Actions Private Preview Developer Instructions v3

Last edited 7/27/2023 by Irene Wang

Overview

Developers have the power to enhance users' productivity by building Actions that streamline task completion, minimizing context switching across various applications. This feature not only benefits users by enabling them to accomplish tasks more efficiently, but it also enhances the visibility and user engagement of your app by seamlessly integrating it into their workflow.

During this private preview phase, we are introducing Actions on content in the Microsoft 365 app. This empowers users to take immediate action on content files through your app, expanding the range of interactions users can have with their content.

This developer documentation provides a comprehensive guide on building Actions as part of the Actions private preview program. It covers the following key areas:

- Understanding Actions
- Steps to build and test Actions
- Guidelines

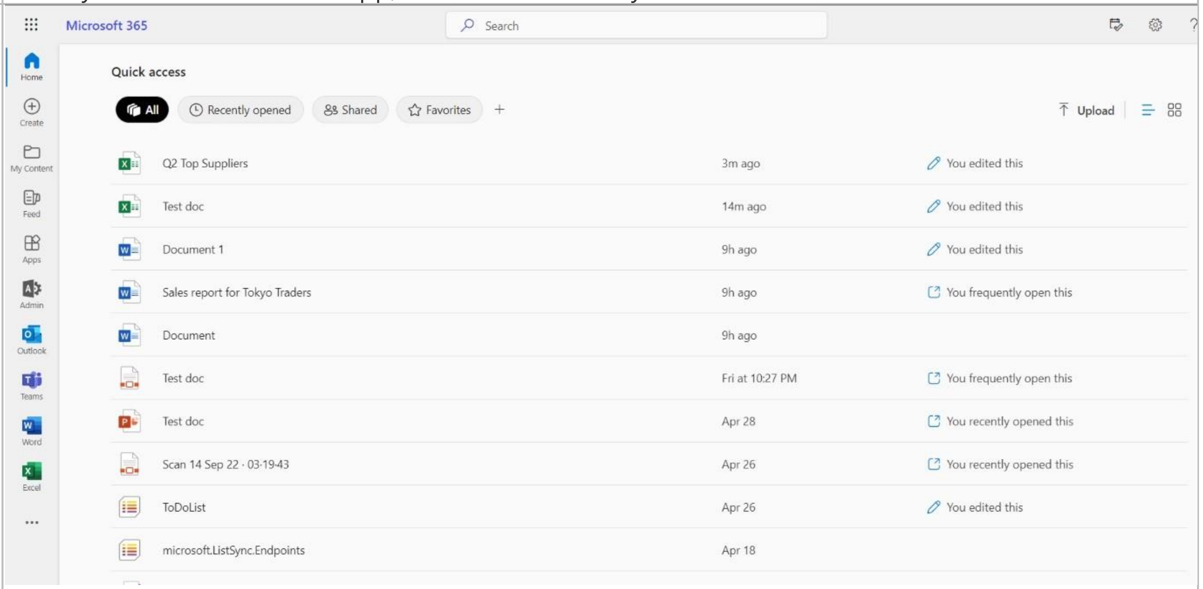
Thank you for participating in this private preview! For any questions or feedback, please reach out to Irene Wang. (irenewang@microsoft.com)

What is an Action?

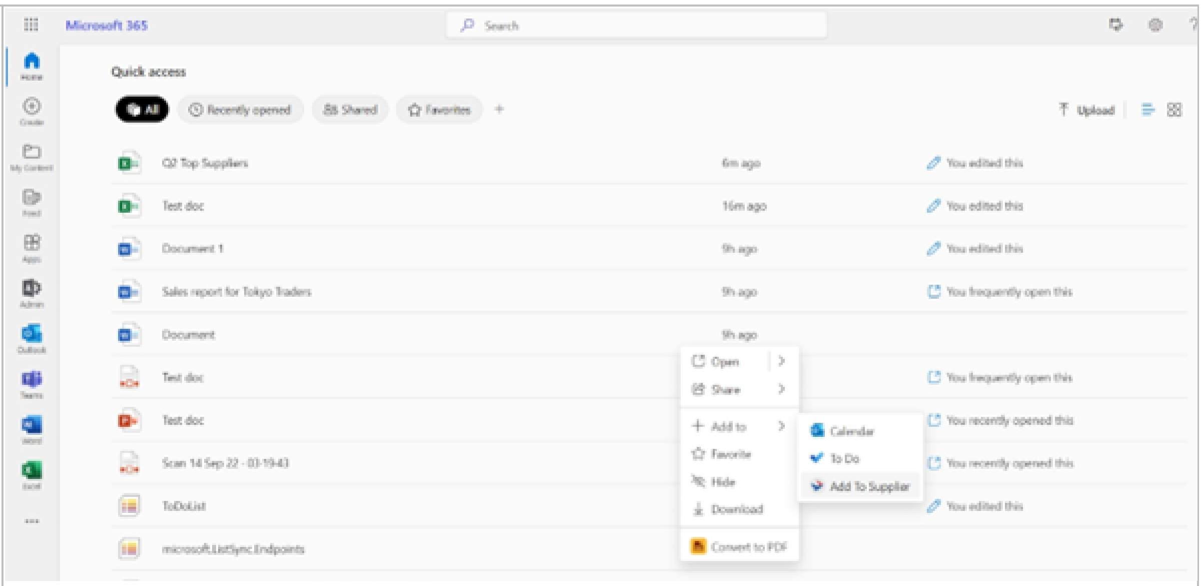
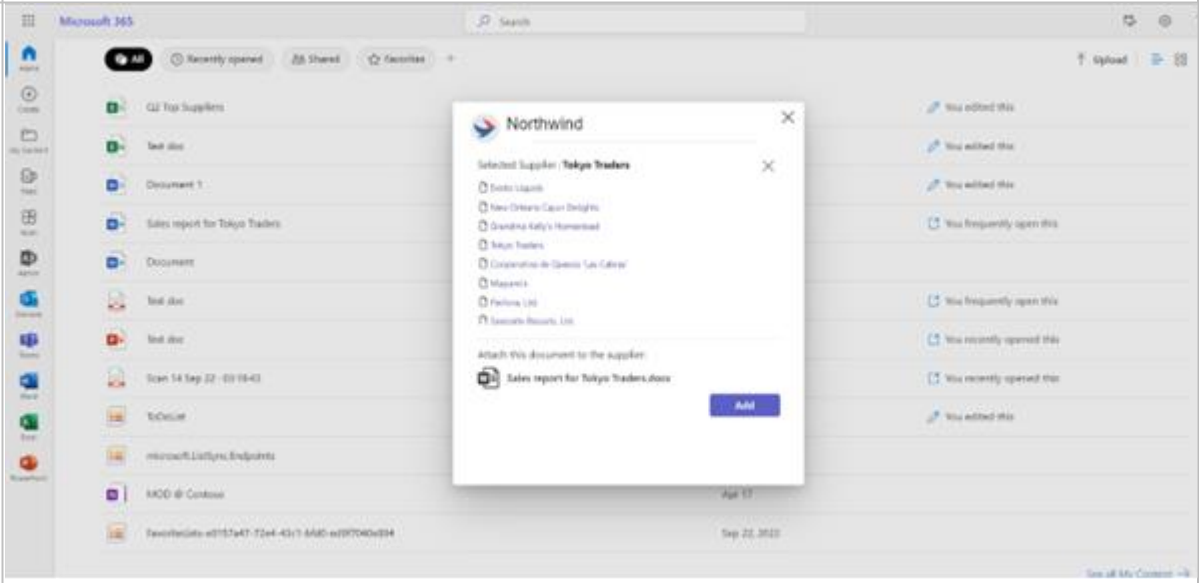
Actions aim to integrate your app into your user's workflow by enabling easy discoverability and seamless interaction with their content. By directing users to your app with their intent and contextual content, Actions enable efficient task completion. This integration not only enhances the visibility and engagement of your app but also offers these benefits with minimal development effort.

