The Fabric React package includes the supported Fabric Core styles used in the Fabric React components. Microsoft recommends that you import the Fabric Core styles from the Fabric React package instead of

⁷ <https://developer.microsoft.com/fabric>

⁸ <https://docs.microsoft.com/javascript/api/office-ui-fabric-react?branch=live&view=office-ui-fabric-react-latest>

⁹ <https://github.com/OfficeDev/office-ui-fabric-react>

from the **@microsoft/sp-office-ui-fabric-core** package to ensure that the right styles are used in your component.

Because the **@microsoft/sp-office-ui-fabric-core** package is already installed in your solution by the Yeoman generator, Microsoft recommends that you uninstall that package if you decide to use Fabric components and reduce your component bundle size.

```
npm uninstall @microsoft/sp-office-ui-fabric-core --save
```

You can then import the core styles from the Sass declarations available in the Fabric React package.

```
@import '~office-ui-fabric-react/dist/sass/_References.scss';
```

Understand the approach and its limitations

Fabric components in your solution are locked to the specific version of Fabric React that you installed. You need to update the Fabric React package to get any new components if they are available in a newer package version. Because the Fabric components are included in the component bundle, it may increase the size of your component bundle. However, this approach is the only approach officially supported when Office UI Fabric React is being used in SharePoint Framework solutions.

The CSS challenge with Office UI Fabric

The following concepts and references provide insights on the challenge with using Office UI Fabric in the context of client-side web parts.

Global CSS styles

How to avoid Global CSS styles at all costs is a big problem. Today, both Fabric Core and Fabric React have global styles. A lack of any native solutions from the browser to manage the style scoping makes this a very difficult problem.

- **Scope CSS** is in early stages of discussion.
- **iFrames** are not a good option to isolate styles.
- **Web Components** is another standard that talks about scoped styles but is still in the discussion stage.
- **The React: CSS in JS** discussion explains the problem well.

CSS specificity

Higher CSS specificity styles override lower specificity styles, but the key thing to understand is how the specificity rules apply. In general, the precedence order from highest to lowest is as follows:

- The style attribute (for example, `style="background:red;"`).
- ID selectors (`#myDiv{}`).
- Class selectors (`.myCssClass{}`), attribute selectors (`[type="radio"]`), and pseudo-classes (`:hover`).
- Type selectors (`h1`).

Loading order

If two classes are applied on an element and have the same specificity, the class that is loaded later takes precedence. As shown in the following code, the button appears red. If the order of the style tags is changed, the button appears green. This is an important concept, and if not used correctly, the user experience can become load order-dependent, making them inconsistent.

```
<style>
    .greenButton { color: green; }
</style>
<style>
    .redButton { color: red; }
</style>
<body>
    <button class="greenButton redButton"></button>
</body>
```

Single part app pages

Single part app pages provide the ability to host SharePoint Framework web parts or Teams applications in SharePoint Online with a locked layout. End users cannot modify or configure a page that uses the Single Part App Page layout.

App pages have following characteristics:

- Single Part App Pages cannot be edited by end users using a browser.
- Currently support hosting only single web part or Microsoft Teams application.
- Page layout can only be changed programmatically from normal page layout to a Single Page App Page.
- End-users cannot parameterize exposed web part or Teams application.

A web part can be configured to be exposed as an app page. This configuration is performed in the web part manifest file by adjusting the **supportedHosts** value. Web part will be exposed as an option in the upcoming app pages picker user interface, if the **supportedHosts** value contains **SharePointFullPage** value.

The following JSON web part manifest demonstrates a scenario where the web part is included in all supported platforms by updating all different values for the **supportedHosts** parameter.

```
{
    "$schema": "https://developer.microsoft.com/json-schemas/spfx/client-
    side-web-part-manifest.schema.json",
    "id": "eb7ac2da-d8eb-4118-9f4f-19ce595d3ad3",
    "alias": "AllPlatformsWebPart",
    "componentType": "WebPart",
    // The "*" signifies that the version should be taken from the package.
    json
        "version": "*",
        "manifestVersion": 2,
        // If true, the component can only be installed on sites where Custom
        Script is allowed.
        // Components that allow authors to embed arbitrary script code should
```

```
set this to true.  
// https://support.office.com/en-us/article/Turn-scripting-capabilities-  
on-or-off-1f2c515f-5d7e-448a-9fd7-835da935584f  
"requiresCustomScript": false,  
"supportedHosts": ["SharePointWebPart", "SharePointFullPage", "Team-  
sTab"],  
"preconfiguredEntries": [{  
    "groupId": "5c03119e-3074-46fd-976b-c60198311f70", // Other  
    "group": { "default": "Other" },  
    "title": { "default": "All Platforms" },  
    "description": { "default": "This web part is visible in all plat-  
forms" },  
    "officeFabricIconFontName": "Page",  
    "properties": {  
        "description": "allPlatforms"  
    }  
}]  
}
```

Use the Single Part App page in your tenant

You will need to perform the following steps to enable the Single Part App Page layout in your SharePoint site.

- Create a new page.
- Add a web part on the page and configure that as needed.
- Change the page layout type as **SingleWebPartAppPage**.

If you need to further modify the page, you can change the page layout back to **as Article** to enable content editing.

Change page layout using JavaScript in browser console

You can change an existing page to use Single Page App Page layout by using browser developer tools. You can simply enable the developer tools and execute the following code to change an existing page to use a **SingleWebPartAppPage** layout.

You will need to adjust the tenant and page name based on your environment.

```
var siteUrl = 'https://contoso.sharepoint.com/sites/marketing/';  
var pageUrl = 'SitePages/page.aspx'  
fetch(siteUrl + '_api/contextinfo', {  
    method: 'POST',  
    headers: {  
        accept: 'application/json;odata=nometadata'  
    }  
})  
.then(function (response) {  
    return response.json();  
})  
.then(function (ctx) {
```

```
        return fetch(siteUrl + "_api/web/getfilebyurl('" + pageUrl + "')/ListItemAllFields", {
            method: 'POST',
            headers: {
                accept: 'application/json;odata=nometadata',
                'X-HTTP-Method': 'MERGE',
                'IF-MATCH': '*',
                'X-RequestDigest': ctx.FormDigestValue,
                'content-type': 'application/json;odata=nometadata',
            },
            body: JSON.stringify({
                PageLayoutType: "SingleWebPartAppPage"
            })
        })
    })
    .then(function(res) {
        console.log(res.ok ? 'DONE' : 'Error: ' + res.statusText);
    });
}
```

Change page layout using PnP PowerShell

You can also use **PnP PowerShell**¹⁰ to update the page layout for the existing page with the following script.

Note: PnP PowerShell is an open-source tool with active community providing support for it. Microsoft provides no SLA for the open-source tool support.

You will need to adjust the tenant and page name based on your environment.

```
Connect-PnPOnline -Url https://contoso.sharepoint.com/sites/marketing
$item2 = Get-PnPListItem -List "Site Pages" -Query
"<View><Query><Where><Eq><FieldRef Name='FileLeafRef' /><Value
Type='Text'>page.aspx</Value></Eq></Where></Query></View>"
$item2["PageLayoutType"] = "SingleWebPartAppPage"
$item2.Update()
Invoke-PnPQuery
```

With the SharePoint Framework, you can use modern web technologies and tools in your preferred development environment to build productive experiences and apps that are responsive and mobile-ready.

Lesson review questions

1. What is SharePoint Framework (SPFx) and why should you use it?
2. What purpose does the web part properties pane play in allowing you to configure options for custom experiences?
3. When should you use a single part app page vs. a web part?

¹⁰ <https://docs.microsoft.com/en-us/powershell/sharepoint/sharepoint-pnp/sharepoint-pnp-cmdlets?view=sharepoint-ps>

Correct/suggested answers:

1. The SharePoint Framework (SPFx) is a page and web part model that provides full support for client-side SharePoint development, easy integration with SharePoint data, and support for open source tooling. With the SharePoint Framework, you can use modern web technologies and tools in your preferred development environment to build productive experiences and apps that are responsive and mobile-ready.
 - SharePoint was launched as an on-premises product in 2001. Over time, a large developer community has extended and shaped it in many ways. For the most part, the developer community followed the same patterns and practices that the SharePoint product development team used, including web parts, SharePoint feature XML, and more. Many features were written in C#, compiled to DLLs, and deployed to on-premises farms. That architecture worked well in environments with only one enterprise, but it didn't scale to the cloud, where multiple tenants run side-by-side. As a result, we introduced two alternative models: client-side JavaScript injection, and SharePoint Add-ins. Both of these solutions have pros and cons.
2. The property pane is where you can define properties to customize your web part. The property pane is client-side driven and provides a consistent design across SharePoint. The property pane properties are defined in **propertyPaneSettings**. A property pane has three key metadata: a page, an optional header, and at least one group. Pages provide you the flexibility to separate complex interactions and put them into one or more pages. Pages contain a header and groups. Headers allow you to define the title of the property pane. Groups allow you to define the various sections or fields for the property pane through which you want to group your field sets.
3. App pages are exposed as an option in the modern page creation capability when you chose the used page layout for your page. All web parts you configure to be available as an app page are also available on the Create page capability.
 - Web part can be configured to be exposed as an app page. This configuration is performed in the web part manifest file by adjusting the **supportedHosts** value. Web part will be exposed as an option in the upcoming app pages picker user interface, if the **supportedHosts** value contains "**SharePointFullPage**" value.

SharePoint Framework Extension

Lesson introduction

SharePoint Framework (SPFx) extensions are client-side components that run inside the context of a SharePoint page. You can deploy extensions to SharePoint Online, and you can use modern JavaScript tools and libraries to build them.

With SPFx extensions, you can customize more facets of the SharePoint experience, including notification areas, toolbars, and list data views. SPFx extensions are available in all Office 365 subscriptions for production usage.

In this lesson, you'll learn about SPFx extensions including the Application Customizer extension, the ListView Command Set extension, and the Field Customizer extension.

After this lesson, you should be able to:

- Identify the appropriate tool to create an SPFx extension project.
- Describe page placeholders from the Application Customizer.
- Explain the ListView Command Set extension.
- Describe the Field Customizer extension.

SharePoint Framework extensions

SharePoint Framework extensions enable you to extend the SharePoint user experience within modern pages and document libraries, while using the familiar SharePoint Framework tools and libraries for client-side development. Specifically, the SharePoint Framework includes three new extension types:

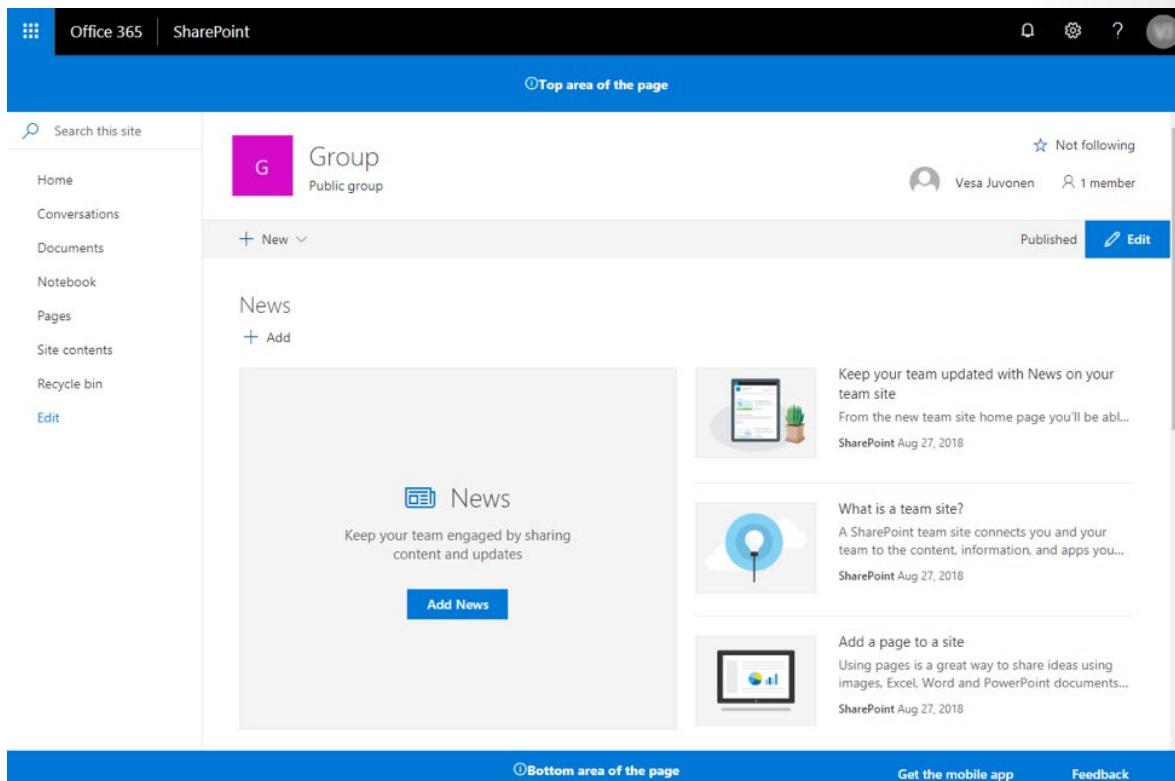
- **Application Customizers:** Adds scripts to the page and accesses well-known HTML element placeholders and extends them with custom renderings.
- **Command Sets:** Extends the SharePoint command surfaces to add new actions and provides client-side code that you can use to implement behaviors.
- **Field Customizers:** Provides modified views to data for fields within a list.

You can build extensions alongside common scripting frameworks, such as AngularJS and React, in addition to plain JavaScript projects. For example, you can use React along with components from Office UI Fabric React to create experiences based on the same components used in Office 365.

The Application Customizer extension

Application Customizers

Application Customizers provide access to well-known locations on SharePoint pages that you can modify based on your business and functional requirements. For example, you can create dynamic header and footer experiences that render across all the pages in SharePoint Online.



This model is similar to using a **UserCustomAction** collection in a Site or Web object to modify the page experience via custom JavaScript. The key difference or advantage with SPFx extensions is that your page elements won't change if changes are made to the HTML/Document Object Model (DOM) structure in SharePoint Online.

Get access to page placeholders

Application Customizer extensions are supported with Site, Web, and List scopes. You can control the scope by deciding where or how the Application Customizer is registered in your SharePoint tenant.

Note:

Feature xml-based registration of Application Customizer is only supported with web or list level. You can however activate Application Customizer more widely either using tenant wide deployment of extensions capability or by associating Application Customizer to the **UserCustomAction** collection in Site object level.

When the Application Customizer exists in the scope and is being rendered, you can use the following method to get access to the placeholder.

```
// Handling the Bottom placeholder
if (!this._bottomPlaceholder) {
    this._bottomPlaceholder =
        this.context.placeholderProvider.tryCreateContent(
            PlaceholderName.Bottom,
            { onDispose: this._onDispose });
}
...
```

After you get the placeholder object, you have full control over what is presented to the end user.

Notice that you're requesting a well-known placeholder by using the corresponding well-known identifier. In this case, the code is accessing the footer section of the page by using the Bottom identifier.

Use Yeoman to Generate an Application Customizer Project

Following are two examples of using Yeoman to generate an Application Customizer project.

The image below is an example of an Application Customizer project being generated via Yeoman.

- Select **Extension** as the client-side component type to be created.
- Select **Application Customizer** as the extension type to be created.

HelloWorldApplicationCustomizer.manifest.json is an example of a necessary file that defines your extension type and a unique identifier for your extension. This unique identifier is needed when debugging and deploying your extension to SharePoint.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/spfx/client-side-extension-manifest.schema.json",
  "id": "05966ad1-55c7-47f6-9ff5-4fee8bff838a",
  "alias": "HelloWorldApplicationCustomizer",
  "componentType": "Extension",
  "extensionType": "ApplicationCustomizer",
  //The "*" signifies that the version should be taken from the package.
  json
  "version": "*",
  "manifestVersion": 2,
  //If true, the component can only be installed on sites where Custom
  Script is allowed.
  // Components that allow authors to embed arbitrary script code should
  set this to true.
  "requiresCustomScript": false
}
```

Code the Application Customizer

When you code your Application Customizer, you'll open the TypeScript (TS) file such as **HelloWorldApplicationCustomizer.ts** in the **src\extensions\helloWorld** folder.

Notice that the base class for the Application Customizer is imported from the **sp-application-base** package, which contains SharePoint framework code required by the Application Customizer.

```
import { override } from '@microsoft/decorators';
import { Log } from '@microsoft/sp-core-library';
import{
  BaseApplicationCustomizer
} from '@microsoft/sp-application-base';
```

```
import { Dialog } from '@microsoft/sp-dialog';
```

The logic for your Application Customizer is contained in the **onInit** method, which is called when the client-side extension is first activated on the page. This event occurs after **this.context** and **this.properties** are assigned. As with web parts, **onInit()** returns a promise that you can use to perform asynchronous operations.

Note:

The class constructor is called at an early stage, when **this.context** and **this.properties** are undefined. Custom initiation logic is not supported here.

The following are the contents of **onInit()** in the default solution. This default solution writes a log to the Dev Dashboard, and then displays a simple JavaScript alert when the page renders.

```
@override
public onInit(): Promise<void> {
    Log.info(LOG_SOURCE, 'Initialized ${strings.Title}');

    let message: string = this.properties.testMessage;
    if (!message) {
        message = '(No properties were provided.)';
    }

    Dialog.alert(`Hello from ${strings.Title}:\n\n${message}`);

    return Promise.resolve();
}
```

Note:

SharePoint Framework Dev Dashboard is an additional UI dashboard, which can be started with CTRL+F12. This is developer-oriented logging information, which you can take advantage as developer.

If your Application Customizer uses the **ClientSideComponentProperties** JSON input, it is deserialized into the **BaseExtension.properties** object. You can define an interface to describe it. The default template looks for a property called **testMessage**. If that property is provided, it outputs it in an alert message.

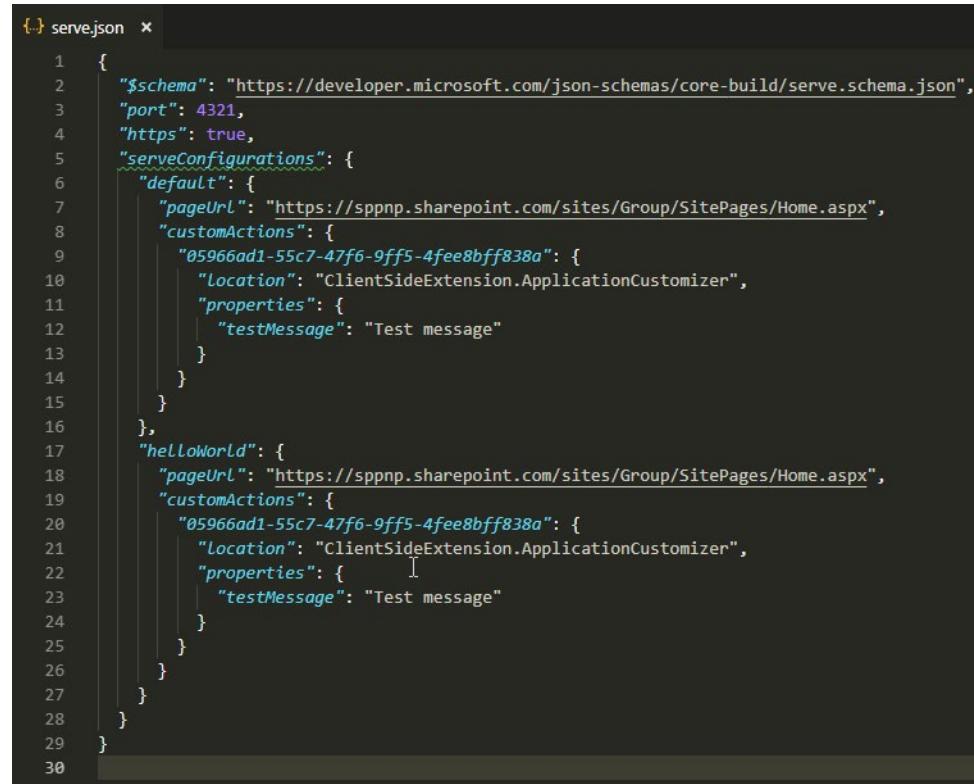
Debug the Application Customizer

You cannot use the local Workbench to test SharePoint Framework extensions. You need to test and develop them directly against a live SharePoint Online site. For debugging, you don't have to deploy your customization to the app catalog.

Follow these steps for debugging:

1. Open the **serve.json** file in the **config** folder. This file will be updated with the default settings matching your project. Notice that there's a specific GUID mentioned under the **customActions** element. This is automatically updated to match your component when the project is scaffolded. If you end up adding new components or change the properties for the component, you will need to update this file for testing.

2. Update **pageURL** to match your own tenant, which you want to use for testing. You can use any URL with modern experience. This could be, for example, a welcome page of a new group associated team site, such: <https://sppnp.sharepoint.com/sites/yoursite/SitePages/Home.aspx>
3. Your **serve.json** file would be similar to the following (updated with your tenant details):



```
{-> serve.json *}
1  {
2    "$schema": "https://developer.microsoft.com/json-schemas/core-build/serve.schema.json",
3    "port": 4321,
4    "https": true,
5    "serveConfigurations": {
6      "default": {
7        "pageUrl": "https://sppnp.sharepoint.com/sites/Group/SitePages/Home.aspx",
8        "customActions": {
9          "05966ad1-55c7-47f6-9ff5-4fee8bff838a": {
10            "location": "ClientSideExtension.ApplicationCustomizer",
11            "properties": {
12              "testMessage": "Test message"
13            }
14          }
15        }
16      },
17      "helloWorld": {
18        "pageUrl": "https://sppnp.sharepoint.com/sites/Group/SitePages/Home.aspx",
19        "customActions": {
20          "05966ad1-55c7-47f6-9ff5-4fee8bff838a": {
21            "location": "ClientSideExtension.ApplicationCustomizer",
22            "properties": {
23              "testMessage": "Test message"
24            }
25          }
26        }
27      }
28    }
29  }
30 }
```

4. Switch to your console, ensuring you are still in the **app-extension** directory, and then enter the following command: `gulp trust-dev-cert`

Note:

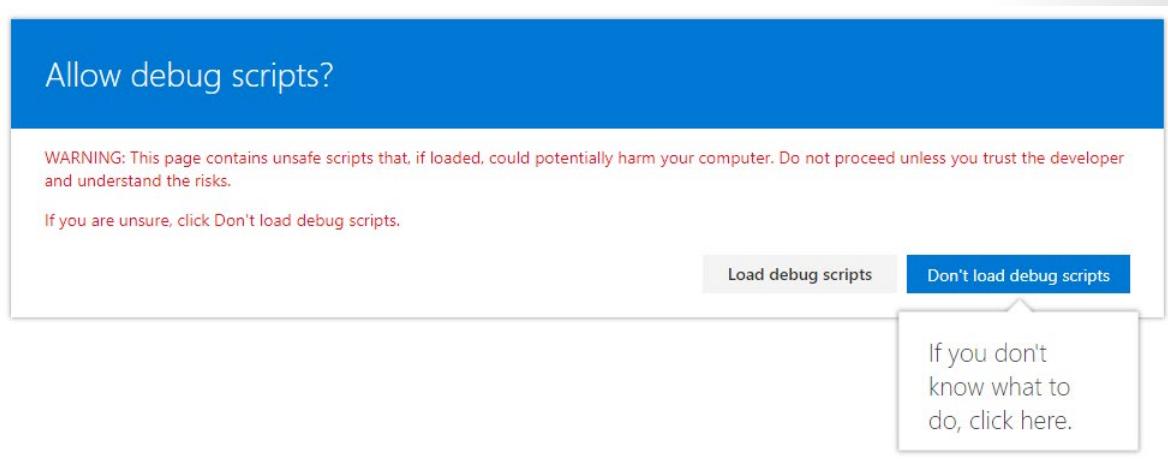
The developer certificate must be installed **ONLY once in your development environment**, so you can skip this step if you have already executed that in your environment.

5. Compile your code and host the compiled files from your local computer by running the following command: `gulp serve`

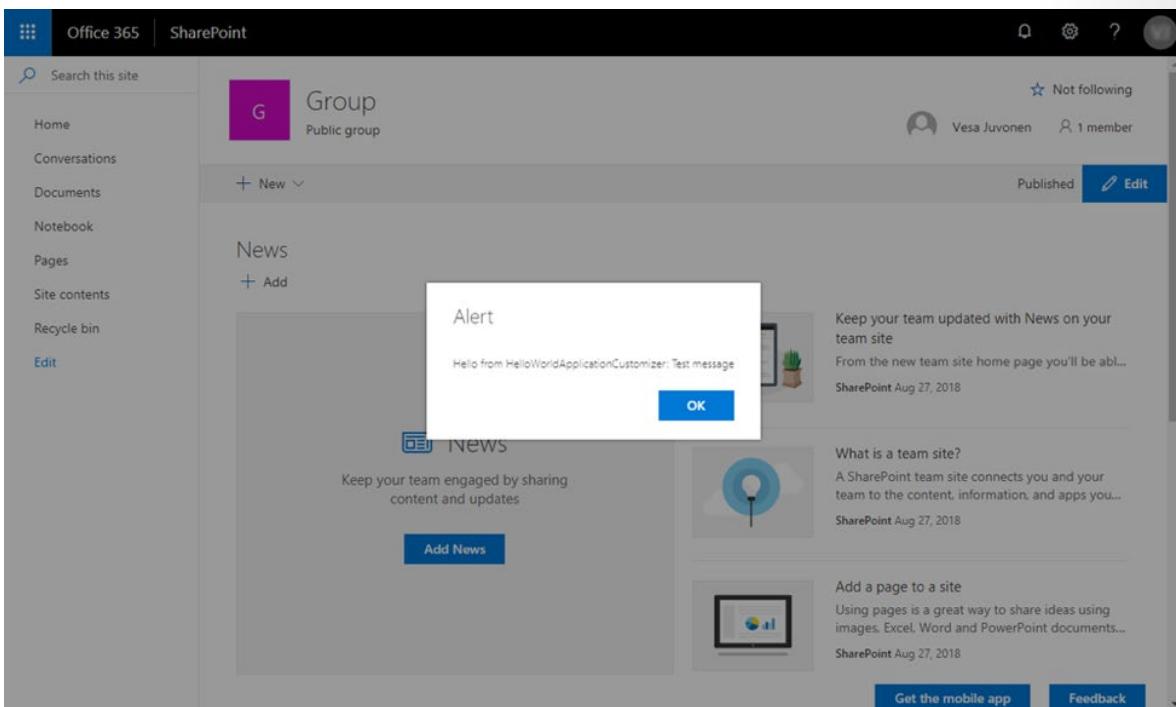
If you do not have the SPFx developer certificate installed, Workbench notifies you that it is not configured to load scripts from localhost. If this happens, stop the process that is currently running in the console window, run the `gulp trust-dev-cert` command in your project directory console to install the developer certificate, and then run the `gulp serve -nobrowser` command again. This process is documented in the [Set up your development environment¹¹](#) article.

1. When the code compiles without errors, it serves the resulting manifest from <https://localhost:4321> and also starts your default browser with needed query parameters.
2. Move to your browser and select **Load debug scripts** to continue loading scripts from your local host.

¹¹ <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/set-up-your-development-environment>



3. The dialog message will appear on your page:



This dialog is thrown by your SharePoint Framework Extension. Note that because you provided the **testMessage** property as part of the debug query parameters, it's included in the alert message. You can configure your extension instances based on the client component properties, which are passed for the instance in runtime mode.

Note:

If you have issues with debugging, double-check the **pageTitle** setting in the **serve.json** file.

Test your code

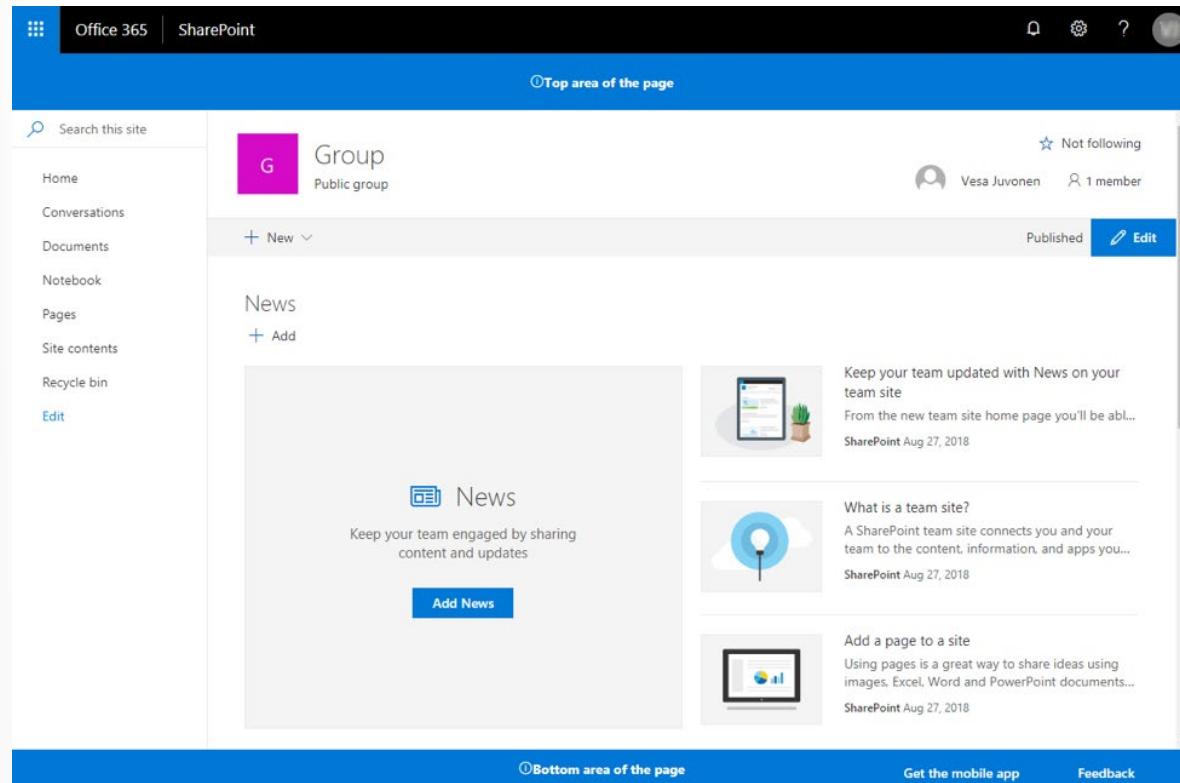
When testing your extension, follow the steps below:

1. Move to the **serve.json** file in the **config** folder. You must adjust the settings because Microsoft introduced two new properties for the extension. Update the properties section in the file to have *Top* and *Bottom* messages.

```
...  
"properties": {  
    "Top": "Top area of the page",  
    "Bottom": "Bottom area of the page"  
}  
...
```

2. Switch to the console window running `gulp serve` and check for errors. Gulp reports any errors in the console; you'll need to fix them before you proceed. If you have the solution already running, restart it to apply the updated settings from the **serve.json** file: `gulp serve`
3. Select **Load debug scripts** to continue loading scripts from your local host.

4. The custom header and footer content will appear in your page:

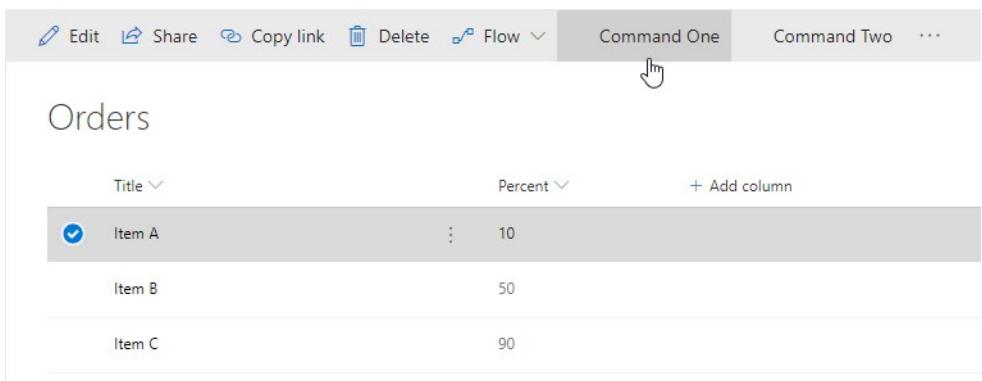


The ListView Command Set extension

Command Sets

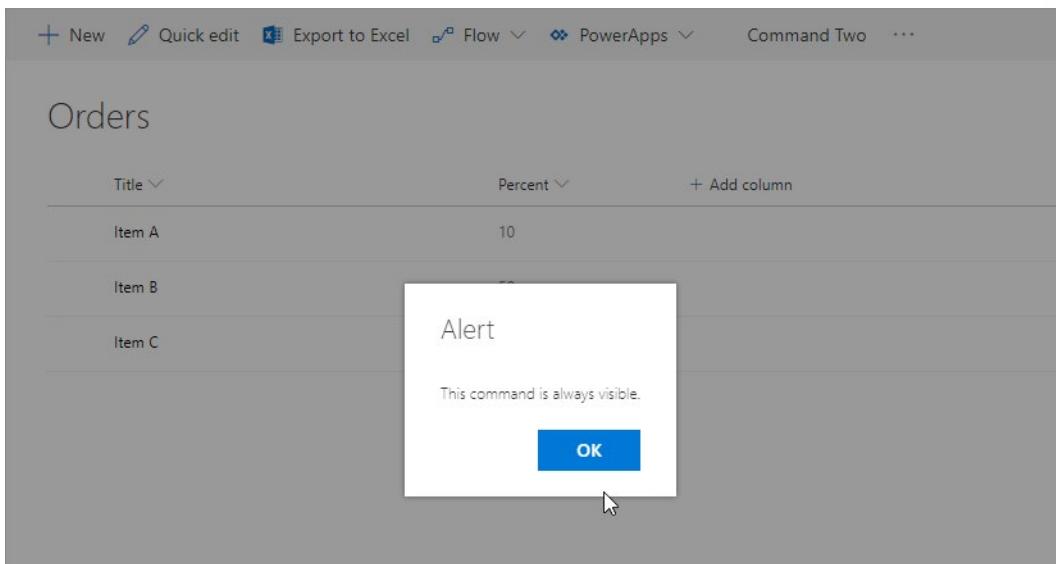
Command Set extensions provide the ability to add new actions that execute client-side code to implement behavior to the SharePoint command bar.

The image below shows a simple example of a ListView Command Set extension deployed with new actions added. These actions are named **Command One** and **Command Two**.



A screenshot of a SharePoint list view titled "Orders". The list contains three items: Item A (Title), Item B (Title), and Item C (Title). To the right of each item is a "Percent" column with values 10, 50, and 90 respectively. Above the list, a horizontal toolbar includes standard actions like Edit, Share, Copy link, Delete, Flow, and a "Command One" button, which is highlighted. Below the toolbar, there is a "Command Two" button and an ellipsis (...).

When the Command One button is selected, the client-side code executes to display a dialog alert as shown in the image below.

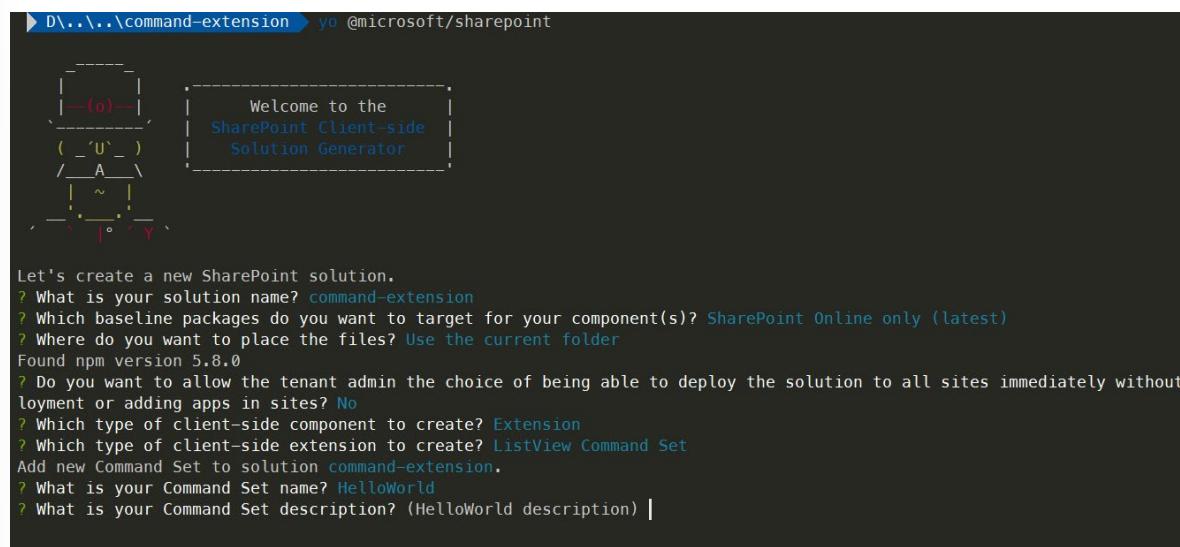


A screenshot of the same SharePoint list view as above, but with an alert dialog box overlaid. The dialog has a title "Alert" and a message "This command is always visible." It features a blue "OK" button at the bottom. The background list is partially visible behind the dialog.

Use Yeoman to Generate Command Set Extension Project

When generating a Command Set extension project via Yeoman, you will be prompted to enter values to create the solution..

- Select **Extension** as the client-side component type to be created.
- Select **ListView Command Set** as the extension type to be created.



D:\...\command-extension> yo @microsoft/sharepoint

Welcome to the
SharePoint Client-side
Solution Generator

Let's create a new SharePoint solution.

? What is your solution name? command-extension

? Which baseline packages do you want to target for your component(s)? SharePoint Online only (latest)

? Where do you want to place the files? Use the current folder

Found npm version 5.8.0

? Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without
deployment or adding apps in sites? No

? Which type of client-side component to create? Extension

? Which type of client-side extension to create? ListView Command Set

Add new Command Set to solution command-extension.

? What is your Command Set name? HelloWorld

? What is your Command Set description? (HelloWorld description) |

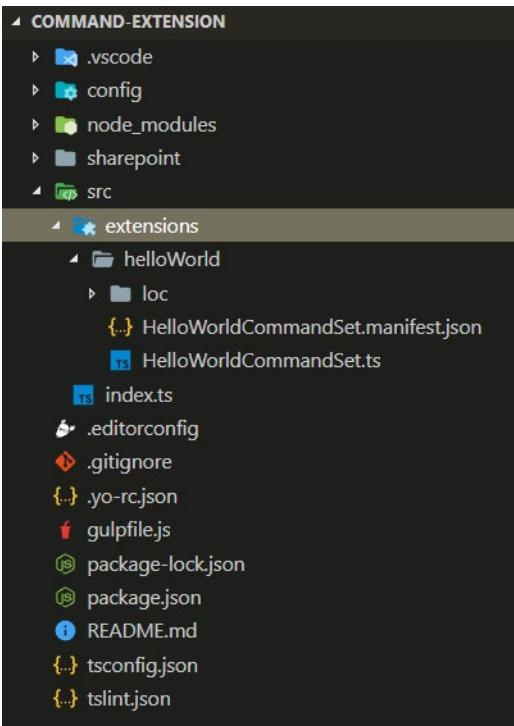
You'll receive a success message indicating the scaffolding has successfully completed. The image below shows what will appear.



_+#####!
#####|
###/ (#|(@) | Congratulations!
#####| \ | Solution command-extension is created.
###/ /###| (@) | Run gulp serve to play with it!
#####| #| / |
###| /##|(@)
#####|
**=+####!

D:\...\command-extension>

Notice that the default solution structure is similar to the solution structure of client-side web parts. This is the basic SharePoint Framework solution structure, with similar configuration options across all solution types.



The **HelloWorldCommandSet.manifest.json** file in the **src\extensions\helloWorld** folder defines the extension type and a unique identifier id for the extension.

You'll need this unique identifier later when debugging and deploying your extension to SharePoint.

Notice the actual command definitions in the manifest file. These are the actual buttons that are exposed based on the registration target. In the default template, you find two different buttons: **Command One** and **Command Two**.

```
...
"items": {
    "COMMAND_1": {
        "title": { "default": "Command One" },
        "iconImageUrl": "icons/request.png",
        "type": "command"
    },
    "COMMAND_2": {
        "title": { "default": "Command Two" },
        "iconImageUrl": "icons/cancel.png",
        "type": "command"
    }
}
...

```

Currently, images are not properly referenced unless you are referring to them from absolute locations in a Content Delivery Network (CDN) within your manifest.

Code the ListView Command Set

When you code your ListView command set, you'll open the TS file such as **HelloWorldCommandSet.ts** in the **src\extensions\helloWorld** folder.

Notice in the code below that the base class for the ListView Command Set is imported from the **sp-list-view-extensibility** package, which contains SharePoint Framework code required by the ListView Command Set.

```
import { override } from '@microsoft/decorators';
import { Log } from '@microsoft/sp-core-library';
import {
  BaseListViewCommandSet,
  Command,
  IListViewCommandSetListViewUpdatedParameters,
  IListViewCommandSetExecuteEventParameters
} from '@microsoft/sp-listview-extensibility';
import { Dialog } from '@microsoft/sp-dialog';
```

The behavior for your custom buttons is contained in the **onListViewUpdated()** and **OnExecute()** methods.

The **onListViewUpdated()** event occurs separately for each command (for example, a menu item) whenever a change happens in the ListView, and the UI needs to be re-rendered. The event function parameter represents information about the command being rendered. The handler can use this information to customize the title or adjust the visibility, for example, if a command should only be shown when a certain number of items are selected in the list view. This is the default implementation.

When using the method **tryGetCommand**, you get a Command object, which is a representation of the command that shows in the UI. You can modify its values, such as title, or visible, to modify the UI element. SPFx uses this information when re-rendering the commands. These objects keep the state from the last render, so if a command is set to **visible = false**, it remains invisible until it is set back to **visible = true**.

```
@override
public onListViewUpdated(event: IListViewCommandSetListViewUpdatedParameters): void {
  const compareOneCommand: Command = this.tryGetCommand('COMMAND_1');
  if (compareOneCommand) {
    // This command should be hidden unless exactly one row is
    selected.
    compareOneCommand.visible = event.selectedRows.length === 1;
  }
}
```

The **OnExecute()** method defines what happens when a command is executed (for example, the menu item is selected). In the default implementation, different messages are shown based on which button was selected.

```
@override
public onExecute(event: IListViewCommandSetExecuteEventParameters): void {
  switch (event.itemId) {
    case 'COMMAND_1':
      Dialog.alert(`\$ {this.properties.sampleTextOne}`);
```

```
        break;
    case 'COMMAND_2':
        Dialog.alert(` ${this.properties.sampleTextTwo} `);
        break;
    default:
        throw new Error('Unknown command');
    }
}
```

Debug the ListView Command Set

Just as it was detailed for debugging Application Customizer extensions earlier in this lesson, the same applies for ListView Command Set extensions. You cannot use the local Workbench to test SharePoint Framework extensions. You need to test and develop them directly against a live SharePoint Online site. For debugging, you don't need to deploy your customization to the app catalog.

The steps below are listed to show the process for testing and debugging:

1. Go to any SharePoint list in your SharePoint Online site by using the modern experience or create a new list. Copy the URL of the list to your clipboard as you will need it in the following step.
2. Because our ListView Command Set is hosted from localhost and is running, you can use specific debug query parameters to execute the code in the list view. Open the **serve.json** file from the **config** folder. Update the **pageUrl** attributes to match a URL of the list where you want to test the solution. After edits your **serve.json** appears more like the following:

```
{
    "$schema": "https://developer.microsoft.com/json-schemas/core-build/serve.schema.json",
    "port": 4321,
    "https": true,
    "serveConfigurations": {
        "default": {
            "pageUrl": "https://sppnp.sharepoint.com/sites/Group/Lists/Orders/AllItems.aspx",
            "customActions": {
                "bf232d1d-279c-465e-a6e4-359cb4957377": {
                    "location": "ClientSideExtension.ListViewCommandSet.CommandBar",
                    "properties": {
                        "sampleTextOne": "One item is selected in the list",
                        "sampleTextTwo": "This command is always visible."
                    }
                }
            }
        },
        "helloWorld": {
            "pageUrl": "https://sppnp.sharepoint.com/sites/Group/Lists/Orders/AllItems.aspx",
            "customActions": {
                "bf232d1d-279c-465e-a6e4-359cb4957377": {
                    "location": "ClientSideExtension.ListViewCommandSet.CommandBar",
                    "properties": {
                        "sampleTextOne": "One item is selected in the list",
                        "sampleTextTwo": "This command is always visible."
                    }
                }
            }
        }
}
```

```
        }
    }
}
}
```

3. Compile your code and host the compiled files from the local machine by running this command:
`gulp serve`
4. When the code compiles without errors, it serves the resulting manifest from `https://localhost:4321`. This will also start your default browser within the URL defined in `serve.json` file. In Windows, you can control which browser window is used by activating the browser you want before executing this command.
5. Accept the loading of debug manifests by selecting **Load debug scripts** when prompted.

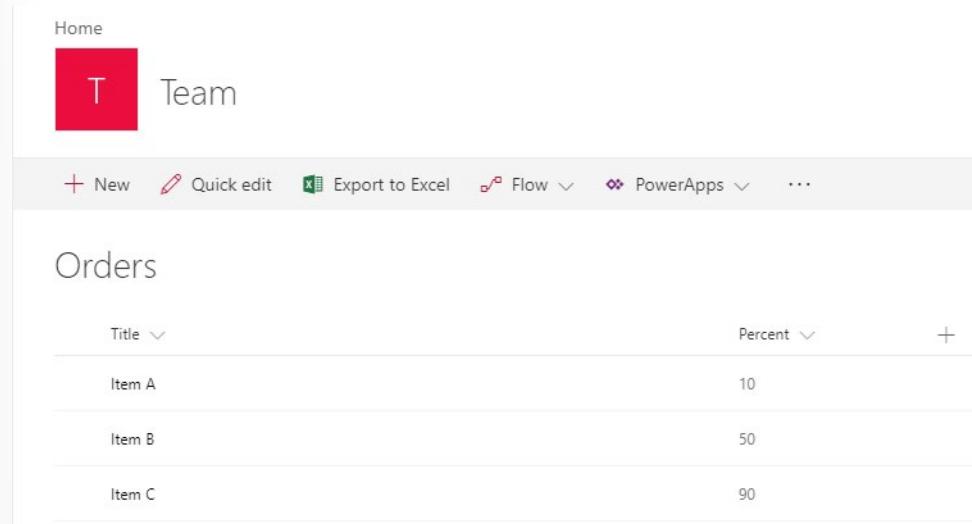
Command set extensions provide the ability to add new actions that execute client-side code to implement behavior to the SharePoint command bar.

The Field Customizer extension

Field Customizers

In classic SharePoint, client-side rendering (JSLink) was used to customize the rendering of fields on list view pages. To customize the rendering of fields in modern SharePoint, Field Customizers are a good replacement for the JSLink customization of fields.

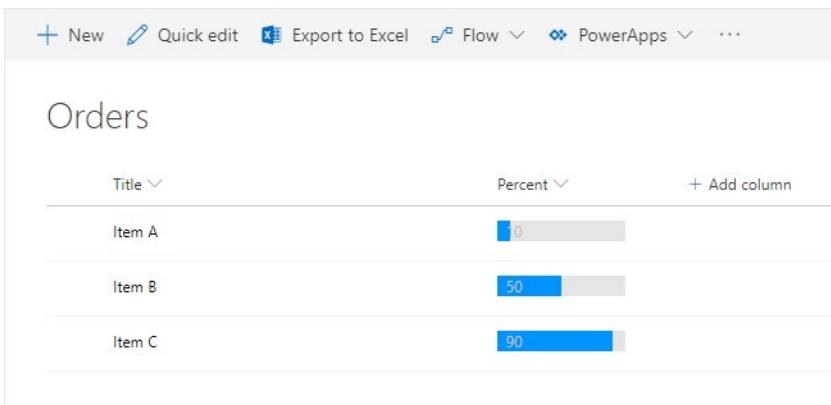
For example, you may have a number field that you want to display as a percent progress bar. This sample image shows a list called **Orders** that is rendering the Percent number fields as-is in SharePoint Online.



The screenshot shows a SharePoint list titled "Orders". The list contains three items: "Item A" with a value of 10, "Item B" with a value of 50, and "Item C" with a value of 90. The "Percent" column is displayed as plain text. The SharePoint ribbon navigation bar is visible at the top, showing "Home" and "Team". The list header includes standard actions like "New", "Quick edit", "Export to Excel", "Flow", "PowerApps", and a more options menu.

Title	Percent
Item A	10
Item B	50
Item C	90

You can override the list field rendering by creating and deploying a SPFx Field customizer extension. Below is a sample image showing the Percent field now rendering a progress bar for the value instead of just the number text.

**Note:**

One of the main benefits of migrating old-school JSLink customizations to the SharePoint Framework is that it is the only technique supported on modern sites for customizing the UI of "modern" lists and libraries. The rendering infrastructure and enabled no-script flag on "modern" sites means you can't rely on old-school JSLink customizations. Thus, if you really want to extend the "modern" UI, you need to upgrade to the SharePoint Framework.

Use Yeoman to Generate a Field Extension Project

To generate a Field extension project via Yeoman:

- Select **Extension** as the client-side component type to be created.
- Select **Field Customizer** as the extension type to be created.