User scenarios

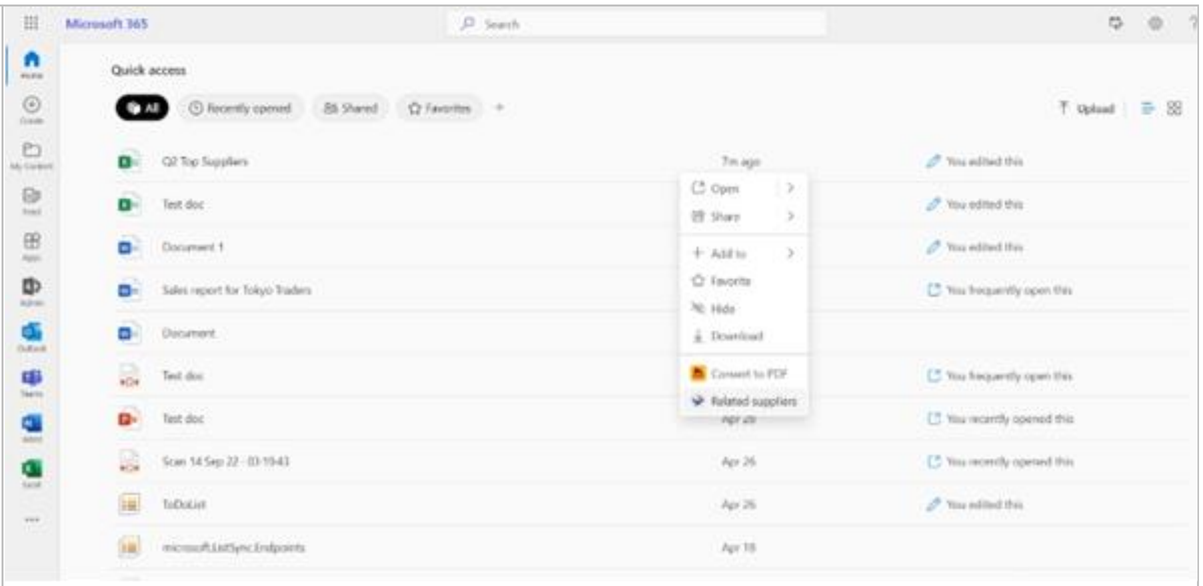
When a user wants to complete a task on a piece of content, an Action can surface your app right when they need it. Let’s walk through a user scenario of how users might interact with your app through Actions you develop.

	Customer scenarios walk through
0	Background: Kat is a supervisor at Northwind Traders with limited time for focused work. She starts her day in the Microsoft 365 app, where she can easily access all of her content.
	
1	Scenario 1: Action opens a dialog
1.a	She sees the latest Sales report for a supplier, 'Tokyo Trader,' and wants to add it as an attachment in the supplier management system app built by Northwind Traders. Right-clicking on the Word document, she chooses the Action 'Add to supplier' built by Northwind Traders.

Q2 Top Suppliers	3m ago	You edited this
Test doc	14m ago	You edited this
Document 1	9h ago	You edited this
Sales report for Tokyo Traders	9h ago	You frequently open this
Document	9h ago	
Test doc	Fri at 10:27 PM	You frequently open this
Test doc	Apr 28	You recently opened this
Scan 14 Sep 22 - 03-19-43	Apr 26	You recently opened this
ToDoList	Apr 26	You edited this
microsoft.ListSync.Endpoints	Apr 18	

	
1.b	A dialog pops up where she selects 'Tokyo Traders' and then clicks 'Add'. She is able to add the attachment quickly, without opening the document or app.
	