```
▶ D:\...\field-extension> yo @microsoft/sharepoint

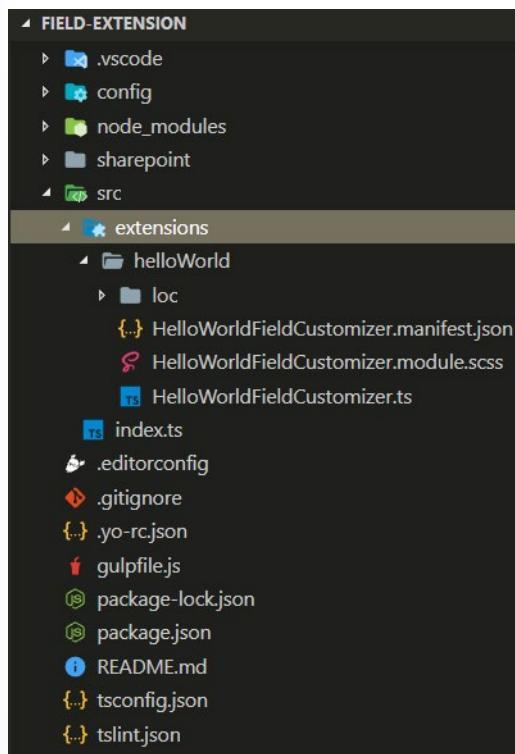
  Welcome to the
  SharePoint Client-side
  Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? field-extension
? Which baseline packages do you want to target for your component(s)? SharePoint Online only (latest)
? Where do you want to place the files? Use the current folder
Found npm version 5.8.0
? Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without running any feature deployment or adding apps in sites? No
? Will the components in the solution require permissions to access web APIs that are unique and not shared with other components in the tenant? No
? Which type of client-side component to create? Extension
? Which type of client-side extension to create? Field Customizer
Add new Field Customizer to solution field-extension.
? What is your Field Customizer name? HelloWorld
? What is your Field Customizer description? HelloWorld description
? Which framework would you like to use? (Use arrow keys)
> No JavaScript framework
  React
```

You'll receive a message indicating the scaffolding has successfully completed.

!SharePoint client-side solution scaffolded successfully](./../Linked_Image_Files/field_customizer_extension_image_4.png)

The default solution structure is similar to the solution structure of client-side web parts. This is the basic SharePoint Framework solution structure, with similar configuration options across all solution types.



Code the Field Customizer

When you code your field customizer, you'll open the TS file such as **HelloWorldFieldCustomizer.ts** in the **src\extensions\helloWorld** folder.

Notice in the code below that the base class for the Field Customizer is imported from the **sp-list-view-extensibility** package, which contains SharePoint Framework code required by the Field Customizer.

```
import { Log } from '@microsoft/sp-core-library';
import { override } from '@microsoft/decorators';
import {
  BaseFieldCustomizer,
  IFieldCustomizerCellEventParameters
} from '@microsoft/sp-listview-extensibility';
```

The logic for your Field Customizer is contained in the **OnInit()**, **onRenderCell()**, and **onDisposeCell()** methods.

- **OnInit()** is where you perform any setup needed for your extension. This event occurs after **this.context** and **this.properties** are assigned, but before the page DOM is ready. As with web parts, **OnInit()** returns a promise that you can use to perform asynchronous operations; **onRenderCell()** is not called until your promise has resolved. If you don't need that, simply return `Promise.resolve<void>();`.
- **onRenderCell()** occurs when each cell is rendered. It provides an **event.domElement** HTML element where your code can write its content.

- **onDisposeCell()** occurs immediately before the **event.cellDiv** is deleted. It can be used to free any resources that were allocated during field rendering. For example, if **onRenderCell()** mounted a React element, you must use **onDisposeCell()** to free it; otherwise, a resource leak occurs.

The following are the contents of **onRenderCell()** and **onDisposeCell()** in the default solution:

```

@Override
    public onRenderCell(event: IFieldCustomizerCellEventParameters): void {
        // Use this method to perform your custom cell rendering.
        const text: string = `${this.properties.sampleText}: ${event.
        fieldValue}`;
        event.domElement.innerText = text;
        event.domElement.classList.add(styles.cell);
    }
@Override
    public onDisposeCell(event: IFieldCustomizerCellEventParameters): void {
        // This method should be used to free any resources that were
        // allocated during rendering.
        // For example, if your onRenderCell() called ReactDOM.render(),
        then you should
        // call ReactDOM.unmountComponentAtNode() here.
        super.onDisposeCell(event);
}

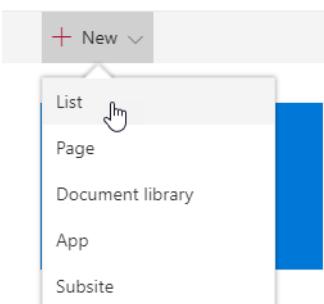
```

Debug the Field Customizer

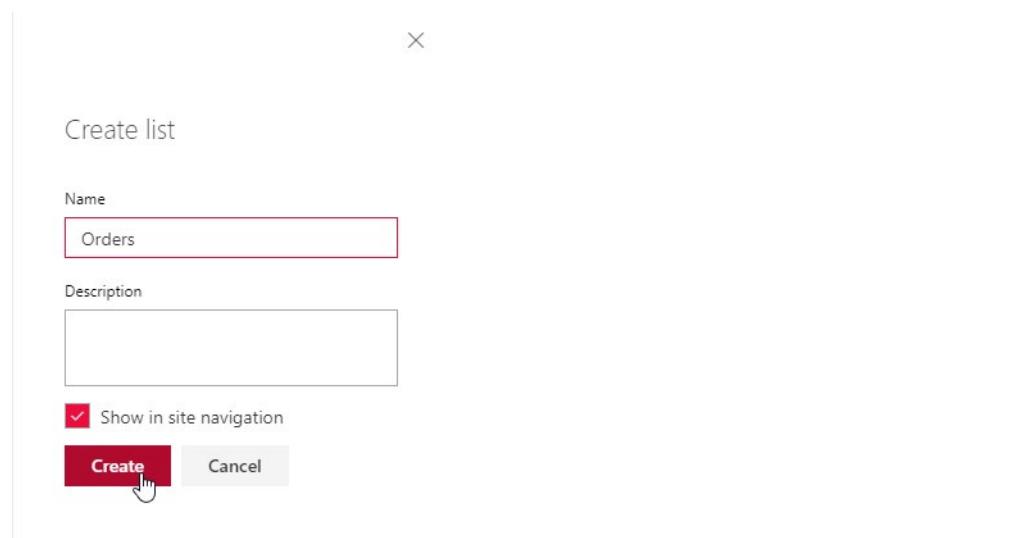
Same as other SPFx extensions, you cannot use the local Workbench to test SharePoint Framework Extensions. You need to test and develop them directly against a live SharePoint Online site. You don't have to deploy your customization to the app catalog to do this, which makes the debugging experience simple and efficient.

The steps below are an example of how to debug your field customizer solution:

1. To test your extension, first create the field to test the customizer in. Go to the site in your SharePoint Online tenant where you want to test the field customizer.
2. Move to the **Site Contents** page.
3. On the toolbar, select **New**, and then select **List**.



4. Create a new list named **Orders**, and then select **Create**.



5. Select the "+" button, and then select **Number** to create a new Number field for the list.

The screenshot shows the SharePoint ribbon with 'Home' selected. Below the ribbon, a red square icon with a white 'T' is visible next to the word 'Team'. The main content area shows a list titled 'Orders'. At the top left of the list are buttons for '+ New', 'Quick edit', 'Export to Excel', 'Flow', 'PowerApps', and '...'. To the right of the list is a '+' button. A dropdown menu is open, listing field types: 'Single line of text', 'Multiple lines of text', 'Number' (which is highlighted with a hand cursor icon), 'Yes/No', and 'Person'.

6. Set the name of the field to **Percent**, and then select **Save**.

Create a column

Learn more about column creation.

Name *

Description

Type

Number

Number of decimal places

Automatic

Show as percentage

Default value

Enter a number

Use calculated value ⓘ

More options

Save Cancel

7. Add a few items with different numbers in the percent field. You'll modify the rendering later in this tutorial, so the different numbers will be presented differently based on your custom implementation.

Home

T Team

+ New Quick edit Export to Excel Flow PowerApps ...

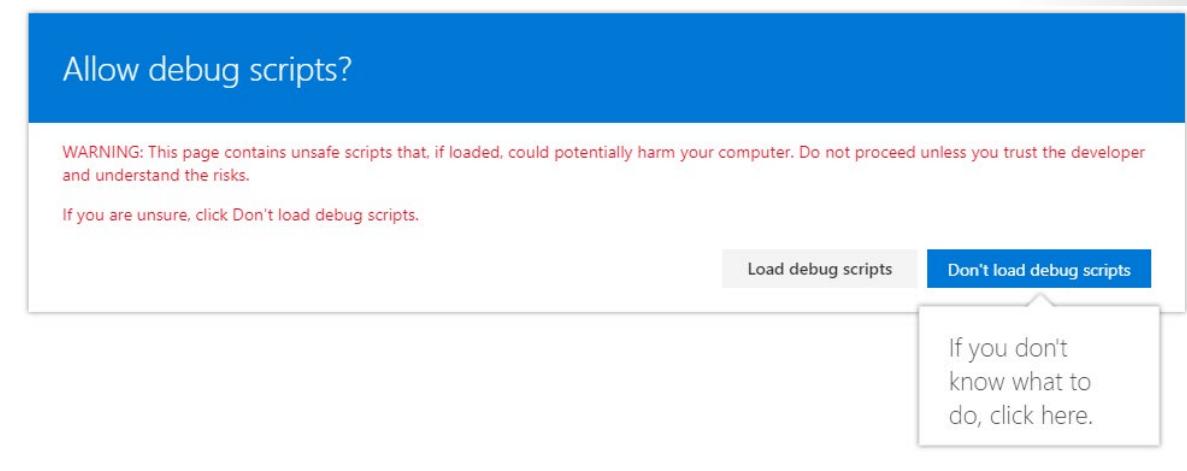
Orders

Title	Percent	
Item A	10	
Item B	50	
Item C	90	

8. Move back to Visual Studio Code and open **serve.json** file from config folder. Update **InternalFieldName** attribute as **Percent** based on the name of the field you just created. Update the **pageUrl** attributes to match a URL of the list you created in the previous steps. After edits your **serve.json** looks like:

```
{  
  "$schema": "https://developer.microsoft.com/json-schemas/core-build/serve.schema.json",  
  "port": 4321,  
  "https": true,  
  "serveConfigurations": {  
    "default": {  
      "pageUrl": "https://sppnp.sharepoint.com/sites/Group/Lists/Orders/AllItems.aspx",  
      "fieldCustomizers": {  
        "Percent": {  
          "id": "b909e395-f714-421f-94e0-d638daf210",  
          "properties": {  
            "sampleText": "Value"  
          }  
        }  
      }  
    },  
    "helloWorld": {  
      "pageUrl": "https://sppnp.sharepoint.com/sites/Group/Lists/Orders/AllItems.aspx",  
      "fieldCustomizers": {  
        "Percent": {  
          "id": "b909e395-f714-421f-94e0-d638daf210",  
          "properties": {  
            "sampleText": "Value"  
          }  
        }  
      }  
    }  
  }  
}
```

9. Compile your code and host the compiled files from the local machine by running this command:
`gulp serve`
10. When the code compiles without errors, it serves the resulting manifest from <https://localhost:4321>.
11. This will also start your default browser within the URL defined in **serve.json** file. In Windows, you can control which browser window is used by activating the browser you want before executing this command. Accept the loading of debug manifests by selecting **Load debug scripts** when prompted.



12. Notice how the **Percent** values are now presented with an additional prefix string as **Value:**, which is provided as a property for the Field Customizer.

Orders		
Title	Percent	+ Add column
Item A	Value: 10	
Item B	Value: 50	
Item C	Value: 90	

With SharePoint Framework extensions, you can customize more facets of the SharePoint experience, including notification areas, toolbars, and list data views. SPFx extensions are available in all Office 365 subscriptions for production usage.

SPFx extensions are client-side components that run inside the context of a SharePoint page. You can deploy extensions to SharePoint Online, and you can use modern JavaScript tools and libraries to build them.

Lesson review questions

1. What is the best extension to use for creating a custom header or footer in SharePoint?
2. What kind of customizer would be used to add an extra command to kick off a business process?
3. What type of extension would you use for overriding the field rendering of SharePoint list views?

Correct/suggested answers:

1. An Application Customizer using page placeholders is the best extension to use for creating a custom header or footer in SharePoint.

2. A Command Set extension is what you would use to create an extra command to kick off a business process. Command set extensions provide the ability to execute client-side code that you define in your solution.
3. A field customizer extension is what you would use to override the field rendering of SharePoint list views.

Package and Deploy an SPFx Solution

Lesson introduction

There are different ways to deploy and activate SharePoint Framework (SPFx) solutions on SharePoint sites. The deployment process depends on the type of SPFx solution you are developing and deploying.

In this lesson, you'll learn the process to package and deploy a SPFx solution including preparing a package for deployment, packaging a solution, tenant-scoped deployment, domain isolated web parts, and deploying a solution.

After this lesson, you should be able to:

- Describe the options for preparing a package for deployment.
- Explain the options for bundling and packaging a solution.
- Describe the requirements of tenant-scoped solution deployment.
- Describe the requirements of domain isolated web parts.
- Discuss the options to deploy a solution.

Prepare a package for deployment

Provision SharePoint assets with your solution package

At times, you may need to provision a SharePoint list or a document library along with your client-side solution package so that the list or library is available for your client-side components, such as web parts. SharePoint Framework toolchain allows you to package and deploy SharePoint items with your client-side solution package. These items are then provisioned when the client-side solution is installed on a site.

Provision items using JavaScript code

While it is possible to create SharePoint items using JavaScript code in your component, such as web parts, it is limited to the context of the current user using that component. If the user doesn't have sufficient permissions to create or modify SharePoint items, the JavaScript code does not provision those items. In such cases, when you want to provision SharePoint items in an elevated context, you must package and deploy the items along with your solution package.

Provision SharePoint items in your solution

The following SharePoint assets can be provisioned along with your client-side solution package:

- Fields.
- Content types.
- List instances.
- List instances with custom schema.

Fields

A field or a site column represents an attribute or piece of metadata, that the user wants to manage for the items in the list (or content type) to which they added the column. It is a reusable column definition,

or template, that you can assign to multiple lists across multiple SharePoint sites. Site columns decrease rework and help you ensure consistency of metadata across sites and lists.

For example, if you define a site column named **Customer**, users can add that column to their lists and reference it in their content types. This ensures that the column has the same attributes—at least to start with—wherever it appears.

Here's an example of a new **DateTime** field:

```
<Field ID="{1511BF28-A787-4061-B2E1-71F64CC93FD5}"  
      Name="DateOpened"  
      DisplayName="Date Opened"  
      Type="DateTime"  
      Format="DateOnly"  
      Required="FALSE"  
      Group="Financial Columns">  
    <Default>[today]</Default>  
</Field>
```

Content types

A content type is a reusable collection of metadata (columns), behavior, and other settings for a category of items or documents in a SharePoint list or document library. Content types enable you to manage the settings for a category of information in a centralized, reusable way.

For example, imagine a business situation in which you have three different types of documents: expense reports, purchase orders, and invoices. All three types of documents have some characteristics in common: they are all financial documents and contain data with values in currency. But each type of document has its own data requirements, its own document template, and its own workflow.

One solution to this business problem is to create four content types. The first content type, **Financial Document**, encapsulates data requirements that are common to all financial documents in the organization. The remaining three, **Expense Report**, **Purchase Order**, and **Invoice**, could inherit common elements from **Financial Document**. In addition, they could define characteristics that are unique to each type, such as a particular set of metadata, a document template to be used when creating a new item, and a specific workflow for processing an item.

You can refer to the schema and attributes in the **ContentType Element (ContentType)**¹² documentation to define a new content type in your solution.

The following is an XML example of a content type:

```
<ContentType ID="0x010042D0C1C200A14B6887742B6344675C8B"  
           Name="Cost Center"  
           Group="Financial Content Types"  
           Description="Financial Content Type">  
  <FieldRefs>  
    <FieldRef ID="{1511BF28-A787-4061-B2E1-71F64CC93FD5}" />  
    <FieldRef ID="{060E50AC-E9C1-4D3C-B1F9-DE0BCAC300F6}" />  
  </FieldRefs>  
</ContentType>
```

¹² <https://msdn.microsoft.com/en-us/library/aa544268.aspx>

List instances

Lists are a key, underlying feature of a SharePoint site. They enable teams to gather, track, and share information. Many applications rely on lists created at the site for data storage to implement their behaviors. A list instance is a predefined SharePoint list that has a well-known identifier. You can customize and add items to these lists, create additional lists from the list templates already available, and create custom lists with just the settings and columns that you choose.

SharePoint provides several list templates such as contact list, calendar, task list, and more. You can use these templates to create new list instances for your web parts or other components. For example, you can define a list instance **Finance Documents** based on the **Document Library** template to store associated documents with your web part.

You can refer to the schema and attributes in the **ListInstance Element (List Instance)**¹³ documentation to define a list instance in your solution.

The following is an example of a list instance definition:

```
<ListInstance  
    FeatureId="00bfea71-e717-4e80-aa17-d0c71b360101"  
    Title="Finance Records"  
    Description="Finance documents"  
    TemplateType="101"  
    Url="Lists/FinanceRecords">  
</ListInstance>
```

List instances with custom schema

You can use a custom list schema definition to define your fields, content types, and views used in your list instance. You use the **CustomSchema**¹⁴ attribute in the **ListInstance** element to reference a custom schema for the list instance.

For example, you can define a list instance **Finance Documents** with a content type **Financial Document** that could encapsulate data requirements that are common to all financial documents in the organization.

The following is an XML example of a list instance definition that uses a custom schema:

```
<ListInstance  
    CustomSchema="schema.xml"  
    FeatureId="00bfea71-de22-43b2-a848-c05709900100"  
    Title="Cost Centers"  
    Description="Cost Centers"  
    TemplateType="100"  
    Url="Lists/CostCenters">  
</ListInstance>
```

This is the custom XML schema definition that defines a content type for the list instance defined previously:

```
<List xmlns:ows="Microsoft SharePoint" Title="Basic List" EnableContent-  
Types="TRUE" FolderCreation="FALSE" Direction="$Resources:Direction;"
```

¹³ <https://msdn.microsoft.com/en-us/library/office/ms476062.aspx>

¹⁴ <https://msdn.microsoft.com/en-us/library/office/ms476062.aspx>

```
Url="Lists/Basic List" BaseType="0" xmlns="http://schemas.microsoft.com/sharepoint/">
  <MetaData>
    <ContentTypes>
      <ContentTypeRef ID="0x010042D0C1C200A14B6887742B6344675C8B" />
    </ContentTypes>
    <Fields></Fields>
    <Views>
      <View BaseViewID="1" Type="HTML" WebPartZoneID="Main" DisplayName="$Resources:core,objectiv_schema_mwsidcamlidC24;" DefaultView="TRUE" MobileView="TRUE" MobileDefaultView="TRUE" SetupPath="pages\viewpage.aspx" ImageUrl="/_layouts/images/generic.png" Url="AllItems.aspx">
        <XslLink Default="TRUE">main.xsl</XslLink>
        <JSLink>clienttemplates.js</JSLink>
        <RowLimit Paged="TRUE">30</RowLimit>
        <Toolbar Type="Standard" />
        <ViewFields>
          <FieldRef Name="LinkTitle"></FieldRef>
          <FieldRef Name="SPFxAmount"></FieldRef>
          <FieldRef Name="SPFxCostCenter"></FieldRef>
        </ViewFields>
        <Query>
          <OrderBy>
            <FieldRef Name="ID" />
          </OrderBy>
        </Query>
      </View>
    </Views>
    <Forms>
      <Form Type="DisplayForm" Url="DispForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
      <Form Type="EditForm" Url="EditForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
      <Form Type="NewForm" Url="NewForm.aspx" SetupPath="pages\form.aspx" WebPartZoneID="Main" />
    </Forms>
  </MetaData>
</List>
```

Create SharePoint items in your solution

The solution package uses SharePoint Features to package and provision the SharePoint items. A feature is a container that includes one or more SharePoint items to provision. A feature contains a Feature.xml file and one or more element manifest files. These XML files are also known as feature definitions.

Typically, a client-side solution package contains one feature. This feature is activated when the solution is installed on a site. It is important to note that the site administrators install your solution package and not the feature.

A feature is primarily constructed by using the following XML files.

Element manifest file

The element manifest file contains the SharePoint item definitions and is executed when the feature is activated. For example, the XML definitions to create a new field, content type, or list instance is in the element manifest.

The following is an example of an element manifest XML file that defines a new **DateTime** field.

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field ID="{1511BF28-A787-4061-B2E1-71F64CC93FD5}">
    Name="DateOpened"
    DisplayName="Date Opened"
    Type="DateTime"
    Format="DateOnly"
    Required="FALSE"
    Group="Financial Columns">
    <Default>[today]</Default>
  </Field>
</Elements>
```

Any supported files that accompany the element manifest are element files. For example, the list instance schema is an element file that is associated with a list instance defined in an element manifest.

The following is an example of a custom list instance XML schema:

```
<List xmlns:ows="Microsoft SharePoint" Title="Basic List" EnableContent-
Types="TRUE" FolderCreation="FALSE"
      Direction="$Resources:Direction;" Url="Lists/Basic List" Base-
Type="0" xmlns="http://schemas.microsoft.com/sharepoint/">
  <MetaData>
    <ContentTypes>
      <ContentTypeRef ID="0x010042D0C1C200A14B6887742B6344675C8B" />
    </ContentTypes>
  </MetaData>
</List>
```

Upgrade actions file

As its name suggests, this is the file that includes any upgrade actions when the solution is updated on the site. As part of the upgrade actions, the action could specify to include one or more element manifests as well. For example, if the upgrade requires a new field to be added, the field definition is available as an element manifest and is associated in the upgrade actions file.

The following is an example of an upgrade actions file that applies an element manifest file during the upgrade:

```
<ApplyElementManifests>
  <ElementManifest Location="9c0be970-a4d7-41bb-be21-4a683586db18\ele-
ments-v2.xml" />
</ApplyElementManifests>
```

Configure the SharePoint feature

To include the XML files, you must first define the feature configuration in the **package-solution.json** configuration file under the **config** folder in your project. The **package-solution.json** contains the key metadata information about your client-side solution package and is referenced when you run the package-solution gulp task that packages your solution into an .sppkg file.

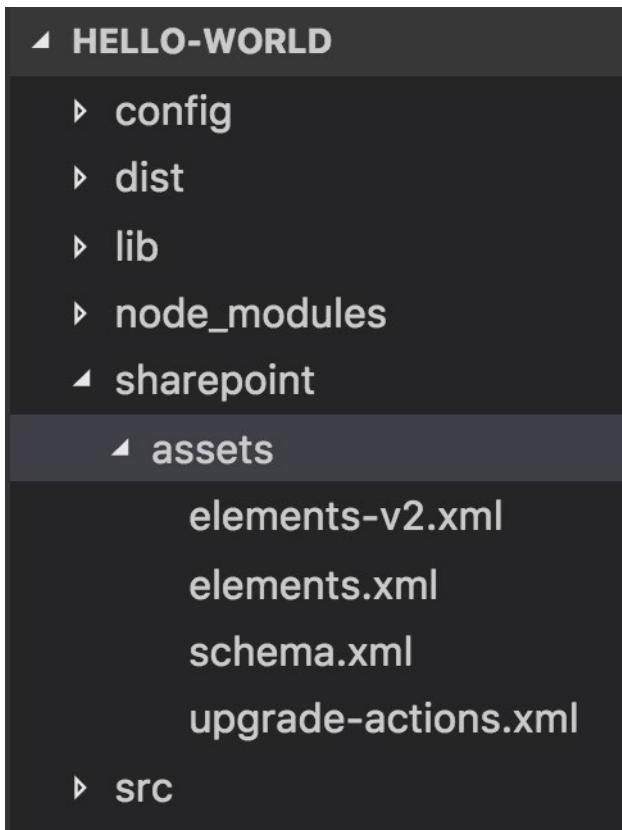
```
{
  "solution": {
    "name": "hello-world-client-side-solution",
    "id": "26364618-3056-4b45-98c1-39450adc5723",
    "version": "1.1.0.0",
    "features": [
      {
        "title": "hello-world-client-side-solution",
        "description": "hello-world-client-side-solution",
        "id": "d46cd9d6-87fc-473b-a4c0-db9ad9162b64",
        "version": "1.1.0.0",
        "assets": {
          "elementManifests": [
            "elements.xml"
          ],
          "elementFiles": [
            "schema.xml"
          ],
          "upgradeActions": [
            "upgrade-actions-v1.xml"
          ]
        }
      }
    ],
    "paths": {
      "zippedPackage": "solution/hello-world.sppkg"
    }
  }
}
```

The features JSON object contains the metadata about the feature, as shown in the following table.

Property	Description
id	Unique identifier (GUID) of the feature.
title	Title of the feature.
description	Description of the feature.
assets	An array of XML files used in the feature.
elementManifests	An array of element manifest files defined within the assets property.
elementFiles	An array of element files defined within the assets property.
upgradeActions	An array of upgrade action files defined within the assets property.

Create the feature XML files

The toolchain looks for the XML files as defined in the configuration under a special folder, **sharepoint\assets**, in your client-side solution project.



The configurations defined in the **package-solution.json** is what maps the XML files here to its appropriate feature XML file when the package-solution gulp task is executed.

Package SharePoint items

After you define your feature in the **package-solution.json** and create the respective feature XML files, you can use the following gulp task to package the SharePoint items along with your .sppkg package:

```
gulp package-solution
```

This command packages one or more client-side component manifests, such as web parts, along with the feature XML files referenced in the **package-solution.json** configuration file.

Note:

You can use the **-ship** flag to package minified versions of your components.

Upgrade SharePoint items

You may include new SharePoint items or update existing SharePoint items when you upgrade your client-side solution. Because provisioning SharePoint items uses features, you use the feature **Upgrade-Actions** XML file to define a list of upgrade actions.

The **upgradeActions** JSON object array in the **package-solution.json** references the feature XML file(s) associated with the upgrade actions for your feature. At minimum, an upgrade action file defines the element manifest XML file that executes when upgrading the feature.

When you upgrade a SharePoint Framework solution, you must also update version attributes for both the solution and any features that include the upgrade actions. A solution version increase tells SharePoint end users that there is a new version of the package available. A feature element version increase ensures that the tasks defined in the upgrade actions are being processed as part of the solution upgrade.

The following is an example of an upgrade action file that applies an element manifest file during the upgrade:

```
<ApplyElementManifests>
  <ElementManifest Location="9c0be970-a4d7-41bb-be21-4a683586db18\elements-v2.xml" />
</ApplyElementManifests>
```

This is the corresponding element-v2.xml that defines a new Currency field to be provisioned during the upgrade:

```
<?xml version="1.0" encoding="utf-8"?>
<Elements xmlns="http://schemas.microsoft.com/sharepoint/">
  <Field ID="{060E50AC-E9C1-4D3C-B1F9-DE0BCAC300F6}"
    Name="Amount"
    DisplayName="Amount"
    Type="Currency"
    Decimals="2"
    Min="0"
    Required="FALSE"
    Group="Financial Columns" />
</Elements>
```

Sub-elements

Upgrade actions in client-side solutions support the following sub-elements.

AddContentTypeField

This sub-element adds a new field to an existing provisioned content type. It propagates the change from the site content type to all child lists and content types within the site. For example:

```
<AddContentTypeField
  ContentTypeId="0x010100A6F9CE1AFE2A48f0A3E6CB5BB770B0F7"
  FieldId="{B250DCFD-9310-4e2d-85F2-BE2DA37A57D2}"
  PushDown="TRUE" />
```

ApplyElementManifests

This sub-element adds a new element to an existing feature. When a feature is upgraded, it provisions all non-declarative elements that are referenced in the specified element manifests.

VersionRange

This sub-element specifies a version range to which specified upgrade actions apply.

Update SharePoint Framework packages

SharePoint client-side development tools use the **npm¹⁵** package manager to manage dependencies and other required JavaScript helpers. npm is typically included as part of Node.js setup.

When you create a new client-side solution, the Yeoman generator for SharePoint fetches the latest SharePoint Framework packages required for your client-side project. As you build your project, your existing packages may be outdated by new versions of one or more packages available.

Based on the **release notes¹⁶** for a particular release or the latest package, you may decide to update your SharePoint Framework packages used in your project. SharePoint Framework packages include both the npm packages you have installed in your project (for example: **@microsoft/sp-core-library¹⁷**) and npm packages installed globally (for example: **@microsoft/generator-sharepoint¹⁸**).

Note:

While it may not be required, Microsoft recommends that you update the SharePoint Framework packages often so that you can get the latest changes and fixes that have been released.

Find outdated packages

To find the outdated packages in your client-side project, including SharePoint Framework and other packages your project depends on, run the following command in a console in the same directory as your project: `npm outdated`

The command lists the following information about the packages your project depends on. It looks up the required information from the **package.json** file located in the root of your project directory and npm registry. This command lists:

- Current version installed in your project.
- Version requested by your project (available in package.json).
- Latest version available.

¹⁵ <https://www.npmjs.com/>

¹⁶ <https://github.com/SharePoint/sp-dev-docs/wiki>

¹⁷ <https://www.npmjs.com/package/@microsoft/sp-core-library>

¹⁸ <https://www.npmjs.com/package/@microsoft/generator-sharepoint>

```
[→ react-todo-basic git:(dev) ✘ npm outdated
Package           Current  Wanted  Latest  Location
@microsoft/sp-build-web    0.7.0   0.7.0   0.9.0  react-todo-simple
@microsoft/sp-client-base   0.4.0   0.4.0   0.7.0  react-todo-simple
@microsoft/sp-client-preview 0.5.0   0.5.0   0.9.0  react-todo-simple
@microsoft/sp-module-interfaces 0.4.0   0.4.0   0.7.0  react-todo-simple
@microsoft/sp-webpart-base   0.1.0   0.1.0   0.4.0  react-todo-simple
@microsoft/sp-webpart-workbench 0.5.0   0.5.0   0.8.0  react-todo-simple
office-ui-fabric-react      0.50.0  0.50.0  1.0.0  react-todo-simple
react                  0.14.8  0.14.8  15.4.2 react-todo-simple
react-addons-update        0.14.8  0.14.8  15.4.2 react-todo-simple
react-dom                0.14.8  0.14.8  15.4.2 react-todo-simple
→ react-todo-basic git:(dev) ✘ ]
```

To identify the SharePoint Framework packages, look for the package names that start with the following npm scope and prefix: @microsoft/sp-

Along with the framework packages, you may also need to update react and office-ui-fabric-react packages. Make sure you read the [release notes¹⁹](#) for that specific release to determine which packages require updates and plan accordingly.

Using the “npm outdated” command with a project

Starting from Feature Pack 2, SharePoint 2016 supports SharePoint Framework solutions. SharePoint 2016 uses an older version of the SharePoint Framework than the version available in SharePoint Online.

When scaffolding new projects, the SharePoint Framework Yeoman generator prompts you to choose if your solution should be using the latest version of the SharePoint Framework and be working only with SharePoint Online, or if it should use an older version of the SharePoint Framework and work with both SharePoint 2016 and SharePoint Online.

When you run the npm outdated command in a project that targets both SharePoint Online and SharePoint 2016, it shows you the latest versions of the SharePoint Framework packages. These versions, however, work only with SharePoint Online. If you update your solution to use these latest packages, it will no longer work with SharePoint 2016.

When working with SharePoint Framework solutions compatible with SharePoint hosted on-premises, you should always verify which patch level the target SharePoint farm has and which version of the SharePoint Framework it supports.

Update packages

When updating packages to newer versions, you should always use your package manager (npm or Yarn). You should not edit the **package.json** file manually. If you follow the recommended practice of using a lock file, your changes to the **package.json** file will be ignored.

Start with identifying which packages need updating and which newer version you want to use. Note that it might not always be possible for you to use the latest version of the given package because it might be incompatible with other SharePoint Framework dependencies, such as TypeScript.

¹⁹ <https://github.com/SharePoint/sp-dev-docs/wiki>

For each package that you want to update, run the following command: `npm install mypackage@newversion --save`

For example, if you were using AngularJS version v1.5.9 and wanted to update to version 1.6.5, you would run: `npm install angular@1.6.5 --save`

Updating the package by using npm installs the specified version of the package in your project and updates the version number in the **package.json** file dependencies and the lock file used in your project.

After the packages are installed, execute the following command to clean up any old build artifacts: `gulp clean`

Furthermore, it is recommended to delete all old packages by deleting the **node_modules** folder and download (ie: reinstall) all dependencies with the updated **package.json**: `npm install`

Without this step, old versions may conflict with the newer versions the next time you build the project.

Update your code

Depending on breaking API changes, you may have to update your existing project code and config files. For each release, the **release notes**²⁰ highlight any breaking changes and the modifications required to your existing code. You must make sure you update your code with those fixes.

You can always build the project to see if you have any errors and warnings by running the command in a console in your project directory: `gulp build`

Update Yeoman generator

If you have installed the SharePoint Framework Yeoman generator globally, you can find out if it requires updating by running the following command: `npm outdated -g`

The command lists the following information about the packages installed globally on your machine. It looks up this information from the versions installed on your machine and the npm registry.

- Current version installed globally on your machine.
- Version requested by you when you installed.
- Latest version available.

Package	Current	Wanted	Latest	Location
@microsoft/generator-sharepoint	0.3.2	0.3.2	0.4.1	
gulp	3.9.0	3.9.1	3.9.1	
n	2.1.2	2.1.4	2.1.4	
npm	4.1.1	4.1.2	4.1.1	
typescript	2.0.3	2.1.5	2.1.5	
typings	1.4.0	2.1.0	2.1.0	
webpack	1.13.0	1.14.0	1.14.0	

To identify the generator package, look for the following package name: @microsoft/generator-sharepoint

²⁰ <https://github.com/SharePoint/sp-dev-docs/wiki>

Update generator package

Open your favorite console and execute the following command to update the generator to its latest published version: `npm install @microsoft/generator-sharepoint@latest -g`

This command updates the Yeoman generator for SharePoint to the latest published version along with its dependencies. You can validate this by executing the following command in the console: `npm ls @microsoft/generator-sharepoint -g --depth=0`

Package a solution

Each package configuration file includes some optional settings to override the places where the task looks for various source files and manifests, as well as defining the location to write the package. Additionally, it includes a required solution definition, which instructs the packager on the relationships between various components.

The package-solution gulp task looks at **/config/package-solution.json** for various configuration details, including some generic filepaths, as well as defining the relationship between components (WebParts and Applications) in a package.

The schema for the configuration file is:

```
interface IPackageSolutionTaskConfig {  
    paths?: {  
        packageDir?: string;  
        debugDir?: string;  
        zippedPackage?: string;  
        featureXmlDir?: string;  
        manifestsMatch?: string;  
        manifestDir?: string;  
        sharepointAssetDir?: string;  
    };  
    solution?: ISolution;  
}
```

Solution definition (ISolution)

Each solution file must have a **name** that identifies the package in the SharePoint UI. Additionally, each package must contain a globally unique identifier (**id**), which is used internally by SharePoint. Optionally, you may also specify a **version** number in the format "X.X.X.X", which is used to identify various versions of the package when upgrading.

Note:

The versioning system only applies to Feature Framework and SharePoint Feature definitions included in the package. Code and assets from the new version of the package are available as soon as new version of the package is added to App Catalog with no need to update the app on sites.

The solution definition also optionally contains a list of SharePoint Feature definitions.

Note:

If this is omitted or empty, the task creates a single Feature for every component (a 1:1 mapping).

The schema for the solution file is:

```
interface ISolution {  
  name: string;  
  id: string;  
  title?: string;  
  supportedLocales?: string[];  
  version?: string;  
  features?: IFeature[];  
  iconPath?: string;  
  skipFeatureDeployment?: boolean;  
}
```

Feature definition (IFeature)

It's important to note that there is a definition for creating a SharePoint feature, and that some of these options are exposed in the UI. Like the solution, each feature contains a mandatory **title**, **description**, **id**, and **version** number (in the X.X.X.X format). Note that the feature **id** should also be a globally unique identifier.

Each feature can also contain any number of components that are activated when the feature is activated. This is defined via a list of **componentIds**, which are globally unique identifiers that *must* match the **id** in the component's manifest file. If this list is undefined or empty, the packager includes *every* component in the feature.

The schema for the feature is:

```
interface IFeature {  
  title: string;  
  description: string;  
  id: string;  
  version?: string;  
  componentIds?: string[];  
  components: IComponent[];  
  assets: ISharepointAssets<string>;  
}
```

File paths

- **packageDir**: Packaging root folder. Defaults to **./sharepoint**. All other paths are relative to this folder.
- **debugDir**: Folder to write the raw package to disk for debugging. Defaults to **solution/debug**
- **zippedPackage**: Name of the sppkg file to create (including extension). Defaults to **ClientSolution.sppkg**
- **featureXmlDir**: Folder containing the custom feature XML to import into the package. Defaults to **feature_xml**

Note:

All files in this folder are included in the SPPKG; however, you must create a .rels file for your custom feature for it to be included in the package manifest.

- **manifestsMatch**: Glob used to find manifest files. Looks in **dist/** when running in normal, but in **deploy/** for production.

- **manifestDir**: Path to the folder where manifests are stored. Defaults to **buildConfig.distFolder**
- **sharepointAssetDir**: Directory containing SharePoint assets (such as feature elements, element manifests, and upgrade actions), which are automatically included in the SharePoint package. Defaults to **assets**

The schema for the file paths is:

```
interface IPackageSolutionTaskConfig {  
    paths?: {  
        packageDir?: string;  
        debugDir?: string;  
        zippedPackage?: string;  
        featureXmlDir?: string;  
        manifestsMatch?: string;  
        manifestDir?: string;  
        sharepointAssetDir?: string;  
    };  
    solution?: ISolution;  
}
```

Default configuration

```
{  
    "solution": {  
        "name": "spfx-hello-world-client-side-solution",  
        "id": "77fd2eed-55b0-42bf-8b4d-f66730cb0c34",  
        "version": "1.0.0.0"  
    },  
    "paths": {  
        "zippedPackage": "solution/spfx-hello-world.sppkg"  
    }  
}
```

Custom feature XML

To support provisioning of various SharePoint resources (such as List Templates, Pages, or Content Types), custom feature XML may also be injected into the package. This is used to provision resources necessary for applications, but may also be used for web parts. The documentation for Feature XML is located at [Feature.xml Files²¹](#).

By default, the packaging task looks for the custom feature XML in **./sharepoint/feature_xml**. Every file in this folder is included in the final application package. However, the task relies on the contents of the **_rels/** folder to determine which custom features are defined.

Essentially, the assumption is that each **.xml.rels** file is related to a **feature.xml** of the same name at the top level of the **feature_xml/**, then adds a relationship to the **AppManifest.xml.rels** file that includes that feature in the package.

²¹ <https://msdn.microsoft.com/en-us/library/office/ms475601.aspx?f=255&MSPPError=-2147217396>

Tenant-scoped solution deployment

Tenant-Wide Deployment of SharePoint Framework extensions

The Tenant-Wide Deployment option for SharePoint Framework extensions is supported for application customizers and for ListView Command Sets. It provides tenant app catalog managers with an easy option for managing which extensions are activated by default across the tenant or based on web/list templates used in the sites.

Note:

SharePoint Framework extensions are supported with modern experiences regardless of the actual site template used when the content site was created.

When developers create a new SharePoint Framework extension solution using standard SharePoint Framework Yeoman packages, the solution package includes automation to activate the extension across the tenant.

Warning:

Starting from SharePoint Framework v1.6, default scaffolding will automatically create example files in SharePoint Solution to activate extension across the tenant if you choose to use the tenant-scoped deployment option.

You need to configure your SharePoint Framework solution to use a tenant-scoped deployment option (detailed in the following section), allowing it to automatically activate extensions across the tenant using Tenant-Wide Deployment functionality. This means that the **skipFeatureDeployment** attribute in the **package-solution.json** must be set as true.

Tenant-scoped solution deployment option

You can configure your SPFx components to be immediately available across the tenant when the solution package is installed to the tenant app catalog. You can configure this by using the **skipFeatureDeployment** attribute in the **package-solution.json** file.

When the solution has this attribute enabled, it provides the tenant administrator with the option to enable the solution automatically across all site collections and sites in the tenant when the solution package is installed to the tenant app catalog.

You must update to the latest SharePoint Framework Yeoman template version to use this capability. You can update your global installation by executing `npm install -g @microsoft/generator-sharepoint`. Tenant-wide deployment is not available for SharePoint 2016 Feature Pack 2 as it only supports SPFx 1.1, and this deployment option was released in version 1.4. If you use a SPFx webpart older than 1.4 you can upgrade with instructions via the [Office 365 CLI²²](#)

Solution-specific requirements

When using a tenant-scoped solution, any feature framework definitions in the SharePoint Framework solution are ignored. If the solution contains feature framework definitions (for example, for creating a custom list), you should not use this solution-specific option.

²² <https://aka.ms/o365cli>

Configure solution to be available across the tenant

The SharePoint Framework Yeoman template asks a specific question related to this option:

Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites immediately without running any feature deployment or adding apps in sites? (y/N)

This question directly impacts the **skipFeatureDeployment** attribute in the package-**solution.json** file.

```
λ yo @microsoft/sharepoint

          Welcome to the
          SharePoint Client-side
          Solution Generator

Let's create a new SharePoint solution.
? What is your solution name? auto-deploy
? Where do you want to place the files? Use the current folder
? Do you want to allow the tenant admin the choice of being able to deploy the solution to all sites
immediately without running any feature deployment or adding apps in sites? (y/N)|
```

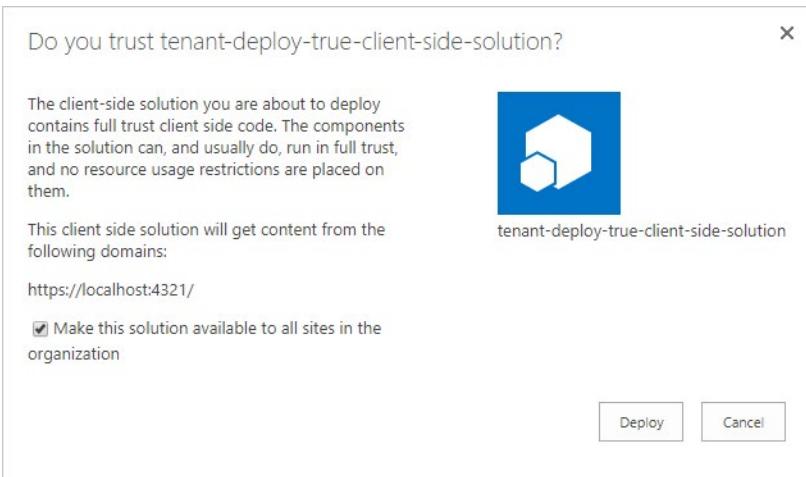
In the following example JSON configuration, **skipFeatureDeployment** is set to **true**, which indicates that the solution can be centrally deployed across the tenant.

```
{
  "solution": {
    "name": "tenant-deploy-client-side-solution",
    "id": "dd4fec4-6f7e-47f1-a0e2-97de8890e3fa",
    "version": "1.0.0.0",
    "skipFeatureDeployment": true
  },
  "paths": {
    "zippedPackage": "solution/tenant-deploy-true.sppkg"
  }
}
```

Approve tenant-wide deployment in app catalog

When you set the **skipFeatureDeployment** attribute set to **true** and deploy the tenant app catalog, the administrator is given an option to deploy the solution centrally across the tenant.

By default, the **Make this solution available to all sites in the organization** check box is not selected. If the administrator selects the check box, components in the solutions automatically become visible and available across the tenant.



Notice that because the solution and site-specific upgrade actions are only available when you use the feature framework, there's no specific upgrade option for centrally deployed solutions. These solutions can be updated by updating the solution-specific assets in the CDN and by updating the package in the app catalog. This automatically updates all existing component instances across the tenant to use the latest component assets, such as JavaScript files and updated CSS files.

Domain isolated web parts

Using the domain isolated web parts capability, you can build web parts that securely communicate with APIs secured with Azure AD without exposing the access token to other components on the page or even scripts in the tenant.

Secure with isolated web parts

You can use the API management page to specify which APIs in your tenant can be accessed by scripts, and with which permissions. This allows your SharePoint Framework solutions to safely access APIs secured with Azure AD. Using SharePoint Framework, you can easily retrieve an access token for the specific API. While this significantly simplifies communicating with APIs secured with Azure AD, it allows all scripts—not just specific SharePoint Framework solutions—to obtain an access token for any of the approved APIs. If one of the scripts you use in your tenant is exploited, then it can access any of the approved APIs on behalf of the current user.