2	Scenario 2: Action opens page
2.a	On the same page, she notices the 'Q2 Top suppliers' Excel sheet and wants to see which suppliers she works with are on this list. She right-clicks on the Excel file, then clicks on the Action 'Related suppliers'.

Northwind
Selected Supplier : Tokyo Traders
Exotic Liquids
New Orleans Cajun Delights
Grandma Kelly's Homestead
Tokyo Traders
Cooperativa de Quesos 'Las Cabras'
Mayumi's
Pavlova, Ltd.
Specialty Biscuits, Ltd.
Attach this document to the supplier:
Sales report for Tokyo Traders.docx
Add


Microsoft 365

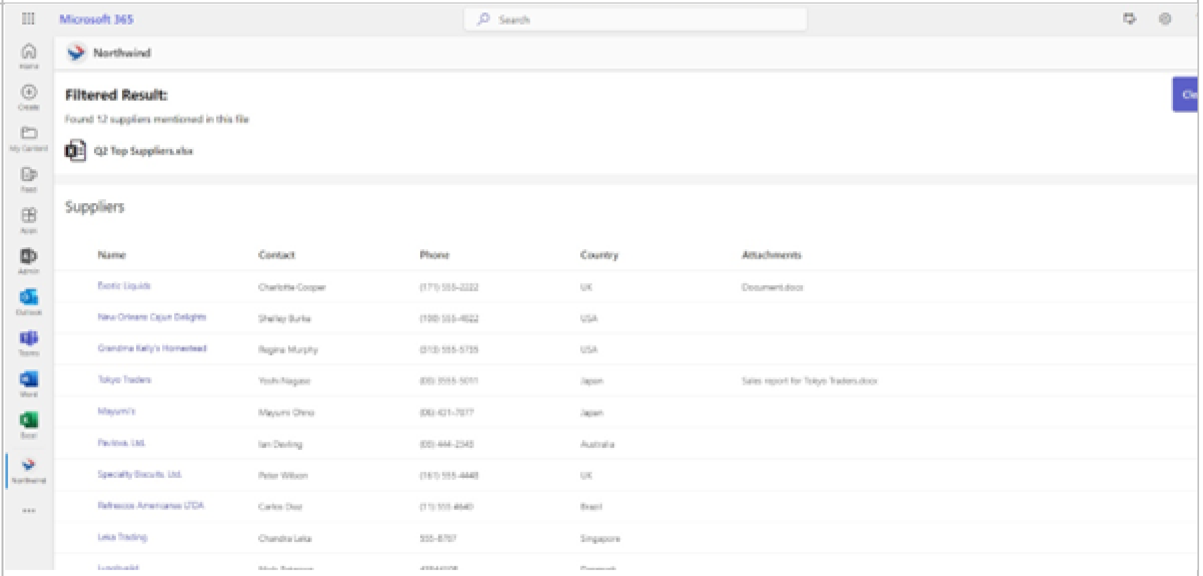
Quick access

All | Recently opened | Shared | Favorites

Upload

Q2 Top Suppliers	7m ago	You edited this
Test doc		You edited this
Document 1		You edited this
Sales report for Tokyo Traders		You frequently open this
Document		
Test doc		You frequently open this
Test doc	Apr 26	You recently opened this
Scan 14 Sep 22 - 03-19-43	Apr 26	You recently opened this
ToDoList	Apr 26	You edited this
microsoft.ListSync.Endpoints	Apr 18	

2.b The Northwind app opens, displaying the list of suppliers filtered to show only those that appear in the document. This saves her time opening up the app and the Excel file and checking each item manually.


Microsoft 365

Northwind

Filtered Result:

Found 12 suppliers mentioned in this file

Q2 Top Suppliers.xlsx

Suppliers

Name	Contact	Phone	Country	Attachments
Exotic Liquids	Charlotte Cooper	(171) 555-2222	UK	Document.docx
New Orleans Cajun Delights	Shelley Burke	(100) 555-4822	USA	
Grandma Kelly's Homestead	Regina Murphy	(313) 555-5735	USA	
Tokyo Traders	Yoshi Nagase	(03) 3555-5011	Japan	Sales report for Tokyo Traders.docx
Mayumi's	Mayumi Ohno	(06) 431-7877	Japan	
Pavlova, Ltd.	Ian Devling	(03) 444-2343	Australia	
Specialty Biscuits, Ltd.	Peter Wilson	(161) 555-4448	UK	
Refrescos Americanas LTDA	Carlos Diaz	(11) 555 4640	Brazil	
Leka Trading	Chandra Leka	555-8787	Singapore	

Understand how Actions work

From a developer's perspective, an Action is built using a combination **of Intent + Object + Handler**. When a user intends to accomplish a task, it can be represented as

intent + object, with the **intent** denoting the verb describing the user's desired action, and the **object** representing the noun on which the action is to be performed.
As a developer, your role is to receive the user's **intent + object** input and construct the corresponding **handler** that facilitates task completion for the users.

To build an Action, you will define the **intent, object,** and **handler** of your actions in the **manifest**. And in your handler, use Teams JS V2 to receive the Action information to create a seamless user experience for performing users specific tasks.

Concept Name	**Description**	**What is supported in private preview**
Intent	"intent" refers to the objective or purpose that a user intends to perform or achieve. User intent is typically represented by a verb, such as "open," “add to”, "create", "share," or any other action that a user intends to perform. This "intent" enables the M365 platform to display the Actions in locations that most align with the user's needs and intentions. This includes but not limited to, where Actions show up and how Actions are grouped or ordered.	We currently enable three main intents for Actions. Among them, "**Open**" and "**AddTo**" are the commonly used intents within the Microsoft 365 app. Additionally, with the "**custom**" intent, developers have the flexibility to build Actions to fulfill any user task. **In short: “open”, “addTo”, “custom”**
Object	"object" refers to the specific entity or item on which the user intends to act. It represents the noun or context content that defines what the user wants to perform an action on. For now, an "object' is piece of content (file). Later, we plan to extend it to more entity types, for example, emails, people, app, etc.	Currently, Actions can be triggered on **content objects**, like Word, PowerPoint, Excel, and PDF, which reside in **OneDrive and SharePoint** that accessible through Microsoft Graph.
Handlers	A "handler" refers to the method or mechanism to fulfill the user's intent and perform the desired action on the specified object. It is responsible for implementing the logic and functionality of the Action, ensuring a seamless and meaningful user experience. To support your users in the most meaningful way, we offer multiple types of handlers that you can build. You have the choice to direct users to the app’s page or	**openPage:** This handler allows you to directly guide users to your app's launch page (aka personal tab). By utilizing the openPage handler, you can effectively drive users to your app's dedicated pages, providing them with a rich and expansive interface to accomplish their goals. **openDialog**: This handler directs users to a dialog (aka Task Module), offering a dedicated and contextualized interface for interacting

	enable them to complete tasks within a dialog. Furthermore, we have plans to introduce additional types of handlers in the future.	with your app's features without opening the full app. This ensures a focused and efficient workflow, allowing users to complete tasks seamlessly within their current context.

For this private preview, Actions are supported in Microsoft365 App (on web at www.microsoft365.com and on Windows app store at https://www.microsoft.com/store/productId/9WZDNCRD29V9).

Steps to build Actions

3 steps to build an Action:

Step1: Define Actions in the manifest
Step2: Retrieve Action information through context object
Step3: Access content object through Graph API

Details of each step are explained below.

Step1: Define Actions in the manifest

To make an action show up in the context menu when the user right-clicks on a piece of content in Microsoft 365 app, you simply add a json payload in the Manifest, where you define the user intent, object type and the handler that will trigger.

Here is a sample manifest that builds the two Actions in the above user experience section.
Link to the sample manifest in Github.

Sample manifest with 2 Actions:

```
{
  "$schema": "https://raw.githubusercontent.com/OfficeDev/microsoft-teams-app-schema/preview/DevPreview/MicrosoftTeams.schema.json",

  "manifestVersion": "devPreview",


   ...
  "actions": [
    {
      "id": "completeLinkedTask",
      "displayName": "Mark complete",
      "intent": "custom",
      "description": "Test action with open intent and openPage handler",
      "handlers": [
        {
          "type": "openPage",
          "supportedObjects": {
            "file": {
              "extensions": ["xlsx", "doc", "docx", "pdf", "pptx", "ppt"]
            },
            "folder": null
          },
          "pageInfo": {
            "pageId": "index",
            "subPageId": ""
          }
        }
      ]
    },
    {
      "id": "addTodoTask",
      "displayName": "Add todo task",
      "intent": "addTo",
      "description": "Add this file with a short note to my to do list",
      "handlers": [
        {
          "type": "openDialog",
          "supportedObjects": {
            "file": {
              "extensions": ["xlsx", "doc", "dot", "docx", "pdf", "pptx",
"ppt"]
            },
            "folder": null
          },
          "dialogInfo": {
            "dialogType": "url",
            "url": "${{TAB_ENDPOINT}}/index.html#/dialogPage",
            "width": "small",
            "height": "large"
          }
        }
      ]
    }
  ],
  ...
```

```
    }
```

Notes:

- Action "Add to supplier", which uses "addTo" intent, supports files with extensions ".docx", ".doc". And it has handler type "openDialog". It will open an HTML based dialog as from the url address.
- Action "Related suppliers", uses "custom" intent, supports files with extensions ".xlsx". And it has handler type "openPage". It will open the App, and navigate to the page as defined in the pageId and subpageId.

Reference:

$schema

The `https://` URL referencing the JSON Schema for the manifest.

During preview, please use developer preview manifest schema

```
"$schema": "https://raw.githubusercontent.com/OfficeDev/microsoft-teams-app-schema/preview/DevPreview/MicrosoftTeams.schema.json",
```

manifestVersion

During preview, please use developer preview manifest schema.

```
"manifestVersion": "devPreview",
```

actions

Defines the actions for this app.
The object is an array of action objects. This block is required only for solutions that provides Actions.
Each action item is an object with the following structure:

Name	Type	Required	Description
id	String	Y	An identifier string in the default locale that is used to catalog actions. Must be unique across all actions for this app. Example: "openDocInContoso"
displayName	String	Y	A display name for the action. Capitalize first letter. Example: "Add to suppliers", "Open in Contoso"
description	String	Y	A longer display name for the action.
intent	Enum	Y	An enum string that describes the intent of the action. Possible values: open, addTo, custom.

icons	Array of objects	N	Array of objects containing icons needed for the action. Example:
			`"icons": [`
			`{ "size": 32, "url": "https://url/for/icon.png" },`
			`{ "size": 48, "url": "https://url/for/icon1_5x.png" },`
			`{ "size": 64, "url": "https://url/for/icon2x.png" }`
			`],`
			(Currently not supported)

handlers	Array of objects	Y	An array of handlers object, defining how actions can be handled. If an app has more than one handler, only one experience will show up at one entry point. The hub will decide which action to show up based on which experience is supported.

actions.handlers

Defines the handlers of the action.The handlers is an array of handler objects. Each action must have at least one handler.
Each action handler is an object with the following structure:

Name	Type	Required	Description
supportedObjects	Object	N	Objects defining what objects can trigger this action.
type	Enum	Y	An enum defining the handler type. Possible values: openPage, openDialog, etc
dialogInfo	Object	N	Required if the handler type is openDialog. Object containing metadata of the dialog handler.
pageInfo	Object	N	Required if the handler type is openPage. Object containing metadata of the page to open.
url	String	N	Required if the handler type is openUrl. **(Currently not supported)**
botInfo	Object	N	Required if the handler type is openBot. **(Currently not supported)**

actions.handlers.supportedObjects

Objects defining what objects can trigger this action. Currently we are supporting files that is stored in ODSP (OneDrive and SharePoint). More types will come later.