Isolated web parts introduce a new way to isolate access to APIs secured with Azure AD and ensure that only specific SharePoint Framework web parts are allowed to obtain an access token for the particular API.

How isolated web parts work

Note:

The isolated web parts capability is available only in SharePoint Framework v1.8.0 and later.

Solutions using the isolated web parts capability include a specific flag set in the project metadata inside the .sppkg file. When deploying these solutions to the app catalog, it specifies all API permission requests as isolated. After approving an isolated API permission request, SharePoint will create a separate Azure AD application in the Azure AD linked to the Office 365 tenant.

This Azure AD application is specific to the SharePoint Framework solution that requested API permissions and will include set OAuth permissions as requested by that solution. The return URL of that Azure AD application, which is used by the OAuth implicit flow, will be set to a unique domain tied to that specific SharePoint Framework application. When added to a page, all web parts from solutions using isolated permissions will be displayed using an iframe pointing to a unique domain tied to that particular SharePoint Framework solution. This method allows the SharePoint Framework to enforce unique API permissions and ensure that no other solution or script in the tenant can obtain an access token to these APIs.

Scaffold a project that uses isolated permissions

If the solution requires API permissions that should be isolated and not available to other components, the SharePoint Framework Yeoman generator will prompt you during the scaffolding of a new SharePoint Framework project.

If you answer **Yes**, then the generator will add a flag to your project's configuration in the **config/package-solution.json** file, by setting the **isDomainIsolated** property to **true**. Because the isolated web parts capability applies only to web parts, the generator will only allow you to create web parts in your project.

Note:

Theoretically, you can manually create a SharePoint Framework extension in a project that uses isolated permissions. This presents a security risk and is something you should never do. If the extension you add communicates with APIs secured with Azure AD, it wouldn't be able to retrieve the access token in an isolated way and would fail on runtime.

Communicate with APIs

Even though the web part uses isolated permissions, there is nothing specific to how you obtain an access token to an API secured with Azure AD in your code. Even though isolated web parts will be loaded inside an iframe pointing to a unique domain during runtime, you can still communicate with SharePoint REST API as easily as you would in non-isolated web parts.

Deploy solutions with isolated web parts

Solutions with isolated web parts are deployed the same way as regular SharePoint Framework solutions. The only difference is that the API permission requests are deployed as isolated. This is clearly visible on the API management page, where the API permission requests are grouped per solution to which they apply. API permissions granted on the tenant-level can be used by any SharePoint Framework solution or piece of script on the tenant. Isolated permissions on the other hand, can only be used by the solution that requested them.

Use isolated web parts

When added to the page, isolated web parts are displayed using an iframe. This iframe points to a unique domain assigned to the SharePoint Framework solution containing the web part. This domain is also referenced in the return URL of the Azure AD application created to host the isolated permissions for that particular SharePoint Framework solution. Using the unique domain ensures that only web parts from that particular SharePoint Framework solution are able to obtain an access token for the isolated set of permissions.

Upgrade existing project to use isolated permissions

If you're upgrading an existing SharePoint Framework project to v1.8.0 and want to use the isolated permissions capability, you can do so in the **config/package-solution.json** file by setting the **isDomainIsolated** property to **true**. You should ensure that your project contains only web parts.

After changing the project to use isolated permissions, you should redeploy your project. This will issue new API permission requests, all isolated to your solution, which the tenant admin must approve.

Remove solutions with isolated permissions

When approving isolated API permissions, SharePoint creates a dedicated Azure AD application specific to the SharePoint Framework solution that requested permissions. When the SharePoint Framework solution is removed from the app catalog, the API permissions and the Azure AD application are not removed.

Optimize SharePoint Framework builds for production

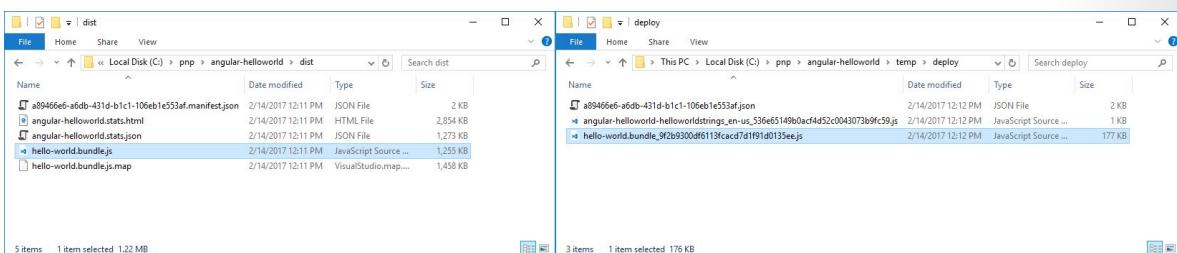
When deploying SharePoint Framework solutions to production, you should always use a release build of your project that is optimized for performance. This topic illustrates the main differences between debug and release builds and shows how you can optimize your bundle for use in production environments.

Use release builds in production

When building a SharePoint Framework project, you can choose whether you want to build it in a debug or release mode. By default, SharePoint Framework projects are built in debug mode, which makes it easier to debug code. But when your code is finished and works as expected, you should build it in release mode to optimize it for running in the production environment.

For more information about building your project in release mode, see **SharePoint Framework tool-chain**²³.

The main difference between the output of a debug and release build is that the release version of the generated bundle is minified and significantly smaller in size than its debug equivalent. To illustrate the difference, compare the size of the debug and release version of a SharePoint Framework project with a web part using Angular.



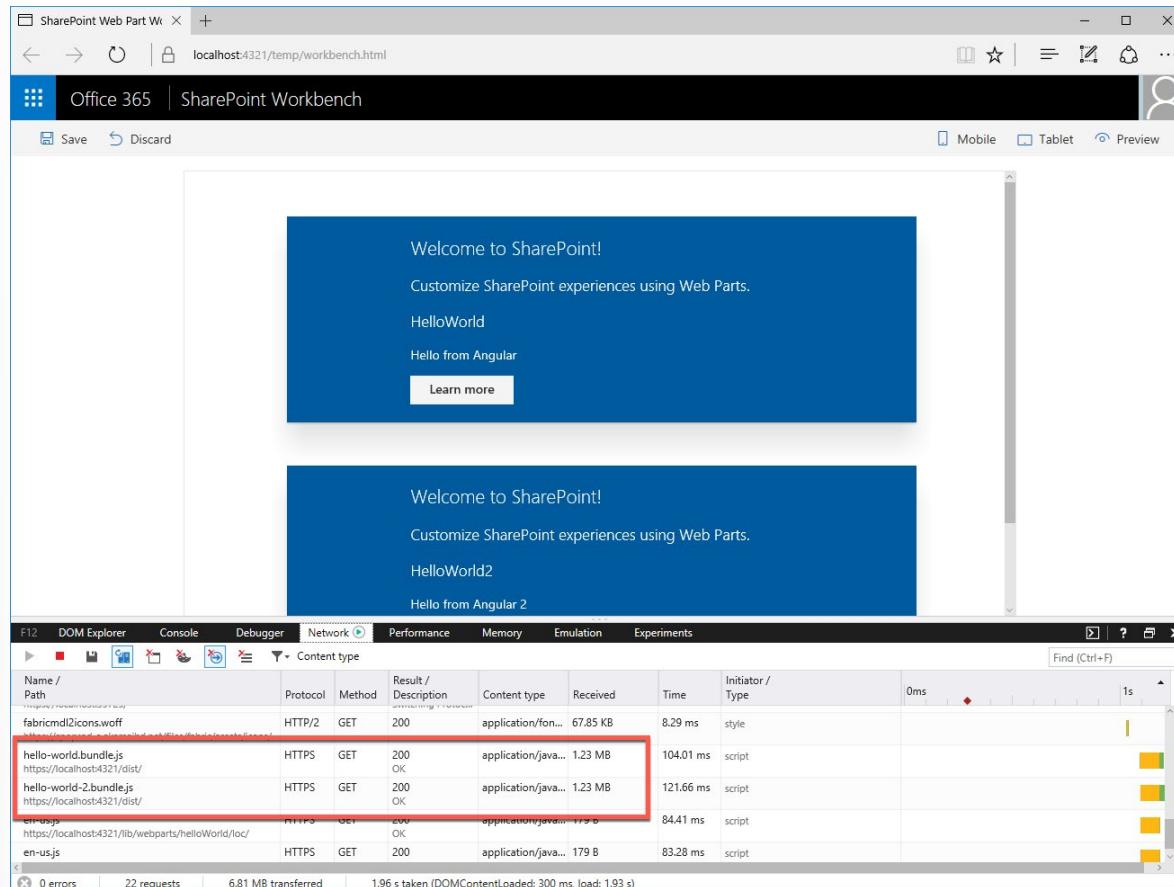
The debug version of the bundle is 1255 KB while its release equivalent is only 177 KB. The difference in size between the debug and release version of the generated bundle depends on the libraries used in your project. The release build is always significantly smaller than a debug build, which is why you should always deploy the output of release builds to production.

²³ <https://docs.microsoft.com/en-us/sharepoint/dev/spfx/toolchain/sharepoint-framework-toolchain>

Don't include third-party libraries in the bundle

When building SharePoint Framework solutions, you can benefit from many existing JavaScript libraries to solve common problems. Using existing libraries allows you to be more productive and lets you focus on adding value for your organization, instead of on developing generic functionality.

By default, when referencing third-party libraries in your project, SharePoint Framework includes them in the generated bundle. As a result, users working with your solution end up downloading the same library multiple times, once with each component. This causes the total page size to grow significantly, taking longer to load and leading to a poor user experience, particularly on slower networks.



When working with third-party libraries, you should always consider loading them from an external location—either a public CDN or a hosting location owned by your organization. This allows you to exclude the library from your bundle, significantly decreasing its size.

Additionally, if the hosting location from which you load the library is optimized for serving static assets, users working with your solution only need to load the library once. On subsequent requests, or when using your solution in the future, the web browser reuses the previously cached copy of the library rather than downloading it again. As a result, the page with your solution loads significantly faster.

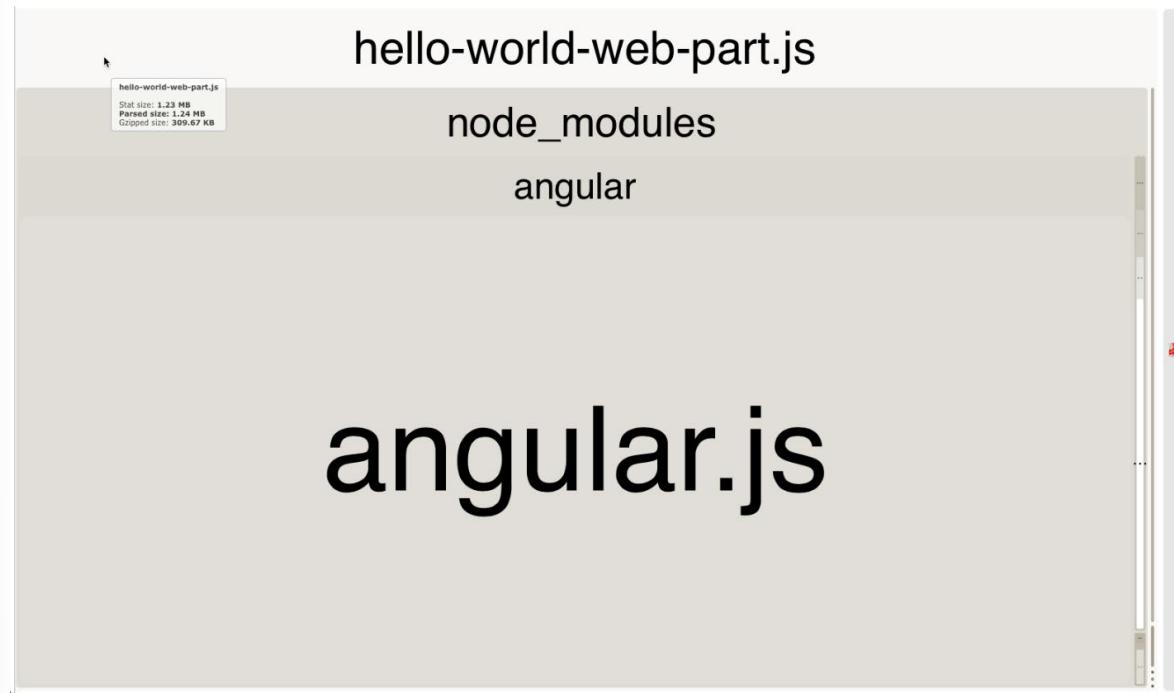
Verify the contents of your bundle

To better understand the size of the generated bundles, you can extend the Webpack configuration in your project and tell the SharePoint Framework to generate bundle statistics.

1. First, install the `webpack-bundle-analyzer` package in your project by executing the following in the command line: `npm install webpack-bundle-analyzer --save-dev`
2. Next, change the contents of the **gulpfile.js** file in your project to:

```
'use strict';
const gulp = require('gulp');
const path = require('path');
const build = require('@microsoft/sp-build-web');
const bundleAnalyzer = require('webpack-bundle-analyzer');
build.configureWebpack.mergeConfig({
  additionalConfiguration: (generatedConfiguration) => {
    const lastDirName = path.basename(__dirname);
    const dropPath = path.join(__dirname, 'temp', 'stats');
    generatedConfiguration.plugins.push(new bundleAnalyzer.BundleAnalyzerPlugin({
      openAnalyzer: false,
      analyzerMode: 'static',
      reportFilename: path.join(dropPath, `${lastDirName}.stats.html`),
      generateStatsFile: true,
      statsFilename: path.join(dropPath, `${lastDirName}.stats.json`),
      logLevel: 'error'
    }));
    return generatedConfiguration;
  }
});
build.initialize(gulp);
```

3. The next time you bundle your project by using the gulp bundle task, you can find a bundle stats files generated in the temp/stats folder in your project. One of the generated stats files is a treemap showing the different scripts included in the generated bundle. You can find this visualization in the **./temp/stats/[solution-name].stats.html** file.



Using the Webpack bundle analyzer treemap helps you conveniently verify that the generated bundle doesn't contain any unnecessary scripts. It also helps you understand how the included scripts affect the total bundle size. Keep in mind that the displayed size reflects the debug build and will be significantly smaller for a release build.

More detailed information used to generate the visualization is included in the [./dist/\[solution-name\].stats.json](#) file. By using this file, you can find out why a specific script has been included in the bundle or if a particular script is used in multiple bundles. With this information, you can optimize your bundles, improving the loading time of your solution.

Choose your primary client-side library

If there are multiple components on the same page, or even on different pages across the portal, and they all use the same library loaded from the same URL, the web browser also reuses the copy it has already cached, which lets the portal load faster.

This is why it is essential for organizations to rationalize which libraries and versions they use and where they load from, not only for a specific project but for the whole organization. Such a policy allows users working with the different applications to be more productive by loading applications faster. By reusing the previously downloaded assets, the load on the network is limited, freeing its bandwidth for other purposes.

Reference only the necessary components

Sometimes when working with external libraries, you might actually not need the whole library but only a small portion of it. Including the whole library unnecessarily adds to the size of your bundle, increasing its load time. Instead, you should always consider loading only the specific parts of a library that you actually need.

To illustrate this, take the Lodash library as an example. Lodash is a collection of utilities helping you to perform certain operations in your code. When working with Lodash, you generally only need a few specific methods rather than the complete library.

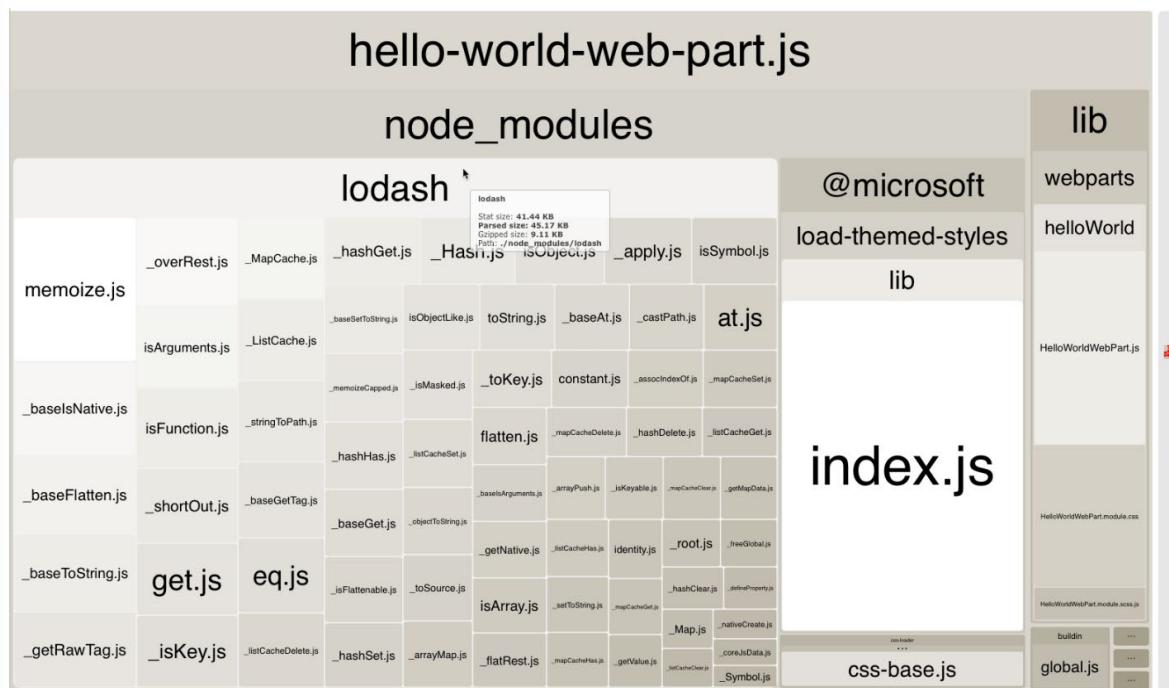
However, if you referenced the whole library by using the following code, it adds 527 KB to your unoptimized bundle:

```
import * as _ from 'lodash';
```



Instead, if you referenced only the specific Lodash method by using the following code, it adds 45 KB to your unoptimized bundle:

```
const at: any = require('lodash/at');
```



Referencing specific methods instead of the whole library comes with a price.

Currently, Lodash doesn't support loading specific methods inside of SharePoint Framework projects by using the **import** notation. Instead, you have to use a **require** statement, which doesn't offer you the typesafety capabilities that using the **import** statement does. Other libraries may present similar limitations. Eventually, it is up to you to decide if loading significantly more code into your bundles is worth preserving the typesafety capabilities.

Note:

Some of the Lodash methods are provided with the SharePoint Framework in the **@microsoft/sp-lodash-subset** library. Before using Lodash, verify if the method that you want to use isn't already available in the **@microsoft/sp-lodash-subset** library, which is already available as a part of the SharePoint Framework and does not need to be included in your bundle.

Lesson review questions

1. What commands are necessary to bundle, package, and ship your solution?
2. Where in the SharePoint Tenant should solution packages be stored and deployed from?
3. When should you use domain isolated web parts?

Correct/suggested answers:

1. Commands:
 - Command to ensure solution builds correctly: `gulp build`
 - Command to bundle and package solution:
 - `gulp bundle`
 - `gulp package-solution`

You can add the `--ship` flag to package a minified version of your components: `gulp package-solution --ship`

2. Application customizers and ListView command sets can be deployed tenant-wide. This allows app catalog managers to manage which extensions are activated by default across the tenant or are based on web/list templates in the sites.
3. Use domain isolated web parts when you need to isolate access to APIs secured with Azure AD. You can build web parts that securely communicate with APIs secured with Azure AD without exposing the access token to other components on the page or even scripts in the tenant.

Consumption of Microsoft Graph and Third Party APIs

Lesson introduction

Microsoft Graph provides a unified programmability model that you can use to build apps for organizations and consumers that interact with the data of millions of users. When building SharePoint Framework (SPFx) solutions, you can connect to Microsoft Graph by using the MSGraphClient. SPFx allows you to specify which Azure AD applications and permissions your solution requires, and a tenant administrator can grant the necessary permissions if they haven't yet been granted.

In this lesson, you'll learn about the consumption of Microsoft Graph, and review the consumption of third-party APIs secured with Azure AD from within SPFx.

When building SharePoint Framework (SPFx) solutions, you might need to connect to an API secured using Azure Active Directory (Azure AD).

After this lesson, you will be able to:

- Describe the purpose of the MSGraphClient object.
- Describe the methods for granting permissions to Microsoft Graph.
- Explain the purpose of the AadHttpClient object.
- Describe the methods for granting permissions to consume a third-party API.

Use the MSGraphClient to connect to Microsoft Graph

Developers can use SharePoint Framework (starting from v.1.4.1) to consume Microsoft Graph REST APIs, or any other REST API that's registered in Azure Active Directory (Azure AD).

MSGraphClient overview

When building SharePoint Framework solutions, you can easily connect to Microsoft Graph by using the MSGraphClient.

The MSGraphClient is a new HTTP client introduced in SharePoint Framework v1.6.0 that simplifies connecting to Microsoft Graph inside SharePoint Framework solutions. The MSGraphClient wraps the existing **Microsoft Graph JavaScript Client Library**²⁴, offering you the same capabilities you have when using the client library in other client-side solutions.

While you could use the Microsoft Graph JavaScript Client Library in your solution directly, the MSGraphClient handles authenticating against Microsoft Graph for you, which allows you to focus on building your solution.

Types of content you can access using Microsoft Graph

The following are the types of content you can access using the Microsoft Graph REST API to access data in Office 365 using the SharePoint Framework.

- User profile details

²⁴ <https://www.npmjs.com/package/@microsoft/microsoft-graph-client>

- Calendar of events
- Planner tasks

The following are examples of supported use cases for using Microsoft Graph with the SharePoint Framework.

User-centric use cases in v1.0

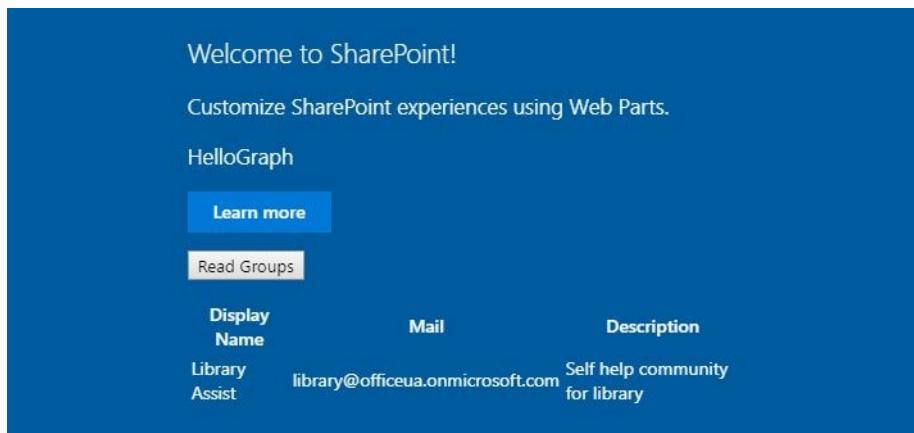
- Get the profile and photo of a user.
- Get the profile information about the user's manager and IDs of her direct reports, all stored in Azure AD.
- Access the user's files on OneDrive for Business, find the identity of the last person who modified a file there, and navigate to that person's profile.
- Access the user's calendar on Exchange Online and determine the best time for other users to meet with her team in the next two weeks.
- Subscribe to and track changes in the user's calendar, and tell user when she is spending more than 80% of her time in meetings.
- Set automatic replies when the user is away from the office.
- Get the people who are most relevant to the user, based on communication, collaboration, and business relationships.
- Get the latest sales projection from a chart in an Excel file in the user's OneDrive for Business.
- Find the tasks assigned to the user in Planner.

Office 365 group use cases in v1.0

- Run a report on Office 365 groups in an organization and identify the group with the most communication among group members.
- Find the plans of this Office 365 group, and the assignment of tasks in that plan.
- Start a new conversation in the Office 365 group to determine whether members want to create another group to share the workload.
- Get the default notebook for the group and create a page to note the outcome of the investigation.

One example might be a SharePoint Framework web part deployed and being configured to use the MSGraphClient to enable searching for users in the current tenant directly from the web part.

The following image shows a SharePoint Framework web part example that retrieves and displays a list of Office 365 groups.



Use the MSGraphClient in your SPFx solutions

The MSGraphClient can be used in both client-side web part and in SharePoint Framework extension solutions.

Note:

The single sign-on for the MSGraphClient is currently available only in SharePoint Online. You can leverage the MSGraphClient for on-premises developments, but your users will be required to sign in again within the web part.

To use the MSGraphClient in your SharePoint Framework solution, add the following import clause in your main web part file:

```
import { MSGraphClient } from '@microsoft/sp-http';
```

The MSGraphClient is exposed through the MSGraphClientFactory available on the web part context. To get a reference to the MSGraphClient, in your code you would add:

```
export default class HelloWorldWebPart extends BaseClientSideWebPart<IHelloWorldWebPartProps> {
  public render(): void {
    // ...
    this.context.msGraphClientFactory
      .getClient()
      .then((client: MSGraphClient): void => {
        // use MSGraphClient here
      });
    // ...
  }
}
```

After you have the reference to the MSGraphClient instance, start communicating with Microsoft Graph by using its JavaScript Client Library syntax:

```
export default class HelloWorldWebPart extends BaseClientSideWebPart<IHelloWorldWebPartProps> {
  public render(): void {
    // ...
```

```
        this.context.msGraphClientFactory
            .getClient()
            .then((client: MSGraphClient): void => {
                // get information about the current user from the Microsoft Graph
                client
                    .api('/me')
                    .get((error, response: any, rawResponse?: any) => {
                        // handle the response
                    });
            });
        }
        // ...
    }
}
```

Use the Microsoft Graph TypeScript types

When working with Microsoft Graph and TypeScript, you can use the **Microsoft Graph TypeScript types**²⁵ to help you catch errors in your code faster. The Microsoft Graph TypeScript types are provided as a separate package.

1. To install the Microsoft Graph TypeScript types, you would need to run: `npm install @microsoft/microsoft-graph-types --save-dev`
2. After installing the package in your project, you need to import it to your web part file: `import * as MicrosoftGraph from '@microsoft/microsoft-graph-types';`
3. Enter the objects retrieved from Microsoft Graph, for example:

```
export default class HelloWorldWebPart extends BaseClientSideWebPart<IHelloWorldWebPartProps>
{
    public render(): void {
        // ...
        this.context.msGraphClientFactory
            .getClient()
            .then((client: MSGraphClient): void => {
                // get information about the current user from the Microsoft Graph
                client
                    .api('/me')
                    .get((error, user: MicrosoftGraph.User, rawResponse?: any) => {
                        // handle the response
                    });
            });
        }
        // ...
    }
}
```

²⁵ <https://www.npmjs.com/package/@microsoft/microsoft-graph-types>

Grant permissions to Microsoft Graph

By default, the service principal has no explicit permissions granted to access Microsoft Graph. However, if you request an access token for Microsoft Graph, you get a token with the **user_impersonation** permission scope, which can be used for reading information about the users (that is, **User.Read.All**).

When building SharePoint Framework solutions, you can connect to Microsoft Graph by using the `MSGraphClient`. Microsoft Graph provides a unified programmability model that you can use to build apps for organizations and consumers that interact with the data of millions of users.

Understand the AadHttpClient

Azure AD secures a variety of resources, from Office 365 to custom line-of-business applications built by the organization. To connect to these resources, applications must obtain a valid access token that grants them access to a particular resource. Applications can obtain an access token as part of the OAuth authorization flow.

Client-side applications that are incapable of storing a secret, such as SharePoint Framework solutions, use a specific type of OAuth flow named OAuth implicit flow.

Developers building client-side solutions are responsible for implementing authorization by using the OAuth implicit flow in their application. In SharePoint Framework solutions, that's already done as part of the framework through the `MSGraphClient` and the `AadHttpClient`, both of which are introduced in SharePoint Framework v1.4.1.

Connect to Azure AD-secured APIs in SharePoint Framework solutions

By using the `AadHttpClient`, you can easily connect to APIs secured using Azure AD without having to implement the OAuth flow yourself.

As part of the SharePoint Framework, a specific process is defined for how developers can request permissions and tenant administrators can manage permissions to resources secured with Azure AD. The following schema illustrates this process.

Developers who build a SharePoint Framework solution that requires access to specific resources secured with Azure AD list these resources along with the required permission scopes in the solution manifest. When deploying the solution package to the app catalog, SharePoint creates permission requests and prompts the administrator to manage the requested permissions. For each requested permission, tenant administrators can decide whether they want to grant or deny the specific permission.

All permissions are granted to the whole tenant and not to a specific application that has requested them. When the tenant administrator grants a specific permission, it is added to the SharePoint Online Client Extensibility Azure AD application, which is provisioned by Microsoft in every Azure AD and which is used by the SharePoint Framework in the OAuth flow to provide solutions with valid access tokens.

Discover available applications and permissions

The target Azure AD that secures your Office 365 tenant determines which applications you can request permissions for in your solution. The list of available applications might depend on the Office 365 license that the organization is using and which line-of-business applications they registered in Azure AD. If you have sufficient permissions, there are several ways to see which applications and permission scopes are available in your tenant.

Use Azure portal or Azure AD admin center

One way to see the available applications in Azure AD is by navigating to the [Azure portal²⁶](#) or the [Azure AD admin center²⁷](#).

1. In the Azure AD admin center, in the left navigation, select **Enterprise applications**.

2. On the **Enterprise applications** blade, in the **Manage** group, select **All applications**.

3. To quickly find the application to which you want to connect, you can filter the overview either by application type (Microsoft Applications or Enterprise Applications), or you can search for it by using its name or ID. For example, if you want to request additional permissions to Microsoft Graph, search for **graph** in the search box.

²⁶ <https://ms.portal.azure.com/>

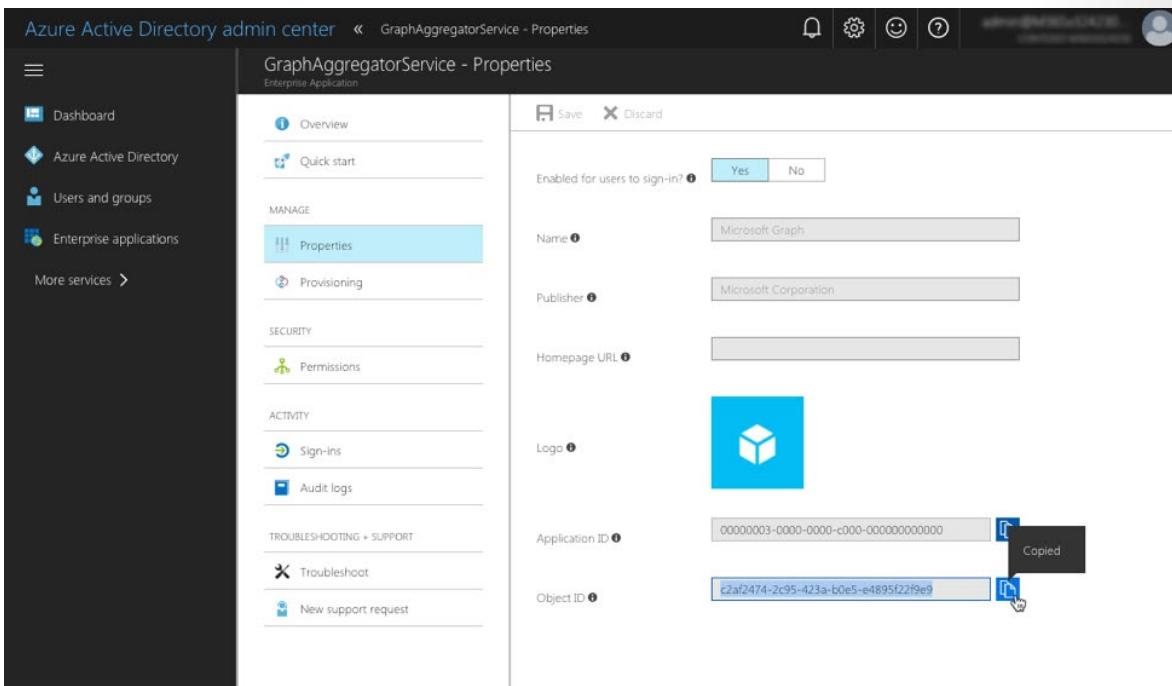
²⁷ <https://aad.portal.azure.com/>

The screenshot shows the Azure Active Directory admin center interface. On the left, there's a navigation sidebar with options like Dashboard, Azure Active Directory, Users and groups, Enterprise applications, and More services. The main area is titled "Enterprise applications - All applications" and shows a list of applications. One application, "GraphAggregat", is listed with its details: Name (GraphAggregat), Application ID (c2af2474-2c95-423a...), Object ID (00000003-0000-0000...), and Publisher (Microsoft Corporation). There are filters at the top for Show (Microsoft Applications), Applications status (Any), Application visibility (Any), and a search bar for graph.

4. After you find the application, select it to get additional information. On the application's blade, in the **Manage** group, select **Properties**.

The screenshot shows the "GraphAggregatorService" application blade in the Azure Active Directory admin center. The left sidebar has the same navigation as before. The main area is titled "GraphAggregatorService" and "Enterprise Application". Under the "MANAGE" section, the "Properties" option is selected and highlighted with a cursor. To the right, there's a summary card with "Total Users" (0) and a chart titled "App usage between 20/11/2017 and 20/12/2017" showing usage over time.

5. From the list of properties, copy the value of the **Object ID** property, which you need to request additional permission scopes to Microsoft Graph. Alternatively, you can copy the application's **Name** and use it in the permission request instead.



Note:

While the **Object ID** is unique for each tenant, the application's **Name** is the same across all tenants. If you want to build your solution once and deploy it to different tenants, you should use the application's Name when requesting additional permissions in your solution.

Connect to Azure AD applications using the AadHttpClient

Starting with version 1.4.1, SharePoint Framework simplifies connecting to APIs secured with Azure AD. Using the AadHttpClient, you can easily connect to APIs secured with Azure AD without having to implement authentication and authorization yourself.

Internally, the AadHttpClient implements the Azure AD OAuth flow using ADAL JS by using the SharePoint Online Client Extensibility service principal to obtain a valid access token. The SharePoint Online Client Extensibility service principal is provisioned by Microsoft and is available in the Azure AD of all Office 365 tenants. To use the AadHttpClient in your SharePoint Framework solution:

1. Add the following import clause in your main web part file: `import { AadHttpClient, HttpClientResponse } from '@microsoft/sp-http';`
2. Get a new instance of the AadHttpClient, passing the resource to which you want to connect as parameters:

```
export default class HelloWorldWebPart extends BaseClientSideWebPart<IHelloWorldWebPartProps>
{
  public render(): void {
    // ...
    this.context.aadHttpClientFactory
      .getClient('https://contoso.azurewebsites.net')
      .then((client: AadHttpClient): void => {
        // connect to the API
      });
  }
}
```

```
// ...  
}
```

Note:

Each instance of the AadHttpClient is linked to the specific resource, which is why you need to create a new instance of the client for each resource to which you want to connect.

After you instantiate the AadHttpClient for your resource, issue a web request to communicate with your API; in this example, the API returns a list of orders represented by the custom **Order** interface defined elsewhere in the project:

```
export default class HelloWorldWebPart extends BaseClientSideWebPart<IHelloWorldWebPartProps> {  
    public render(): void {  
        // ...  
        this.context.aadHttpClientFactory  
            .getClient('https://contoso.azurewebsites.net')  
            .then((client: AadHttpClient): void => {  
                client  
                    .get('https://contoso.azurewebsites.net/api/orders',  
                        AadHttpClient.configurations.v1)  
                    .then((response: HttpClientResponse): Promise<Order[]>  
                        => {  
                            return response.json();  
                        })  
                    .then((orders: Order[]): void => {  
                        // process data  
                    });  
            })  
        // ...  
    }  
}
```

Considerations

The following are some considerations that you should take into account when working with web API permissions.

Request permissions via SharePoint Framework solutions

Currently, it's only possible to request additional permissions through a SharePoint Framework solution. The request is initiated when the solution package (.sppkg) containing a permissions request is deployed in the app catalog. After the request is initiated, it can be approved or denied by the tenant administrator.

Granted permissions apply to all solutions

Although permissions to Azure AD resources are requested by a SharePoint Framework solution, once granted, they apply to the whole tenant and can be leveraged by any solution in that tenant.

Removing the solution doesn't revoke permissions

Removing the solution that initially requested the particular permission doesn't revoke the granted permission. Tenant administrators must manually revoke permissions granted through SharePoint Framework application requests.

Revoking previously granted permissions doesn't invalidate issued access tokens

Revoking previously granted permissions doesn't invalidate access tokens issued to users. Instead, these access tokens remain valid until they expire.

Permission request doesn't affect solution deployment

Regardless of whether the administrator denies or approves permissions requested by the solution, the solution can be deployed and used on sites. When building solutions that require additional permissions, you should never assume that the requested permissions have been granted.

Control the SharePoint Online Client service principal

All permissions granted through web API requests are stored with the SharePoint Online Client Extensibility Azure AD application. If tenant administrators don't want developers to use the web API request model and the MSGraphClient and AadHttpClient in their solutions, they can disable the SharePoint Online Client Extensibility service principal through PowerShell by using the **Disable-SPOTenantServicePrincipal** cmdlet.

The service principal can be re-enabled by using the Enable-SPOTenantServicePrincipal cmdlet. Alternatively, it's also possible to enable and disable the SharePoint Online Client Extensibility service principal through the Office 365 CLI by using the **spo serviceprincipal set²⁸** command.

Grant permissions to consume a third-party API

When building SharePoint Framework solutions, you might want to consume public APIs, such as stock or weather information, which connects anonymously.

Connect to anonymous APIs using the HttpClient

The easiest way to connect to anonymous APIs in your SharePoint Framework solutions is by using the HttpClient provided as a part of the SharePoint Framework.

For example, to get dummy information from the Typicode service, you would execute:

```
this.context.httpClient
    .get('https://jsonplaceholder.typicode.com/todos/1', HttpClient.configurations.v1)
    .then((res: HttpClientResponse): Promise<any> => {
        return res.json();
    })
    .then((response: any): void => {
        console.log(response);
```

²⁸ <https://pnp.github.io/office365-cli/cmd/spo/serviceprincipal/serviceprincipal-set/>

```
});
```

Similar to the SPHttpClient you use for connecting to SharePoint APIs, the HttpClient offers you similar capabilities for performing the most common web requests. If necessary, you can use its options to configure requests. For example, to specify request headers, you would use the following code:

```
this.context.httpClient
    .get('https://jsonplaceholder.typicode.com/todos/1', HttpClient.configurations.v1,
    {
        headers: [
            ['accept', 'application/json']
        ]
    })
    .then((res: HttpClientResponse): Promise<any> => {
        return res.json();
    })
    .then((response: any): void => {
        console.log(response);
    });
});
```

Considerations for using the HttpClient

When using the HttpClient, there are a few things that you should take into account.

Authentication cookies not included

While the HttpClient is very similar to the SPHttpClient, it doesn't include authentication cookies in its requests. So if you were to use it to connect to SharePoint APIs, your requests would fail with a **401 Unauthorized** response.

Part of the SharePoint Framework

The HttpClient is part of the SharePoint Framework, and you don't need any additional dependencies to start using it. It is already available on the page, which is why using it doesn't cause additional performance overhead on runtime.

The SharePoint Framework allows you to specify which Azure AD applications and permissions your solution requires, and a tenant administrator can grant the necessary permissions if they haven't yet been granted.

This lesson covered the consumption of third-party APIs secured with Azure AD from within SPFx, including granting permissions to consume a third-party API.

Lesson review questions

1. What types of content can you access using the MSGraphClient to integrate within your SharePoint Framework solutions?
2. To use the MSGraphClient in your SharePoint Framework solution, what import clause will you need to add in your main web part file?

3. What is the purpose of the AadHttpClient object?
4. What are the methods for granting permissions to consume a third-party API?

Correct/suggested answers:

1. User profile information, calendar items, Office 365 groups, and OneDrive files are examples of content you can access using the MSGraphClient.
2. import { MSGraphClient } from '@microsoft/sp-http';
3. Azure AD secures a variety of resources, from Office 365 to custom line-of-business applications built by the organization. To connect to these resources, applications must obtain a valid access token that grants them access to a particular resource. Applications can obtain an access token as part of the OAuth authorization flow. Client-side applications that are incapable of storing a secret, such as SharePoint Framework solutions, use a specific type of OAuth flow named OAuth implicit flow. Developers building client-side solutions are responsible for implementing authorization by using the OAuth implicit flow in their application. In SharePoint Framework solutions, that's already done as part of the framework through the MSGraphClient and the AadHttpClient, both of which are introduced in SharePoint Framework v1.4.1. By using the AadHttpClient, you can easily connect to APIs secured by using Azure AD without having to implement the OAuth flow yourself.
4. The easiest way to connect to third-party APIs is by using the HttpClient provided as part of the SharePoint Framework.

Web Parts as Teams Tabs

Lesson introduction

Starting with SharePoint Framework v1.8, you can build tabs for Microsoft Teams using SharePoint Framework (SPFx) tooling and use SharePoint as a host for your solutions.

In this lesson, you'll learn about SPFx web parts as Teams tabs, including creating an SPFx web part and deploying an SPFx web part as a Teams tab.

After this lesson, you will be able to:

- Describe the considerations for creating an SPFx web part to be a Teams tab.
- Explain the options for deploying an SPFx web part as a Teams tab.

Create a SPFx web part

SharePoint Framework in Teams

You can implement your Microsoft Teams tabs using SharePoint Framework. For SharePoint developers, this significantly simplifies the development process for Teams tabs because SharePoint Framework web parts are hosted within SharePoint without any need for external services such as Azure.

Building Microsoft Teams Tabs using SharePoint Framework

You can build tabs for Microsoft Teams using SharePoint Framework tooling and use SharePoint as a host for your solutions.

The following are benefits of using SharePoint Framework as the platform for your Microsoft Teams tabs:

- The development model is similar to SharePoint Framework web parts.
- Technically, any web part can be exposed as a tab in Microsoft Teams.
- You have different scoping options for exposing your custom tab as a web part and tab in your tenant.
- Your tab will be executed in the context of the underlying SharePoint site behind the specific team. This means that you can take advantage of any SharePoint-specific APIs or functionalities in your web part.

Development process

You can start developing Microsoft Teams tabs by simply using the SharePoint Framework 1.8 or later packages. The following are the high-level steps involved in the development process.

1. Create a SharePoint Framework solution with a client-side web part.
2. Add “**TeamsTab**” to the **supportedHosts** property of the web part manifest: “**supportedHosts**”: [“**SharePointWebPart**”, “**TeamsTab**”],
3. Deploy the web part using a tenant-scoped deployment option to your SharePoint app catalog.
4. Create the Microsoft Teams app manifest file, **manifest.json**, and save it in the ./teams folder of the solution.

5. Zip the contents of the **./teams** folder; this is the Microsoft Teams app package.
6. Use the Microsoft Teams app package to side-load the application in Microsoft Teams.

Creating the Microsoft Teams manifest manually for a web part and deploying it to Microsoft Teams

One of the most important files needed to deploy a web part as a Teams tab in Microsoft Teams is the app manifest.

To side-load a SharePoint Framework web part as a Microsoft Teams application, you must create a Microsoft Teams application manifest file with specific values. The code below is a sample manifest template file that can be used as a starting template.

```
{
  "$schema": "https://developer.microsoft.com/en-us/json-schemas/teams/v1.2/MicrosoftTeams.schema.json",
  "manifestVersion": "1.2",
  "packageName": "{{SPFX_COMPONENT_ALIAS}}",
  "id": "aa3fecf0-1fd0-4751-abal-12314dc3a22f",
  "version": "0.1",
  "developer": {
    "name": "Parker Porcupine",
    "websiteUrl": "https://products.office.com/en-us/sharepoint/collaboration",
    "privacyUrl": "https://privacy.microsoft.com/en-us/privacystatement",
    "termsOfUseUrl": "https://www.microsoft.com/en-us/servicesagreement"
  },
  "name": {
    "short": "{{SPFX_COMPONENT_NAME}}"
  },
  "description": {
    "short": "{{SPFX_COMPONENT_SHORT_DESCRIPTION}}",
    "full": "{{SPFX_COMPONENT_LONG_DESCRIPTION}}"
  },
  "icons": {
    "outline": "{{SPFX_COMPONENT_ID}}_outline.png",
    "color": "{{SPFX_COMPONENT_ID}}_color.png"
  },
  "accentColor": "#004578",
  "configurableTabs": [
    {
      "configurationUrl": "https://{{teamSiteDomain}}{{teamSitePath}}/_layouts/15/TeamsLogon.aspx?SPFX=true&dest={{teamSitePath}}/_layouts/15/teamhostedapp.aspx%3FopenPropertyPane=true%26teams%26componentId={{SPFX_COMPONENT_ID}}%26forceLocale={{locale}}",
      "canUpdateConfiguration": true,
      "scopes": [
        "team"
      ]
    }
  ]
}
```

```
        }
    ],
    "validDomains": [
        "*login.microsoftonline.com",
        "*sharepoint.com",
        "*sharepoint-df.com",
        "spoppe-a.akamaihd.net",
        "spoprod-a.akamaihd.net",
        "resourceseng.blob.core.windows.net",
        "msft.spoppe.com"
    ],
    "webApplicationInfo": {
        "resource": "https://teamSiteDomain",
        "id": "00000003-0000-0ff1-ce00-000000000000"
    }
}
```

The following are important attributes to update in the manifest file:

- **packageName**: The technical name of the package.
- **id**: The unique package id. Generate a new GUID for this attribute.
- **name:Short**: The name of your app (no more than 30 chars).
- **description:Short**: A short description of your app.
- **description:full**: A full description of your app.
- **icons:outline**: The outline icon is used in these places: the app bar and messaging extensions the user has marked as a “favorite.” This icon must be 32x32 pixels.
- **icons:color**: The color icon is used throughout Microsoft Teams (in app and tab galleries, bots, flyouts, and so on). This icon should be 192x192 pixels.
- **configurationUrl**: You should adjust the sample **configurationUrl** to include the manifest id of your web part component by updating the GUID after the **componentId** query parameter.
- **canUpdateConfiguration**: Setting this property to true means that you allow modification of the properties of the web part. Set this value to false to prevent this.

Notice in the template manifest file example above, there are multiple placeholders in the code. You would need to replace these values from the SharePoint Framework web part manifest file (**./src/web-parts/[webpartfoldername]/[webpartname].manifest.json**). The table below is a guide of the placeholder mappings.

manifest.json string	Property in SPFx component manifest
{{SPFX_COMPONENT_ALIAS}}	alias
{{SPFX_COMPONENT_NAME}}	preconfiguredEntries[0].title
{{SPFX_COMPONENT_SHORT_DESCRIPTION}}	preconfiguredEntries[0].description
{{SPFX_COMPONENT_LONG_DESCRIPTION}}	preconfiguredEntries[0].description
{{SPFX_COMPONENT_ID}}	Id

Notice the **componentId** query parameter in the **configurationUrl**. You should not update any other sections of the URL, because it's dynamically replaced when the tab is rendered from the context of SharePoint.

There are also two other placeholders that need to be replaced in the Teams manifest file for the icons. The example below is the current which need to be replaced with the actual icon files names that would be in the solution project "teams" folder.

```

    "icons": {
        "outline": "{{SPFX_COMPONENT_ID}}_outline.png",
        "color": "{{SPFX_COMPONENT_ID}}_color.png"
    },

```

The following JSON structure demonstrates the sample manifest file with the placeholders replaced.

```

{
    "$schema": "https://developer.microsoft.com/en-us/json-schemas/teams/v1.2/MicrosoftTeams.schema.json",
    "manifestVersion": "1.2",
    "packageName": "SpFxTeamsTogetherWebPart",
    "id": "aa3fecf0-1fd0-4751-aba1-12314dc3a22f",
    "version": "0.1",
    "developer": {
        "name": "Parker Porcupine",
        "websiteUrl": "https://products.office.com/en-us/sharepoint/collaboration",
        "privacyUrl": "https://privacy.microsoft.com/en-us/privacystatement",
        "termsOfUseUrl": "https://www.microsoft.com/en-us/servicesagreement"
    },
    "name": {
        "short": "SPFx Teams Together"
    },
    "description": {
        "short": "SPFx Teams Together short description",
        "full": "SPFx Teams Together long description"
    },
    "icons": {
        "outline": "b7771434-9587-4a79-9990-48c310f78a3d_outline.png",
        "color": "b7771434-9587-4a79-9990-48c310f78a3d_color.png"
    },
    "accentColor": "#004578",
    "configurableTabs": [
        {
            "configurationUrl": "https://{teamSiteDomain}{teamSitePath}/_layouts/15/TeamsLogon.aspx?SPFX=true&dest={teamSitePath}/_layouts/15/teamshostedapp.aspx%3FopenPropertyPane=true%26teams%26componentId=b7771434-9587-4a79-9990-48c310f78a3d%26forceLocale={locale}",
            "canUpdateConfiguration": true,
            "scopes": [
                "team"
            ]
        }
    ],
    "validDomains": [

```

```
    "* .login.microsoftonline.com",
    "* .sharepoint.com",
    "* .sharepoint-df.com",
    "spoppe-a.akamaihd.net",
    "spoprod-a.akamaihd.net",
    "resourceseng.blob.core.windows.net",
    "msft.spoppe.com"
],
"webApplicationInfo": {
    "resource": "https://teamSiteDomain",
    "id": "00000003-0000-0ff1-ce00-000000000000"
}
}
```

Updating the web part manifest to make it available for Microsoft Teams

The next step in the development process is modifying the manifest JSON file for the web part you want to make available to Teams. Below is an example of the **supportedHosts** properties modified to include **"TeamsTab"**.

```
{
    "$schema": "https://developer.microsoft.com/json-schemas/spfx/client-
side-web-part-manifest.schema.json",
    "id": "8eb36405-e408-4578-8340-faf16d647d83",
    "alias": "MyFirstTeamsTab",
    "componentType": "WebPart",
    // The "*" signifies that the version should be taken from the package.
    json
        "version": "*",
        "manifestVersion": 2,
        // If true, the component can only be installed on sites where Custom
        Script is allowed.
        // Components that allow authors to embed arbitrary script code should
        set this to true.
        // https://support.office.com/en-us/article/Turn-scripting-capabilities-
        on-or-off-1f2c515f-5d7e-448a-9fd7-835da935584f
        "requiresCustomScript": false,
    "supportedHosts": ["SharePointWebPart", "TeamsTab"],
    "preconfiguredEntries": [
        {
            "groupId": "5c03119e-3074-46fd-976b-c60198311f70", // Other
            "group": { "default": "Other" },
            "title": { "default": "MyFirstTeamsTab" },
            "description": { "default": "MyFirstTeamsTab description" },
            "officeFabricIconFontName": "Page",
            "properties": {
                "description": "MyFirstTeamsTab"
            }
        }
    ]
}
```

```
}
```

Updating code to be aware of the Microsoft Teams context

The next step in the process would be to update your web part code file. For example, the `src\web-parts[webpartfoldername][webpartname].ts` file would need to be modified with the edits necessary to make the solution aware of the Microsoft Teams context, if it's used as a tab.

1. This import statement would need to be added at the top of web part code file:
`import * as microsoftTeams from '@microsoft/teams-js';`
2. After that, the following private variable would need to be added inside your web part class. You will be storing information about the Microsoft Teams context in this variable. In the example below, the private variable was added to the **MyFirstTeamsTabWebPart** class.

```
export default class MyFirstTeamsTabWebPart extends BaseClientSideWebPart<IMyFirstTeamsTabWebPartProps> {
    // This variable has been added
    private _teamsContext: microsoftTeams.Context;
```

3. The next step is to add a new `onInit` method inside the class, just below the private variable that was added with the following content:

```
protected onInit(): Promise<any> {
    let retVal: Promise<any> = Promise.resolve();
    if (this.context.microsoftTeams) {
        retVal = new Promise((resolve, reject) => {
            this.context.microsoftTeams.getContext(context => {
                this._teamsContext = context;
                resolve();
            });
        });
    }
    return retVal;
}
```

4. Next step is to update the **render** method as follows. Notice how the code is rendering different content depending on whether the code is rendered as a tab in Microsoft Teams or as a web part in SharePoint.

```
public render(): void {
    let title: string = "";
    let subTitle: string = "";
    let siteTabTitle: string = "";
    if (this._teamsContext) {
        // We have teams context for the web part
        title = "Welcome to Teams!";
        subTitle = "Building custom enterprise tabs for your business.";
        siteTabTitle = "We are in the context of following Team: " + this._teamsContext.teamName;
    }
    else
    {
```

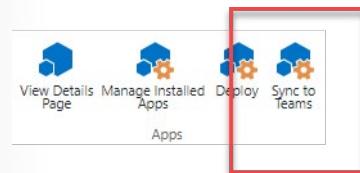
```
// We are rendered in normal SharePoint context
title = "Welcome to SharePoint!";
subTitle = "Customize SharePoint experiences using Web Parts.";
siteTabTitle = "We are in the context of following site: " + this.context.pageContext.web.title;
}
this.domElement.innerHTML =
<div class="${ styles.myFirstTeamsTab }">
  <div class="${ styles.container }">
    <div class="${ styles.row }">
      <div class="${ styles.column }">
        <span class="${ styles.title }">${title}</span>
        <p class="${ styles.subTitle }">${subTitle}</p>
        <p class="${ styles.description }">${siteTabTitle}</p>
        <p class="${ styles.description }">Description property value - ${escape(this.properties.description)}</p>
        <a href="https://aka.ms/spfx" class="${ styles.button }">
          <span class="${ styles.label }">Learn more</span>
        </a>
      </div>
    </div>
  </div>
</div>;
}
```

Deploy a SPFx web part as a Teams tab

There are multiple options to deploy Microsoft Teams tabs. Because both SharePoint and Microsoft Teams have their own app catalogs, deployment requires operations on both services. Visibility of the new functionality can be controlled by the deployment steps taken.

Tenant deployment

You can use the **Sync to Teams** button in the App Catalog ribbon to automatically create the Microsoft Teams app manifest and app package and install it in the Microsoft Teams store. This will make your solution available for all users in your tenant and Microsoft Teams teams.



This capability is ideal when you have a simple implementation that you want expose automatically in Microsoft Teams. However, Teams solutions can be complex and contain multiple capabilities such as tabs, bots, and actionable cards. For such complex solutions, crafting the Teams manifest manually and deploying it in the Teams App Catalog directly is the only option.

Alternative deployment options

There are two other alternative deployment options for deploying your solution:

- Single team deployment (make solution available to one specific team).
- Tenant-wide deployment (make solution available to all teams within organization).

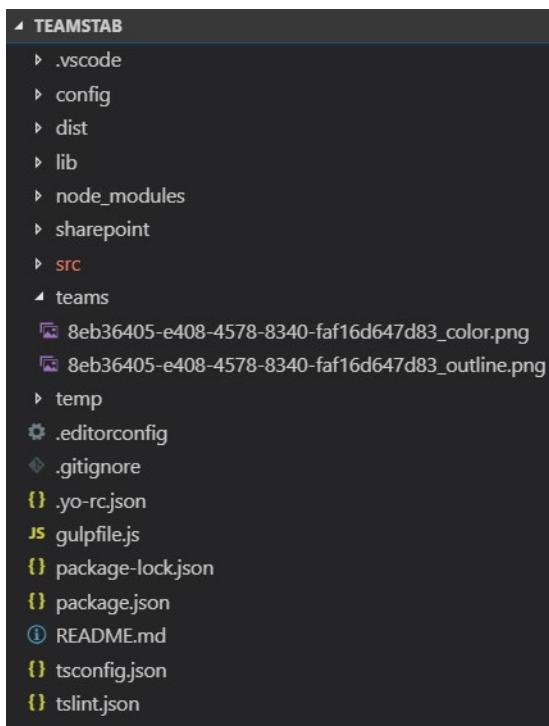
Single team packing and deploying solution

The following alternative solution deployment method will allow you to make a solution available only to one specific team in your tenant. The steps involve include:

1. Building a SharePoint Framework solution the normal way:

```
gulp build  
gulp bundle --ship  
gulp package-solution --ship
```

2. Locate the `./teams` folder in the project folder:



3. The folder will contain two image files. Add the manifest file created to this folder and call it ***manifest.json**.
4. Zip contents of the folder into a zip file. The zip file should only contain the manifest.json and the two images.
5. Add the solution to the App Catalog, and make sure to select the option **Make this solution available to all sites in the organization** before selecting **Deploy**.

Tenant-wide packing and deploying solution

To deploy tenant-wide, you would run through this packaging and deployment process instead.

1. First step would be to run the build command for the SPFx web part solution.
2. The following commands below would need to be executed to build and bundle the solution. This creates a release build of the project by using a dynamic label as the host URL for the assets. This URL is automatically updated based on the tenant CDN settings.

```
gulp build  
gulp bundle --ship
```

3. The next command to execute is the following task to package the solution. This creates an updated **teams-tab-webpart.sppkg** package in the **sharepoint/solution** folder.

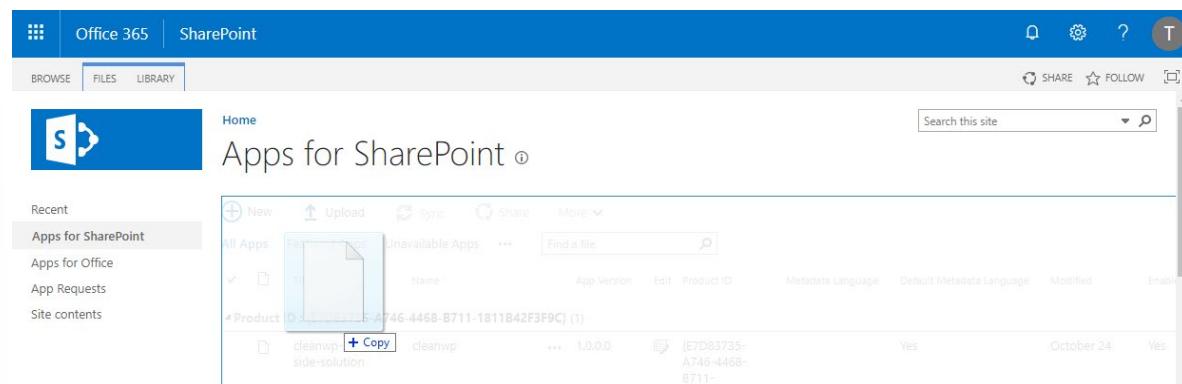
```
gulp package-solution --ship
```

4. Next, the packaged solution that gets generated from the package-solution task would need to be deployed to the SharePoint App Catalog site collection.

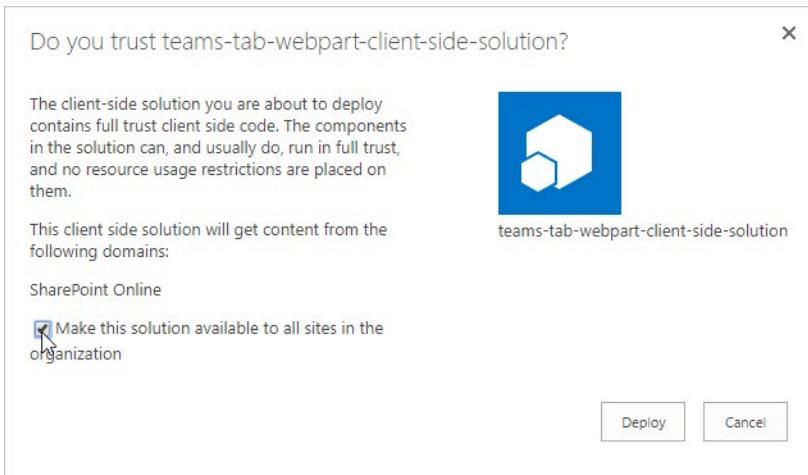
Note:

If you do not have an app catalog, an app catalog can be created through the SharePoint Online admin center.

5. From the app catalog site (for example <https://www.contoso.com/sites/apps>), the **mypackagename.sppkg** would need to be uploaded. This deploys the client-side solution package. Because this is a full-trust client-side solution, SharePoint displays a dialog box that asks you to trust the client-side solution to deploy.



6. The **domain** list in the prompt says **SharePoint Online**. This is because the content is either served from the Office 365 CDN or from the App Catalog, depending on the tenant settings. Ensure the **Make this solution available to all sites in the organization** option is selected, so the web part can be used from the Microsoft Teams side.



7. Next step would be to select **Deploy**. If there are errors when you are deploying, you'll be able to see any exceptions or issues in the package by looking at the **App Package Error Message** column in the App Catalog.
8. Once the solution is deployed successfully, the web part becomes automatically available across the SharePoint Online sites.

Note:

In this example, you used a tenant-wide deployment option of the SharePoint Framework solution. This ensures that the development and usage experience is as easy as possible. You could also deploy the solution as site scope, but in that case you'd need to ensure that the solution was deployed on the SharePoint site behind Microsoft Teams before you could use it.

Make the web part available in Microsoft Teams

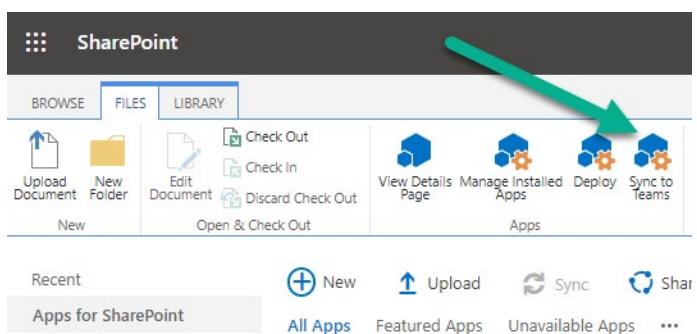
To make your web part available in Microsoft Teams, you must synchronize your solution with teams.

Note:

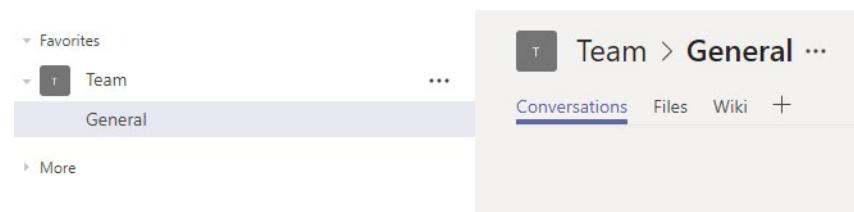
In this example, you use the automatic deployment option for the solution from the SharePoint App Catalog. If you want to provide alternative settings for your solutions, you can perform these steps manually.

These next steps below are an example of the process involved to make your web part available in Microsoft Teams.

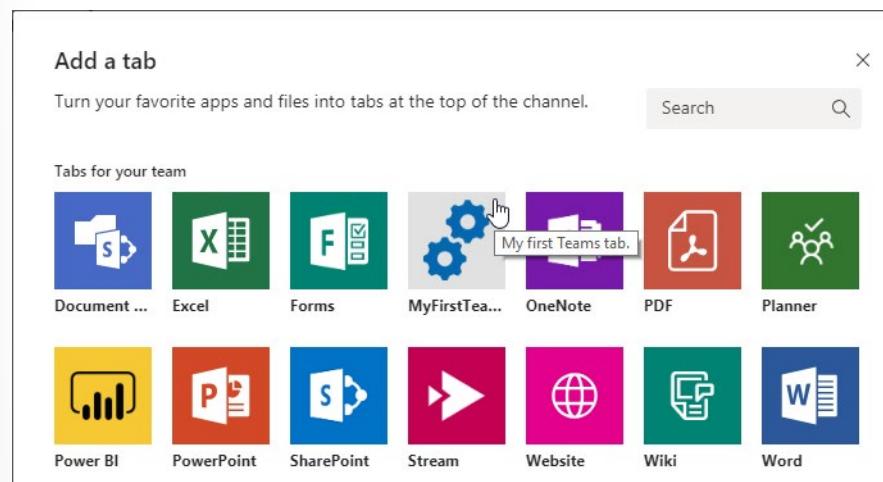
1. You would need to select the **deployed solution** in the SharePoint tenant App Catalog and select the **Sync to Teams** button on the **Files** tab.



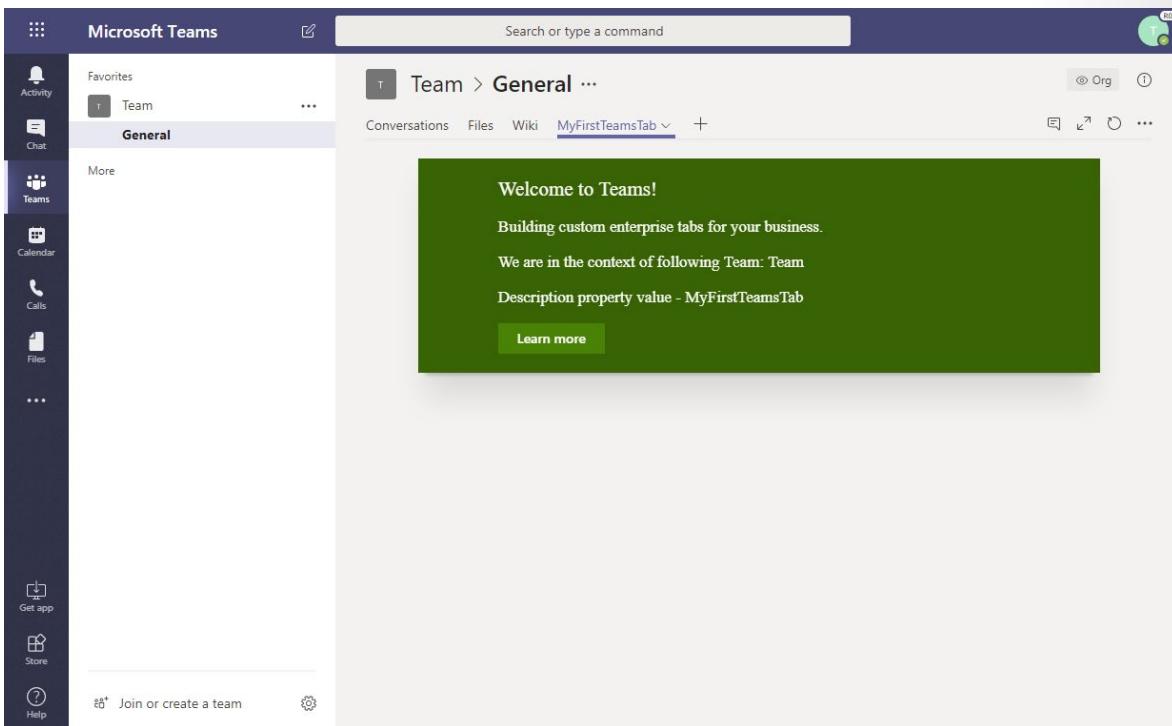
2. The next step is to move to a channel in Teams that you would want to make the solution available to.



3. You would proceed with selecting + to add a new tab on the channel.
4. In the **Add a tab** dialog box, there will be list that you'd scroll to the bottom of the list and select **More Apps**.
5. Select **Upload a custom app** > **Upload for** from the list of app categories.
6. Select the Microsoft Teams application ZIP file. This is the file that contains the **manifest.json** and two image files.
7. The application will appear next to the tenant name. Sometimes you'll need to refresh the page for the app to appear if you are using the browser Microsoft Teams client.
8. Select the custom tab, called **MyFirstTeamTab**, in the list.



Your custom tab has been added on the Microsoft Teams channel, and you can see how the code is reacting in the Microsoft Teams context. The theme of the web part comes from the underlying SharePoint site by default.



Update your app package

When you make changes to your app and create a new package, you might encounter an error when selecting "sync to teams." A notification that says "Failed to sync solution to teams" appears in the top-right corner of your page.

If this happens, there are additional steps you can follow to resolve the issue. You would need to delete the app from Microsoft Teams and then try to sync it again:

1. Open Microsoft Teams.
2. When viewing the team channel, select + to add a tab.
3. Select the **More apps** link at the top.
4. Find the app in the list, and select the ... menu.
5. Select **Delete** to remove the app from Microsoft Teams. After that you should be able to sync the new version to Microsoft Teams.

This lesson covered SPFx web parts as Teams tabs, including creating an SPFx web part and deploying an SPFx web part as a Teams tab.

Lesson review questions

1. What are the considerations for creating an SPFx web part to be a Teams tab?
2. What are the options for deploying an SPFx web part as a Teams tab?

Correct/suggested answers:

1. Considerations:

- You can build tabs for Microsoft Teams using SharePoint Framework tooling and use SharePoint as a host for your solutions.
- There are following benefits of using SharePoint Framework as the platform for your Microsoft Teams tabs:
 - The development model is similar to SharePoint Framework web parts.
 - Technically, any web part can be exposed as a tab in Microsoft Teams.
 - You have different scoping options for exposing your custom tab as a web part and tab in your tenant.
 - Your tab will be executed in the context of the underlying SharePoint site behind the specific team. This means that you can take advantage of any SharePoint-specific APIs or functionalities in your web part.
- 2. There are multiple options for deploying a Microsoft Teams tab. Tenant deployment allows you to use the Sync with Teams button in the App Catalog ribbon that will automatically create the Microsoft Teams app manifest and app package and install it in the Microsoft Teams store. This will make your solution available for all users in your tenant and Microsoft Teams teams. Alternative deployment options provide a way to deploy your solution, such allowing you to make a solution available only to one specific team in your tenant.

Branding and Theming in SharePoint

Lesson introduction

Branding and theming are options to customize a SharePoint site. When building SharePoint Framework customizations, you use theme colors so that your customizations look like a part of the site.

In this lesson, you'll learn options for branding and theming in SharePoint, including SharePoint themes, site designs, and site scripts.

After this lesson, you will be able to:

- Explain the options for SharePoint site theming.
- Describe the options for site designs and site scripts.

SharePoint site theming

Like the Microsoft brand palette, the SharePoint themes are designed to build on the Microsoft brand, while at the same time allowing for flexibility to enliven your partnerships without dominating them. They reveal shared goals and personality, and they reflect diversity and ability to optimize the SharePoint experience.

Themes provide a quick and easy way to apply lightweight branding to a SharePoint site. A theme lets a site owner, or a user who has designer rights, customize a site by changing the site layout, color palette, font scheme, and background image.

The theming experience in SharePoint was redesigned to simplify the process of customizing sites. The themes user interface was redesigned and there is a set of new file formats related to themes. The **Change the look** wizard is the entry point to the theming experience where site owners or users with designer rights can change the look and feel of the sites. They can select a design for the site. Then, they can customize the design by changing the site layout, background, color palette, or font scheme. They can preview the site before applying the design.

These features include:

- The ability to define custom themes and make them available to site owners. Themes are defined in a **JSON schema**²⁹ that stores color settings and related metadata for each theme.
- An online **Theme Generator tool**³⁰ that you can use to define new custom themes.
- A simplified set of default themes, with six light themes and two dark themes presently available.
- An updated color palette, with 12 light colors and 6 dark colors, as well as 16 supplementary themes.
- Control over which themes are available for use on pages within your sites. For example, you can define custom themes based on your organization's branding or identity, and make those the only available themes within your sites.

These capabilities are available to administrators via **PowerShell cmdlets**³¹, and to developers via the **SharePoint client-side object model (CSOM)**³² or the **SharePoint REST API**³³.

²⁹ <https://docs.microsoft.com/en-us/sharepoint/dev/declarative-customization/site-theming/sharepoint-site-theming-json-schema>

³⁰ <https://aka.ms/themedesigner>

³¹ <https://docs.microsoft.com/en-us/sharepoint/dev/declarative-customization/site-theming/sharepoint-site-theming-powershell>

³² <https://docs.microsoft.com/en-us/sharepoint/dev/declarative-customization/site-theming/sharepoint-site-theming-csom>

³³ <https://docs.microsoft.com/en-us/sharepoint/dev/declarative-customization/site-theming/sharepoint-site-theming-rest-api>

Theming experience components

The theming experience includes the following:

Color palette

A color palette, or color scheme, defines the combination of colors that are used in a site. Thirty-two color palettes are installed with SharePoint. They are available to users when they choose to modify a design in the design gallery.

Color palettes are XML files (.spcolor files). They are stored in the Theme Gallery of the root site of the site collection in the 15 folder ([http:// SiteCollectionName/_catalogs/theme/15/](http://SiteCollectionName/_catalogs/theme/15/)).

Font scheme

A font scheme defines the fonts that are used in a site. Seven font schemes are included in SharePoint. They are available to users when they choose to modify a design in the design gallery.

Font schemes are XML files (.spfont files). They are stored in the Theme Gallery of the root site of the site collection in the 15 folder ([http:// SiteCollectionName/_catalogs/theme/15/](http://SiteCollectionName/_catalogs/theme/15/)).

Background image

The background image that is used in the site.

Master page

A master page defines the chrome (the shared framing elements) of a site. The theming experience lets users select the master page to apply to a site.

Master page preview

A master page preview is used to render a preview image of the selected theme components. Each master page must have a corresponding master page preview file for the master page to be available in the theming experience.

Master page preview files (.preview files) are stored in the Master Page Gallery of the site ([http:// SiteName/_catalogs/masterpage/](http://SiteName/_catalogs/masterpage/)).

Composed look

A composed look, or design, is the color palette, font scheme, background image, and master page that determine the look and feel of a site. Design and theme can be used interchangeably to describe the overall look of a site.

The theming experience uses the Composed Looks list to determine the available designs. You can create additional designs by creating list items in the Composed Looks list. To access the composed looks list, on the Site Settings page, under Web Designer Galleries, choose Composed looks.

Change the look

The Change the look wizard is the entry point to the theming experience that lets users change the look and feel of their site. To access the Change the look wizard, choose the Settings icon, and then choose Change the look.

Design gallery

The design gallery is the first page in the Change the look wizard. The design gallery shows a thumbnail view of available designs.

Site designs and site scripts

Use site designs and site scripts to automate provisioning new or existing modern SharePoint sites that use your own custom configurations.

When people in your organization create new SharePoint sites, you often need to ensure some level of consistency. For example, you may need proper branding and theming applied to each new site. You may also have detailed site provisioning scripts, such as using the PnP provisioning engine, that need to be applied each time a new site is created.

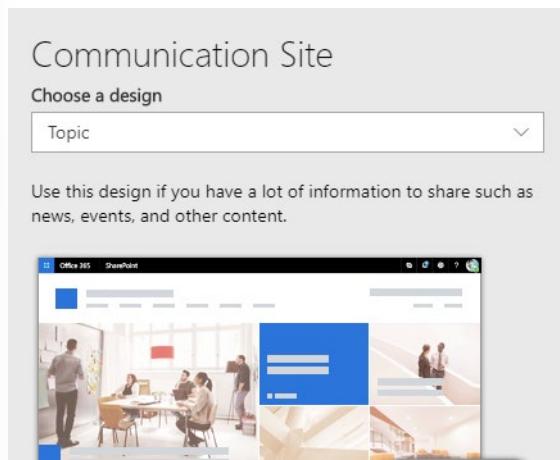
Site designs

Site designs are like templates. They can be used each time a new site is created to apply a consistent set of actions. They can also be applied to existing modern sites (group-connected Team and Communication sites). Most actions typically affect the site itself, such as setting the theme or creating lists. But a site design can also include other actions, such as recording the new site URL to a log or sending a tweet.

You create site designs and register them in SharePoint to one of the modern template sites: the Team site or the Communication site. You can see how this works in the following steps.

1. Go to the SharePoint start page on your developer tenant.
2. Select **Create site**.
3. You'll see the two modern template sites: **Team site** and **Communication site**.
4. Select **Communication site**.
5. The Communication site has a **Choose a design** option, which offers the following default site designs:
 - Topic
 - Showcase
 - Blank

Note:
The Team site template contains only one default site design named **Team site**.
6. For each default site design, there is a title, a description, and an image.



When a site design is selected, SharePoint creates the new site and runs site scripts for the site design. The site scripts detail the work, such as creating new lists or applying a theme. These script actions run in the background. A notification bar is displayed, which the site creator can select to view the status of the actions being applied.

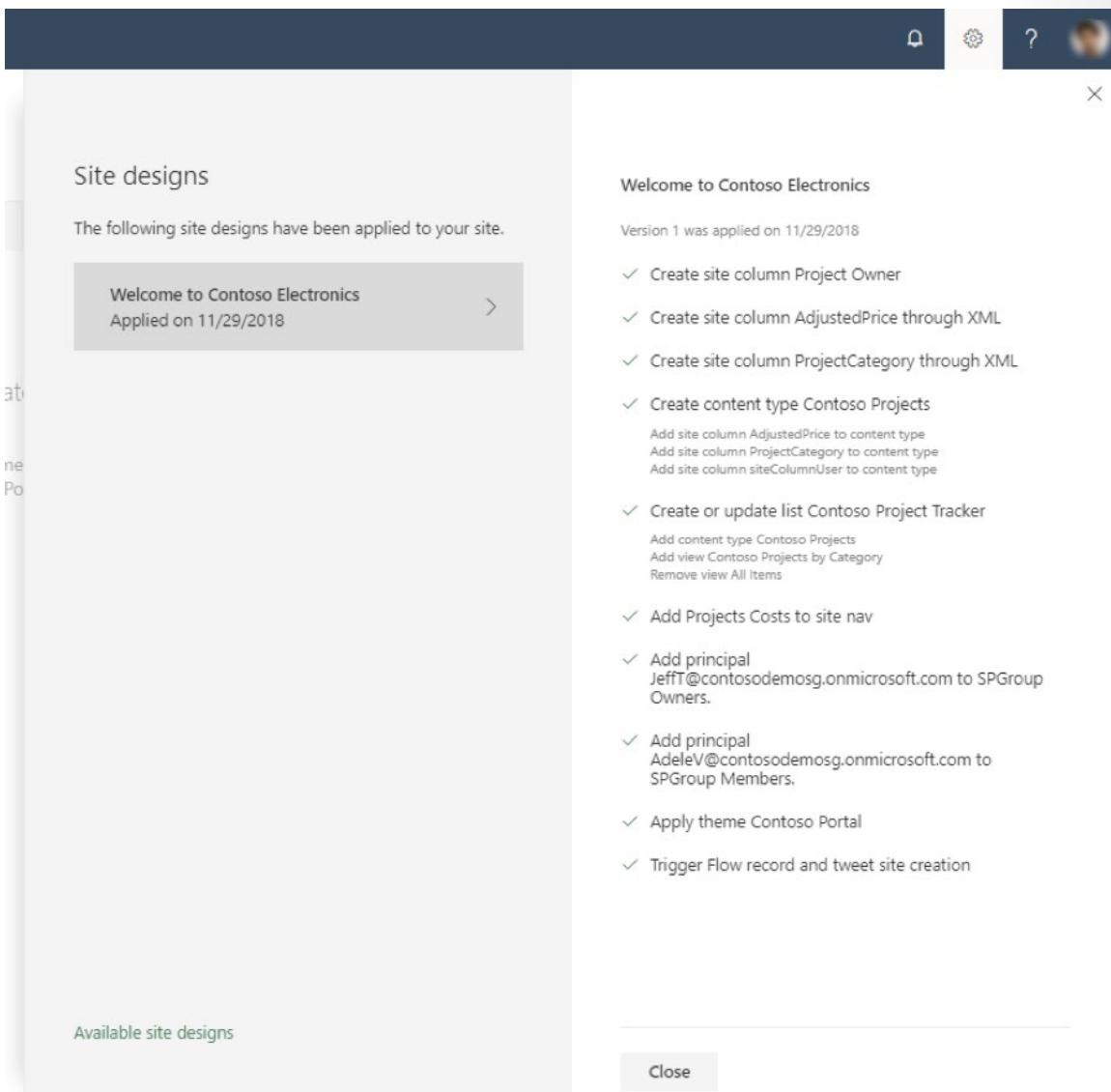
A screenshot of the SharePoint Home page for the "Contoso Strategy 2019" site. The top navigation bar shows the site logo and name, and the SharePoint logo. A yellow notification bar at the top states: "Applying site design. We're updating your site based on the design you chose. View progress". The main content area displays the site's branding with a blue "CS" logo and the title "Contoso Strategy 2019 Classified". Below the title are navigation links: Home, Documents, Pages, Site contents, and Edit. A secondary navigation bar at the bottom includes options: + New, Page details, and a color palette.

When the scripts are complete, the notification bar message changes, allowing the site creator to either refresh the page to see the results of the applied scripts or view the site script details.

A screenshot of the SharePoint Home page for the "Contoso Strategy 2019" site after the site design has been applied. The yellow notification bar now says: "Site design applied. Refresh this site to see the changes." The rest of the page content is identical to the previous screenshot, showing the site's branding and navigation links.

The site design information panel can be invoked by a site owner at any time to see what site designs have been applied to the site (and their script details), as well as to apply new or updated site designs.

When the actions in the scripts are completed, SharePoint displays detailed results in a progress pane.



Site script

Site scripts are JSON files that specify an ordered list of actions to run when creating the new site. The actions are run in the order listed.

The following JSON example is a script that has two top-level actions. First, it applies a theme that was previously created named **Contoso Explorers**. It then creates a **Customer Tracking** list.

```
{
  "$schema": "https://developer.microsoft.com/json-schemas/sp/site-design-script-actions.schema.json",
  "actions": [
    {
      "verb": "applyTheme",
      "themeName": "Contoso Explorers"
    }
  ]
}
```

```
        },
        {
            "verb": "createSPList",
            "listName": "Customer Tracking",
            "templateType": 100,
            "subactions": [
                {
                    "verb": "SetDescription",
                    "description": "List of Customers and Orders"
                },
                {
                    "verb": "addSPField",
                    "fieldType": "Text",
                    "displayName": "Customer Name",
                    "isRequired": false,
                    "addToDefaultView": true
                },
                {
                    "verb": "addSPField",
                    "fieldType": "Number",
                    "displayName": "Requisition Total",
                    "addToDefaultView": true,
                    "isRequired": true
                },
                {
                    "verb": "addSPField",
                    "fieldType": "User",
                    "displayName": "Contact",
                    "addToDefaultView": true,
                    "isRequired": true
                },
                {
                    "verb": "addSPField",
                    "fieldType": "Note",
                    "displayName": "Meeting Notes",
                    "isRequired": false
                }
            ]
        },
        "version": 1
    }
}
```

Each action in a site script is specified by a verb value in the JSON. In the previous script, the first action was specified by the applyTheme verb. Next, the createSPList verb creates the list. Notice that the createSPList verb contains its own set of verbs that run additional actions on only the list.

Available actions include:

- Creating a new list or library (or modifying the default one created with the site)
- Creating site columns, creating content types, and configuring other list settings

- Setting site branding properties such as navigation layout, header layout, and header background
- Applying a theme
- Setting a site logo
- Adding links to quick launch or hub navigation
- Triggering a Microsoft Flow
- Installing a deployed solution from the App Catalog
- Setting regional settings for the site
- Adding principals (users and groups) to SharePoint roles
- Setting external sharing capability for the site

Site scripts can be run again on the same site after provisioning. Site scripts are non-destructive, so when they run again, they ensure that the site matches the configuration in the script.

For example, if the site already has a list with the same name that the site script is creating, the site script will only add missing fields to the existing list.

The limit of site script actions was previously capped at 30. This remains the limit for scripts applied synchronously using the **Invoke-SPOSiteDesign**³⁴ command, but based on customer feedback and support for additional actions, this limit has increased to 300 actions (or 100,000 characters) when the scripts are applied asynchronously (either through the UI or using the **Add-SPOSiteDesignTask**³⁵ command).

There is also a limit of 100 site scripts and 100 site designs per tenant.

Use PowerShell or REST to work with site designs and site scripts

You can create site designs and site scripts by using PowerShell or the REST API. The following example creates a site script and a site design that uses the site script:

```
C:\> Get-Content 'c:\scripts\site-script.json' ` 
    -Raw | ` 
    Add-SPOSiteScript ` 
    -Title "Contoso theme and list" ` 
    Id : 2756067f-d818-4933-a514-2a2b2c50fb06 ` 
    Title : Contoso theme and list ` 
    Description : ` 
    Content : ` 
    Version : 0 ` 
C:\> Add-SPOSiteDesign ` 
    -Title "Contoso customer tracking" ` 
    -WebTemplate "64" ` 
    -SiteScripts "2756067f-d818-4933-a514-2a2b2c50fb06" ` 
    -Description "Creates customer list and applies standard theme"
```

³⁴ <https://docs.microsoft.com/en-us/powershell/module/sharepoint-online/Invoke-SPOSiteDesign?view=sharepoint-ps>

³⁵ <https://docs.microsoft.com/en-us/powershell/module/sharepoint-online/Add-SPOSiteDesignTask?view=sharepoint-ps>

In the previous example, the **Add-SPOSiteScript** cmdlet or **CreateSiteScript** REST API returns a site script id. This is used for the **SiteScripts** parameter in the subsequent call to the **Add-SPOSiteDesign** cmdlet or **CreateSiteDesign** REST API.

The **WebTemplate** parameter set to the value 64 indicates registering this site design with the Team site template. If you have disabled modern Group creation, publish your site designs using **WebTemplate 1** so that they display for the “Group-less” Team site template. The value 68 would indicate registering with the Communication site template. The Title and Description parameters are displayed when a user views site designs as they create a new Team site.

Note:

A site design can run multiple scripts. The script IDs are passed in an array, and they run in the order listed.

Scoping

You can configure site designs to only appear for specific groups or people in your organization. This is useful for ensuring that people only see the site designs intended for them. For example, you might want the accounting department to only see site designs specifically for them. And the accounting site designs may not make sense to show to anyone else.

By default, a site design can be viewed by everyone when it is created. Scopes are applied by using the **Grant-SPOSiteDesignRights** cmdlet or the **GrantSiteDesignRights** REST API. You can specify the scope by user or a mail-enabled security group.

The following PowerShell example shows how to give Nestor (a user at the fictional Contoso site) view rights on a site design:

```
Grant-SPOSiteDesignRights ` 
    -Identity 44252d09-62c4-4913-9eb0-a2a8b8d7f863 ` 
    -Principals "nestorw@contoso.onmicrosoft.com" ` 
    -Rights View
```

You have the option to use branding and theming to customize your SharePoint site.

Lesson review questions

1. What is the purpose of SharePoint themes?
2. What is the purpose of site designs?

Correct/suggested answers:

1. Like the Microsoft brand palette, the SharePoint themes are designed to build on the Microsoft brand, while at the same time allowing for flexibility to enliven partnerships without dominating them. They reveal shared goals and personality, and they reflect diversity and ability to optimize the SharePoint experience.
2. Site designs are like a template. They can be used each time a new site is created to apply a consistent set of actions. They can also be applied to existing modern sites (group-connected Team and Communication sites.)

Module 4 Extend Teams

Overview of Building Apps for Microsoft Teams

Lesson introduction

Microsoft Teams is a collaboration workspace in Office 365 that integrates with apps and services that people use to get work done together. When you build an app on the Microsoft Teams Platform, you extend the Microsoft Teams client (web, mobile, and desktop) with web services that you host. Microsoft Teams apps are bridges between the Teams client and your services and workflows - bringing them directly into the context of your collaboration platform. Teams apps primarily focus on increasing collaboration and improving productivity.

In this lesson, you'll learn about the components of a Teams app, options for extending the Microsoft Teams client, App Studio's features for Teams app development, and options for distributing a Teams app.

After this lesson, you should be able to:

- Describe the components of a Teams app.
- Describe Teams platform UI components, including deep links and task modules
- Identify the components of a Teams app package.
- Describe the purpose of a Teams app manifest.
- Explain App Studio functionality and features.
- Explain the Teams toolkit extension
- Describe options for distributing a Teams app.

Components of a Microsoft Teams app

Apps for Microsoft Teams are bridges between the Microsoft Teams client and your services and workflows - bringing them directly into the context of collaboration. Apps for Microsoft Teams can be as simple or as complex as needed, from sending notifications to channels or users to complex multi-surface apps incorporating conversational bots, natural language processing, and embedded web experiences.

You can build apps for an individual, your team, your organization, or for all Microsoft Teams users everywhere.

What can Microsoft Teams apps do?

Apps built on the Teams platform primarily focus on increasing collaboration and improving productivity. The best apps for Microsoft Teams focus on helping people express themselves and work better together. Here are a few scenarios that are well-suited for Teams apps:

- **Collaborate on items in external systems:** One of the core scenarios for a custom Teams app is to bring information or items into Teams from some other place and have a conversation around it. You can push information into Teams, enable your users to search for and pull it on demand, or make it available in an embedded web view.
- **Trigger workflows from conversations:** Often conversations result in the need to kick off some workflow or complete some action: take a note, review a pull request, convert to a sales lead, etc. Your Teams app can put access to that workflow right inside of Teams.
- **Notify your team of important events:** Sick of email notifications? Send notifications to Teams instead. Send notifications directly to users, to a channel, to the activity feed, or to anyone who subscribes to them.
- **Embed functionality from other sites/services:** Sometimes you just need to make your app easier to discover. Embed your existing single page app or build something from scratch for Teams.

In each of these scenarios, Teams apps allow users to collaborate and be productive without unnecessary context-switching.

Components of a Teams app

Teams apps are a combination of capabilities and entry point. A Microsoft Teams app consists of three primary components:

- **The Microsoft Teams client (web, desktop or mobile)** provides the extension points and UI elements your users will interact with.
- **Your Teams App Package** creates the app installed by your users. It contains metadata about your app (name, icons, etc.) and pointers to the web services you host that power your app.
- **Your web services**, hosted by you, provide the APIs and logic that power your app. Changes to your web service do not require you to re-install your app in the Teams client.

Teams apps capabilities

Capabilities are the extension points for building apps on the Microsoft Teams platform.

There are multiple ways to extend Teams, so every app is unique: Some only have one capability (such as a webhook), while others have a few to give users options. For instance, your app could display data in a central location (tab) and present that same information through a conversational interface (bot).

Your Teams app can have one or all of the following core capabilities:

- **Tabs**
- **Messaging extensions**
- **Bots**
- **Webhooks and connectors**

Teams apps entry points

The Teams platform provides a flexible set of entry points where people can discover and use your app. Your app can be as simple as embedding an existing website in a personal tab or a multi-faceted app that users interact with across several entry points.

The most successful apps feel native to Teams, so it's important to carefully plan your app's entry points.

Teams, channels, and group chats

Here's how Teams app capabilities are commonly used in collaborative contexts:

- **Tabs** provide a full-screen embedded web experience configured for the team, channel, or group chat. All members interact with the same web-based content, so a stateless single page app experience is typical.
- **Messaging extensions** are shortcuts for inserting external content into a conversation or taking action on messages without leaving Teams. Link unfurling provides rich content when sharing content from a common URL.
- **Bots** interact with members of the conversation through chat and responding to events (like adding a new member or renaming a channel). Conversations with a bot in these contexts are visible to all members of the team, channel, or group, so bot conversations should be relevant to everyone.
- **Webhooks and connectors** allow an external service to post messages into a conversation and users to send messages to a service.
- **Microsoft Graph REST API** for getting data about teams, channels, and group chats to help automate and manage Teams processes.

Personal apps

Personal apps focus on interactions with a single user. The experience in this context is unique to each user. Users can pin personal apps to the left navigation rail for quick access.

Here's how Teams capabilities are commonly used in personal contexts:

- **Bots** have one-on-one conversations with a user. Bots that require multi-turn conversations or provide notifications relevant only to a specific user are best suited in personal contexts.
- **Tabs** provide a full-screen embedded web experience that's meaningful to individual users.

UI components

Apps typically exhibit one or more standard Teams UI components. Building out your app using these components leads to rich experiences that feel native to Teams users.

Cards

Cards are UI containers defined by JSON that can contain formatted text, media, controls (like dropdowns and radio buttons) and buttons that trigger an action.

Card actions can send payloads to your app's API, open a link, initiate authentication flows, or send messages to conversations. The Teams platform supports multiple cards, including Adaptive Cards, hero cards, thumbnail cards, and more. You can combine card collections and display in a list or carousel.

Task modules

Task modules provide modal experiences in Teams. They are especially useful for initiating workflows, collecting user input, or displaying rich information such as videos or Power BI dashboards. In task modules, you can run custom front-end code, display an <iframe> widget, or show an Adaptive Card.

When considering how you want to build your app, remember that modals are natural for users to enter information or complete tasks compared to a tab or a conversation-based bot experience.

Deep links

Your app can create URL deep links to help navigate your user through your app, and the Teams client. You can create a deep link for most entities within Teams, and some (like a new meeting request) allow you to pre-populate information using query strings in the URL.

For example, your conversational bot could send a message to a channel with a deep link to a task module that results in a card being sent as a one-to-one message to a user, that in turn contains a deep link to create a new meeting with a specific user at a certain date/time. Use deep links to connect across the various extension points available to your app, keeping your user in the correct context at all times.

Web-based content

Web-based content is a webpage you host that can be embedded in a tab or task module.

Note

Keep in mind that the Microsoft Teams Platform is not a hosting service; the web services powering your app must be hosted by you and accessible by HTTPS over the internet.

It is also important to keep in mind that any functionality you expose in a Microsoft Teams app is publicly available over the internet unless you take additional steps to secure it. If you are providing access to confidential or protected information you'll want make sure your services are at a minimum authenticating the endpoint connecting to your app, or **authenticating your users¹**.

Microsoft Teams platform elements

The Microsoft Teams Platform provides flexible UI/UX features for apps to take advantage of. These elements allow you to create rich experiences that feel native to the Teams client.

Cards and card actions

A card is a UI container for short and related pieces of information. They contain actionable snippets of content that you can add to a conversation through a bot, a connector, or app. Cards can contain combinations of text, graphics, and buttons to allow you to communicate with your audience.

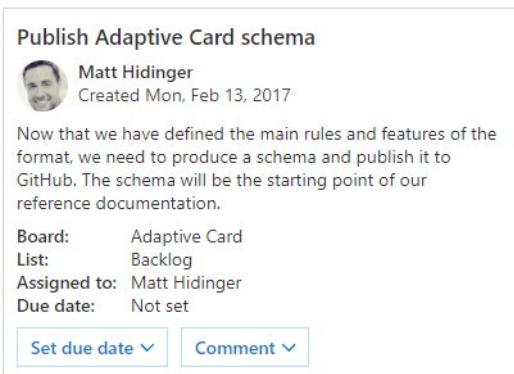
Cards can include buttons that can trigger card actions. Card actions can send payloads to your app's API, open links, initiate authentication flows, or send messages to conversations.