Name	Type	Required	Description
file	Object	N	Supported file types
file.extensions	Array of strings	N	Array of strings. File extensions of the file types the action can trigger, e.g. “.pdf”, “.docx”.
folder	Object	N	Object indicates that the file handler is available when a folder is selected. **(Currently not supported)**
supportsMultiSelect	Boolean	N	Boolean indicates if multiple files can be selected. **(Currently not supported)**

actions.handlers.pageInfo

Required if the handler type is openPage. Object containing metadata of the page to open.

Name	Type	Required	Description
pageId	String	N	Id of the page (in the app) that the action will direct the user to.
subpageId	String	N	Id of the subpage (in the app) that the action will direct the user to.

actions.handlers.dialogInfo

Required if the handler type is openPage. Object containing metadata of the page to open.

Name	Type	Required	Description
dialogType	Enum	Y	Enum string define the type of the dialog. Values can be: “url” for html based dialog (supported now), or “adaptiveCard” for adaptive card based dialog. (not supported yet)
width	String	Y	Dialog width - either a number in pixels or default layout such as 'large', 'medium', or 'small'.

height	String	Y	Dialog height - either a number in pixels or default layout such as 'large', 'medium', or 'small'.

actions.handlers.botInfo (Not supported yet)

Required if the handler type is openPage. Object containing metadata of the page to open.

Name	Type	Required	Description
botId	String	Y	The unique Microsoft app ID for the bot as registered with the Bot Framework. This may well be the same as the overall app ID.
fetchTask	Boolean	N	A Boolean value that indicates if it should fetch the task module dynamically.

Step2: Build the handler that retrieves Action information through context object

Based on the handler type:

- With open page handler type, users will be directed to your app's launch page based on the pageId and subPageId you defined in the manifest.
- As for the open dialog handler, you will create an HTML-based dialog that enables users to complete tasks seamlessly within the dialog itself.

Once your app's page or dialog is launched, your app can access contextual information about the invoked Action from the actionInfo property of the Context object (returned from a call to app.getContext()).

2.a. What is ActionInfo in context object?

The Microsoft Teams JavaScript client library (TeamsJS) has been updated to enable your app to determine when a page or dialog was opened from an Action, and the content that user was triggering this Action.

app.Context (Context interface)
A new property actionInfo has been added to the app.Context object to represent the action by which your app was invoked.
actionInfo (ActionInfo interface)
Contains: • actionId : `string` (Maps to the action id supplied inside the manifest)

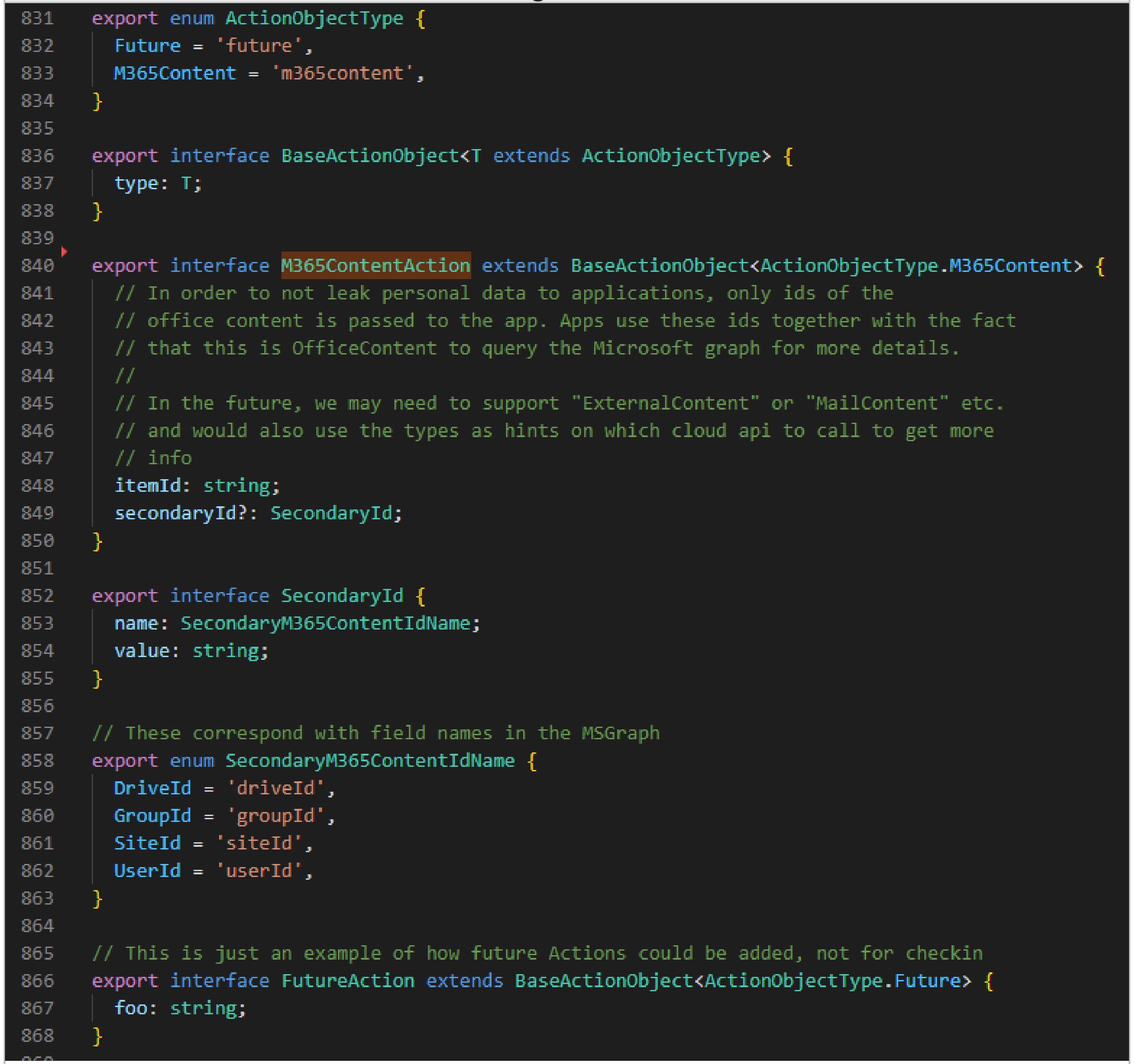
- actionObjects : `BaseActionObject<M365Content>[]` (Array of corresponding action objects)