The Microsoft Teams platform supports multiple types of cards including Adaptive Cards, Hero Cards, Thumbnail Cards and more. **Adaptive cards²** are the recommended card type for new Teams development.

¹ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/authentication/authentication>

² <https://docs.microsoft.com/en-us/microsoftteams/platform/task-modules-and-cards/cards/cards-reference>

Here's an example of an Adaptive Card:



The screenshot shows a GitHub commit titled "Publish Adaptive Card schema". It includes a user profile picture of Matt Hidinger, the date "Created Mon, Feb 13, 2017", and a description: "Now that we have defined the main rules and features of the format, we need to produce a schema and publish it to GitHub. The schema will be the starting point of our reference documentation." Below the description are four status fields: "Board: Adaptive Card", "List: Backlog", "Assigned to: Matt Hidinger", and "Due date: Not set". At the bottom are two buttons: "Set due date" and "Comment".

Deep links

You can create links to information and features in the Teams client that help navigate your user through your app and Teams. You can create a deep link for most entities within Teams, and some (like a new meeting request) allow you to pre-populate information using query strings in the URL. Deep links may be useful when:

- Navigating the user to content within one of your app's tabs. For instance, your app may have a bot that sends messages notifying the user of an important activity. When the user taps the notification, the deep link navigates to the tab so the user can view more details about the activity.
- Your app automates or simplifies certain user tasks, such as creating a chat or scheduling a meeting, by pre-populating the deep links with required parameters. This avoids the need for users to manually enter information.
- Your tab needs to link to other content in Teams. This could include linking to a channel, message, another tab or even to open a scheduling dialog.

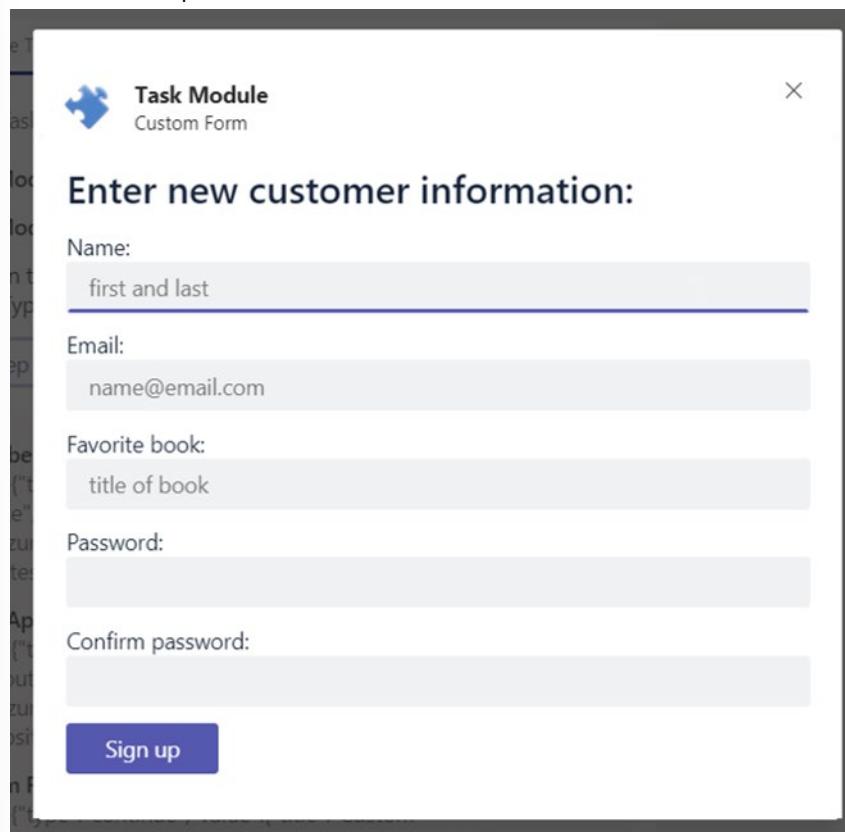
Task modules

Task modules³ allow you to create modal popup experiences in your Teams application. Inside the popup you can run your own custom HTML/JavaScript code, show an <iframe> widget such as a YouTube or Microsoft Stream video, or display an Adaptive card.

Task modules are especially useful for initiating and completing tasks or displaying rich information like videos or Power BI dashboards. A popup experience is often more natural for users initiating and completing tasks compared to a tab or a conversation-based bot experience.

³ <https://docs.microsoft.com/en-us/microsoftteams/platform/task-modules-and-cards/what-are-task-modules>

Here's an example of a task module that collects information from a user:



Task modules can be invoked in three ways:

- **Channel or personal tabs.** Using the Microsoft Teams Tabs SDK you can invoke task modules from buttons, links or menus on your tab.
- **Bots.** Buttons on cards sent from your bot. This is particularly useful when you don't need everyone in a channel to be aware of an interaction between a user and a bot. For example, when having users respond to a poll in a channel it's not terribly useful to see a record of that poll being created. A task module could allow an individual to interact with the bot and set up the poll.
- **Deep link URLs.** You can create URLs to invoke a task module from anywhere.

The `TaskInfo` object contains the metadata for a task module. You must define either the `url` property (to display an embedded iFrame) or `card` (to display an Adaptive Card).

Embedded content pages

A content page is a webpage that is rendered within the Teams client. Content pages are typically part of personal or channel/group custom tabs or embedded as webviews inside task modules.

Extensible points in the Teams client

There are multiple places where the Microsoft Teams client can be extended to allow users to interact with your app. Depending on your scenario you may choose to focus on a single extension point (like a personal conversational bot) or combine multiple extension points. The following are options for extending the Teams client.

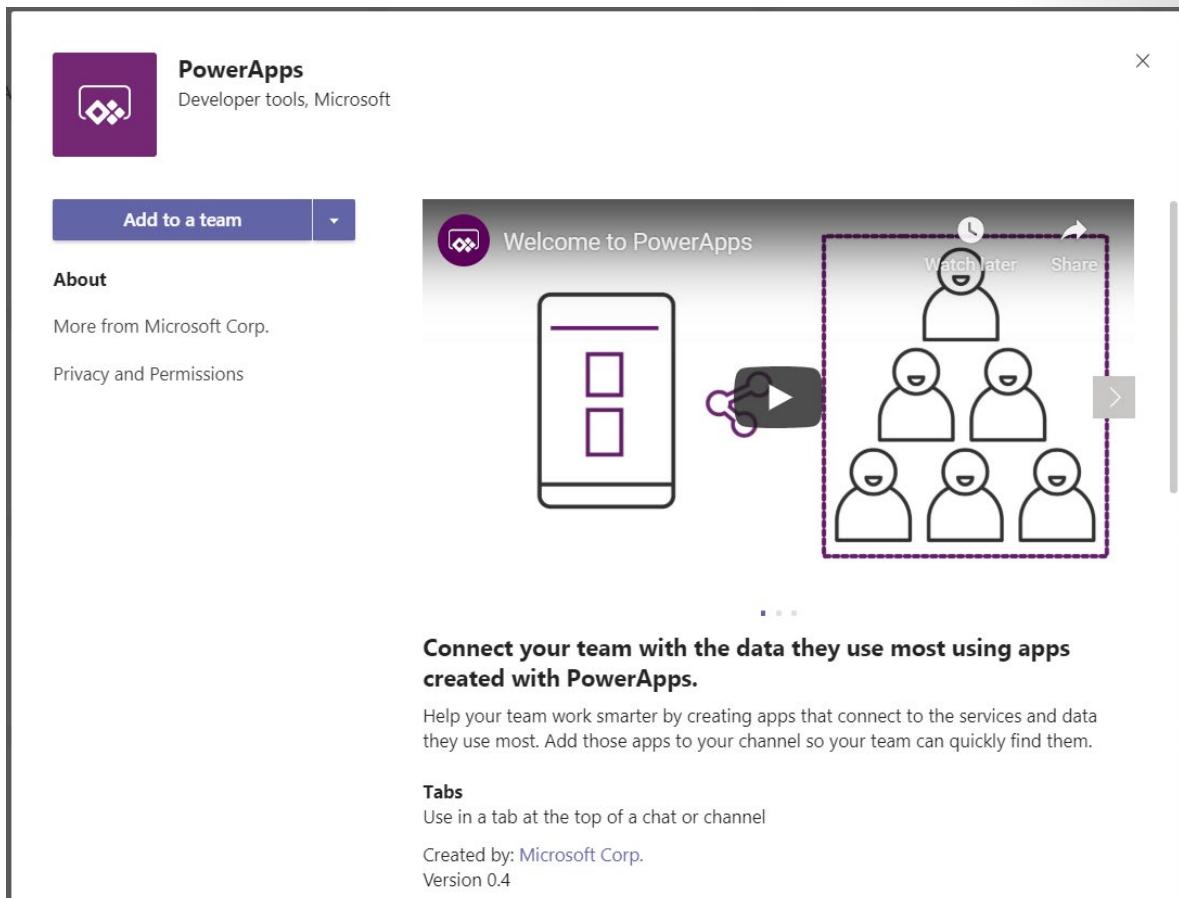
Teams, channels and group chats

Teams, channels and group chats allow multiple people to collaborate in Microsoft Teams. Apps in this context make themselves available to all members of the group or conversation, typically focusing on enabling additional collaborative workflows or unlocking new social interactions. Your app will have access to APIs allowing it to get information about the members in the conversation, the channels in a team, and metadata about the team or conversation.

Teams, channels and group chats can be extended with:

- **Conversational bots** that interact with members of the conversation through chat and by responding to events (like a new member being added, or a channel being renamed). All conversations with a bot in this context are visible to all members of the channel or group, so you'll need to ensure the conversation is relevant to everyone.
- **Configurable Tabs** that provide a full-screen embedded web experience configured for the channel or group chat they are installed in. All members will interact on the same shared web-app, so a stateless single page app experience is typical.

For example, the PowerApps app for Microsoft Teams provides a tab experience for group chats or channels:



- **Webhooks and Connectors** that enable external services to post messages to the conversation. Cards and card actions allow for the use of rich, actionable messages. Webhooks provide a simple, unauthenticated, one-way method to post messages to a channel, while Connectors provide a slightly more robust back-and-forth experience.

Personal apps

Personal apps are the portion of your Teams app focusing on interactions with a single user. The experience is unique to each individual user. This portion of your app can be pinned to the left-navigation rail - enabling one-click access for your users.

Personal apps can contain:

- **Conversational bots** that can have a private one-to-one conversation with the user. If your app needs to have a multi-turn conversation with a user, or provide a notification relevant only to a single user, it is typically best to have that interaction in a personal app.
- **Personal Tabs** that provide a full-screen embedded web experience.

The OneNote app for Microsoft Teams provides both a tab experience for group chats and channels, as well as a personal app experience that allows users to keep track of their personal notes.

The screenshot shows the Microsoft OneNote app integrated into a Microsoft Teams window. On the left, there's a sidebar with a purple icon containing a white 'N', the title 'OneNote', and a subtitle 'Content management, Files + documents, Productivity'. Below this is a dropdown menu with 'Open' at the top, followed by 'Add to a team' and 'Add to a chat'. A 'Privacy and Permissions' link is also visible. To the right, a main content area displays a promotional image for OneNote with the text 'Note-taking made easier for everyone...' and 'Organize in one place,'. There's a play button icon over the image. At the bottom, there's a section titled 'Use OneNote notebooks to collaborate on digital content with your team.' followed by a description and links for 'Tabs', 'Personal app', and 'Created by: Microsoft Corp.'

OneNote
Content management, Files + documents, Productivity

Open

Add to a team
Add to a chat

Privacy and Permissions

Note-taking made easier for everyone... Watch later Share

Organize in one place,

Use OneNote notebooks to collaborate on digital content with your team.

Use OneNote notebooks to collaborate on digital content and share it with your team.

Tabs
Use in a tab at the top of a chat or channel

Personal app
Keep track of important content and info

Created by: Microsoft Corp.

Messages

Messages are the heart of collaboration in Teams. **Messaging extensions** allow users to interact with your web service through buttons and forms from the compose message area, the command box, or directly from a message. Your app can respond by presenting a form to the user to collect more information, sending a reply to the original message, or sending a message directly to the user. Your app can also help users craft more effective messages by enabling them to search, or take action, in an external system

and insert the results in a rich, structured format complete with actionable buttons. We'll review messaging extensions further in Lesson 04.

App Studio for Microsoft Teams app development

App Studio is a tool that is used to create app packages for Microsoft Teams apps, design and preview bot cards, and find documentation. App Studio streamlines the process of creating manifests and packages. It also provides tools such as the Card editor and a React control library.

App Studio is a Teams app which can be found in the Teams store. It is also available for **direct download**⁴.

Your app package

Your Microsoft Teams app package creates the application that will be installed by your users. The app package is a .zip file containing the following:

- A **manifest file** named `manifest.json`, which specifies attributes of your app and points to required resources for your experience, such the location of its tab configuration page or the Microsoft app ID for its bot. The manifest file is the most significant part of a Microsoft Teams app package.
- A transparent "outline" icon and a full "color" icon. Visit [Icons⁵](#) for more information.

Note: Any time you make a change to your app manifest, you'll need to re-upload your app package, and update your app in Teams by re-installing it.

App Studio features

App Studio provides tools that make it easier to create and modify your app package, test and preview UI elements, and more.

Conversation

Use the **Conversation** tab to test your bot cards. This tab shows you how your cards look when they run in Teams. Test a tab by selecting the *Send me this card* button, located on the Card editor tab. You can also search for App Studio documentation in this tab.

Manifest editor

The **Manifest editor** tab in App Studio simplifies creating the manifest. You describe the app, upload your icons, add app capabilities, and produce a .zip file that you then upload to Teams for testing or distribution.

Note that manifests have to conform to the **Teams App schema⁶**.

Note:

App Studio does not produce functional code for your app or host your app. Your app must already be hosted and running at the URL listed in the manifest for the app upload process to result in a working Teams app.

⁴ <https://aka.ms/InstallTeamsAppStudio>

⁵ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/apps/apps-package>

⁶ <https://docs.microsoft.com/en-us/microsoftteams/platform/resources/schema/manifest-schema>

Card editor

Use the **Card editor** tab to make the card creation process easier and less error-prone. You can build Hero cards or Thumbnail cards by entering data in a form, and you can verify and test the resulting card by using a bot.

The Card editor also provides the corresponding JSON, C#, or Node.js code for the card that you can add to your app's source code.

To verify an existing card in Teams, you can paste the JSON for that card into the **JSON** tab under **Add card info** and send it to yourself to verify the appearance of the card in a chat.

React Control Library

App Studio provides several categories of UI controls that follow the Teams design principles. Use the React control library to create an app that fits the Teams client experience.

This React control library will be deprecated in the future. Consider using the **Fluent-UI react controls as an alternative**⁷ (formerly Stardust UI).

Microsoft Teams Toolkit Extension

The Microsoft Teams Toolkit enables you to create custom Teams apps directly within the Visual Studio Code environment or the Visual Studio integrated development environment (IDE). The toolkit guides you through the process and provides everything you need to build, debug, and launch your Teams app.

Installing the Teams Toolkit

The toolkit extension is available to install on:

- Visual Studio Code
- Visual Studio integrated development environment (IDE)

Option 1: Visual Studio Code

The Microsoft Teams Toolkit for Visual Studio Code is available for download from the **Visual Studio Marketplace**⁸ or directly as an extension within Visual Studio Code.

Note: After installation, you should see the Teams Toolkit in the Visual Studio Code activity bar. If not, right-click within the activity bar and select **Microsoft Teams** to pin the toolkit for easy access.

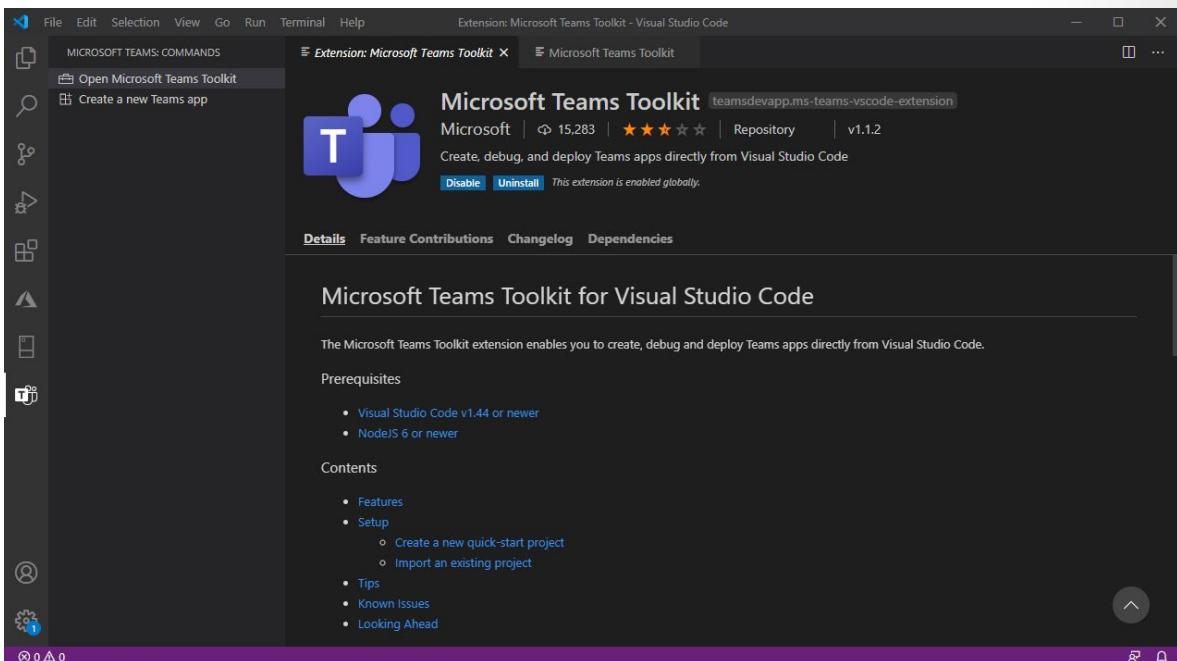
The teams toolkit in Visual Studio Code allows you to:

- Set up a new project
- Import an existing project
- Configure your app
- Package your app
- Run your app locally or in Team

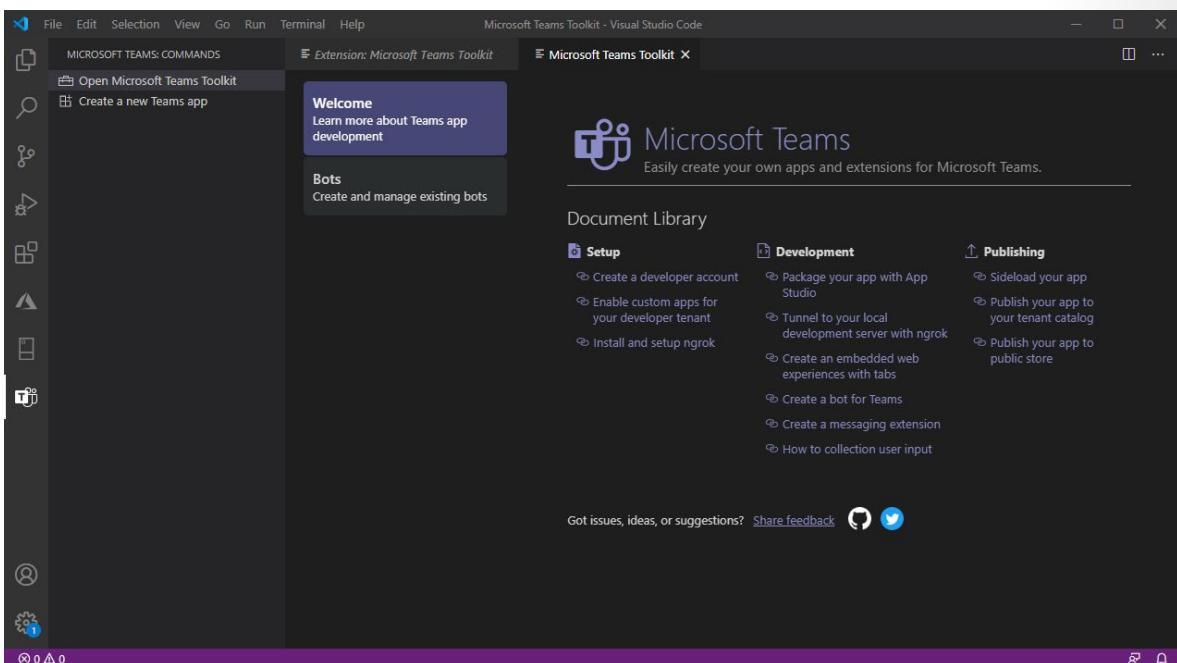
⁷ <https://microsoft.github.io/fluent-ui-react/>

⁸ <https://aka.ms/teams-toolkit>

Microsoft Teams Toolkit extension page



Microsoft Teams Toolkit welcome page



Option 2: Visual Studio integrated development environment (IDE)

The Microsoft Teams Toolkit for Visual Studio is available for download from the **Visual Studio Marketplace**⁹ or directly from the **Extensions** menu within Visual Studio.

The teams toolkit in Visual Studio allows you to:

- Set up a new project
- Configure your app
- Package your app
- Run your app in Teams
- Validate your app
- Publish your app

Installing the toolkit

Prerequisites

1. **Enable developer preview**¹⁰
2. Make sure the **ASP.NET and web development** module has been added to your Visual Studio instance. You can check by following the steps in the **Modify Visual Studio by adding or removing workloads and component**¹¹ documentation.



3. If you would like test your app by deploy it from Visual Studio, you'll need to have IIS (Internet Information Services) installed in your development environment. Visual Studio does not include IIS and it isn't included in the default Windows 10, Windows 8, or Windows 7 configuration; however, you can download the latest version from the Microsoft download center.

Internet Information Services (IIS) 10.0 Express

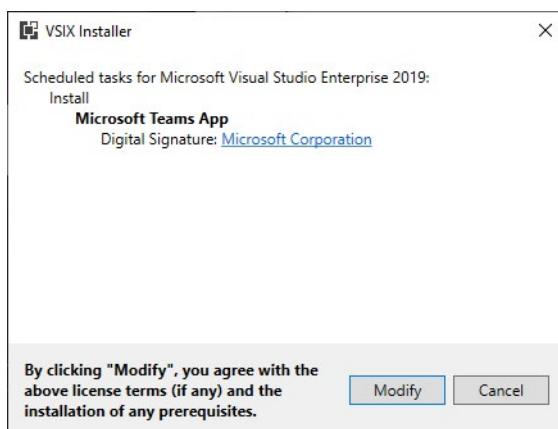
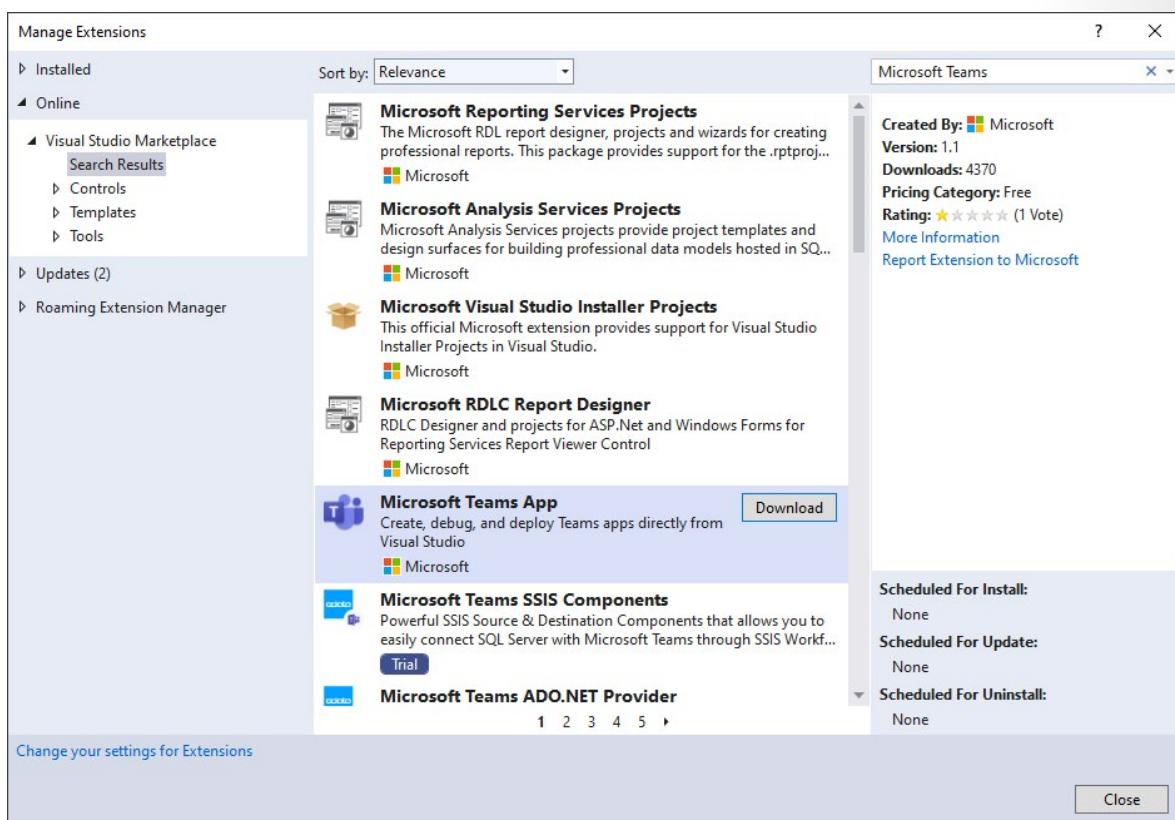


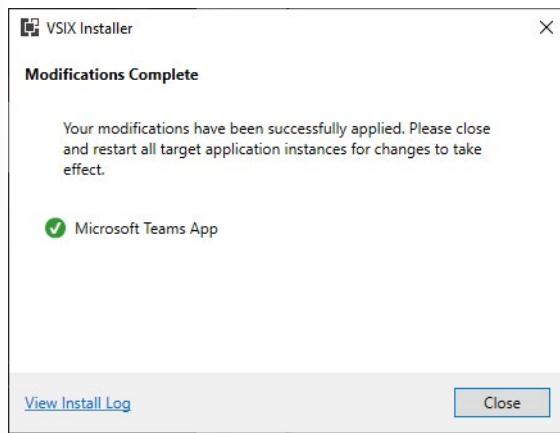
⁹ <https://marketplace.visualstudio.com/items?itemName=TeamsDevApp.vsteamstemplate>

¹⁰ <https://docs.microsoft.com/en-us/microsoftteams/platform/resources/dev-preview/developer-preview-intro#enable-developer-preview>

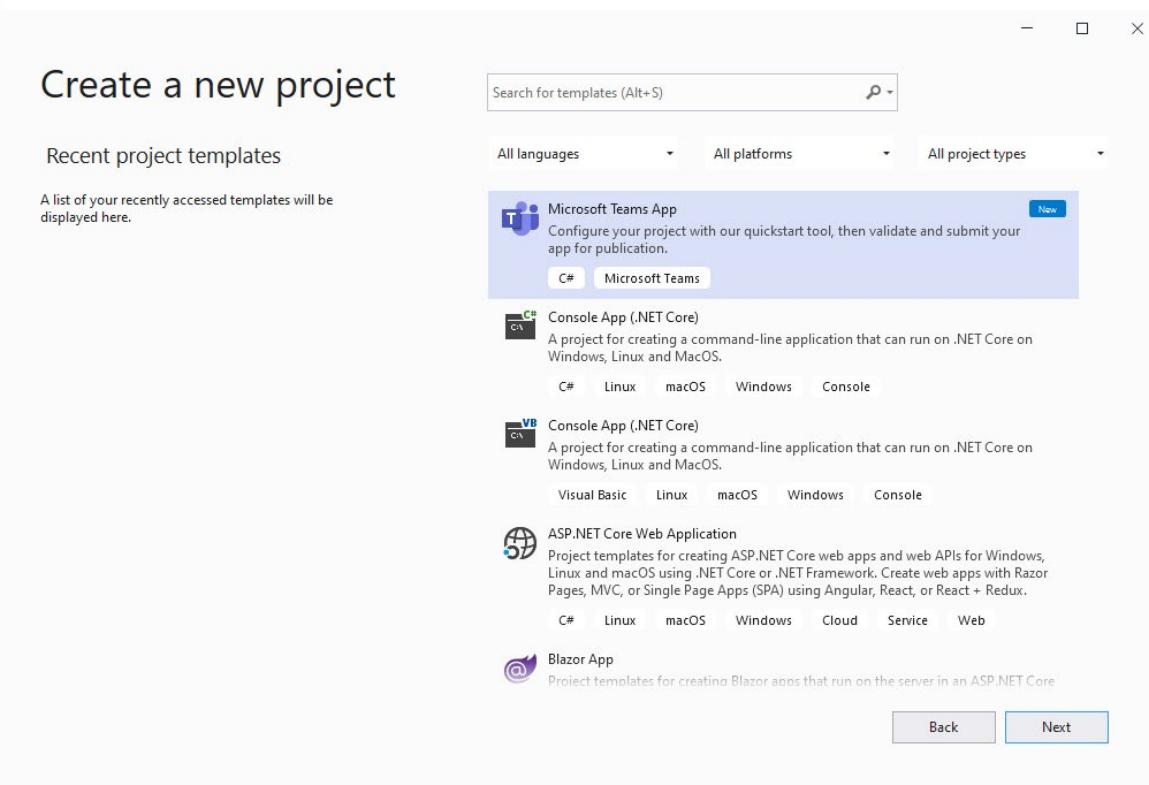
¹¹ <https://docs.microsoft.com/en-us/visualstudio/install/modify-visual-studio?view=vs-2019&preserve-view=true>

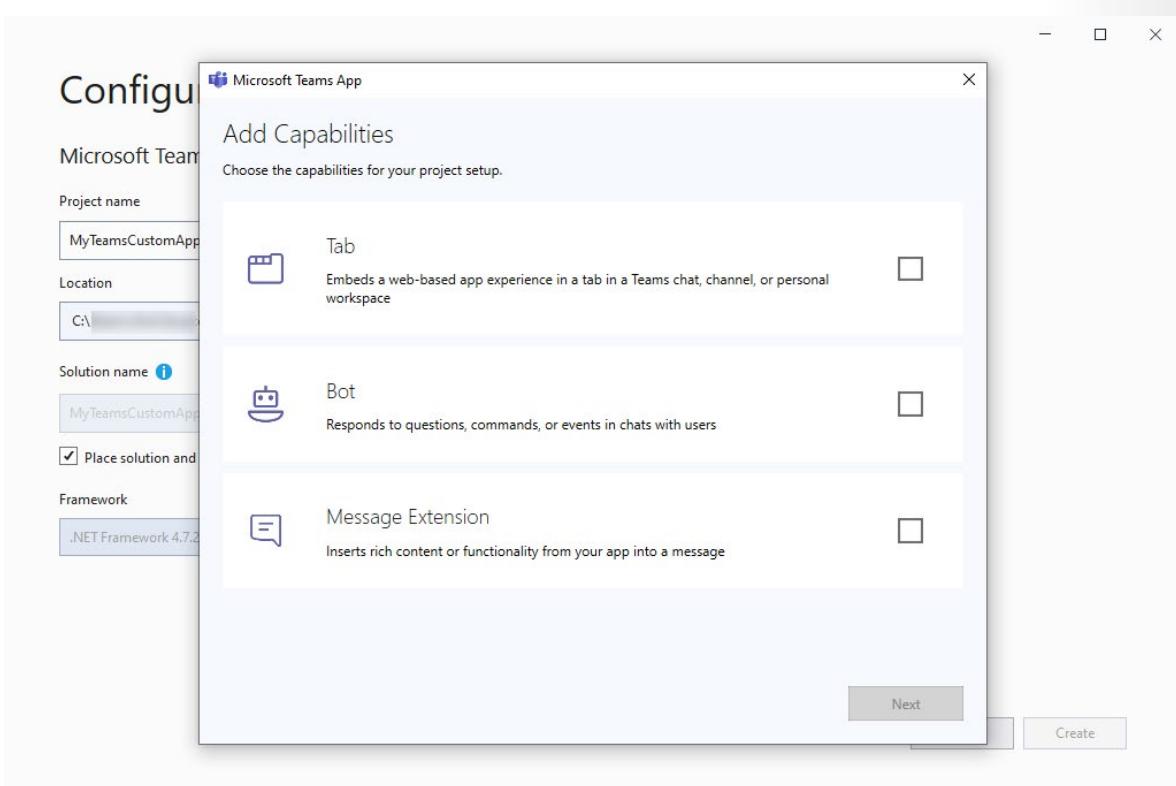
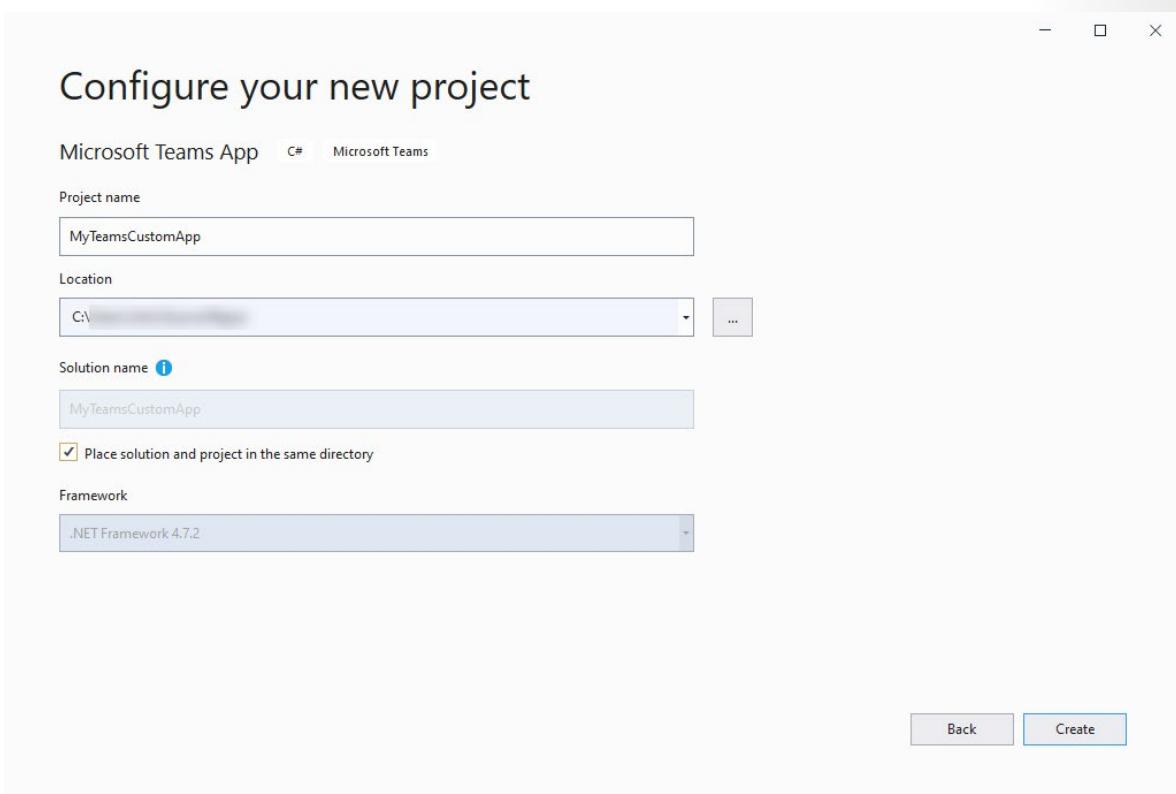
Microsoft Teams App extension installation for Visual Studio





Creating new Microsoft Teams app project in Visual Studio





Lesson review questions

1. What options do you have for distributing your Teams app?
2. What is the purpose of a task module in a tab?
3. What is a Teams app manifest?

Correct/suggested answers:

1. You can share your app package directly, publish your app to your organizational app catalog, or publish your app to the public App Store.
2. A task module simplifies workflows that require data entry. Task modules allow your app to gather user input in a Teams-aware pop-up window.
3. A Teams app manifest is a text file that contains the metadata required to present your app to users. Manifests define an app's user experience.

Options for distributing a Microsoft Teams app

You have three options for distributing a Microsoft Teams app, depending on your target audience.

- **Share your app package directly.** You can choose to share your app package directly with users. This is particularly useful if your app is directed towards a limited audience (just a couple of teams or individuals), as well as during development and testing of your app.
- **Publish your app to your organizational app catalog.** If your app is applicable to a specific organization (or if you've customized your app to meet an organization's specific needs), a tenant administrator can upload your app to the organization's app catalog. This makes your app available for anyone in the organization to install (but does not automatically install it).
- **Publish your app to the public App Store.** If your app is intended for all Teams users everywhere, you can submit your app for publication in the public app store. You'll need to go through a rigorous review process, so make sure you've dotted your i's and crossed your t's.

When distributing your app you need to take into consideration not just your desired audience, but the IT policies in place in the organization you want to share your app with. Each organization has complete control over determining which apps will be uploaded to their organizational app catalog, and which apps are available to install from the app store.

Publish to an organizational app catalog

You can use the Microsoft Teams Tenant Apps Catalog to test and distribute line-of-business applications to your organization.

The Teams Tenant Apps Catalog lets you distribute line-of-business applications that were built specifically for your organization and that you rely on to complete critical business functions.

Once you have the app package, you can add it to the enterprise app catalog. While all users in the tenant can view the app catalog, only global admins and Teams service admins can publish and manage it.

Users in your organization can view apps in the catalog and install them for teams of which they are a member.

Publish to AppSource

To include your solution in AppSource—formerly known as Office Store—you submit it to the Seller Dashboard. You need to create an individual or company account if you have not already done so for other Windows apps or Office extensibility types.

Note:

Registering as a developer can be a time-consuming process and is not covered in this lesson.

The process to submit an app to AppSource, at a high level, is as follows:

- **Register as a Microsoft app developer¹².**
- **Register as a developer in the Seller Dashboard¹³.**
- Validate your app by following the **app guidance¹⁴**.
- Submit your app package, test notes, and app metadata to teamsubm@microsoft.com.
- Once the Teams validation team approves you, use the Seller Dashboard to submit your Teams app package to AppSource.
- Monitor the Seller Dashboard to track **approval¹⁵**.

The app you create vs. the app your users install

Your app may take advantage of multiple extensibility points in the Teams client, and work in a variety of scopes. The app package you distribute to users will define all of these as a single entity. However, because all app installations in Microsoft Teams are context-specific, the entirety of your app may not always be installed for all users.

For example, imagine your app contains a conversational bot that works in both a personal and team conversations, as well as both a personal tab and a channel tab. When your app is installed, it will be installed in a specific context - if a user installs the app in a team, they have not necessarily installed the personal portion of your app. This can be a bit confusing at first, just remember to never expect that all portions of your app will be installed and configured in any given context.

¹² <https://developer.microsoft.com/store/register>

¹³ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/apps-publish>

¹⁴ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/office-store-approval>

¹⁵ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/deploy-and-publish/office-store-approval>

Webhooks in Microsoft Teams

Lesson introduction

Webhooks allow you to connect your web services to the channels and teams inside Microsoft Teams. Outgoing webhooks allow users to send text messages to your web service. Incoming webhooks are a type of connector that allows users to subscribe to a feed.

In this lesson, you'll learn about outgoing and incoming webhooks in Microsoft Teams, when to use them, their limitations and the differences between them.

After this lesson, you should be able to:

- Describe when to use webhooks.
- Explain the limitations of webhooks.
- Describe the differences between incoming webhooks and outgoing webhooks.

Webhooks and Connectors in Microsoft Teams

Outgoing and incoming webhooks both allow you to connect your web services to channels and teams inside Microsoft Teams.

Outgoing webhooks

Outgoing webhooks allow your users to send text messages from a channel to your web services without having to use the Microsoft Bot Framework.

When you add an outgoing webhook to a team, it acts like bot, listening in channels for messages that use **@mention**, sending notifications to external web services, and responding with rich messages that can include cards and images.

You can use outgoing webhooks for custom workflows and commands, such as kicking off a build or checking the latest set of site issues. They are best suited for completing team-specific workloads that don't require large amounts of information to be collected or exchanged.

Incoming webhooks

Connectors allow users to subscribe to receive notifications and messages from your web services.

Incoming webhooks work as a type of connector. They allow external apps to share content in team channels, and you can use them as tracking and notification tools.

For example, you can create an incoming webhook in your DevOps channel and configure your build, deployment and monitoring services to send alerts.

Incoming webhooks use an HTTPS POST operation to send messages in correctly formatted JSON, typically as a card. Because you enable and disable them using a Teams UI, incoming webhooks give you a quick and easy way to connect a channel to your service.

Outgoing webhooks in Microsoft Teams

Outgoing webhooks post data from Microsoft Teams to any chosen service capable of accepting a JSON payload.

Once an outgoing webhook is added to a team, it acts like bot, listening in channels for messages that use **@mention** to mention the outgoing webhook by name. Once a user triggers the outgoing webhook using **@mention**, the outgoing webhook sends the message to your service, and the service has five seconds to send a response to the message. The response can include text, cards, or images.

You configure outgoing webhooks on a per-team basis.

NOTE: A Teams app used in multiple Teams would require configuring a unique outgoing webhook for each Team.

Outgoing webhooks help limit access to authorized users. The security token used by an outgoing webhook is only scoped to the team to which it was added.

Key features

Feature	Description
Scoped Configuration	Webhooks are scoped at the team level. You'll need to go through the setup process for each team you want to add your outgoing webhook to.
Reactive Messaging	Users must use @mention for the webhook to receive messages. Currently users can only message an outgoing webhook in public channels and not within the personal or private scope
Standard HTTP message exchange	Responses will appear in the same chain as the original request message and can include any Bot Framework message content (rich text, images, cards, and emojis). Note: Although outgoing webhooks can use cards, they cannot use any card actions except for openURL .
Teams API method support	In Teams, outgoing webhooks send an HTTP POST to a web service and process a response back. They cannot access any other APIs, such as to retrieve the roster or list of channels in a team.

Limitations

- Outgoing webhooks do not have access to non-messaging APIs, such as team roster membership.
- Outgoing webhooks cannot post into channels **proactively**¹⁶.
- Although outgoing webhooks can use cards, they cannot use button actions such as `imBack` or `invoke`.

Sample code

The following outgoing webhook code samples can be found on GitHub:

- Node.js:** [OfficeDev/msteams-samples-outgoing-webhook-nodejs](https://github.com/OfficeDev/msteams-samples-outgoing-webhook-nodejs)¹⁷
- C#:** [OfficeDev/microsoft-teams-sample-outgoing-webhook](https://github.com/OfficeDev/microsoft-teams-sample-outgoing-webhook)¹⁸

¹⁶ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/bots/bot-conversations/bots-conv-proactive>

¹⁷ <https://github.com/OfficeDev/msteams-samples-outgoing-webhook-nodejs>

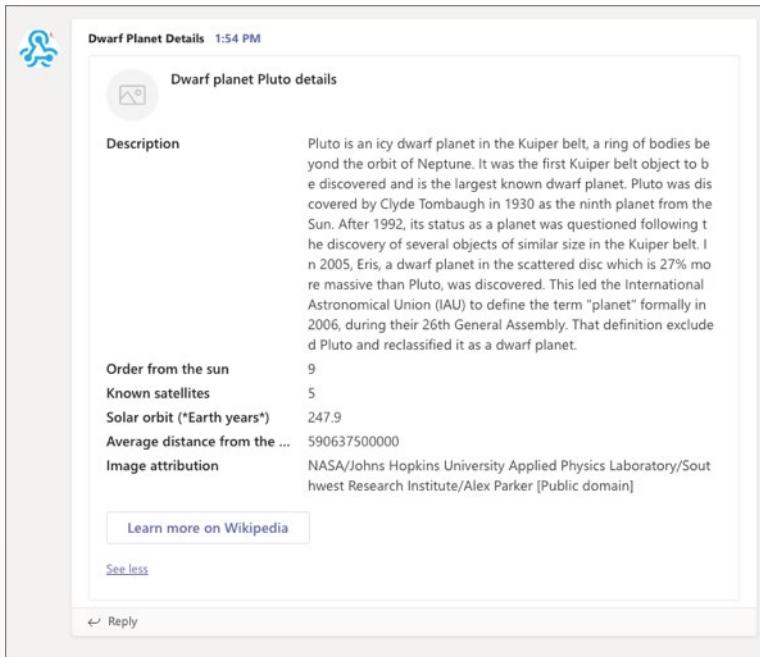
¹⁸ <https://github.com/OfficeDev/microsoft-teams-sample-outgoing-webhook>

Incoming webhooks in Microsoft Teams

Incoming webhooks function as connectors, providing a simple way for an external service to share content in team channels. They are useful for tracking and notification.

Microsoft Teams provides a unique URL to which you send a JSON payload with the message that you want to POST, typically in a card format.

Here is an example of a scenario that allows users to use an incoming webhook to request information about planets within Microsoft Teams. A card is displayed after users



Note: If you include a card in a message sent to an incoming webhook, it must be an Office 365 Connector card; adaptive cards aren't supported when sending messages to incoming webhooks.

Key features

Feature	Description
Scoped configuration	Incoming webhooks are scoped and configured at the channel level. In contrast, outgoing webhooks are scoped and configured at the team level.
Secure resource definitions	Messages are formatted as JSON payloads, which prevents the injection of malicious code, because there is no code execution on the client.
Actionable messaging support	If you send messages using cards, you must use the actionable message card format. Actionable message cards are supported in all Office 365 groups, including Teams.
Independent HTTPS messaging support	Cards help you present information in a clear and consistent way. Any tool or framework that can send HTTPS POST requests can send messages to Teams by using an incoming webhook.

Feature	Description
Markdown support	All text fields in actionable messaging cards support basic Markdown. Don't use HTML markup in your cards; HTML is ignored and treated as plain text.

Lesson review questions

1. Should you configure outgoing webhooks on a per-team or a per-channel basis? Why?
2. Describe 2 limitations of outgoing webhooks.
3. True or false: Incoming webhooks are scoped and configured at the channel level.

Correct/suggested answers:

1. Per team. They are best suited for completing team-specific workloads that don't involve collecting or exchanging large amounts of information. The security token used by an outgoing webhook is only scoped to the team to which it was added.
2. Limitations:
 - Outgoing webhooks do not have access to non-messaging APIs, such as team roster membership.
 - Outgoing webhooks cannot post into channels proactively.
 - Although outgoing webhooks can use cards, they cannot use button actions such as imBack or invoke.
3. True. Incoming webhooks are scoped and configured at the channel level.

Tabs in Microsoft Teams

Lesson introduction

This lesson introduces you to tabs, allowing you to create embedded web experiences within Microsoft Teams. You create tabs whenever you need to give users information or workflows that Microsoft Teams doesn't provide natively.

You can create two types of tabs - static and configurable. Static tabs provide content for individual users, while configurable tabs provide a single type of content - such as meeting reminders or change notifications - for an entire team.

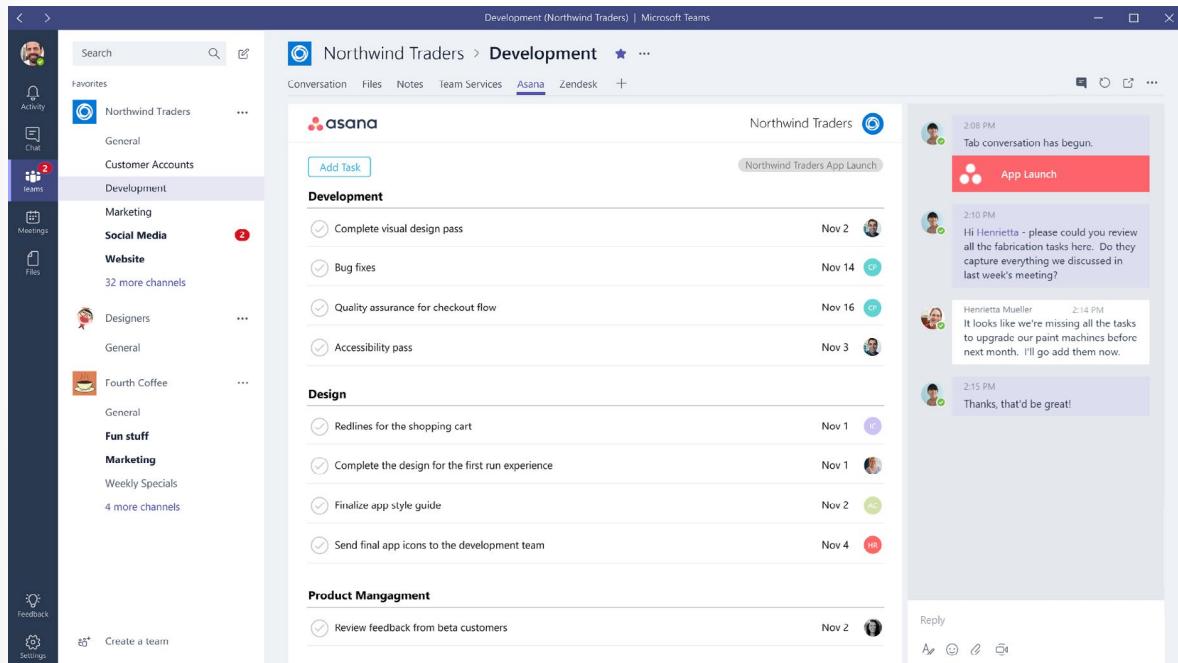
In this lesson, you'll learn about tabs in Microsoft Teams, the process of creating tabs, and how to avoid some testing and visual design mistakes that can prevent your tabs from working on mobile devices.

After this lesson, you should be able to:

- Explain when to use tabs.
- Describe the capabilities of personal tabs.
- Describe the capabilities of channel and group tabs.
- Explain the requirements for tabs for mobile clients.

What is a tab in Microsoft Teams?

In Microsoft Teams, tabs display rich interactive web content. You can build a tab from scratch or adapt an existing web app.



Teams provides two types of tabs:

- Static tabs support an individual user as personal tabs.
- Configurable tabs become part of a channel or group and provide a single type of information to a team. Configurable tabs have a configuration page in addition to a content page.

Tab scope

Microsoft Teams determines where a tab can be used based on its scope. You set the scope in the app manifest by using one of these values:

- **Teams** (team scope): Tabs in channels allow teams to interact with your shared experience. Currently, all tabs in channels are configurable tabs - a user configures the content of your tab experience when the tab is first added to a channel.
- **Group chat** (groupchat scope): You can also use configurable tabs in group chats, defined as conversations between two or more users.
- **Personal** (personal scope): Tabs in the personal scope allow users to interact privately with your experience. Currently, all personal tabs are static tabs - content that is relevant to individual users.

Requirements and considerations for tab pages in Teams

Tabs display web pages, but not all web pages render properly in a tab. Pages loaded in a custom tab need to:

- Allow Teams to display them in an **iframe¹⁹** (via the X-Frame-Options and/or Content-Security-Policy headers). Iframes provide nested browsing hosted by the parent window (`window.parent`). Many standard webpages don't allow iframes.
- Handle **authentication²⁰** differently, such as using a popup. Most websites simply redirect to a login provider, which typically dead-ends tabs hosted in an iframe in order to prevent clickjacking.
- Handle **cross-domain²¹** navigation differently, because the Teams client needs to validate the origin against a static `validDomains` list in the app manifest when loading or communicating with the tab.
- Use the Teams client visual themes.
- Make calls to the **Teams client SDK²²** (`microsoftTeams.initialize()`), which gives Teams a communication channel with the hosted page and more visibility into its operations.

Static tabs

A static tab provides content for individual users. For example, say you have a notetaking app. You can add a tab that holds a user's personal notes. That way, users can refer to their notes without having to share them with the entire team.

A static tab is a **content page²³** that you declare in your manifest. However, unlike a configurable tab, it doesn't require a configuration page.

Creating a static tab involves the following steps:

- Declare the tab in the manifest of your app package. For guidance, see **Extend your Teams app with a custom tab²⁴**.
- Create the content page. Teams displays the page in an iframe. Note that static and configurable tabs impose the same restraints on content.

¹⁹ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/tabs/tabs-content>

²⁰ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/authentication/auth-flow-tab>

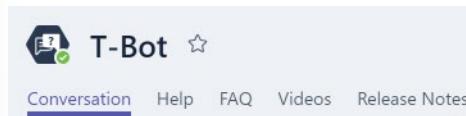
²¹ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/tabs/cross-domain>

²² <https://docs.microsoft.com/en-us/javascript/api/overview/msteams-client>

²³ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/tabs/tabs-content>

²⁴ <https://docs.microsoft.com/en-us/microsoftteams/platform/tabs/how-to/add-tab>

Currently, you can add one or more static tabs to your app's "personal scope" experience, which users access either by using the app bar or alongside your app's bot conversation.



Creating tab content

Content pages in Teams, regardless of scope or type, must follow the guidelines in [Create a content page²⁵](#).

Static tabs on mobile clients

Static tabs should also follow the guidance for [personal apps on mobile²⁶](#). They also need to follow the [responsive web design²⁷](#) principles and work well on any screen resolution.

Configurable tabs

A configurable tab becomes part of a channel and provides a single type of information to a team.

For example:

- The **Planner** tab for a channel contains a single plan; the Power BI tab maps to a specific report. Users can drill down to the relevant context, but they should not be able to navigate outside the tab.
- The **Power BI** tab doesn't enable navigation to other Power BI reports, but it does enable the **Go to website** button that launches the report in the main Power BI website.

Configurable tab scope

You define configurable tabs in the app manifest, and you scope them to `team` or `groupchat`, which determines where Teams uses them.

- **Teams** (team scope): Tabs in channels allow teams to interact with your shared experience. Currently, all tabs in channels are configurable.
- **Group chat** (groupchat scope): Conversations between two or more users.

Creating a configurable tab

Creating a configurable tab involves the following:

- **Create the configuration page²⁸**: For configurable tabs, you must provide a configuration page to present options and gather information so users can customize the content in, and experience with, your tab. You can also enable users to update or remove a tab after they add it.
- **Create the content page²⁹**: A content page is an HTML page that you host. You can also provide a page for users to specify what happens to content when they remove a tab.

²⁵ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/tabs/tabs-content>

²⁶ <https://docs.microsoft.com/en-us/microsoftteams/platform/resources/design/framework/tabs-mobile>

²⁷ https://www.w3schools.com/html/html_responsive.asp

²⁸ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/tabs/tabs-configuration>

²⁹ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/tabs/tabs-content>

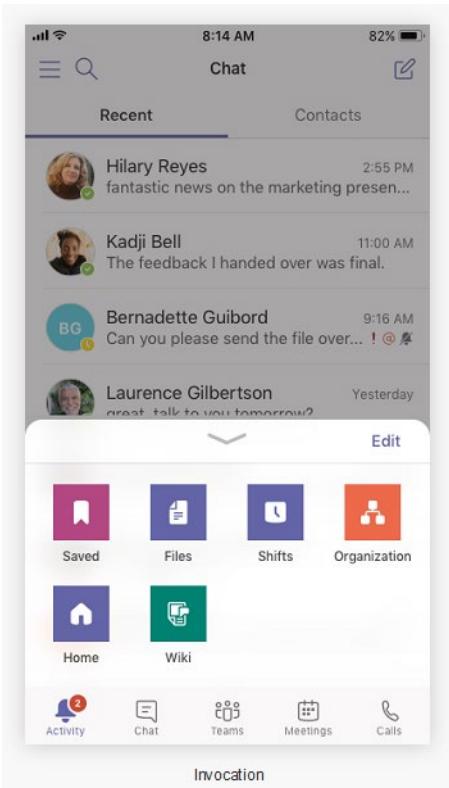
As mentioned earlier in this module, you can also allow users to create and share **deep links to items within your tab³⁰**, such as a link to an individual task within a tab that contains a task list.

Tabs on mobile

Custom tabs can be part of a channel, group chat, or personal app (apps that contain static tabs and/or a one-to-one bot).

Personal apps are available on mobile clients in the App Drawer. Users can only install personal apps from a desktop or web client, and they can take up to 24 hours to appear on mobile clients.

Group and channel tabs are available on mobile clients as well. The default behavior is currently to use the `websiteUrl` to launch your tab in a browser window. However, users can load them onto a mobile client by selecting the ... overflow menu next to the tab and choosing **Open**, which uses your `contentUrl` to load the tab inside the Teams mobile client.



Developer considerations for mobile support

When you're building an app that includes a tab, you need to consider (and test) how your tab will function on the Android and iOS Teams clients. The following sections outline some of the key scenarios you need to consider.

³⁰ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/deep-links>

Testing on mobile clients

You need to ensure that your tab works properly on mobile devices of various sizes and qualities. For Android devices, you can use the **DevTools**³¹ to debug your tab while it's running. Microsoft recommends that you test on high and low performance devices, plus a tablet.

Responsive design

Because your tab can be opened on devices with a wide range of screen sizes, it needs to follow **responsive design**³² principles. All of the tab's key constructs should be accessible on mobile devices, and without distortion. Ensure that all buttons and links are easily accessible using finger-based navigation.

Authentication

For authentication to work on mobile clients, upgrade your Teams JS SDK to at least version 1.4.1.

Low bandwidth & intermittent connections

Mobile clients often need to function with low bandwidth and intermittent connections. Your app should handle any timeouts appropriately by providing a contextual message to the user. You should also display progress indicators to alert your users to any long-running processes.

Design considerations for mobile

Team's mobile platform allows apps to be an immersive experience with the app content taking up all of the screen apart from main Teams navigation. To create an immersive experience that fits seamlessly within the Microsoft Teams client, follow the guidelines below.

Layouts

Be careful when choosing a layout for your tab. Consider the kind of information you're presenting, and choose a layout that organizes it for easy consumption. Here are some options:

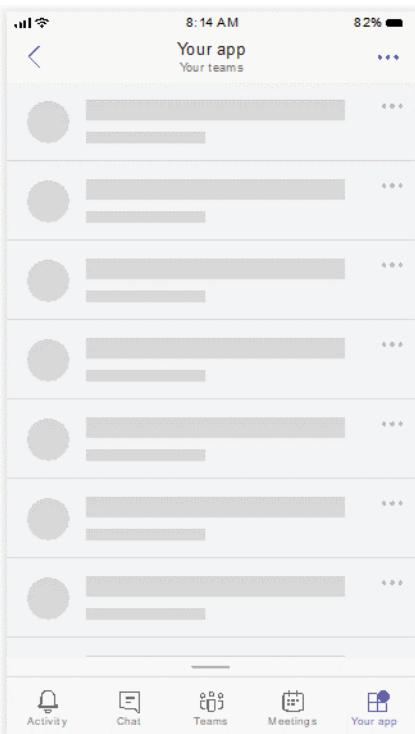
- **Single canvas:** Provides one large working area. The Wiki app uses this layout. If your app doesn't separate content into smaller components, this is a good fit.

³¹ <https://docs.microsoft.com/en-us/microsoftteams/platform/resources/dev-preview/developer-preview-tools>

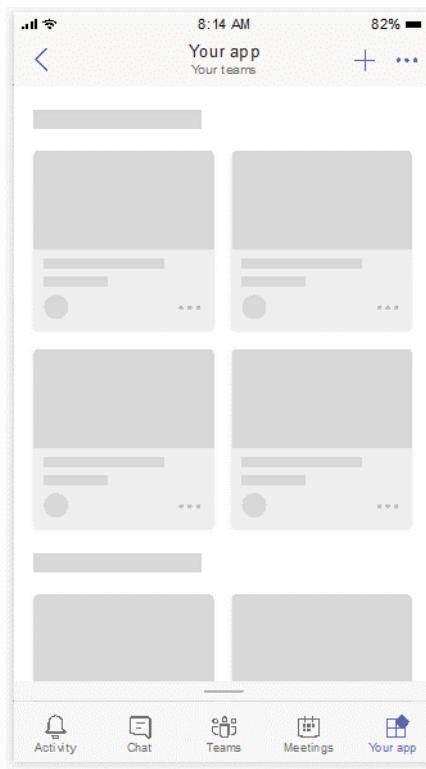
³² https://www.w3schools.com/html/html_responsive.asp



- **List:** Great for sorting and filtering large quantities of data and keeping the most important items at the top. You can use sortable columns. Actions can be added to each list item under the ellipsis menu.

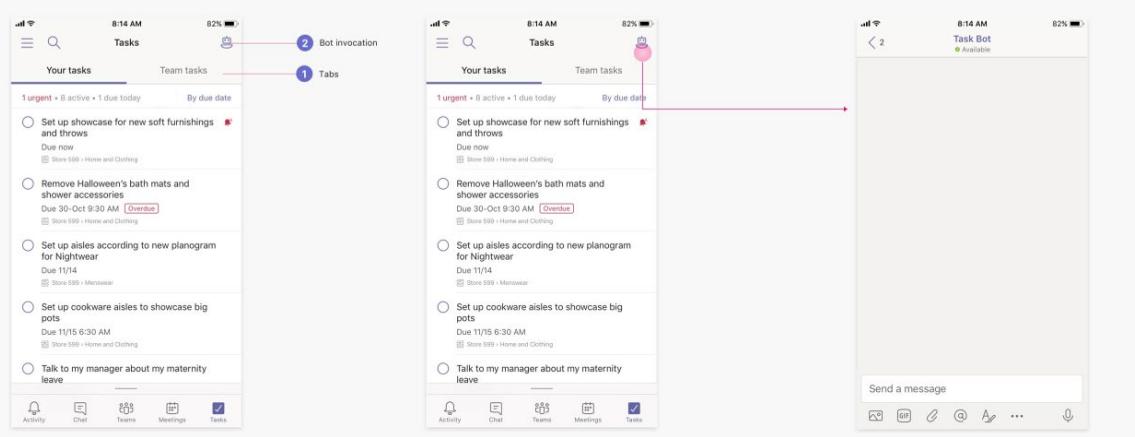


- **Grid:** Useful for showing visual elements such as photographs. It helps to include a filter or search control at the top.



Tabs with bots on mobile

Here's an example of a personal app that uses two static tabs and a bot.



Lesson review questions

1. Name at least three factors that you need to consider when designing a tab for mobile.
2. List at least four reasons why web pages may not render well in a tab.
3. Describe a scenario where you would choose a static tab instead of a configurable tab.

Correct/suggested answers:

1. The need to test on mobile devices; responsive design so content renders properly; authentication; low bandwidth; and intermittent connections
2. They don't work well with iframes; they don't allow authentication; they don't allow cross-domain navigation; they don't use the Teams client visual themes; and they don't call the Teams client SDK.
3. Scenarios:
 - Static tabs support an individual user as personal tabs.
 - Configurable tabs become part of a channel or group and provide a single type of information to a team. Configurable tabs have a configuration page in addition to a content page.

Messaging Extensions in Microsoft Teams

Lesson introduction

Messaging extensions allow users to query your service for information, and post information to your service. You use messaging extensions in Microsoft Teams whenever you select a command located at the bottom of the compose textbox, such as **Emoji**, **Giphy**, or **Sticker**.

You can create two types of messaging extensions in Microsoft Teams: search-based and action-based. Search-based extensions allow users to query your service for information, and action-based extensions allow them to post information to your service.

This lesson introduces you to search-based and action-based messaging extensions. You'll learn where messaging extensions can be invoked from, when to use each type of message extension command, and the process of creating messaging extensions.

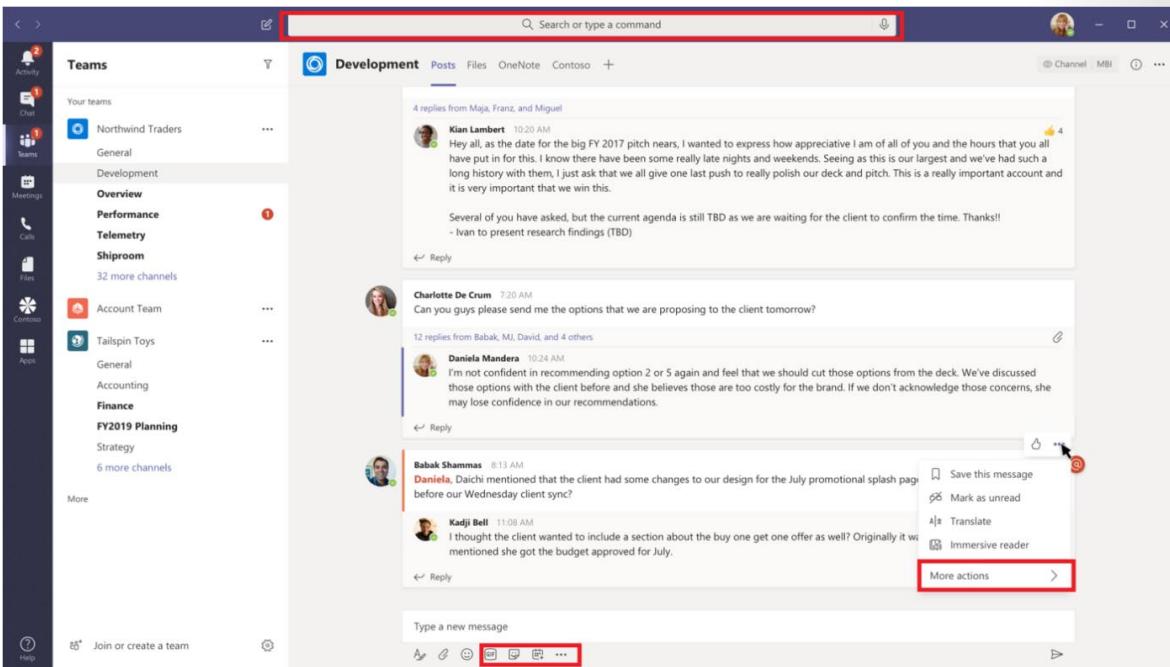
After this lesson, you should be able to:

- Explain messaging extensions and where they can be invoked from.
- Describe when to use the different messaging extension command types based on requirements.
- Describe search-based messaging extensions.
- Describe action-based messaging extensions with adaptive cards and with parameters.

What are messaging extensions for Microsoft Teams?

Messaging extensions allow users to query your service for information using search commands, or post information to your service using action commands. They can be invoked from the compose message area, the command box, or directly from a message. You can then send the results of that interaction back to the Microsoft Teams client, typically in the form of a richly formatted card.

The following screenshot of the Teams UI highlights the 3 options users have for invoking messaging extensions.



A few messaging extensions are built in, such as **Emoji**, **Giphy**, and **Sticker**. Users can select the **More Options (...)** button to see other messaging extensions, including those added from the app gallery or uploaded manually.

Types of messaging extensions

There are two types of messaging extensions:

- **Search-based messaging extensions** allow users to query your service and view results in the form of a card, right within a message.

Example: **Lookup a work item**



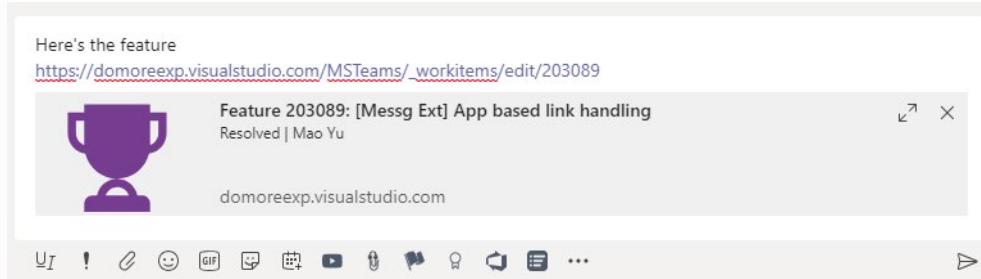
- **Action-based messaging extensions** allow you present your users with a modal popup to collect or display information. When they submit the form, your web service can respond by inserting a message into the conversation directly, or by inserting a message into the compose message area and allowing the user to submit the message. You can even chain multiple forms together for more complex workflows.

Example: **Create a work item**

Link Unfurling

You also have the option to invoke your service when a URL is pasted in the compose message area. This functionality, known as **link unfurling**, allows you to subscribe to receive an invoke when URLs containing a particular domain are pasted into the compose message area. Your web service can “unfurl” the URL into a detailed card, providing more information than the standard website preview card. You can even add buttons to allow your users to immediately take action without leaving the Microsoft Teams client. This works very similarly to a search command, with the URL serving as the search term.

In the following screenshot, a user has pasted in a URL for a work item in Azure DevOps, which the messaging extension has resolved into a card.



When to use messaging extensions

Here are a few example scenarios for which messaging extensions can be useful:

- **Scenario:** I need some external system to do something and I want the result of the action to be sent back to my conversation.
 - **Example:** Reserve a resource and let the channel know what day/time you reserved it for.
- **Scenario:** I need to find something in an external system, and I want to share the results with my conversation.
 - **Example:** Search for a work item in Azure DevOps and share it with the group as an adaptive card.
- **Scenario:** I need to complete a complex task involving multiple steps (or lots of information) in an external system, and the results should be shared with a conversation.
 - **Example:** Create a bug in your tracking system based on a Teams message, assign that bug to Bob, then send a card to the conversation thread with the bug's details.

Overview of creating messaging extensions

At a high level, you'll need to complete the following steps to create a messaging extension.

1. Prepare your development environment
2. Create and deploy your web service (during development, use a tunneling service like ngrok to run it locally)
3. Register your web service with the Bot Framework
4. Create your app package
5. Upload your package to Microsoft Teams

Creating your web service, creating your app package, and registering your web service with the Bot Framework can be done in any order. Because those three pieces are so intertwined, no matter which

order you do them in you'll need return to update the others. Your registration needs the messaging endpoint from your deployed web service, and your web service needs the ID and password created from your registration. Your app manifest also needs that ID to connect Teams to your web service.

As you're building your messaging extension, you'll regularly be moving between changing your app manifest, and deploying code to your web service. When working with the app manifest, keep in mind that you can either manually manipulate the JSON file, or make changes through App Studio. Either way, you'll need to re-deploy (upload) your app in Teams when you make a change to the manifest, but there's no need to do so when you deploy changes to your web service.

Defining a messaging extension in your app manifest

A messaging extension consists of your hosted web service and your app manifest, which defines where users can invoke your service from within the Teams client. You can update your app manifest manually or using App Studio.

Note:

The manifest refers to messaging extensions as `composeExtensions`. This is to maintain backward compatibility.

The extension definition is a `composeExtension` object that has the structure shown in the following table.

Property name	Purpose	Required?
<code>botId</code>	The unique Microsoft app ID for the bot as registered with the Bot Framework. This should typically be the same as the ID for your overall Teams app.	Yes
<code>scopes</code>	Array declaring whether this extension can be added to personal or team scopes (or both).	Yes
<code>canUpdateConfiguration</code>	Enables Settings menu item.	No
<code>commands</code>	Array of commands that this messaging extension supports. You are limited to 10 commands.	Yes

Messaging extension command types

When you define a messaging extension in your app manifest, you can specify up to ten different commands. Each command defines a `type` based on whether you're creating a *search-based* messaging extension or an *action-based* messaging extension, in addition to the options that users will have for invoking the command from the Teams client.

For search-based extensions the command type is `query`, and for action-based extensions the command type is `action`. If not present, the default command type value is set to `query`.

Note that you can currently only create a single messaging extension per app, and the commands for an extension can include both search- and action-based commands.

Example

The example below is a simple messaging extension object in the app manifest with a search command. This is not the entire app manifest file, just the part specific to messaging extensions.

```
...
"composeExtensions": [
  {
    "botId": "abcd1234-1fc5-4d97-a142-35bb662b7b23",
    "canUpdateConfiguration": true,
    "commands": [
      {
        "id": "searchCmd",
        "description": "Search your Todo's",
        "title": "Search",
        "initialRun": true,
        "parameters": [
          {
            "name": "searchKeyword",
            "description": "Enter your search keywords",
            "title": "Keywords"
          }
        ]
      }
    ]
  }
]
...
]
```

Your Web Service

Once invoked, your web service will receive an HTTPS message with a JSON payload including all the relevant information. You'll respond with a JSON payload, letting the Teams client know what interaction to enable next. The nature of how your web service can be invoked and respond will depend on whether you're using a search-based messaging extension or an action-based messaging extension.

Creating search-based messaging extensions

Creating a search-based messaging extension involves defining the commands for the messaging extension in your app manifest and setting up your service to receive and respond to queries.

Define search-based messaging extension commands

Choose command invoke locations

The first thing you need to decide is where users can invoke your search command from within the Teams client. Search commands can be invoked from one or both of the following locations:

- The buttons at the bottom of the compose message area
- By @mentioning your app in the command box

Search commands cannot be invoked from messages.

When invoked from the compose message area, your user will have the option of sending the results to the conversation. When invoked from the command box, the user can interact with the resulting card, or copy it for use elsewhere.

Add the command to your app manifest

In the app manifest, your command item is an object with the structure shown in the following table.

Property name	Purpose	Required?	Minimum manifest version
id	Unique ID that you assign to this command. The user request will include this ID.	Yes	1.0
title	Command name. This value appears in the UI.	Yes	1.0
description	Help text indicating what this command does. This value appears in the UI.	Yes	1.0
type	Set the type of command. Possible values include query and action. If not present, the default value is set to query	No	1.4
initialRun	Optional parameter, used with query commands. If set to true, indicates this command should be executed as soon as the user chooses this command in the UI.	No	1.0

Property name	Purpose	Required?	Minimum manifest version
fetchTask	Optional parameter, used with action commands. Set to true to fetch the Adaptive card or web URL to display within the task module (https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/task-modules/task-modules-overview). This is used when the input to the action command is dynamic as opposed to a static set of parameters. Note that if set to true, the static parameter list for the command is ignored.	No	1.4
parameters	Static list of parameters for the command.	Yes	1.0
parameter.name	The name of the parameter. This is sent to your service in the user request.	Yes	1.0
parameter.description	Describes this parameter's purposes or example of the value that should be provided. This value appears in the UI.	Yes	1.0
parameter.title	Short user-friendly parameter title or label.	Yes	1.0
parameter.inputType	Set to the type of input required. Possible values include text, textarea, number, date, time, toggle. Default is set to text.	No	1.4

Property name	Purpose	Required?	Minimum manifest version
context	Optional array of values that defines the context in which the message action is available. Possible values are message, compose, or commandBox. Default is ["compose", "commandBox"].	No	1.5

For search-based messaging extensions, the `type` parameter should be set to `query`.

You can add the command to your app manifest manually, or using the Manifest Editor in App Studio.

Testing your messaging extension

You test your messaging extension by uploading your app. See [Upload an app package to Microsoft Teams](#)³³ for more information.

To open your messaging extension, navigate to any of your chats or channels. Choose the **More options** (...) button in the compose box, and choose your messaging extension.

Receiving queries

When the user performs a query, Microsoft Teams issues a synchronous HTTP request to your service, sending a standard Bot Framework Activity object. In addition to the standard bot activity properties, the payload contains the following request metadata.

Property name	Purpose
<code>type</code>	Type of request; must be <code>invoke</code> .
<code>name</code>	Type of command issued to your service. Currently the following types are supported: <code>composeExtension/query</code> , <code>composeExtension/querySettingUrl</code> , <code>composeExtension/setting</code> , <code>composeExtension/selectItem</code> , <code>composeExtension/queryLink</code>
<code>from.id</code>	ID of the user that sent the request.
<code>from.name</code>	Name of the user that sent the request.
<code>from.aadObjectId</code>	Azure Active Directory (AD) object ID of the user that sent the request.
<code>channelData.tenant.id</code>	Azure AD tenant ID.
<code>channelData.channel.id</code>	Channel ID (if the request was made in a channel).
<code>channelData.team.id</code>	Team ID (if the request was made in a channel).
<code>clientInfo</code> entity	Additional metadata about the client, such as locale/language and type of client.

The request parameters are found in the `value` object, which includes the following properties.

³³ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/apps/apps-upload>

Property name	Purpose
commandId	The name of the command invoked by the user, matching one of the commands declared in the app manifest.
parameters	Array of parameters. Each parameter object contains the parameter name, along with the parameter value provided by the user.
queryOptions	Pagination parameters are skip (skip count for this query) and count (number of elements to return).

This invoke is triggered:

- As characters are entered into the search box.
- If initialRun is set to true in your app manifest, you'll receive the invoke message as soon as the search command is invoked.

At that point, your code has 5 seconds to provide an HTTP response to the request. During this time, your service can perform additional lookup, or any other business logic needed to serve the request.

Receiving requests from links inserted into the compose message box

As mentioned previously, as an alternative (or in addition) to searching your external service, users can insert a URL into the compose message box to query your service.

Responding to queries

Your service should respond with the results matching the user query. The response must indicate an HTTP status code of 200 OK and a valid application/json object with the body shown in the following table.

Property name	Purpose
composeExtension	Top-level response envelope.
composeExtension.type	Type of response. The supported types are result (displays a list of search results), auth (asks the user to authenticate), config (asks the user to set up the messaging extension), and message (displays a plain text message).
composeExtension.attachmentLayout	Specifies the layout of the attachments. Used for responses of type result. Currently the supported types are list (a list of card objects containing thumbnail, title, and text fields), and grid (a grid of thumbnail images).
composeExtension.attachments	Array of valid attachment objects. Used for responses of type result. Currently the supported types are: application/vnd.microsoft.card.thumbnail, application/vnd.microsoft.card.hero, application/vnd.microsoft.teams.card.o365connector, and application/vnd.microsoft.card.adaptive.

Property name	Purpose
composeExtension.suggestedActions	Suggested actions. Used for responses of type auth or config.
composeExtension.text	Message to display. Used for responses of type message.

Response card types

The following attachment types are supported:

- **Thumbnail card**³⁴
- **Hero card**³⁵
- **Office 365 Connector card**³⁶
- **Adaptive card**³⁷

See [Cards](#)³⁸ for an overview of all card types.

To learn how to use Thumbnail and Hero cards, see [Add cards and card actions](#)³⁹.

Default query

If you set `initialRun` to `true` in the manifest, Teams issues a “default” query when the user first opens the messaging extension. Your service can respond to this query with a set of prepopulated results. This can be useful for displaying, for instance, recently viewed items, favorites, or any other information that is not dependent on user input.

The default query has the same structure as any regular user query, except with a parameter `initialRun` whose string value is `true`.

Authentication

If your service requires user authentication, you need to sign in the user before he or she can use the messaging extension.

Defining action-based messaging extensions

Using action-based messaging extensions involves defining the extension commands in your app manifest, creating and sending the task module for users to interact with, and sending a response to the user’s submit action. The following sections provide an overview of each of these processes.

Define action-based messaging extensions

Before creating your command, you’ll need to decide:

- Where can the action command be triggered from?

³⁴ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards/cards-reference>

³⁵ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards/cards-reference>

³⁶ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards/cards-reference>

³⁷ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards/cards-reference>

³⁸ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards/cards>

³⁹ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards-actions>

- How will the task module be created?
- Will the final message or card be sent to the channel from a bot, or will it be inserted into the compose message area for the user to submit?

Choosing a location for invoking an action command

The first thing you need to decide is where your action command can be invoked from. By specifying the context in your app manifest, your command can be invoked from one or more of the following locations:

- The buttons at the bottom of the compose message area.
- By @mentioning your app in the command box.

Note:

You cannot respond with a bot message inserted directly into the conversation if your messaging extension is invoked from the command box.

- Directly from an existing message via the ... **overflow** menu on a message.

Note:

The initial invoke to your bot will include a JSON object containing the message from which it was invoked, which you can process before presenting users with a task module.

When using an adaptive card, your messaging extension needs to handle an initial invoke event from the user, create the task module, and return it back to the client.

Deciding how the task module will be created

In addition to choosing where your command can be invoked from, you must also decide how to populate the form in the task module for your users. You have three options for creating the form that is rendered inside the task module:

- Static list of parameters - This is the simplest option. You can define a list of parameters (input fields) in your app manifest the Teams client will render. You cannot control the formatting with this option.
- Adaptive card - You can choose to use an adaptive card, which provides greater control over the UI, but still limits you on the available controls and formatting options.
- Embedded web view - If you need complete control over the UI and controls, you can choose to embed a custom web view in the task module.

If you choose to create your task module with a static list of parameters, the first call to your messaging extension will be when a user submits the task module. When using an embedded web view or an adaptive card, your messaging extension will need to handle an initial invoke event from the user, create the task module, and return it back to the client.

Choosing how to send the final message

In most cases, your action command inserts a card into the compose message box. Your user can then decide to send it into the channel or chat. The message in this case comes from the user, and your bot can't edit or update the card.

If your messaging extension is triggered from the compose box or directly from a message, your web service can insert the final response directly into the channel or chat. In this case, the Adaptive card comes from the bot, the bot will be able to update it, and the bot can also reply to the conversation

thread if needed. You will need to add the bot object to your app manifest using the same ID and by defining the appropriate scopes.

Adding the command to your app manifest

Once you decide how users will interact with your action command, add the command to your app manifest. To do this, add a new `composeExtension` object to the top level of your app manifest JSON. You can either do that in App Studio, or manually.

An action-based messaging extension command has the following properties.

Property name	Purpose	Required?	Minimum manifest version
<code>id</code>	Unique ID that you assign to this command. The user request will include this ID.	Yes	1.0
<code>title</code>	Command name. This value appears in the UI.	Yes	1.0
<code>type</code>	Must be <code>action</code> .	No	1.4
<code>fetchTask</code>	<code>true</code> for an Adaptive card or embedded web view for your task module, <code>false</code> for a static list of parameters or when loading the web view by a <code>taskInfo</code> .	No	1.4
<code>context</code>	Optional array of values that defines where the messaging extension can be invoked from. Possible values are <code>message</code> , <code>compose</code> , or <code>commandBox</code> . Default is <code>["compose", "commandBox"]</code> .	No	1.5

Create and send the task module

If you are not populating your task module with parameters defined in your app manifest, you'll need to create the task module to be presented to your users. You can use either an Adaptive Card or an embedded web view.

Your service will receive an `Activity` object of type `composeExtension/fetchTask`, and you'll need to respond with a `task` object containing either the adaptive card or a URL to the embedded web view.

Respond to the submit action

If you allow your users to send a response back from the task module, you'll need to handle the submit action. Once a user submits the task module, your web service will receive a `composeExtension/submitAction` invoke message with the command id and parameter values set. Your app will have five

seconds to respond to the invoke, otherwise the user will receive an “Unable to reach the app” error message, and any reply to the invoke will be ignored by the Teams client.

Response options

You have the following options for responding.

- No response - You can choose to use the submit action to trigger a process in an external system, and not provide any feedback to the user. This can be useful for long-running processes, and you may choose to provide feedback in another manner (for example, with a **proactive message**⁴⁰).
- **Another task module**⁴¹ - You can respond with an additional task module as part of a multi-step interaction. This can be useful when you need to collect large amounts of information, you need to dynamically change what information you're collecting based on user input, or you need to validate the information submitted by the user and potentially resend the form with an error message if something is wrong.
- **Card response**⁴² - You can respond with a card that the user can then interact with and/or insert into a message. This is the most common way to respond.
- **Adaptive Card from bot**⁴³ - Insert an Adaptive Card directly into the conversation. This can be very useful in scenarios where you need to gather information from your users before creating an adaptive card response, or when you're going to need to update the card after someone interacts with it.
- **Request the user authenticate**⁴⁴
- **Request the user provide additional configuration**⁴⁵

The table below shows which types of responses are available based on the invoke location (command-Context) of the messaging extension. For authentication or configuration, once the user completes the flow the original invoke will be re-sent to your web service.

Response Type	compose	command bar	message
Card response	x	x	x
Another task module	x	x	x
Bot with Adaptive Card	x		x
No response	x	x	x

Lesson review questions

1. Name the types of messaging extensions and provide one example for a scenario that would be an appropriate use of each type.
2. Where can search-based messaging extensions be invoked from within the Teams client?
3. How can you collect information from the user for an action-based messaging extension?

⁴⁰ <https://docs.microsoft.com/en-us/microsoftteams/platform/bots/how-to/conversations/send-proactive-messages>

⁴¹ <https://docs.microsoft.com/en-us/microsoftteams/platform/messaging-extensions/how-to/action-commands/respond-to-task-module-submit?tabs=dotnet>

⁴² <https://docs.microsoft.com/en-us/microsoftteams/platform/messaging-extensions/how-to/action-commands/respond-to-task-module-submit?tabs=dotnet>

⁴³ <https://docs.microsoft.com/en-us/microsoftteams/platform/messaging-extensions/how-to/action-commands/respond-to-task-module-submit?tabs=dotnet>

⁴⁴ <https://docs.microsoft.com/en-us/microsoftteams/platform/messaging-extensions/how-to/add-authentication>

⁴⁵ <https://docs.microsoft.com/en-us/microsoftteams/platform/messaging-extensions/how-to/add-configuration-page>

Correct/suggested answers:

1. Types:
 - Search-based: Query your service for information and insert that into a message. For example, allow users to search for work items in a dev ops system.
 - Action-based: Collect information from the user and post it to a third-party service. For example, allow users to create a new work item in a dev ops system.
2. Search commands can be invoked from one or both of the following locations:
 - The buttons at the bottom of the compose message area. User will have the option of sending the results to the conversation.
 - By @mentioning your app in the command box. User can interact with the resulting card, or copy it for use elsewhere.
3. You have three options for creating the form that is rendered inside the task module:
 - Static list of parameters - This is the simplest option. You can define a list of parameters (input fields) in your app manifest the Teams client will render. You cannot control the formatting with this option.
 - Adaptive card - You can choose to use an adaptive card, which provides greater control over the UI, but still limits you on the available controls and formatting options.
 - Embedded web view - If you need complete control over the UI and controls, you can choose to embed a custom web view in the task module.

If you choose to create your task module with a static list of parameters, the first call to your messaging extension will be when a user submits the task module. When using an embedded web view or an adaptive card, your messaging extension will need to handle an initial invoke event from the user, create the task module, and return it back to the client.

Conversational Bots in Microsoft Teams

Lesson introduction

Conversational bots allow you to provide chat capabilities to your users. You can create several types of chats, such as group chats with more than two people, or chats between a single user and a bot.

This lesson describes conversational bots and when to use them, and provides an overview of the capabilities of conversational bots in Microsoft Teams apps.

After this lesson, you should be able to:

- Explain when to use conversational bots in a Microsoft Teams app.
- Describe the scoping options for bots.
- Explain when to invoke a task module from a bot.

Overview of Bots in Teams

Bots are apps that users interact with in a conversational way. Users can enter text or graphics (such as cards or images), or use speech. Every interaction between a user and a bot generates an *activity* object. The Bot Framework Service sends the activity objects between the user's *channel* - a bot-connected app such as Teams - and the bot. Channels may include additional information in the activities they send.

This course assumes you know how bots use activity objects. If you don't, see [How bots work⁴⁶](#).

Conversational bots in Teams apps

With Teams apps, you can make the bot the star of your experience, or just a helper. Bots are distributed as part of your broader app package, which can include other capabilities such as tabs or messaging extensions.

Conversational bots allow users to interact with your web service through text, interactive cards, and task modules. They're incredibly flexible — conversational bots can be scoped to handling a few simple commands or complex, artificial intelligence powered and natural language processing virtual assistants.

Types of conversations

A bot appears just like any other team member; you interact with it in a conversation. The difference is that it has a hexagonal avatar icon and is always online.

Bots in Teams support several kinds of conversations (called *scopes* in the app manifest):

- *teams*: Also called channel conversations
- *personal*: Conversations between a bot and a single user
- *groupChat*: A conversation between a bot and 2 or more users

For the bot to work in a particular scope, you need to list it in the manifest.

Bots behave differently depending on the kind of conversation they're involved in. Each scope will provide unique opportunities and challenges for your conversational bot.

⁴⁶ <https://docs.microsoft.com/en-us/azure/bot-service/bot-builder-basics?view=azure-bot-service-4.0&tabs=csharp>

In a channel

Channels contain threaded conversations between multiple people — potentially lots of people (currently, up to two thousand). This potentially gives your bot massive reach, but individual interactions need to be concise. Traditional multi-turn interactions probably won't work well. Instead, look to use interactive cards or task modules, or potentially move the conversation to a one-to-one conversation if you need to collect lots of information.

Your bot will also only have access to messages where it's @mentioned directly, you cannot retrieve additional messages from the conversation using Microsoft Graph and elevated organization-level permissions.

In a group chat

Group chats are non-threaded conversations between three or more people. They tend to have fewer members than a channel, and are more transient. Similar to a channel, your bot will only have access to messages where it's @mentioned directly.

In a one-to-one chat

This is the traditional way for a conversational bot to interact with a user. They can enable incredibly diverse workloads. **Bots in single user conversations⁴⁷** do not require an @mention - the user can just type their message.

Teams App Bot Scenarios

There are a variety of scenarios that are appropriate for conversational bots in Teams. Just remember to consider whether a conversation-based interface is the best way to present your functionality.

Some scenarios where bots excel in a channel or group chat include:

- **Notifications**, particularly if you provide an interactive card for users to take additional action.
- **Feedback scenarios** like polls and surveys.
- Interactions that can be resolved in a **single request/response cycle**, where the results are useful for multiple members of the conversation.
- **Social/fun bots** — get an awesome cat image, randomly pick a winner, etc.

In private one-to-one chats, Q&A bots, bots that initiate workflows in other systems, bots that tell jokes, and bots that take notes are just a few examples of scenarios for conversational bots.

Bot conversation basics

Each message is an `Activity` object of type `messageType: message`. When a user sends a message, Teams posts the message to your bot. More specifically, it sends a JSON object to your bot's messaging endpoint. Your bot examines the message to determine its type and responds accordingly.

Basic conversation is handled through the Bot Framework Connector, a single REST API to enable your bot to communicate with Microsoft Teams and other channels. The Bot Builder SDK provides easy access to this API, additional functionality to manage conversation flow and state, and simple ways to incorporate cognitive services such as natural language processing (NLP).

⁴⁷ <https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/bots/bot-conversations/bots-conv-personal>

Message content

Your bot can send rich text, pictures, and cards. In turn, users can send rich text and pictures to your bot. You specify the types of content your bot can handle in the Teams settings page for your bot. The following tables lists examples of content that your bot can send and receive.

Format	From user to bot	From bot to user	Notes
Rich text	Yes	Yes	n/a
Pictures	Yes	Yes	Maximum 1024x1024 and 1 MB in PNG, JPEG, or GIF format; animated GIFs are not supported.
Cards	No	Yes	See the Teams Card Reference (https://docs.microsoft.com/en-us/microsoftteams/platform/concepts/cards/cards-reference) for supported cards.
Emojis	No	Yes	Teams currently supports emojis via UTF-16 (such as U+1F600 for grinning face).

Receiving messages

Depending on the scopes you declare, your bot can receive messages in the following contexts:

- **Personal chat:** Users can interact in a private conversation with a bot by selecting the added bot in the chat history, or by typing its name or app ID in the **To:** box in a new chat.
- **Channels:** Users can mention a bot (@**botname**) in a channel if it has been added to the team. Note that additional replies to a bot in a channel require mentioning the bot. It will not respond to replies unless it's mentioned.

Subscribe to conversation events

Microsoft Teams sends notifications to your bot for events that happen in scopes where your bot is active. You can capture these events in your code and take action on them, such as the following:

- Trigger a welcome message when your bot is added to a team
- Trigger a welcome message when a new team member is added or removed
- Trigger a notification when a channel is created, renamed or deleted
- When a bot message is liked by a user

Sending messages

To send a text message, specify the string you want to send as the activity. You can send a single message response or send multiple responses at once.

Updating messages

Rather than have your messages be static snapshots of data, your bot can dynamically update messages inline after sending them. You can use dynamic message updates for scenarios such as poll updates, modifying available actions after a button press, or any other asynchronous state change.

The new message need not match the original in type. For instance, if the original message contained an attachment, the new message can be a simple text message.

Note:

You can only update content sent in single-attachment messages and carousel layouts. Posting updates to messages with multiple attachments in list layout is not supported.

Adding notifications to your message

Notifications alert users about new tasks, mentions and comments related to what they are working on or need to look at by inserting a notice into their Activity Feed. You can set notifications to trigger from your bot message, but whether or not a notification is raised will ultimately depend on the individual user's Teams settings and you cannot programmatically override these settings. The type of notification can be either a banner or both a banner and an email.

Sending proactive messages

Often your bot will be sending messages to users in response to messages it receives, but your bot can also send messages proactively. A proactive message is a message that is sent by a bot to start a conversation. You may want your bot to start a conversation for a number of reasons, including:

- Welcome messages for personal bot conversations
- Poll responses
- External event notifications

Sending a message to start a new conversation thread is different than sending a message in response to an existing conversation: when your bot starts a new a conversation, there is no pre-existing conversation to post the message to. In order to send a proactive message you need to:

- **Decide what you're going to say⁴⁸**
- **Obtain the user's unique Id and tenant Id⁴⁹**
- **Send the message⁵⁰**

Sending proactive messages to users can be a very effective way to communicate with your users. However, from their perspective this message can appear to come to them completely unprompted, and in the case of welcome messages will be the first time they've interacted with your app. As such, it is very important to use this functionality sparingly (don't spam your users), and to provide them with enough information to let them understand why they are being messaged.

Proactive messages generally fall into one of two categories, welcome messages or notifications.

⁴⁸ <https://docs.microsoft.com/en-us/microsoftteams/platform/bots/how-to/conversations/send-proactive-messages>

⁴⁹ <https://docs.microsoft.com/en-us/microsoftteams/platform/bots/how-to/conversations/send-proactive-messages>

⁵⁰ <https://docs.microsoft.com/en-us/microsoftteams/platform/bots/how-to/conversations/send-proactive-messages>

Combining channel and private interactions with your bot

When interacting in a channel, your bot should be smart about taking certain conversations offline with a user. For instance, suppose a user is trying to coordinate a complex task with a set of team members, such as scheduling. Rather than have the entire sequence of interactions visible to the channel, consider sending a personal chat message to the user. Your bot should be able to easily transition the user between personal and channel conversations without losing state.

Note: Don't forget to update the channel when the interaction is complete to notify the other team members.