M365ContentAction (M365ContentAction interface)

Contains:

- itemId: IDs of the content are passed to the app. Apps should use these ids to query the Microsoft graph for more details. (See the below section for how you can use itemId to retrieve the content info)
- secondaryId: Represents an optional secondary identifier for an action in a Microsoft 365 content item.

More details see the reference or the image below



```
export enum ActionObjectType {
  Future = 'future',
  M365Content = 'm365content',
}

export interface BaseActionObject<T extends ActionObjectType> {
  type: T;
}

export interface M365ContentAction extends BaseActionObject<ActionObjectType.M365Content> {
  // In order to not leak personal data to applications, only ids of the
  // office content is passed to the app. Apps use these ids together with the fact
  // that this is OfficeContent to query the Microsoft graph for more details.
  //
  // In the future, we may need to support "ExternalContent" or "MailContent" etc.
  // and would also use the types as hints on which cloud api to call to get more
  // info
  itemId: string;
  secondaryId?: SecondaryId;
}

export interface SecondaryId {
  name: SecondaryM365ContentIdName;
  value: string;
}

// These correspond with field names in the MSGraph
export enum SecondaryM365ContentIdName {
  DriveId = 'driveId',
  GroupId = 'groupId',
  SiteId = 'siteId',
  UserId = 'userId',
}

// This is just an example of how future Actions could be added, not for checkin
export interface FutureAction extends BaseActionObject<ActionObjectType.Future> {
  foo: string;
}
```

2.b. Sample code getting the Action info from context
Applications can access the context object using *app.getContext*. Once an application has the context object, they can do the following things with it.

1. Check if the app was launched using an action:

```
app.getContext().then((context: app.Context) => {
  const actionInfo = context.actionInfo;
  if (actionInfo) {
      // App was launched using an action
  }
})
```

2. Check the actionId was linked to the action that launched the application

```
app.getContext().then((context: app.Context) => {
    const actionInfo = context.actionInfo;
    if (actionInfo && actionInfo.actionId == 'myActionId1`) {
        // Handle specific action
    }
})
```

3. Determine the type of an Action and access its type specific data

```
app.getContext().then((context: app.Context) => {
    const actionInfo = context.actionInfo;
    if (actionInfo) {
        if (actionInfo.actionObject.type == app.ActionObjectType.M365Content)
{
            const m365Action: app.M365ContentAction = actionInfo.actionObject;
            // Get the requested content from Mirosoft Graph by item id:
${m365Action.itemId};
        }
    }
})
```

Link to sample code: get context

Step3 : Access content through Graph API

After obtaining the itemId of the triggering content, you can leverage the Graph API to read or modify the content, facilitating task completion for your users.

Sample code: fetch data, call Graph API

(Optional step) Close the dialog

When constructing an action that directs the user to a dialog, you can utilize the dialog.url.submit( ) API to close the dialog when the user clicks a button.

However, it's important to note that using the submit() API in your app's in the Action triggered dialog will not allow data to be sent back to the launch page.

Test your Actions with your own account

[For private preview] Enable Actions in your tenant

During the private preview, please send an email to irenewang@microsoft.com providing the Tenant ID of the tenant for which you wish to enable Actions. We will then enable the Actions feature for your specified tenant, allowing accounts within the tenant to access this feature. Make sure the account to be used for sideloading to test Actions is part of the tenant. Through sideloading, only the associated account will have access to your Actions, other individuals within the tenant will not have visibility at this stage.

Sideload your app to the test tenant

Once you've updated your app to support actions, you can preview by sideloading it with the TeamsFx CLI a commandline tool.
(Note: in this private preview, side-loading through Teams Toolkit or Teams UI are not enabled. Those will be enabled during public preview.)