Invoking a task module from a bot

Instead of having multiple conversation steps between your bot and a user, task modules can be a better user experience. You can invoke task modules from bots in Microsoft Teams by using buttons on Adaptive and Bot Framework cards (Hero, Thumbnail, and Office 365 Connectors). This prevents your app from having to keep track of bot state and allows the user to interrupt or cancel the sequence. Task modules can also be appropriate when using a bot in a channel or a group to prevent multi-turn interactions and keep everyone in the channel from seeing what another user is doing with the bot.

Lesson review questions

1. What are the differences between a bot and an outgoing webhook?
2. Name the types of conversations you can have with a Teams bot. How does their behavior differ among each type?
3. When is it appropriate to invoke a task module from a bot?

Correct/suggested answers:

1. An outgoing webhook allows you to create a simple bot for basic interaction, such as kicking off a workflow or other simple commands you may need. Outgoing webhooks live only in the team in which you create them, and they're intended for simple processes. Developers do not need to use the Bot Framework to create outgoing webhooks.
2. Types:
 - Teams
 - Personal
 - Group chatDifferences:
 - Bots in Teams and group chat conversations require the user to @ mention the bot to invoke it in a channel.
 - Bots in a one-to-one conversation do not require an @ mention. All messages sent by the user are routed to your bot.
3. Task modules are often a better user experience than multiple conversation steps where you as a developer have to keep track of the bot state. Task modules allow the user to interrupt or cancel the sequence. Task modules can also be appropriate when using a bot in a team or group chat to allow a user to interact with the bot without a multi-turn interaction that notifies the entire channel or group.

Module 5 Extend Office

Office Add-ins

Lesson introduction

The Microsoft Office Add-ins platform enables you to extend the functionality of Office applications. In this lesson, you will explore various ways you can use add-ins to extend and interact with Office applications. You will also learn about configuring your add-in using the add-in's manifest file.

After this lesson, you will know how to:

- Describe the fundamental components and types of Office Add-ins.
- Describe the purpose of the Office Add-ins manifest.
- Describe the task pane and content Office Add-ins.
- Describe dialog boxes.
- Describe components of custom functions.
- Describe Add-in commands.

Overview of the Office Add-ins platform

All Office Add-ins built upon the Office Add-ins platform share a common framework. You can use familiar web technologies such as HTML, CSS, and JavaScript to build a solution that can run in Office across multiple platforms, including on Windows, Mac, iPad, and in a web browser.

The Office Add-ins platform enables you to extend the functionality of Office applications. Creating custom ribbons, connecting to REST endpoints, or UI elements that look like a natural extension of Office and work across platforms.

An Office Add-in can enrich the user's experience within an Office application. Add-ins allow you to provide:

- Additional functionality in Office: Bring external data into Office, automate Office documents, expose third-party functionality in Office clients, and more. For example, use Microsoft Graph API to connect to data that drives productivity.

- Embedded rich, interactive objects: Embed maps, charts, and interactive visualizations that users can add to their own Excel spreadsheets and PowerPoint presentations.

Benefits of Office Add-ins

Office Add-ins don't involve code that runs on the user's device or in the Office client. For an Office Add-in, the host application, for example Excel, reads the add-in manifest and hooks up the add-in's custom ribbon buttons and menu commands in the UI. When needed, it loads the add-in's JavaScript and HTML code, which executes in the context of a browser in a sandbox. Key benefits of Office Add-ins include:

- Cross-platform support: Add-ins can run in Office on Windows, Mac, and iPad/iOS, and in a web browser.
- Centralized deployment and distribution: Office 365 administrators can deploy add-ins across their organization.
- Public availability via AppSource: By publishing your add-in to AppSource, you can make it available to the general public.
- Built using standard web technology: You can develop your add-in using your favorite framework.

Components of an Office Add-in

An Office Add-in includes two basic components: an XML manifest file and a web application. Like any other web application, the add-in must be hosted on a web server. After a user installs the add-in, the Office application reads the manifest to wire up any custom buttons in the UI and determine the location of the add-in's default page. When the add-in is activated, the web application's code is loaded and runs in a sandboxed context.

Manifest

The manifest is an XML file that specifies settings and capabilities of the add-in, such as:

- The add-in's display name, description, ID, version, and default locale.
- How the add-in integrates with Office.
- The permission level and data access requirements for the add-in.

Web application

A web application defines the functionality and user interface (UI), if any, of the add-in.

The web application can be a static HTML page that is displayed inside an Office application or a dynamic web page that interacts with either the Office document or any other Internet resource.

You can use any technologies, both client and server side, that the hosting provider supports (such as ASP.NET, PHP, or Node.js) to create an experience that interacts with Office documents or allows the user to interact with online resources from an Office host application. To interact with Office clients and documents, you use the Office.js JavaScript APIs.

Supported Office Add-in types, hosts, and platforms

Office Add-ins run in various Office applications on certain platforms. Add-ins also have defined types or extension points that determine how the add-in interacts with the Office application. Developers can leverage Office Add-ins to do the following within an Office host application:

Extend functionality (any Office application): You can add new functionality to Office applications via insertable task panes or custom ribbon buttons and menu commands (collectively called "add-in commands")

Create new objects (Excel or PowerPoint): You can embed web-based objects called content add-ins within Excel and PowerPoint documents. With content add-ins, you can integrate rich, web-based data visualizations, media (such as a YouTube video player or a picture gallery), and other external content.

The following table provides an overview of Office applications, platforms, and available add-in types.

Office application	Supported platforms	Available add-in types
Excel	Windows, Mac, iPad, web browser	task pane, content, custom functions (excluding iPad)
OneNote	web browser	task pane
Outlook	Windows, Mac, iOS, Android, web browser	task pane, contextual, UI-less functions, module (Windows only)
PowerPoint	Windows, Mac, iPad, web browser	task pane, content
Project	Windows	task pane
Word	Windows, Mac, iPad, web browser	task pane

Office Add-ins manifest

An Office Add-in's XML manifest file defines the settings and capabilities of the add-in. It describes how your add-in should be activated when an end user installs and uses it with Office documents and applications. You can configure it to control how your add-in is rendered and behaves in the targeted Office applications.

Role of the manifest

The Office applications where your add-in runs (such as Word or Excel) use information from the manifest to render add-in UI and wire up custom buttons or menu entries.

If you publish your add-in to AppSource, the information from the manifest (name, description, author, logo, etc.) is used to create the app entry that displays to potential customers in AppSource. The AppSource validation process reads information from the manifest and validates that your add-in runs on expected platforms.

Elements defined in a manifest

In the manifest, you define key information about the add-in, including:

- Add-in metadata (for example, ID, version, description, display name, default locale).
- Information about how the add-in integrates with Office (for example, target applications, custom functionality, add-in commands).

- Location of images the add-in should use for branding and command iconography.
- Permissions required by the add-in.
- Dimensions of the add-in (for example, default dimensions for content add-ins, requested height for Outlook add-ins).
- Rules that specify when the add-in should activate in a message or appointment (Outlook only).

Determining the required elements for a manifest depends on the type of Office Add-in: Content, Task Pane, Outlook. See **Required elements by Office Add-in type**¹.

When running in Office on the web, your task pane can be navigated to any URL. However, in desktop platforms, if your add-in tries to go to a URL in a domain other than the domain that hosts the start page (as specified in the **SourceLocation** element of the manifest file), that URL opens in a new browser window outside the add-in pane of the Office host application.

To override this (desktop Office) behavior, specify each domain you want to open in the add-in window in the list of domains specified in the **AppDomains** element of the manifest file. If the add-in tries to go to a URL in a domain that is in the list, then it opens in the task pane in both Office on the web and desktop. If it tries to go to a URL that isn't in the list, then, in desktop Office, that URL opens in a new browser window (outside the add-in pane).

The following XML manifest example hosts its main add-in page in the <https://www.contoso.com> domain as specified in the **SourceLocation** element. The **SourceLocation** specifies the source file location(s) for your Office Add-in as a URL between 1 and 2018 characters long. The source location must be an **HTTPS** address, not a file path.

It also specifies the <https://Northwind Traders>² domain in an **AppDomain** element within the **AppDomains** element list. **AppDomains** specifies additional domains that load pages in the add-in window. It also lists trusted domains from which Office.js API calls can be made from IFrames within the add-in. If the add-in goes to a page in the www.northwindtraders.com domain, that page opens in the add-in pane, even in Office desktop.

Note:

AppDomain elements should be used to specify any additional domains other than the one specified in the SourceLocation element.

```
<?xml version="1.0" encoding="UTF-8"?>
<OfficeApp xmlns="http://schemas.microsoft.com/office/appforoffice/1.1" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:type="TaskPaneApp">
    <!--IMPORTANT! Id must be unique for each add-in. If you copy this manifest, ensure that you change this id to your own GUID. -->
    <Id>c6890c26-5bbb-40ed-a321-37f07909a2f0</Id>
    <Version>1.0</Version>
    <ProviderName>Contoso, Ltd</ProviderName>
    <DefaultLocale>en-US</DefaultLocale>
    <DisplayName DefaultValue="Northwind Traders Excel" />
    <Description DefaultValue="Search Northwind Traders data from Excel"/>
    <SupportUrl DefaultValue="[Insert the URL of a page that provides support information for the app]" />
    <AppDomains>
        <AppDomain>https://www.northwindtraders.com</AppDomain>
    </AppDomains>
```

¹ <https://docs.microsoft.com/en-us/office/dev/add-ins/develop/add-in-manifests?tabs=tabid-1>

² <http://www.northwindtraders.com/>

```
<DefaultSettings>
    <SourceLocation DefaultValue="https://www.contoso.com/search_app/
Default.aspx" />
</DefaultSettings>
<Permissions>ReadWriteDocument</Permissions>
</OfficeApp>
```

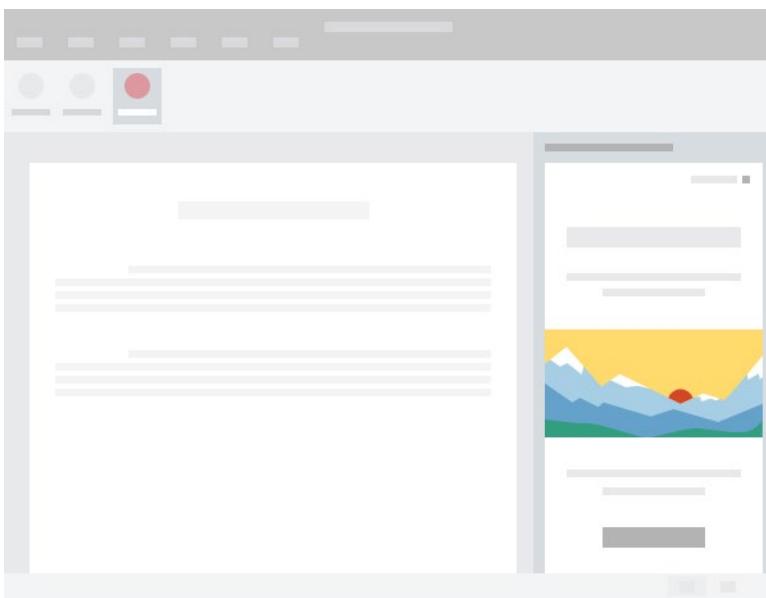
Task panes and content Office Add-ins

Office Add-ins provide several options for how your solution can interact with an Office application. In this topic, we'll discuss two of those options:

- Task pane
- Content

Task pane add-ins

Task panes are interface surfaces that typically appear on the right side of the window within Word, PowerPoint, Excel, and Outlook. Task pane add-ins allow user interaction through a panel displayed within an Office application. Through the task pane interface, you can enable the user to modify documents or emails, view data from a data source, and more. In the following image, the task pane is the panel that displays to the right of the document.



Use task panes when you don't need to embed functionality directly into the document. In newer versions of Word, Excel, and PowerPoint, you can configure the task pane to be displayed automatically when a user opens a file. The user will need to have your add-in installed first to activate this behavior.

Define the task pane add-in type

An add-in's manifest file defines the settings and capabilities of the add-in.

To configure an add-in as a task pane add-in for any Office application except Outlook, set the **xsi:type** attribute to **TaskPaneApp** within the **OfficeApp** element of the manifest file, as shown in the following example:

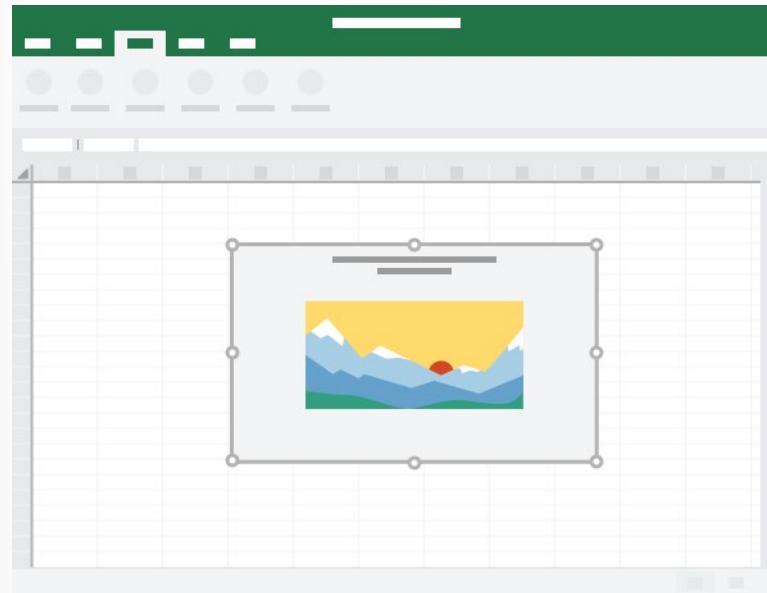
```
<OfficeApp  
...  
xsi:type="TaskPaneApp">  
...  
</OfficeApp>
```

To configure an add-in for Outlook, set the **xsi:type** attribute to **MailApp** within the **OfficeApp** element of the manifest file, as shown in the following example:

```
<OfficeApp  
...  
xsi:type="MailApp">  
...  
</OfficeApp>
```

Content add-ins

Content add-ins can be used to insert an object into an Excel spreadsheet or PowerPoint presentation. That object can be a web-based data visualization, media, or other external content. Use content add-ins when you want to embed functionality directly into the document. In the following image, the content add-in is displayed near the center of the document.



Define the content add-in type

An add-in's manifest file defines the settings and capabilities of the add-in. To configure an add-in as a content add-in, set the **xsi:type** attribute to **ContentApp** within the **OfficeApp** element of the manifest file, as shown in the following example:

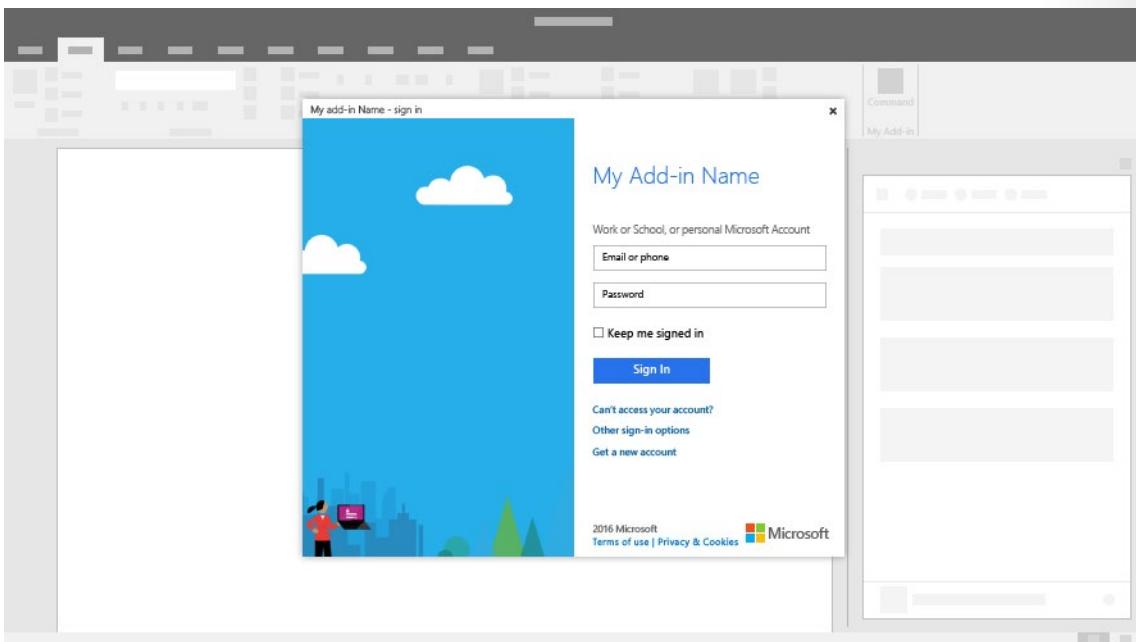
```
<OfficeApp  
...  
xsi:type="ContentApp">  
...  
</OfficeApp>
```

Dialog boxes in Office Add-ins

Dialog boxes are surfaces that float above the active Office application window. The Office Add-ins platform enables you to display a dialog box for your users to:

- Sign into an integrated service (for example, authenticate with Microsoft Account, Google, or Facebook).
- Confirm the user's action.
- Run a task that might be too confined in a task pane (for example, view a video).

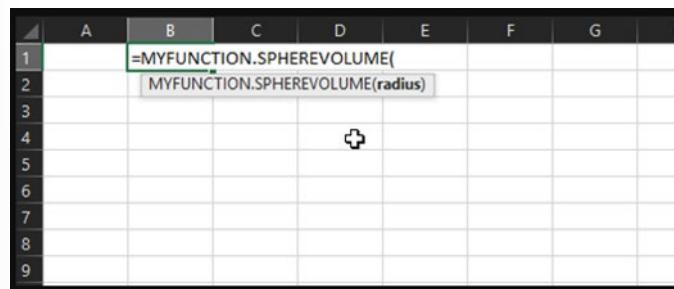
The dialog box is not modal, meaning that your user can continue to interact with the Office application and your add-in while the dialog box is displayed. The following image shows a dialog box being displayed in an Office application.



Custom functions

Custom functions enable developers to add new functions to Excel by defining those functions in JavaScript as part of an add-in. Users within Excel can access custom functions just as they would any native function in Excel, such as SUM().

The following image shows a custom function called **SPHEREVOLUME** being entered in Excel.



The following code sample shows the JavaScript code for the **SPHEREVOLUME()** function shown previously:

```
/**  
 * Returns the volume of a sphere.  
 * @customfunction  
 * @param {number} radius  
 *  
function sphereVolume(radius) {  
    return Math.pow(radius, 3) * 4 * Math.PI / 3;  
}
```

Custom functions are available in Excel on the following platforms:

- Windows (connected to an Office 365 subscription)
- Mac (connected to an Office 365 subscription)
- Web browser

Define the custom function add-in type

If you use the **Yo Office generator**³ to create an Excel custom functions add-in project, you'll find that it creates files that control your functions, your task pane, and your add-in overall.

File	File format	Description
<code>./src/functions/functions.js</code>	JavaScript	Contains the code that defines custom functions.
<code>./src/functions/functions.ts</code>	TypeScript	Contains the code that defines custom functions.
<code>./src/functions/functions.html</code>	HTML	Provides a <script> reference to the JavaScript file that defines custom functions.

³ <https://github.com/OfficeDev/generator-office>

File	File format	Description
./manifest.xml	XML	Specifies the namespace for all custom functions within the add-in and the location of the JavaScript and HTML files that are listed previously in this table. It also lists the locations of other files your add-in might make use of, such as the task pane files and command files.

Script file

The script file (**./src/functions/functions.js** or **./src/functions/functions.ts**) contains the code that defines custom functions and comments that define the function.

The following code defines the custom function `add`. The code comments are used to generate a JSON metadata file that describes the custom function to Excel. The required `@customfunction` comment is declared first, to indicate that this is a custom function. Additionally, you'll notice two parameters are declared, `first` and `second`, which are followed by their `description` properties. Finally, a return description is given.

```
/**
 * Adds two numbers.
 * @customfunction
 * @param first First number.
 * @param second Second number.
 * @returns The sum of the two numbers.
 *
 function add(first, second) {
    return first + second;
}
```

Note that the **functions.html** file, which governs the loading of the custom functions runtime, must link to the current CDN for custom functions. Projects prepared with the current version of the Yeoman generator for Office Add-ins reference the correct CDN. If you are retrofitting a previous custom function project from March 2019 or earlier, you need to copy in the following code to the **functions.html** page:

```
<script src="https://appsforoffice.microsoft.com/lib/1.1/hosted/custom-functions-runtime.js" type="text/javascript"></script>
```

Manifest file

The XML manifest file for an add-in that defines custom functions (**./manifest.xml** in the project that the Yeoman generator for Office Add-ins creates) specifies the namespace for all custom functions within the add-in and the location of the JavaScript, JSON, and HTML files.

To configure an add-in to contain custom functions, the key settings in the manifest are as follows for Excel add-ins:

```
<OfficeApp
  ...
  ...
```

```
xsi:type="TaskPaneApp">
  ...
  <Hosts>
    <Host Name="Workbook"/>
  </Hosts>
  ...
  <VersionOverrides xmlns="http://schemas.microsoft.com/office/task-paneappversionoverrides" xsi:type="VersionOverridesV1_0">
    <Hosts>
      <Host xsi:type="Workbook">
        <AllFormFactors>
          <ExtensionPoint xsi:type="CustomFunctions">
            ...
          </ExtensionPoint>
        </AllFormFactors>
      </Host>
    </Hosts>
  ...
</VersionOverrides>
</OfficeApp>
```

Coauthoring

Excel on the web and Windows connected to an Office 365 subscription allow you to coauthor documents. This feature works with custom functions; if your workbook uses a custom function, your colleague will be prompted to load the custom functions add-in. Once you both have loaded the add-in, the custom function will share results through coauthoring.

Add-in commands

Add-in commands are UI elements that extend the Office UI and start actions in your add-in. You can use add-in commands to add a button on the ribbon or an item to a context menu. When users select an add-in command, they initiate actions such as running JavaScript code, or showing a page of the add-in in a task pane. Add-in commands help users find and use your add-in, which can help increase your add-in's adoption and reuse, and improve customer retention.

Add-in commands in Excel, Word, PowerPoint, and OneNote

You can configure an add-in so that a user can run it by selecting:

- Office application's ribbon or command overflow menu button
 - Key manifest setting: <ExtensionPoint xsi:type="PrimaryCommandSurface">
- Context menu item
 - Key manifest setting: <ExtensionPoint xsi:type="ContextMenu">

An add-in command can also open a submenu with additional commands.

Note:

Content add-ins do not currently support add-in commands.

The following image shows three add-in commands (custom buttons) added to the **Data** tab of the Excel ribbon.

The screenshot shows the Excel ribbon with the Data tab selected. In the Data tab's ribbon group, there is a red box highlighting a custom group called "OData Integration" which contains three buttons: "Get", "Save", and "Compare". Below the ribbon, a data table is displayed with columns D through P. A context menu is open over a specific cell in the table, showing options like "Row with Differences", "Row 5", "Column", "Excel", "OData", "Price", "12", and "19.9".

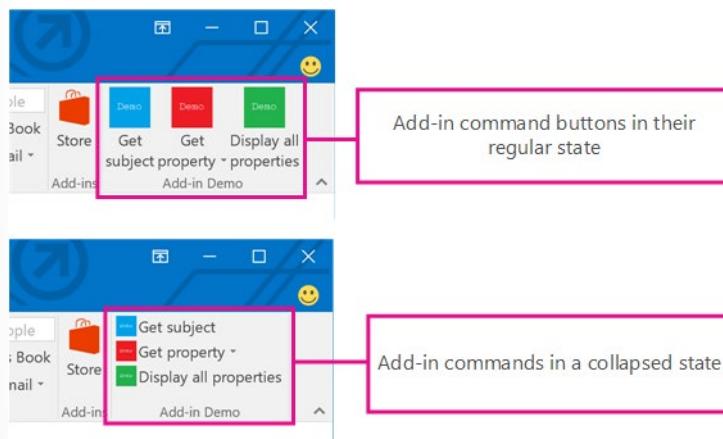
Add-in commands in Outlook

You can configure an add-in so that a user can run it by selecting a button in the Office ribbon or command overflow menu when the user is:

- Reading a message in the reading pane or in a pop-out window.
 - Key manifest setting: <ExtensionPoint xsi:type="MessageReadCommandSurface">
- Composing a message.
 - Key manifest setting: <ExtensionPoint xsi:type="MessageComposeCommandSurface">
- Creating or viewing an appointment or meeting as the organizer.
 - Key manifest setting: <ExtensionPoint xsi:type="AppointmentOrganizerCommandSurface">
- Viewing a meeting as an attendee.
 - Key manifest setting: <ExtensionPoint xsi:type="AppointmentAttendeeCommandSurface">

An add-in command can also open a submenu with additional commands.

The following images show three add-in commands (custom buttons) added to the ribbon in Outlook. In the first image, the buttons are rendered in a regular state; in the second image, the buttons are rendered in a collapsed state.



Where to use add-in commands

As shown in the following table, add-in commands are available in Excel, Outlook, OneNote, PowerPoint, and Word.

Platform	Major Office version	Subscription or one-time purchase?	Notes
Windows	Not applicable	Connected to Office 365 subscription	Not available in One-Note
	2019	One-time purchase	Not available in One-Note
	2016	One-time purchase	Only available in Outlook and Exchange 2016 (requires post-release update) or later. Not available in other Office applications.
	2013	One-time purchase	Only available in Outlook for Exchange 2016 or later. Requires post-release updates for Outlook and Exchange 2016. Not available in other Office applications.
Mac	Not applicable	Connected to Office 365 subscription	Not available in One-Note
	2019	One-time purchase	Not available in One-Note
	2016	One-time purchase	Not available in One-Note
iOS	Not applicable	Connected to Office 365 subscription	Only available in Outlook

Platform	Major Office version	Subscription or one-time purchase?	Notes
Android	Not applicable	Connected to Office 365 subscription	Only available in Outlook
web browser	Not applicable	Not applicable	Available in all supported Office applications

Lesson review questions

1. Which of the following statements is true about Office Add-ins?
 - (A) Add-ins can run only in Office on Windows, Mac, and iPad/iOS.
 - (B) Office 365 administrators can use centralized deployment to deploy add-ins across their organization.
 - (C) A developer publishes their add-in to AppSource to hide it from the general public.
2. The user needs to sign in to access add-in functionality within an Office application. Which Office Add-in platform feature should the developer display to facilitate the sign-in process?
 - (A) Add-in command
 - (B) Custom function
 - (C) Dialog box
3. What file defines the settings and capabilities of an Office Add-in?
 - (A) Dialog file
 - (B) Manifest file
 - (C) web.config file

Correct/suggested answers:

1. (B) A key benefit of Office Add-ins is centralized deployment and distribution: Office 365 administrators can deploy add-ins across their organization.
2. (C) The Office Add-ins platform enables you to display a dialog box for your users to sign in to an integrated service (for example, authenticate with Microsoft Account, Google, or Facebook).
3. (B) An Office Add-in's XML manifest file defines the settings and capabilities of the add-in. You can configure it to control how your add-in is rendered and behaves in the targeted Office applications.

Office JS APIs

Lesson introduction

In your add-in, you can access many built-in properties and features of an Office application using the Microsoft Office JavaScript APIs. You can also use various developer tools to explore the APIs and quickly set up your add-in project.

In this lesson, you'll learn about Office JavaScript APIs, including:

- Microsoft Office Add-ins programming model
- Developer tools:
 - Yeoman generator for Office Add-ins
 - Microsoft Visual Studio
 - Script Lab
 - Manifest validation
- Key API capabilities of:
 - Excel
 - Outlook
 - Word
 - Custom functions

After this lesson, you will know how to:

- Explain the Office Add-in programming model.
- Discuss Office add-in developer tools.
- Describe Excel JavaScript API.
- Describe Outlook JavaScript API.
- Describe Word JavaScript API.
- Describe PowerPoint JavaScript API.
- Describe the capabilities of custom functions.

Office Add-in programming model

The Office Add-in programming model relies on two JavaScript object models:

- Host-specific JavaScript API - Host-specific APIs for Excel and Word provide strongly typed objects that you can use to access specific elements in the host application. For example, the Excel API contains objects that represent worksheets, ranges, tables, charts, and more.
- Common API - Introduced with Office 2013, the Common API enables you to access features such as:
 - UI
 - Dialog boxes
 - Client settings that are common across multiple types of Office applications

Office JavaScript API requirement sets

You may not always have control over the version of Office your users have installed. Depending on the version of Office and platform that your add-in runs on, there are different APIs and features that are supported. For example, Office 2016 supports more capabilities than Office 2013. To handle this situation, requirement sets are provided to help you determine whether an Office host supports the capabilities you need in your Office Add-in.

Requirement sets can be specific to Office hosts, such as an ExcelApi 1.7 set, or common to multiple hosts, such as the Dialog API. Requirement set support varies by Office host and host version.

It's possible to programmatically check if requirement sets are supported by your add-in's Office host, using `isSetSupported`. If the requirement set is supported, `isSetSupported` returns `true` and your add-in can run the additional code that uses the API members from that requirement set. If the Office host doesn't support the requirement set, `isSetSupported` returns `false` and the additional code won't run. The following code shows the syntax to use with `isSetSupported`.

```
if (Office.context.requirements.isSetSupported(RequirementSetName, MinimumVersion))
{
    // Code that uses API members from RequirementSetName.
}
```

Note:

To get the earliest or monthly changes to any of the Office hosts, you can join the Office Insider's program. This program is for Windows PC only and requires an Office 365 subscription. From any Office application, select **File > Account > Office Insider** to quickly join the program.

Use Office JavaScript APIs

To use these APIs, reference them on the Office.js content delivery network (CDN), typically by adding one of the following code statements to your page's `<head>` tag.

```
// Reference the production APIs on the CDN
<script src="https://appsforoffice.microsoft.com/lib/1/hosted/office.js"
type="text/javascript"></script>

// Reference the beta/preview APIs on the CDN
<script src="https://appsforoffice.microsoft.com/lib/beta/hosted/office.js"
type="text/javascript"></script>
```

Along with adding your preferred CDN link, all Office Add-ins require an `Office.onReady()` call. You put your add-in code in this method, and it gets called once the Office.js library has initialized. Inside the `onReady()` method, you can determine which host your add-in is running in by checking the `Office.HostType` enum value (for example, Excel or Word). You can check which platform your add-in is running on with an `Office.PlatformType` enum value (for example, PC or Mac).

If you're using additional JavaScript frameworks that include their own initialization handler or tests, they should be placed within the response to `Office.onReady()`. For example, you would reference JQuery's `$(document).ready()` function as shown in the following code example.

```
Office.onReady(function() {
    // Office is ready.
```

```
$(document).ready(function () {
    // The document is ready.
});
});
```

Make asynchronous calls using proxy objects

When you work with a document, requested read or write actions are batched up using a proxy object. Your API calls don't actually read or update the underlying document until you call the `sync()` method.

For better security, your add-in runs in a sandboxed JavaScript environment that cannot directly access the document, or other add-ins. The Office Add-ins platform provides a `RequestContext` object that, in turn, provides proxy objects to represent the document (such as worksheets, ranges, and tables). The `RequestContext` is typically passed to your code as a parameter named `context`. You can use the `context` object to get any proxy objects you need to work with the document.

Before you can read the properties of a proxy object, you must load the properties to populate the proxy object with data from the Office document. You do this by calling the `load()` method on the proxy object for any properties you need. Then call the `context.sync()` method, which will load all of the requested properties. For example, if you create a proxy range object to work with a user-selected range in an Excel worksheet, and then want to read the selected range's address property, you need to load the address property before you can read it. To request properties of a proxy object to be loaded, call the `load()` method on the object and specify the properties to load.

The following example shows a function that defines a local proxy object (`selectedRange`), loads the address property of that object, and then calls `context.sync()`. When the promise from `context.sync()` is resolved, the code can then read the address property. `Excel.run` is required for this specific host, to load properties that affect platform behavior when the function runs. Not all hosts contain a `run` call.

```
Excel.run(function (context) {
    var selectedRange = context.workbook.getSelectedRange();
    selectedRange.load('address');
    return context.sync()
        .then(function () {
            console.log('The selected range is: ' + selectedRange.address);
        });
}).catch(function (error) {
    console.log('error: ' + error);
    if (error instanceof OfficeExtension.Error) {
        console.log('Debug info: ' + JSON.stringify(error.debugInfo));
    }
});
```

Office Add-in developer tools

You can use Office Add-ins developer tools to create an Office Add-in, explore Office JavaScript APIs, and validate an Office Add-in manifest file. Following are common tools for developing office add-ins:

- Yeoman generator for Office Add-ins
- Visual Studio

- Script Lab
- Manifest validator

Create an Office Add-in

You can create an Office Add-in by using the Yeoman generator for Office Add-ins or Visual Studio.

Yeoman generator for Office Add-ins

The Yeoman generator for Office Add-ins can be used to create a Node.js Office Add-in project that can be managed with Visual Studio Code or any other editor. The generator can create Office Add-ins for:

- Excel
- OneNote
- Outlook
- PowerPoint
- Project
- Word
- Excel custom functions

You have the choice to create the project using HTML, CSS and JavaScript, or using Angular or React. Typescript are options as well.

To create an Office Add-in project with the Yeoman generator, complete the following.

- To globally install Yeoman and the Yeoman generator for Office Add-ins using `npm`, the Node package manager, run the following command: `npm install -g yo generator-office`
- To create an add-in project using the Yeoman generator, run the following command: `yo office`

Visual Studio

Visual Studio can be used to create Office Add-ins for Excel, Word, PowerPoint, or Outlook. An Office Add-in project gets created as part of a Visual Studio solution, which means you can use Visual Studio features like selecting **Start** or choosing **F5** to automatically run your add-in locally on IIS. Office Add-in projects that you create with Visual Studio use HTML, CSS, and JavaScript.

To create an Office Add-in with Visual Studio, create a new C# or Visual Basic project and choose one of the following project types:

- Excel Web Add-in
- Outlook Web Add-in
- PowerPoint Web Add-in
- Word Web Add-in

Script Lab

Script Lab is an add-in that enables you to run Office JavaScript snippets while you're working in an Office program such as Excel or Word. It's available for free via AppSource and is a useful tool to include in your development toolkit as you prototype and verify the functionality you want in your add-in. In

Script Lab, you can access a library of built-in samples to quickly try out APIs or even use a sample as the starting point for your own code. Script Lab offers a number of features to help you explore the Office JavaScript API and prototype add-in functionality.

- **Explore samples :** You can run the samples to instantly see the result in the task pane or document, examine the samples to learn how the API works, and even use samples to prototype your own add-in.****
- **Code and style :** You can customize the HTML markup and CSS to experiment with element placement and styling as you prototype task pane design for your own add-in.****
- **Save and share snippets :** By default, snippets that you open in Script Lab will be saved to your browser cache. To save a snippet permanently, you can export it to a **GitHub gist⁴**. Create a secret gist to save a snippet exclusively for your own use, or create a public gist if you plan to share it with others.****
- **Import snippets :** You can import a snippet into Script Lab either by specifying the URL to the public **GitHub gist⁵** where the snippet YAML is stored or by pasting in the complete YAML for the snippet.****

Office Add-in's manifest validator

The Office Add-ins manifest validator examines your add-in's manifest file to determine if it's correct and complete. If you created your add-in project using the Yeoman generator for Office Add-ins (version 1.1.17 or later), you can validate the manifest by running the following command in the root directory of the project: `npm run validate`

If you didn't use the Yeoman generator to create your add-in project, you can validate your add-in's manifest by completing the following steps:

1. Install Node.js
2. Run the following command in the root directory of your project. Replace **MANIFEST_FILE** with the name of your manifest file: `npx office-addin-manifest validate MANIFEST_FILE`

Excel JavaScript API

The Excel JavaScript APIs give your add-ins access to Excel workbooks. An Excel add-in can edit data, format cells, create visualizations, and control the performance of the workbook.

Object model

To understand the Excel APIs, you must see how the components of a workbook are related to one another.

- A **Workbook** contains one or more **Worksheets**.
- A **Worksheet** gives access to cells through **Range** objects.
- A **Range** represents a group of contiguous cells.
- **Ranges** are used to create and place **Tables**, **Charts**, **Shapes**, and other data visualization or organization objects.
- A **Worksheet** contains collections of those data objects that are present in the individual sheet.

⁴ <https://gist.github.com/>

⁵ <https://gist.github.com/>

- **Workbooks** contain collections of some of those data objects (such as **Tables**) for the entire **Workbook**.

Ranges

A range is a group of contiguous cells in the workbook. The Excel JavaScript APIs typically use A1-style notation (for example, B3 for the single cell in row B and column 3 or C2:F4 for the cells from rows C through F and columns 2 through 4) to define ranges.

Ranges have three core properties: values, formulas, and format. They get or set the cell values, formulas to be evaluated, and the visual formatting of the cells.

Range sample

The following code sample shows how to create sales records. It uses Range objects to set the values, formulas, and formats.

```
// A function to create sales records.
Excel.run(function (context) {
    // Get the active worksheet.
    var sheet = context.workbook.worksheets.getActiveWorksheet();
    // Create the headers and format them to stand out.
    var headers = [
        ["Product", "Quantity", "Unit Price", "Totals"]
    ];
    var headerRange = sheet.getRange("B2:E2");
    headerRange.values = headers;
    headerRange.format.fill.color = "#4472C4";
    headerRange.format.font.color = "white";
    // Create the product data rows.
    var productData = [
        ["Almonds", 6, 7.5],
        ["Coffee", 20, 34.5],
        ["Chocolate", 10, 9.56],
    ];
    var dataRange = sheet.getRange("B3:D5");
    dataRange.values = productData;
    // Create the formulas to total the amounts sold.
    var totalFormulas = [
        ["=C3 * D3"],
        ["=C4 * D4"],
        ["=C5 * D5"],
        ["=SUM(E3:E5)"]
    ];
    var totalRange = sheet.getRange("E3:E6");
    totalRange.formulas = totalFormulas;
    totalRange.format.font.bold = true;
    // Display the totals as US dollar amounts.
    totalRange.numberFormat = [[ "$0.00" ]];
    // Send the calls to the workbook and wait for it to be updated.
    return context.sync();
});
```

```
});
```

A screenshot of an Excel spreadsheet. The table has columns labeled A through E. Row 1 contains headers: Product, Quantity, Unit Price, and Totals. Rows 2, 3, 4, and 5 contain data for Almonds, Coffee, and Chocolate respectively. Row 6 is a total row.

A	B	C	D	E
1	Product	Quantity	Unit Price	Totals
3	Almonds	6	7.5	\$45.00
4	Coffee	20	34.5	\$690.00
5	Chocolate	10	9.56	\$95.60
6				\$830.60

Charts, tables, and other data objects

The Excel JavaScript APIs let your add-in create and manipulate the data tools within Excel. Tables and charts are two of the more commonly used objects, but the APIs support PivotTables, Shapes, Images, and other such constructs.

Create a table

You can create a table using a data-filled range. The APIs layer the table controls (such as filters) and formatting on top of the range.

The following code sample creates a table using the ranges from the previous sample.

```
var productTable = sheet.tables.add("B2:E5", true);
```

A screenshot of an Excel spreadsheet showing the same table as above, but with filter arrows on each header cell (B2, C2, D2, E2). This indicates that the table object has been created and assigned to the variable productTable.

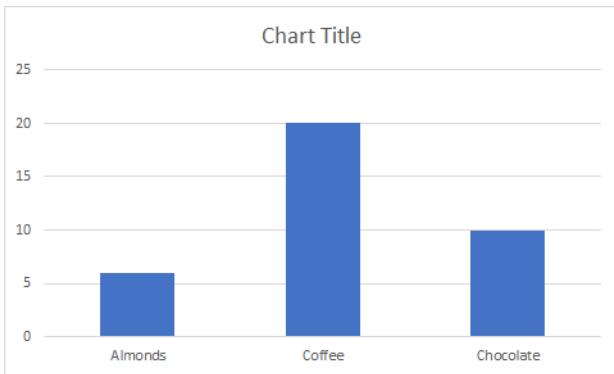
A	B	C	D	E
1	Product	Quantity	Unit Price	Totals
3	Almonds	6	7.5	\$45.00
4	Coffee	20	34.5	\$690.00
5	Chocolate	10	9.56	\$95.60
6				\$830.60

Create a chart

You can create a chart to visualize the data in a range. The Excel APIs offer dozens of chart varieties that can be customized to suit your needs.

The following example creates a simple column chart for three items.

```
var productBarChart = sheet.charts.add(Excel.ChartType.columnStacked,  
sheet.getRange("B3:C5"));
```



Outlook JavaScript API

The Outlook JavaScript APIs give your add-ins access to the user's mailbox, messages, and appointments in Outlook. An Outlook add-in can get the content and properties of a message or appointment. An add-in can also manage the content, formatting, and structure of a message or appointment that is being composed.

Object model

To understand the Outlook APIs, first see how the main components of a mailbox relate to each other.

- The **Mailbox** object enables you to handle authentication, manage a selected item, and view user profile details.
- The **Item** object represents the selected message or appointment that the user is reading or composing.

Using the Outlook APIs, you can manage many properties of an email or appointment. Many of the APIs are organized around the mode the user is in. The following table maps item types and modes.

Item type	Modes
Message	Read/Compose
Appointment/meeting	Attendee (read)/Organizer (compose)

Key features

In addition to task pane add-ins, you can create contextual and module add-ins. Following are few key features of Outlook APIs:

- Authentication options
- Contextual add-ins
- Module add-ins

Authentication

Your Outlook add-in can access information from anywhere on the internet. A few examples include the server that hosts the add-in, your internal network, or elsewhere in the cloud. If that information is protected, your add-in needs a way to authenticate your user. Outlook add-ins provide a number of different methods to authenticate, depending on your specific scenario.

Exchange user identity token

Exchange user identity tokens provide a way for your add-in to establish the identity of the user. By verifying the user's identity, you can authenticate a user into your system once, then accept the user identity token as an authorization for future requests. Consider using user identity tokens if your add-in is used primarily by Exchange on-premises users or needs access to a non-Microsoft service that you control. Your add-in can call `getUserIdentityTokenAsync` to get Exchange user identity tokens.

Access tokens obtained via OAuth2 flows

Add-ins can also access third-party services that support OAuth2 for authorization. Consider using OAuth2 tokens if your add-in needs access to a third-party service outside of your control. For example, using this method, your add-in prompts the user to sign in to the service by using the `displayDialogAsync` method to initialize the OAuth2 flow.

Callback tokens

Callback tokens provide your add-in access to the user's mailbox from your server, either using Exchange Web Services (EWS), or the Outlook REST API. Add-ins obtain callback tokens using one of the `getCallbackTokenAsync` methods. The level of access is controlled by the permissions specified in the add-in manifest.

Authentication summary

The following table summarizes when you should use each type of access token.

Access token	Use if your add-in...
Exchange user identity tokens	Is used primarily by Exchange on-premises users. Needs access to a non-Microsoft service that you control.
OAuth2 access tokens	Needs access to a third-party service outside of your control.
Callback tokens	Needs access to the user's mailbox from your server.

Contextual add-ins

Contextual add-ins are Outlook add-ins that activate based on text in a message or appointment. You may have seen the default contextual add-ins in Outlook, such as Bing Maps or Suggested Meetings. Using contextual add-ins, a user can initiate tasks related to a message without leaving the message itself, which results in an easier and richer user experience.

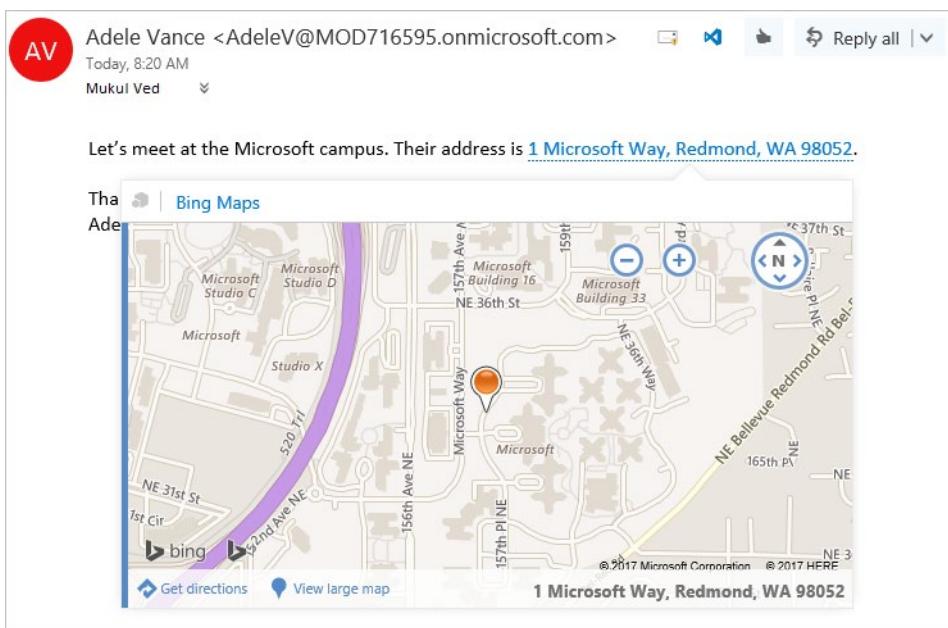
The following table lists a few example tasks based on a user's selection.

User chooses...	Contextual add-in action
Address	Open a map of the location.
String	Open a meeting suggestion add-in.
Phone number	Add number to your contacts.

Note:

At this time, contextual add-ins are not currently available in Outlook on Android and iOS.

The following image shows a contextual add-in displayed in Outlook.



A contextual add-in's manifest must include an `ExtensionPoint` element with an `xsi:type` attribute set to `DetectedEntity`. Within the `ExtensionPoint` element, the add-in specifies the entities or regular expression that can activate it. If an entity is specified, the entity can be any of the properties in the `Entities` object.

As such, the add-in manifest must contain a rule of type `ItemHasKnownEntity` or `ItemHasRegularExpressionMatch`. The following example shows how to specify that an add-in should activate on messages when it detects a phone number.

```
<ExtensionPoint xsi:type="DetectedEntity">
  <Label resid="contextLabel" />
  <SourceLocation resid="detectedEntityURL" />
  <Rule xsi:type="RuleCollection" Mode="And">
    <Rule xsi:type="ItemIs" ItemType="Message" />
    <Rule xsi:type="ItemHasKnownEntity" EntityType="PhoneNumber" High-
      light="all" />
  </Rule>
</ExtensionPoint>
```

After a contextual add-in is associated with an account, it will automatically start when the user selects a highlighted entity or regular expression.

A user launches a contextual add-in through text, either a known entity or a developer's regular expression. Typically, a user identifies a contextual add-in because the entity is highlighted.

Module add-ins

Module add-ins appear in the Outlook navigation bar, right alongside mail, tasks, and calendars. You can use all of the features of the Outlook JavaScript API in your add-in, and create command buttons in the Outlook ribbon so the user can interact with the add-in content.

Note:

Module add-ins are only supported in Outlook 2016 or later on Windows.

Meetings

Subject	Attendees	Hours	Total
Review NDA language	Kristy Woodar; Charlene Potter	4	508
Working meeting on contract with supplier	Norris Frasier; Harlan Lundgren	6	762
Review of pending actions	Mark Cazares; Ruth Dalton; Allison Beasley	4	508

Email

Subject	Recipients	Hours	Total
Product approval overview	Norris Manzo; Susanna Benson; Randy Felts	2	254
Executive approval of product legal requirements	Allison Horne; Susana Franklin	1	127
NDA approval for Level 4 and below	Sonya Hogan; Jodie Norton	3	381

Tasks

Action	Hours	Total
File patents from new product	6	762
Prepare pending action documents for meeting	2	254
Status update for corporate board	8	1016

Mail Calendar Tasks Billable Hours ...