	Steps	**Commands (use in terminal)**
1	Install TeamsFx CLI (NPM instruction)	`$ npm install -g @microsoft/teamsfx-cli`
2	Check you're using TeamsFx-cli version 1.2.2 or later.	`$ teamsfx --version`

3	Sideload your app package with TeamsFx cli	`$ teamsfx m365 sideloading --file-path <path_to_package>`
		
		`Example: teamsfx m365 sideloading --file-path C:\Users\irenewang\Downloads\appPackage.local.zip`

3		

Test your action with Microsoft 365 app

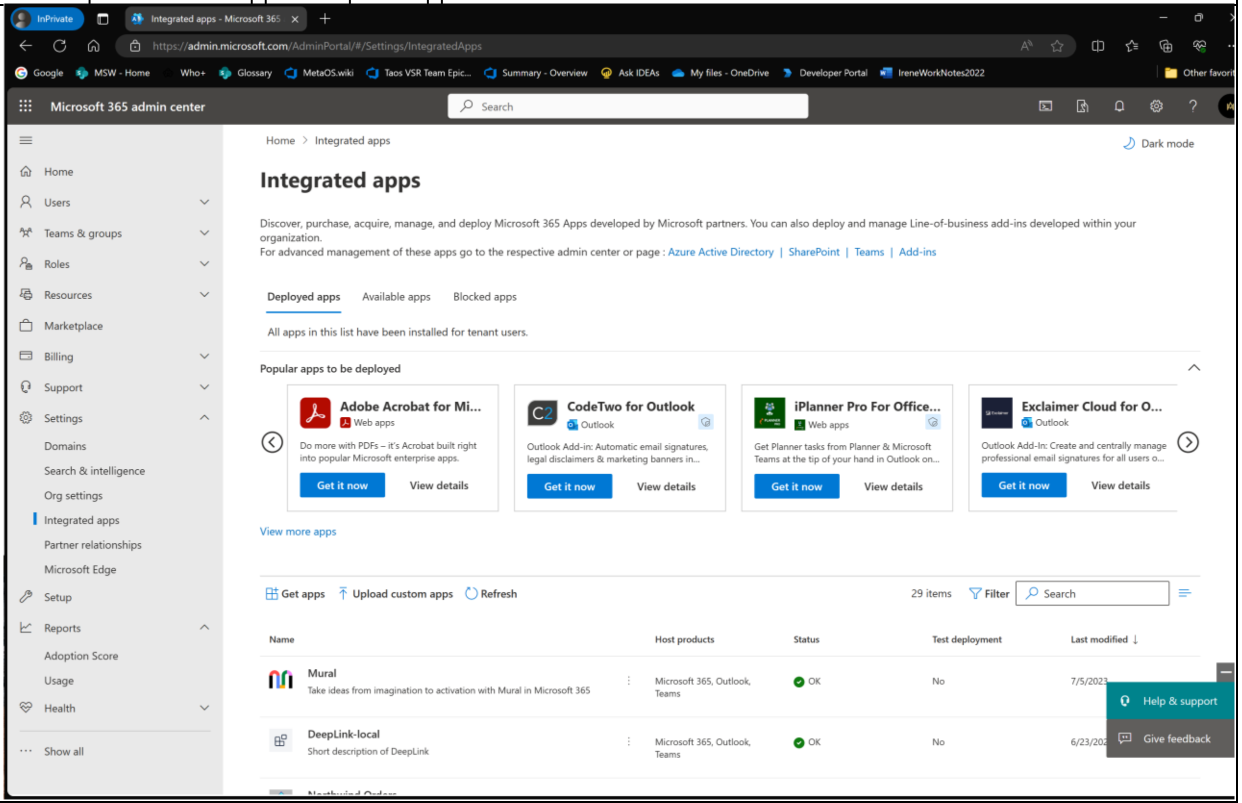
You can now preview your new content action by opening the Microsoft 365 app (https://www.microsoft365.com/) and right-clicking a content file that is supported by your action. Your new content action should appear in the context menu.

Enable Actions for your users

During this private preview, we offer the opportunity for your Actions to be experienced by end users, allowing you to gather early feedback.

To enable this, admins are required to upload the app package containing the Actions via the Microsoft Admin Center (admin.microsoft.com) and acquire this app for targeted release to users in the tenant. Once these steps are completed, users will have access to and can utilize the Actions.

To enable Actions, please provide us with the Tenant ID of the respective tenant. We will ensure that your tenant is added to the allowlist, granting you access to utilize this feature. During the private preview, please send an email to irenewang@microsoft.com providing the Tenant ID of the tenant for which you wish to enable Actions.

Go to Microsoft 365 admin center -> settings -> integrated apps (Or though url: https://admin.microsoft.com/AdminPortal/#/Settings/IntegratedApps) Click "Upload custom apps" to upload apps. Follow instructions 

Integrated apps

Discover, purchase, acquire, manage, and deploy Microsoft 365 Apps developed by Microsoft partners. You can also deploy and manage Line-of-business add-ins developed within your organization.
For advanced management of these apps go to the respective admin center or page : Azure Active Directory | SharePoint | Teams | Add-ins

Deployed apps | Available apps | Blocked apps

All apps in this list have been installed for tenant users.

Popular apps to be deployed

Adobe Acrobat for Mi... — Web apps — Do more with PDFs – it's Acrobat built right into popular Microsoft enterprise apps. — Get it now | View details

CodeTwo for Outlook — Outlook — Outlook Add-in: Automatic email signatures, legal disclaimers & marketing banners in... — Get it now | View details

iPlanner Pro For Office... — Web apps — Get Planner tasks from Planner & Microsoft Teams at the tip of your hand in Outlook on... — Get it now | View details

Exclaimer Cloud for O... — Outlook — Outlook Add-In: Create and centrally manage professional email signatures for all users o... — Get it now | View details

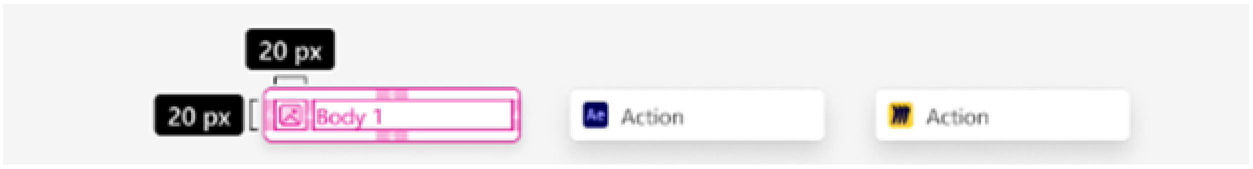
View more apps

Get apps | Upload custom apps | Refresh — 29 items — Filter — Search

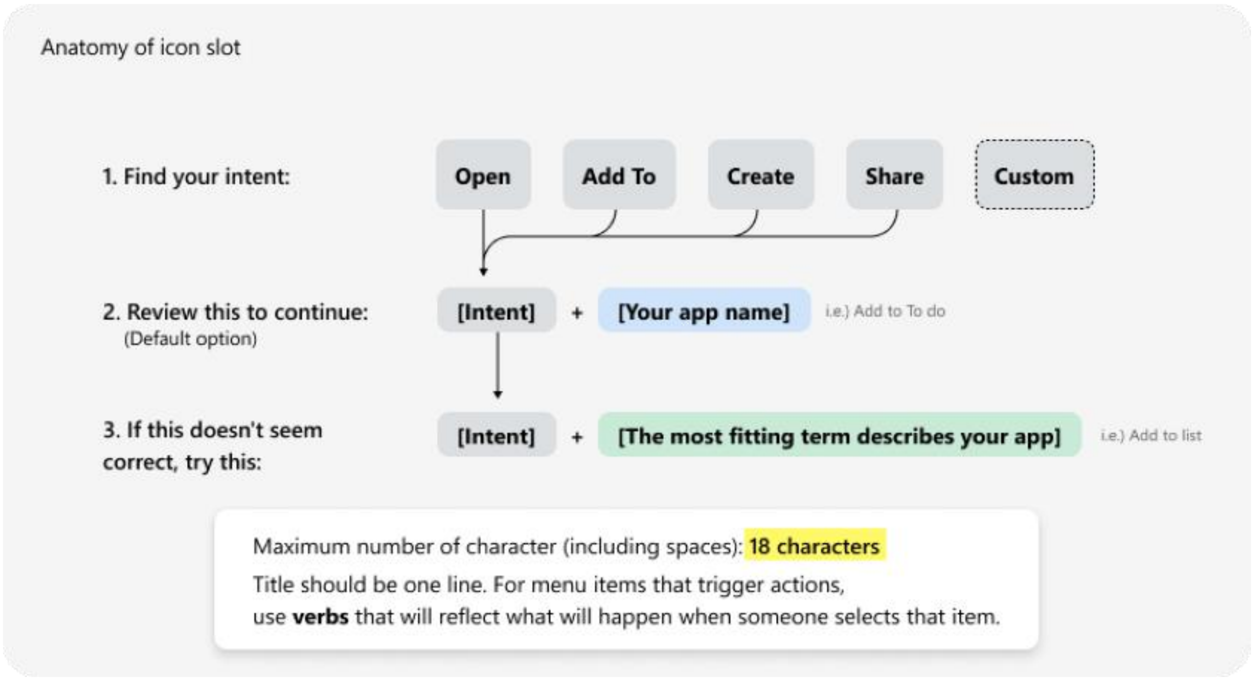
Name	Host products	Status	Test deployment	Last modified ↓
Mural Take ideas from imagination to activation with Mural in Microsoft 365	Microsoft 365, Outlook, Teams	✓ OK	No	7/5/2023
DeepLink-local Short description of DeepLink	Microsoft 365, Outlook, Teams	✓ OK	No	6/23/202

Design guidelines

One single action in the context menu contains: App icon + display name



Choose a display name that is simple and explanatory.



Maximum number of character (including spaces): 18 characters

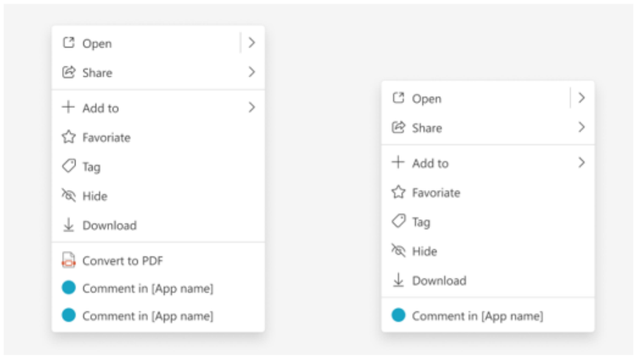
Title should be one line. For menu items that trigger actions, use **verbs** that will reflect what will happen when someone selects that item.

Grouping: Actions with custom intent will show as a flat list in the bottom of the context menu, actions with Open/Add to intent will be grouped into Open and Add to.

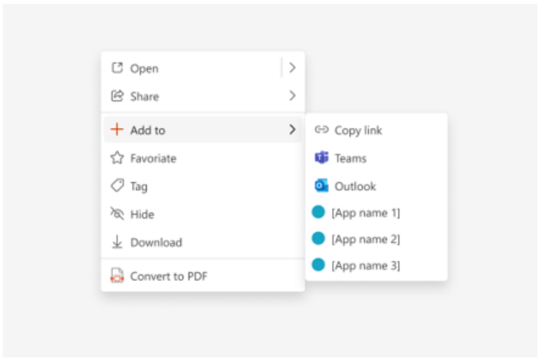
(Note: The placement of actions is determined by the M365 platform. Using intent does not guarantee grouping, and using custom intent does not imply no grouping. We are planning to introduce additional features and experiences to assist users in quickly locating the most relevant and useful actions.)

Actions
In context menu

Flat list showing at the bottom of the context menu



Grouped by intent


Open
Share
Add to
Favoriate
Tag
Hide
Download
Convert to PDF
Copy link
Teams
Outlook
[App name 1]
[App name 2]
[App name 3]