To make a module add-in, include the module extension point in your add-in's manifest file. The following example shows a snippet of a manifest file adjusted to define a module extension.

```
<!-- Add Outlook module extension point. -->
<VersionOverrides xmlns="http://schemas.microsoft.com/office/mailappversionoverrides">
    xsi:type="VersionOverridesV1_0">
        <VersionOverrides xmlns="http://schemas.microsoft.com/office/mailappversionoverrides/1.1">
            xsi:type="VersionOverridesV1_1">
                <!-- Begin override of existing elements. -->
                <Description resid="residVersionOverrideDesc" />
                <Requirements>
                    <bt:Sets DefaultMinVersion="1.3">
                        <bt:Set Name="Mailbox" />
                    </bt:Sets>
                </Requirements>
                <!-- End override of existing elements. -->
                <Hosts>
                    <Host xsi:type="MailHost">
                        <DesktopFormFactor>
                            <!-- Set the URL of the file that contains the
                                JavaScript function that controls the extension. -->
```

```
<FunctionFile resid="residFunctionFileUrl" />
<!-- Module extension point for a ModuleApp -->
<ExtensionPoint xsi:type="Module">
  <SourceLocation resid="residExtensionPointUrl" />
  <Label resid="residExtensionPointLabel" />
  <CommandSurface>
    <CustomTab id="idTab">
      <Group id="idGroup">
        <Label resid="residGroupLabel" />
        <Control xsi:type="Button" id="group.changeToAssociate">
          <Label resid="residChangeToAssociateLabel" />
          <Supertip>
            <Title resid="residChangeToAssociateLabel" />
            <Description resid="residChangeToAssociateDesc" />
          </Supertip>
          <Icon>
            <b:Image size="16" resid="residAssociateIcon16" />
            <b:Image size="32" resid="residAssociateIcon32" />
            <b:Image size="80" resid="residAssociateIcon80" />
          </Icon>
          <Action xsi:type="ExecuteFunction">
            <FunctionName>changeToAssociateRate</FunctionName>
          </Action>
        </Control>
      </Group>
      <Label resid="residCustomTabLabel" />
    </CustomTab>
  </CommandSurface>
</ExtensionPoint>
</DesktopFormFactor>
</Host>
</Hosts>
<Resources>
  ...
</Resources>
</VersionOverrides>
</VersionOverrides>
```

Word JavaScript API

The Word JavaScript APIs give your add-ins access to Word documents. A Word add-in can manage the content, formatting, and structure of a document.

Object model

To understand the Word APIs, you must see how the main components of a document relate to each other.

- A **Document** contains a **Body** and at least one **Section**.

- A **Body** can contain:
 - **Paragraph** (one or more)
 - **Table** (one or more)
 - **List** (one or more)
 - **Text**
 - Objects like images and lists
- A **Section** gives access to its body, headers, and footers.

Note:

You can apply simple formatting to an entire existing document using the Office.js APIs. However, if you wish to apply complex formatting or use rich content objects, you can use Office Open XML (OOXML) to create these effects.

Examples of capabilities in OOXML include applying drop shadows to text or pictures, coercing text boxes into shapes, and inserting Excel charts as live charts in Word documents. Because this is a more advanced skill, we will not cover this subject in its entirety but mention it for developers who are familiar with OOXML.

The following are key scenarios for Word APIs.

Work with document text

Word add-ins use the `Office.onReady()` event handler to start. If the add-in targets Word 2016 or later, it calls the `Word.run()` method to run Word JavaScript APIs. The add-in must pass a function to `Word.run()` that expects a context object to be passed as the parameter. The context object allows the add-in to interact with the Word document. You can use the context object to create any needed proxy objects for the Word document and load the proxies with any properties you wish to use. You can also program actions to run using those properties. As always, a `context.sync()` command then synchronizes the state between the proxy objects and objects in the Word document.

The following example shows code that inserts text after the body text of a Word document. For simplicity, this sample omits the `Office.onReady()` code and focuses on the code within a `Word.run()` code block.

```
// Run a batch operation against the Word JavaScript API.  
Word.run(function (context) {  
    // Create a proxy object for the document body.  
    var body = context.document.body;  
    // Queue a command to load the text property of the proxy body object.  
    body.load("text");  
    // Queue a command to insert text into the end of the Word document  
    // body.  
    body.insertText('This is text inserted after loading the body.text  
    property',  
        Word.InsertLocation.end);  
    // Synchronize the document state by executing the queued commands,  
    // and return a promise to indicate task completion.  
    return context.sync().then(function () {  
        console.log("Body contents: " + body.text);  
    });  
})
```

Search

You can use the APIs to search the document for text that meets your criteria. You can also scope your search to certain types of content, for example, paragraphs or tables.

The following code sample searches for the text “video you” and ignores punctuation. If the text is found, the matches are bolded, highlighted in yellow, and the font color set to purple.

```
// Run a batch operation against the Word object model.  
Word.run(function (context) {  
    // Queue a command to search the document and ignore punctuation.  
    var searchResults = context.document.body.search('video you', {ig-  
    norePunct: true});  
    // Queue a command to load the search results and get the font property  
    values.  
    context.load(searchResults, 'font');  
    // Synchronize the document state by executing the queued commands,  
    // and return a promise to indicate task completion.  
    return context.sync().then(function () {  
        console.log('Found count: ' + searchResults.items.length);  
        // Queue a set of commands to change the font for each found item.  
        for (var i = 0; i < searchResults.items.length; i++) {  
            searchResults.items[i].font.color = 'purple';  
            searchResults.items[i].font.highlightColor = '#FFFF00'; //  
Yellow  
            searchResults.items[i].font.bold = true;  
        }  
        // Synchronize the document state by executing the queued commands,  
        // and return a promise to indicate task completion.  
        return context.sync();  
    });  
})  
.catch(function (error) {  
    console.log('Error: ' + JSON.stringify(error));  
    if (error instanceof OfficeExtension.Error) {  
        console.log('Debug info: ' + JSON.stringify(error.debugInfo));  
    }  
});
```

PowerPoint JavaScript API

With PowerPoint add-ins, you can use familiar web technologies such as HTML, CSS, and JavaScript to build a solution that can run in PowerPoint across multiple platforms.

Object Model

A PowerPoint add-in interacts with objects in PowerPoint by using the JavaScript API for Office, which includes two JavaScript object models:

- **PowerPoint JavaScript API⁶:** The PowerPoint JavaScript API provides strongly typed objects that you can use to access objects in PowerPoint.
- **Common API's⁷:** Introduced with Office 2013, the Common API can be used to access features such as UI, dialog boxes, and client settings that are common across multiple types of Office applications.

PowerPoint add-ins

You can use PowerPoint add-ins to build engaging solutions for your users' presentations across platforms including Windows, iOS, Mac, and in a browser. You can create two types of PowerPoint add-ins:

- Use **content add-ins** to add dynamic HTML5 content to your presentations. For example, see the **LucidChart Diagrams for PowerPoint⁸** add-in, which you can use to inject an interactive diagram from LucidChart into your deck.
- Use **task pane add-ins** to bring in reference information or insert data into the presentation via a service. For example, see the **Pixton Comic Characters⁹** add-in, which you can use to add professional photos to your presentation.

PowerPoint add-in scenarios

The code examples in this article demonstrate some basic tasks for developing add-ins for PowerPoint. Please note the following:

- To display information, these examples use the `app.showNotification``` function, which is included in the Visual Studio Office Add-ins project templates. If you aren't using Visual Studio to develop your add-in, you'll need to replace the `showNotification` function with your own code.
- Several of these examples also use a `Globals` object that is declared beyond the scope of these functions as: `var Globals = {activeViewHandler:0, firstSlideId:0};`

To use these examples, your add-in project must **reference Office.js v1.1 library or later¹⁰**.

Detect the presentation's active view and handle the ActiveViewChanged event

If you are building a content add-in, you need to get the presentation's active view and handle the `ActiveViewChanged` event, as part of your `Office.Initialize` handler.

In PowerPoint on the web, the **Document.ActiveViewChanged¹¹** event will never fire because Slide Show mode is treated as a new session. In this case, the add-in must fetch the active view on load, as shown in the following code sample.

⁶ <https://docs.microsoft.com/en-us/javascript/api/powerpoint>

⁷ <https://docs.microsoft.com/en-us/javascript/api/office>

⁸ <https://appsource.microsoft.com/product/office/WA104380117>

⁹ <https://appsource.microsoft.com/product/office/WA104380907>

¹⁰ <https://docs.microsoft.com/en-us/office/dev/add-ins/develop/referencing-the-javascript-api-for-office-library-from-its-cdn>

¹¹ <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

In the following code sample:

- The `getActiveFileView` function calls the **Document.getActiveViewAsync¹²** method to return whether the presentation's current view is "edit" (any of the views in which you can edit slides, such as **Normal** or **Outline View**) or "read" (**Slide Show** or **Reading View**).
- The `registerActiveViewChanged` function calls the **addHandlerAsync¹³** method to register a handler for the **Document.ActiveViewChanged¹⁴** event.

```
// General Office.initialize function. Fires on load of the add-in.  
Office.initialize = function(){  
    // Gets whether the current view is edit or read.  
    var currentView = getActiveFileView();  
    // Register for the active view changed handler  
    registerActiveViewChanged();  
    // Render the content based off of the currentView.  
    //....  
}  
function getActiveFileView()  
{  
    Office.context.document.getActiveViewAsync(function (asyncResult) {  
        if (asyncResult.status == "failed") {  
            app.showNotification("Action failed with error: " + asyncResult.error.message);  
        }  
        else {  
            app.showNotification(asyncResult.value);  
        }  
    });  
}  
function registerActiveViewChanged() {  
    Globals.activeViewHandler = function (args) {  
        app.showNotification(JSON.stringify(args));  
    }  
    Office.context.document.addHandlerAsync(Office.EventType.ActiveViewChanged, Globals.activeViewHandler,  
        function (asyncResult) {  
            if (asyncResult.status == "failed") {  
                app.showNotification("Action failed with error: " + asyncResult.error.message);  
            }  
            else {  
                app.showNotification(asyncResult.value);  
            }  
        });  
}
```

12 <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

13 <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

14 <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

Navigate to a particular slide in the presentation

In the following code sample, the `getSelectedRange` function calls the **Document.getSelectedDataAsync**¹⁵ method to get the JSON object returned by `asyncResult.value`, which contains an array named **slides**. The **slides** array contains the ids, titles, and indexes of a selected range of slides (or of the current slide, if multiple slides are not selected). It also saves the id of the first slide in the selected range to a global variable.

```
function getSelectedRange() {
    // Get the id, title, and index of the current slide (or selected
    // slides) and store the first slide id *
    Globals.firstSlideId = 0;
    Office.context.document.getSelectedDataAsync(Office.CoercionType.SlideRange, function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.error.message);
        }
        else {
            Globals.firstSlideId = asyncResult.value.slides[0].id;
            app.showNotification(JSON.stringify(asyncResult.value));
        }
    });
}
```

In the following code sample, the `goToFirstSlide` function calls the **Document.goToByIdAsync**¹⁶ method to navigate to the first slide that was identified by the `getSelectedRange` function shown previously.

```
function goToFirstSlide() {
    Office.context.document.goToByIdAsync(Globals.firstSlideId, Office.GoToType.Slide, function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.error.message);
        }
        else {
            app.showNotification("Navigation successful");
        }
    });
}
```

Navigate between slides in the presentation

In the following code sample, the `goToSlideByIndex` function calls the **Document.goToByIdAsync** method to navigate to the next slide in the presentation.

```
function goToSlideByIndex() {
    var goToFirst = Office.Index.First;
```

¹⁵ <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

¹⁶ <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

```

        var goToLast = Office.Index.Last;
        var goToPrevious = Office.Index.Previous;
        var goToNext = Office.Index.Next;
        Office.context.document.goToByIdAsync(goToNext, Office.GoToType.Index,
    function (asyncResult) {
        if (asyncResult.status == "failed") {
            app.showNotification("Action failed with error: " + asyncResult.
error.message);
        }
        else {
            app.showNotification("Navigation successful");
        }
    });
}

```

Get the URL of the presentation

In the following code sample, the `getFileUrl` function calls the **Document.getFileProperties**¹⁷ method to get the URL of the presentation file.

```

function getFileUrl() {
    // Get the URL of the current file.
    Office.context.document.getFilePropertiesAsync(function (asyncResult) {
        var fileUrl = asyncResult.value.url;
        if (fileUrl == "") {
            app.showNotification("The file hasn't been saved yet. Save the
file and try again");
        }
        else {
            app.showNotification(fileUrl);
        }
    });
}

```

Create a presentation

Your add-in can create a new presentation, separate from the PowerPoint instance in which the add-in is currently running. The PowerPoint namespace has the `createPresentation` method for this purpose. When this method is called, the new presentation is immediately opened and displayed in a new instance of PowerPoint. Your add-in remains open and running with the previous presentation.

```
PowerPoint.createPresentation();
```

The `createPresentation` method can also create a copy of an existing presentation. The method accepts a base64-encoded string representation of an .pptx file as an optional parameter. The resulting presentation will be a copy of that file, assuming the string argument is a valid .pptx file. The **FileReader**¹⁸

¹⁷ <https://docs.microsoft.com/en-us/javascript/api/office/office.document>

¹⁸ <https://developer.mozilla.org/docs/Web/API/FileReader>

class can be used to convert a file into the required base64-encoded string, as demonstrated in the example in the next section.

```
var myFile = document.getElementById("file");
var reader = new FileReader();
reader.onload = function (event) {
    // Strip off the metadata before the base64-encoded string
    var startIndex = reader.result.toString().indexOf("base64,");
    var copyBase64 = reader.result.toString().substr(startIndex + 7);
    PowerPoint.createPresentation(copyBase64);
};
// Read in the file as a data URL so we can parse the base64-encoded string
reader.readAsDataURL(myFile.files[0]);
```

Custom functions capabilities

Custom functions have several unique capabilities and restrictions because they run in a separate runtime from other add-in interactions, like task panes.

You can create custom JavaScript or TypeScript functions that can be accessed like built-in Excel functions such as `SUM()`. You can also create streaming custom functions that return a value based on a timer. For example, you can update the current time value in a cell every second. You can make network calls from custom functions as well.

Custom function JavaScript example

The following code sample defines the custom function `add` that accepts two numbers, then returns their sum.

```
/**
 * Adds two numbers.
 * @customfunction
 * @param first First number.
 * @param second Second number.
 * @returns The sum of the two numbers.
 *
function add(first, second) {
    return first + second;
}
```

JSDoc code comment descriptions

The JSDoc tags in the code comments are used to generate a JSON metadata file that describes the custom function to Excel. The JSON metadata file enables Excel to accurately present information to a user and pass expected parameters to your custom function:

- `@customfunction`: Is declared first and indicates that it is a custom function. Required.
- `@param`: Describes the parameter. Include for each parameter defined by the function.
- `@returns`: Describes what the function outputs.

Note:

The comment description "Adds two numbers" is also added to the JSON Metadata file for Excel to display when the user is viewing your custom function.

Custom function runtime restrictions

The custom function runtime only runs JavaScript. There's no document object model (DOM) or local storage, as you would find in a browser-based JavaScript runtime environment. This means you can't load any libraries that use the DOM, such as jQuery. Also, you can't access the Office.js API to interact with the document like you can from a task pane. Instead, the custom functions runtime is optimized for tasks such as performing rapid calculations and generally doesn't need to use some of the Office.js APIs such as formatting tools in Excel.

Custom functions have a webpage that loads the custom functions runtime. Since the custom functions runtime doesn't have a UI, there's nothing for the webpage to display. You'll find the following script tag in the webpage that loads the library for the custom function runtime.

```
<script src="https://appsforoffice.microsoft.com/lib/1.1/hosted/custom-functions-runtime.js" type="text/javascript"></script>
```

Typically, custom functions are combined with a task pane in the same add-in. If you create your add-in project using the Yeoman generator for Office Add-ins, the project will have a webpage for the custom functions, and a web page with UI for the task pane.

Using storage API to communicate with the task pane

Custom function code and task pane code (which uses Office.js) can't call or communicate directly with each other. But you can use a storage API that allows them to share data. A common scenario for using the storage API is when the add-in needs to share a security token for accessing a secure network resource. The user might first call a custom function that requires them to be signed in. After authentication, it receives the security token. Then it shares the security token using the storage API so that later, when the user opens the task pane, the task pane doesn't need to sign them in again.

Alternatively, the user might open the task pane first. In this case, the task pane will sign in the user and share the security token through the storage API. When a custom function is used later, the custom function can get the security token through the storage API.

Storage API example

The following code sample shows how to store and retrieve any value created by the user.

```
/**  
 * @customfunction  
 * @description Stores a value in OfficeRuntime.storage.  
 * @param {any} key Key in the key-value pair you will store.  
 * @param {any} value Value in the key-value pair you will store.  
 *  
 function storeValue(key, value) {  
     return OfficeRuntime.storage.setItem(key, value).then(function (result) {  
         return "Success: Item with key '" + key + "' saved to storage."  
     }, function (error) {  
         return "Error: Unable to save item with key '" + key + "' to storage.  
" + error;  
}
```

```
    } );
}

/**
 * @customfunction
 * @description Gets value from OfficeRuntime.storage.
 * @param {any} key Key of item you intend to get.
 *
function getValue(key) {
    return OfficeRuntime.storage.getItem(key);
}
```

Dialog API

Custom functions have their own Dialog API since they can't access the Office.js API. However, the functionality is similar. The most common scenario is to launch a dialog box to sign in a user and receive a security token.

The following code sample shows how to display a web dialog box from a custom function.

```
OfficeRuntime.displayWebDialog('https://myDomain/myDialog.html', {height: 30,
width: 20});
```

Creating a custom functions project

You can create a custom functions project by using the Yeoman generator for Office Add-ins. Run yo office to start the generator, then choose the **Excel Custom Functions Add-in project** option. Once created, your project will contain a **/src/taskpane/** folder for the task pane source files, and a **/src/functions** folder for the custom function source files.

Note:

You cannot create a custom functions project in Visual Studio.

Lesson review questions

1. Which of the following statements is true about Office JavaScript APIs?
 - (A) Requirement sets can only be specific to Office hosts.
 - (B) The URL to the production APIs on the Office JavaScript CDN is https://appsforoffice.microsoft.com/lib/beta/hosted/office.js.
 - (C) Properties of a proxy object must first be loaded with data from the Office file.
2. A developer wants to use an add-in to visualize data in Excel. What would be a good object to use?
 - (A) Chart
 - (B) Range
 - (C) Value

Correct/suggested answers:

1. (B) The link to the production APIs on the Office JavaScript CDN is https://appsforoffice.microsoft.com/lib/beta/hosted/office.js.

2. You can create a chart to visualize the data in a range. The Excel APIs offer dozens of chart varieties that can be customized to fit your needs.

Customize Office Add-ins

Lesson introduction

You can customize your Office Add-ins and enable your users to customize their experience. In this lesson, you'll learn about customizing Office Add-ins.

After this lesson, you will know how to:

- Describe the options of persisting state and settings.
- Describe Office UI Fabric in Office Add-ins.
- Describe when to use Microsoft Graph in Office Add-ins.
- Describe authentication and authorization in Office Add-ins.

Overview of customizing Office Add-ins

Office Add-ins extend the Office experience by providing contextual functionality that users can access within Office clients. Add-ins empower users to get more done by enabling them to access third-party functionality within Office, without costly context switches.

Your add-in UX design must integrate seamlessly with Office to provide an efficient, natural interaction for your users. Take advantage of **add-in commands**¹⁹ to provide access to your add-in and apply the best practices that we recommend when you create custom HTML-based UI.

Office applications follow a general set of interaction guidelines. The apps share content and have elements that look and behave similarly. This commonality is built on a set of design principles. The principles help the Office team create interfaces that support customers' tasks. Understanding and following them will help you support your customers' goals inside of Office.

Add-in design guidelines

You should consider many factors as you design your Add-in. The Office design principles are used by Office application designers and recommended for add-in design as well:

- Design explicitly for Office.
- Focus on a few key tasks; do them well.
- Favor content over chrome.
- Make it enjoyable and keep users in control.
- Design for all platforms and input methods.

When you design the UI, strive to balance:

- Office branding
- Your own organization's branding
- The look of the target platform (for example, Windows, Mac)

¹⁹ <https://docs.microsoft.com/en-us/office/dev/add-ins/design/add-in-commands>

Persisting add-in state and settings

Office Add-ins are essentially web applications running in the stateless environment of a browser control. As a result, your add-in may need to persist data to maintain the continuity of certain operations or features across sessions of using your add-in. The Office Add-ins platform provides several ways for your add-in to persist state and settings. Your options depend on the Office applications you plan to support and on the type of add-in you plan to develop.

For example, your add-in may have custom settings or other values that it needs to save and reload the next time it's initialized, such as a user's preferred view or default location.

Options to persist state and settings

The Office JavaScript API provides objects for your add-in to save state across user sessions. The following table lists the options, along with the supported add-in types and Office host applications.

Office API object	Supported add-in types	Supported Office hosts	Storage information
CustomProperties	MailApp	Outlook	Item data is stored on the message, meeting request, or appointment the add-in is working on. Data is stored in name/value pairs in a property bag.
CustomXmlParts	TaskPaneApp	Excel (host-specific Excel JavaScript API), Word (Office JavaScript Common API)	Data is stored in a custom XML part of the document or workbook.
RoamingSettings	MailApp	Outlook	Data is stored on the user's Exchange mailbox and associated with the specific add-in. Data is stored in name/value pairs in a property bag.
Settings	ContentApp, TaskPane-App	Excel, PowerPoint, Word	Data is stored on the document, workbook, or presentation the add-in is working on. Data is stored in name/value pairs in a property bag.

Important:

Don't store passwords and sensitive personally identifiable information (PII) on the user's device.

Internally, the data in the property bag accessed with the **Settings**, **CustomProperties**, or **RoamingSettings** objects is stored as a serialized JavaScript Object Notation (JSON) object that contains name/value pairs. The name (key) for each value must be a string, and the stored value can be a JavaScript string, number, date, or object, but not a function.

This example of the property bag structure contains three defined string values named **firstName**, **location**, and **defaultView**.

```
{  
    "firstName": "Erik",  
    "location": "98052",  
    "defaultView": "basic"  
}
```

After the `settings` property bag is saved during the previous add-in session, it can be loaded when the add-in is initialized or at any point after that during the add-in's current session. During the session, the settings are managed entirely in memory using the `get`, `set`, and `remove` methods of the object that corresponds to the kind settings you are creating (**Settings**, **CustomProperties**, or **RoamingSettings**).

You can also use HTML5 web storage and other techniques available through the add-in's underlying browser control.

Office UI Fabric in Office Add-ins

As you build your add-in, you have many UI design factors to consider. Office UI Fabric is a JavaScript front-end framework for building user experiences for Office and Office 365. Fabric provides visually-focused components that developers can extend, rework, and use in their Office Add-in. Although using Office UI Fabric is optional, because Fabric uses the Office Design Language, Fabric's UX components look like a natural extension of Office.

About UI Fabric

Office UI Fabric has two main areas:

- **Fabric Core:** Provides basic elements like font, icons, and color
- **Fabric React components:** Includes Fabric Core elements and adds input, navigation, and notification components, among others

Fabric Core

Fabric Core contains basic elements of the design language such as icons, colors, type, and grid. Fabric core is framework independent. Fabric Core is used by and included with Fabric React.

To start using Fabric Core, reference the CSS in your HTML page, as shown in the following code.

```
<link rel="stylesheet" href="https://static2.sharepointonline.com/files/fabric/office-ui-fabric-core/9.6.1/css/fabric.min.css">
```

You can then use Fabric icons, fonts, and colors. To use a Fabric icon, include the `i` element on your page, and then reference the appropriate classes. You can control the size of the icon by changing the font size. The following example shows how you can include an extra-large table icon in the Office application's primary theme color.

```
<i class="ms-Icon ms-font-xl ms-Icon--Table ms-fontColor-themePrimary"></i>
```

Fabric components

Fabric React provides UX components for input, navigation, notification, and other categories. It builds on and includes Fabric Core.

Recommended components you can use in your add-in are:

- Breadcrumb
- Button
- Checkbox
- ChoiceGroup
- Dropdown
- Label
- List
- Pivot
- TextField
- Toggle

You can use different JavaScript frameworks, such as Angular or React, to build your add-in.

Tip:

If you use the Yeoman generator for Office Add-ins to create a project that references Fabric React, the available project type is Office Add-in Task Pane project using React framework.

Microsoft Graph in Office Add-ins

Integrating data from online service providers increases the value and adoption of your add-ins. Your add-in can connect to Microsoft Graph and access the user's data so they can accomplish more useful and productive scenarios. Example tasks are:

- Read files from OneDrive
- Fetch email attachments
- Get the user's profile

For example, you can build a Microsoft Office Add-in that connects to Microsoft Graph, finds the first three workbooks stored in OneDrive for Business, fetches their filenames, and inserts the names into an Office document using Office.js. You can also build a Microsoft Office Add-in that connects to Microsoft Graph, finds all workbooks stored in OneDrive for Business, fetches all charts in the workbooks using the Excel REST APIs, and inserts an image of a chart into a PowerPoint slide using Office.js.

Microsoft Graph REST APIs provide a way for your add-in to access the user's data in services like:

- Azure Active Directory
- Office 365 services
- Enterprise Mobility and Security services
- Windows 10 services
- Dynamics 365

Authentication and authorization in Office Add-ins

Web applications and therefore, Office Add-ins, allow anonymous access by default, but you can require users to authenticate with a login. For example, you can require that your users be logged in with a

personal Microsoft account, an Office 365 work or school account, or other common account. This task is called user authentication, because it enables the add-in to know who the user is.

Your add-in can also get the user's consent to access their Microsoft Graph data (such as their Office 365 profile, OneDrive files, and SharePoint data) or to access data in other external sources such as Google, Facebook, LinkedIn, SalesForce, and GitHub. This task is called add-in (or app) authorization, because it is the *add-in* that is being authorized, not the user.

You have a choice of two ways to accomplish these authentications:

- **Office Single Sign-on (SSO)**: A system, *currently in preview*, that enables the user's login to Office to also function as a login to the add-in. Optionally, the add-in can also use the user's Office credentials to authorize the add-in to Microsoft Graph. (Non-Microsoft sources are not accessible through this system.)
- **Web Application Authentication and Authorization with Azure Active Directory**: This isn't something new or special. It's just the way that Office add-ins (and other web apps) authenticated users and authorized apps before there was an Office SSO system, and it's still used in scenarios where Office SSO cannot be used.

User authentication without SSO

You can authenticate a user in an Office Add-in with Azure Active Directory (AAD) the same as you would any in any other web application, with one exception: AAD does not allow its login page to open in an iframe. When an Office Add-in is running on *Office on the web*, the task pane is an iframe. This means that you'll need to open the AAD login screen in a dialog box opened with the Office Dialog API. This affects how you use authentication helper libraries.

User authentication with SSO

To use SSO to authenticate the user, your code in a task pane or function file calls the `getAccessToken` method. If the user is not signed in to Office, Office will open a dialog box and navigate to the Azure Active Directory login page. After the user is signed in, or if the user is already signed in, the method returns an access token. The token is a bootstrap token in the **On Behalf Of** flow. However, it can be used as an ID token as well, because it contains several claims that are unique to the current user, including `preferred_username`, `name`, `sub`, and `oid`.

After your code has extracted the desired claim from the token, it uses that value to look up the user in a user table or user database that you maintain. Use the database to store user-relative information such as the user's preferences or the state of the user's account. Since you're using SSO, your users don't sign in separately to your add-in, so you do not need to store a password for the user.

Connect to Microsoft Graph from an Office Add-in

To connect to and use Microsoft Graph, your add-in needs to:

- Authenticate the user
- Be authorized to act on the user's behalf

Access to Microsoft Graph without SSO

You can get authorization to Microsoft Graph data for your add-in by obtaining an access token to Graph from Azure Active Directory (AAD). You can do this without relying on Office SSO.

Access to Microsoft Graph with SSO

To use SSO to get access to Microsoft Graph, your add-in in a task pane or function file calls the **getAccessToken** method. If the user is not signed into Office, Office will open a dialog box and navigate to the Azure Active Directory login page. After the user is signed in, or if the user is already signed in, the method returns an access token. The token is a bootstrap token in the **On Behalf Of** flow. Specifically, it has a scope claim with the value `access_as_user`.

After your code obtains the token, it uses it in the **On Behalf Of** flow to obtain a second token: an access token to Microsoft Graph.

The access token can never give the add-in more or greater permissions than the user has. Users typically only have permissions to data about themselves, their own files and email, and objects that have been shared with them. If your add-in gets Microsoft Graph data about multiple users, then it can be used successfully only by users with admin-level permissions.

Recommended libraries

Depending on your development choices, you can use one of the following libraries for authentication and authorization as appropriate:

- Your server-side is on a .NET-based framework (for example, .NET Core or ASP.NET): use **MSAL.NET**
- Your server-side is node.js-based: use **Passport Azure AD**
- Your add-in uses Implicit flow: use **msal.js**

Lesson review questions

1. An Outlook add-in needs to store data. Which is an Office JavaScript object that the add-in developer might use?
 - (A) CustomProperties
 - (B) CustomXmlParts
 - (C) Settings
2. What is an advantage of Using Office Fabric in an Add-in?
 - (A) Fabric has three (3) main areas.
 - (B) Fabric Core is built on Fabric React.
 - (C) Fabric reflects Office branding.

Correct/suggested answers:

1. (A) CustomProperties is the correct add-in for Outlook. Data is stored on the message or appointment that the add-in is working on. Data is stored in name/value pairs in a property bag.
2. (C) Fabric Core provides basic design elements that reflect or sync with Office branding.

Testing Debugging and Deployment Options for Office Add-ins

Lesson introduction

Understanding your deployment options and how to test during Add-in development. As a best practice is a set-up for success.

As you develop and test your add-in, you can host it locally on your machine. If you use the Yeoman generator for Office Add-ins, your add-in will be hosted using Node.js. If you use Visual Studio, your add-in will be hosted using Internet Information Services (IIS). To make your add-in accessible to other users, host it on a web server, either your own or through a service like Azure.

During your add-in development, you need to verify functionality and fix issues. Several developer tools are available to help you debug your add-in. This lesson covers deployment options, testing, and debugging your add-in.

After this lesson, you will know how to:

- Select deployment options based on requirements.
- Describe testing and debugging concepts for Office Add-ins.

Deployment options for Office Add-in

As you develop your Office Add-in and prepare to make it available to your users, you need to decide which deployment option is best. The following table lists factors you should consider.

Consider...	Examples
Add-in lifecycle stage	Local developer testing, ready for public use
Add-in interaction or feature support	Task pane add-in, content add-in, add-in commands
Target Office applications	Excel, Outlook
Target platforms	Windows, Mac
Scope of user base	Your organization, general public

Deployment options

You have several options for deploying your add-in. The following table notes each option and when it should be used.

Option	Description	Best when...
Sideload	Install your add-in locally.	Developer building and testing add-in
Centralized deployment	Distribute your add-in to users via the Microsoft 365 admin center.	Add-in ready for use in your organization on Office 365 or in a hybrid environment

Option	Description	Best when...
SharePoint catalog	Distribute add-in to users via SharePoint.	Task pane or content add-in ready for use in your organization that's using an on-premises environment; Excel, Word, or PowerPoint is targeted but Mac isn't a target platform
AppSource	Make add-in available to the public.	Add-in ready for public use
Exchange server	Distribute add-in to users via Exchange.	Outlook add-in ready for use in an organization whose environment doesn't use Azure Active Directory identity service
Network share	Make add-in available to network users via a shared folder.	Add-in development and users are on Windows

The following sections provide additional information about the deployment methods that are most commonly used to distribute Office Add-ins to users within an organization.

Centralized Deployment via the Office 365 admin center

The Office 365 admin center makes it easy for an administrator to deploy Office Add-ins to users and groups in their organization. Add-ins deployed via the admin center are available to users in their Office applications right away, with no client configuration required. You can use Centralized Deployment to deploy internal add-ins as well as add-ins provided by ISVs.

SharePoint app catalog deployment

A SharePoint app catalog is a special site collection that you can create to host Word, Excel, and PowerPoint add-ins. SharePoint catalogs don't support new add-in features implemented in the `Version-Overrides` node of the manifest, including add-in commands. Therefore, it's recommended that you use Centralized Deployment via the admin center. Add-in commands, deployed via a SharePoint catalog, open in a task pane by default.

Note:

SharePoint catalogs do not support Office on Mac. To deploy Office Add-ins to Mac clients, you must submit them to [AppSource](#).

Outlook add-in deployment

For on-premises and online environments that do not use the Azure AD identity service, you can deploy Outlook add-ins via the Exchange server.

Outlook add-in deployment requires:

- Office 365, Exchange Online, or Exchange Server 2013 or later
- Outlook 2013 or later

To assign add-ins to tenants, you use the Exchange admin center to upload a manifest directly, either from a file or a URL, or add an add-in from AppSource. To assign add-ins to individual users, you must use Exchange PowerShell.

Deployment options by Office host

The deployment options that are available depend on the Office host that you're targeting and the type of add-in you create.

Deployment options for Word, Excel, and PowerPoint add-ins

Sideload, Office 365 admin center, AppSource and SharePoint catalog are available for the content, task pane and command extension points, with the exception of SharePoint catalog which is not available for the Command extension point.

Note:

SharePoint catalogs do not support Office on Mac.

Deployment options for Outlook add-ins

Sideload, Exchange server, and AppSource are deployment options for both the Mail app and the Command extension points.

Testing and debugging for Office Add-ins

At various points during your add-in's life cycle, you need to verify functionality and fix bugs. You have several options for how you go about testing and debugging your add-in.

To work on and test your add-in, you'll start by sideloading it (that is, installing it locally) then debug to figure out why it's not behaving as expected. Test your add-in to make sure it works properly on platforms and Office application versions that your users are on.

Sideload an Office add-in

You can locally install (sideload) your add-in for testing and debugging on Windows, Mac, and in a web browser. You can also sideload your Excel or Word add-in on an iPad. Use Node.js, Internet Information Services (IIS), or another preferred means to web host your add-in on your development machine.

If you create your project using the Yeoman generator for Office Add-ins, you can run `npm run start` in a command-line prompt to start and sideload your add-in to Excel on Windows or `npm run start:web` to run it in a web browser, although you'll have to manually sideload to Excel in the browser.

If you create your project using Microsoft Visual Studio (VS), you can run the project in VS debug mode and it will automatically sideload to Excel on Windows.

Debug an Office add-in

You can debug your add-in using the following methods:

- A web browser with the browser's built-in developer tools
- Visual Studio, provided you prepared your add-in using this IDE
- Visual Studio Code for custom functions projects only
- Runtime logging on Windows and Mac

The procedure for debugging an Office Add-in varies by platform as well. If you need to debug your add-in on a specific platform, such as Windows and Mac, there are additional tools that may help you.

Windows

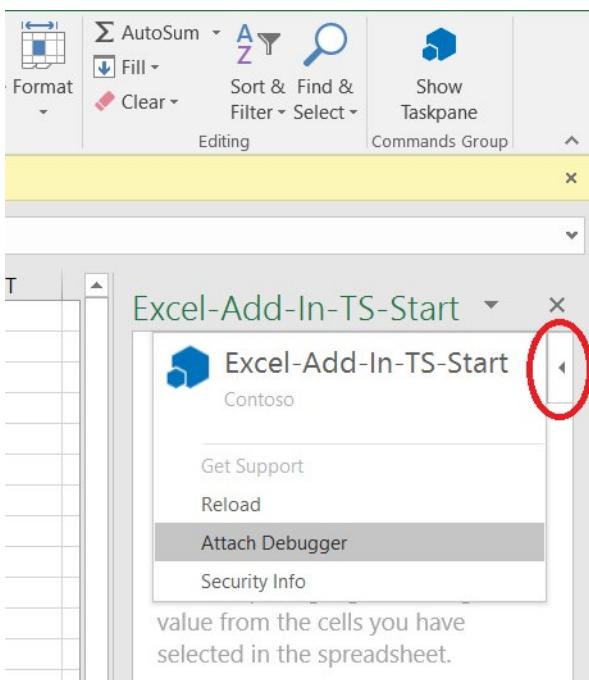
There are developer tools outside of IDEs available to help you debug your add-ins on Windows 10. These are useful when you need to investigate a problem while running your add-in outside the IDE. The tool you use depends on if the add-in is running in Microsoft Edge or Internet Explorer. Your add-in is running in Internet Explorer 11 unless it meets the following criteria to be running in Microsoft Edge:

- Windows 10 (version 1903 or later)
- Office 365 subscription (build 16.0.11629 or later)

For Microsoft Edge, install and use Microsoft Edge DevTools. For Internet Explorer, run F12 developer tools according to your Office version. You can start the F12 developer tools after the add-in is running. The F12 tools are displayed in a separate window and do not use Visual Studio.

- 32-bit Office version: **C:\Windows\System32\F12\IEChooser.exe**
- 64-bit Office version: **C:\Windows\SysWOW64\F12\IEChooser.exe**

An available option to debug task pane add-ins in Office 2016 or later is to attach a debugger. Where the **Attach Debugger** is available through the **Personality** menu. The supported tool is Visual Studio 2015 Update 3 or later. This tool only enables JavaScript debugging.



If the **Personality** menu isn't present or you're already using Visual Studio (VS), you can use **Attach to Process** in VS to debug the add-in in Microsoft Edge or Internet Explorer as applicable. The attach debugger feature will directly attach the debugger to the correct Internet Explorer process for you. You can attach a debugger regardless of whether you are using Yeoman Generator, Visual Studio Code, Node.js, Angular, or another tool.

Mac

For your sideloaded task pane and content add-ins, you can use the Safari Web Inspector on Mac OS High Sierra and Office version 16.9.1 (build 18012504) or later. The supported Office applications are:

- Excel
- Outlook
- PowerPoint
- Word

Add-ins are often cached in Office for Mac, for performance reasons. Normally, the cache is cleared by reloading the add-in. If more than one add-in exists in the same document, the process of automatically clearing the cache on reload might not be reliable. You can clear the cache by using the personality menu of any task pane add-in.

Validate the manifest file

You may want to validate your add-in's manifest file to ensure that it's correct and complete. Validation can also identify issues that are causing the error "**Your add-in manifest is not valid**" when you attempt to sideload your add-in. You can validate your add-in's manifest using any of following options:

- Yeoman generator for Office Add-ins

If you used the **Yeoman generator for Office Add-ins²⁰** to create your add-in, you can also use it to validate your project's manifest file.

- `office-addin-manifest validate` command

If you didn't use the **Yeoman generator for Office Add-ins²¹** to create your add-in, you can validate the manifest by using **office-addin-manifest²²**.

- Libxml

You can validate the manifest file against the XML Schema Definition (XSD) files. This will ensure that the manifest file follows the correct schema, including any namespaces for the elements you are using. If you copied elements from other sample manifests double check that you also **include the appropriate namespaces**. You can use an XML schema validation tool to perform this validation.

Test required Office clients and platforms

Test your add-in in Office versions and on platforms where you or your intended users will be using it.

Private use or limited to your organization

If your add-in will be limited to you or your organization, test it in Office versions and on platforms where you know they'll be used. For example, if you're developing a Word add-in for your organization where your coworkers usually work in Microsoft Edge and Word 2019 on Windows, test your add-in in that browser and version of Word.

²⁰ <https://www.npmjs.com/package/generator-office>

²¹ <https://www.npmjs.com/package/generator-office>

²² <https://www.npmjs.com/package/office-addin-manifest>

Public use

If your add-in will be available to the public through AppSource, you should look over the AppSource validation policies, so the review and validation of your add-in is as smooth as possible. A few key validation requirements are:

- **Browsers:** Internet Explorer 11 and later, Microsoft Edge, Chrome, Firefox, and Safari (Mac).
- **Office:** All applications you specified in the Hosts section of the add-in's manifest configuration file.
- **Operating systems:** Windows, Mac, and iPad. If your Outlook add-in supports mobile, include iOS and Android.

AppSource validation policy 4.12 discusses the expected client and platform support requirements in more detail.

Lesson review questions

1. Which is the best deployment option to test and debug a new Add-in?
 - (A) AppSource
 - (B) Exchange server
 - (C) Sideload
2. What are the three key areas you should validate when publishing an Add-in on AppSource?
 - (A) Browsers, Office applications, Operating systems
 - (B) Browsers, Office applications, Organizations
 - (C) Monitor resolutions, Office applications, Operating systems

Correct/suggested answers:

1. (C) To work on and test your add-in, start by sideloading it (installing it locally), then debug to figure out why it's not behaving as expected.
2. (A) If your Add-in will be available to the public through AppSource, you should look over the AppSource validation policies, so that the review and validation of your add-in is as smooth as possible. A few key validation requirements are:
 - **Browsers:** Internet Explorer 11 and later, Microsoft Edge, Chrome, Firefox, and Safari (Mac)
 - **Office:** All applications you specified in the Hosts section of the add-in's manifest configuration file
 - **Operating systems:** Windows, Mac, and iPad—if your Outlook add-in supports mobile, include iOS and Android

Actionable Messages

Lesson introduction

Whether you are filling out a survey, approving an expense report, or updating a CRM sales opportunity, actionable messages enable you to take quick actions right from within Outlook. Developers can embed actions in their emails or notifications, elevating user engagement with their services and increasing organizational productivity.

After this lesson, you will know how to:

- Describe the features of actionable messages with an Adaptive card.
- Describe scenarios for refreshing an actionable message.

Overview of actionable messages

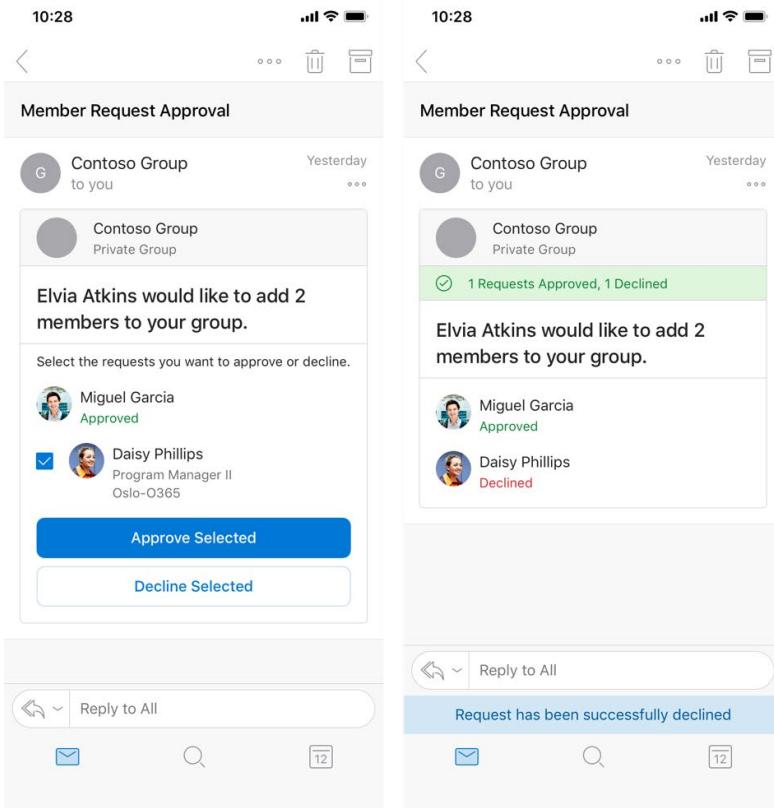
Actionable messages enable users to complete the task on messages from services they use every day right from Outlook. Actionable messages can be posted via a group or inbox connector or can be sent directly over email.

Actionable messages via email

Sending actionable messages via email is supported in the following scenarios.

- The recipient must be an individual, not a group.
- The recipient must be visible on the message. Do not put the recipient in the BCC field.
- The recipient must have a mailbox on Outlook.com or Exchange Online in Office 365.

Several line-of-business solutions send system messages to users that are business critical and requires the user to complete a task. Today, users complete these tasks by visiting a website or switching to another application. This hinders user productivity due to context switching and the extra steps required to complete that task. Examples include expense approvals, bill pay, etc. These are great candidates for enabling actions within the email itself.



Actionable messages with an adaptive card

Adaptive cards communicate information immediately, and their user interface allows users to view and interact with the cards. Adaptive cards allow users to provide a quick response without having to open an app.

These cards can enable greater engagement with users' personal contacts: groups such as customers, company colleagues, club members, friends, and so on.

Users can share Adaptive cards with any number of contacts, across a variety of devices and platforms, including notifications on Windows, Android, and iOS. To create Adaptive cards, you use JSON scripts.

Outlook Actionable Message cards

Outlook Actionable Messages cards are designed using the Adaptive Card format. The Adaptive Card format is a simple yet powerful declarative layout format that provides a lot of flexibility, allowing for visually rich cards.

Adaptive card layout

The typical layout of an Adaptive card consists of a title, perhaps some descriptive text, and then one or more columns of detail. This type of layout works in many situations. Because each column can contain other columns and containers, you can create cards of greater complexity by nesting objects inside each other. By varying the fonts, font sizes, colors, spacing, background images, and so on, you can create different results.

Developers can use **Card Playground tool²³** to play around different Adaptive Card formats. There you will find Adaptive Card samples that can help you get started crafting your own cards and also allows you to send those cards to your own Office 365 email account to see how they look in Outlook.

Refresh adaptive cards

Refresh cards are a very powerful mechanism that allow Action.Http actions to fully update the card on the fly as the action successfully completes.

For examples, using refresh cards with actions that can only be taken a single time. In those cases, the refresh card would not include any action that cannot be taken anymore. Another example is to use refresh cards with actions that change the state of the entity they are performed on. In those cases, the refresh card should include updated information about the entity, and MAY change the set of actions that can be performed.

There are many scenarios that benefit from refresh cards:

- **Approval scenario (e.g. expense report):** Once the request is approved or rejected, the card is refreshed to remove the approve/decline actions and update its content, so it reflects the fact that it's been approved or declined.
- **Task status:** When an action is taken on a task, such as setting its due date, the card refreshes to include the updated due date in its facts.
- **Survey:** Once the question has been answered, the card is refreshed so:
 - It no longer allows the user to respond.
 - It shows updated status, like "Thanks for responding to this survey" alongside the user's actual response.
 - Potentially include a new Action.OpenUrl action that allows the user to consult the survey online.

Lesson review questions

What are the three actions of Adaptive cards?

1. Open URL, Submit, and Input
2. Open URL, Choice Set, and Show Card
3. Open URL, Submit, and Show Card
4. Open URL, Submit, and Show Image

Suggested answer:

The *Open URL* action directs a user to a new or updated product page. The *Submit* action encapsulates data entered by the user and sends it back to your server. The *Show card* action helps you subdivide complex messages into easily understandable chunks.

²³ <https://messagecardplayground.azurewebsites.net/>