

Contents

[Outlook Documentation](#)

[REST APIs](#)

[Overview](#)

[Get Started](#)

[Using the Outlook REST APIs](#)

[.NET Core](#)

[Android](#)

[Angular](#)

[ASP.NET](#)

[Microsoft Flow](#)

[iOS Objective-C](#)

[iOS Swift](#)

[Java](#)

[JavaScript](#)

[Node.js](#)

[PHP](#)

[Python](#)

[React](#)

[React Native](#)

[Ruby](#)

[UWP](#)

[Xamarin](#)

[Concepts](#)

[Overview](#)

[Compare Graph and Outlook](#)

[Reference](#)

[Terms of Use](#)

[Add-ins](#)

[Actionable Messages](#)

[Overview](#)

[Get Started](#)

[Get started with actionable messages](#)

[Actionable Messages via Email](#)

[Actionable Messages via Connectors](#)

[Concepts](#)

[Identity linking](#)

[Invoke an Outlook add-in](#)

[Refresh cards on open](#)

[Security Requirements](#)

[Adaptive Cards](#)

[Actionable Email Developer Dashboard](#)

[Connectors Developer Dashboard](#)

[Card Playground](#)

[Adaptive Card Designer](#)

[MessageCard Reference \(Legacy\)](#)

Outlook Developer documentation

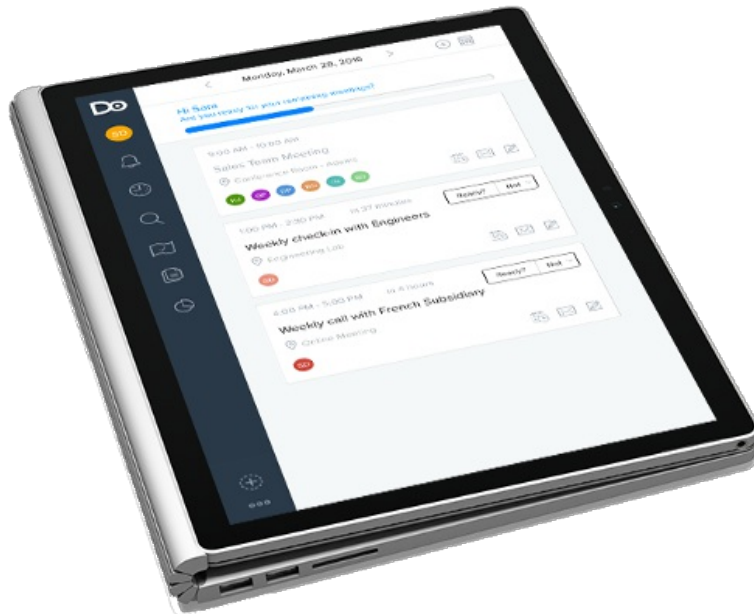
4/18/2022 • 2 minutes to read • [Edit Online](#)

How can I integrate with Outlook?

Outlook provides integrations that allow you to both access Outlook data from your app (REST APIs via Microsoft Graph) or bring your app into Outlook (add-ins or actionable messages).

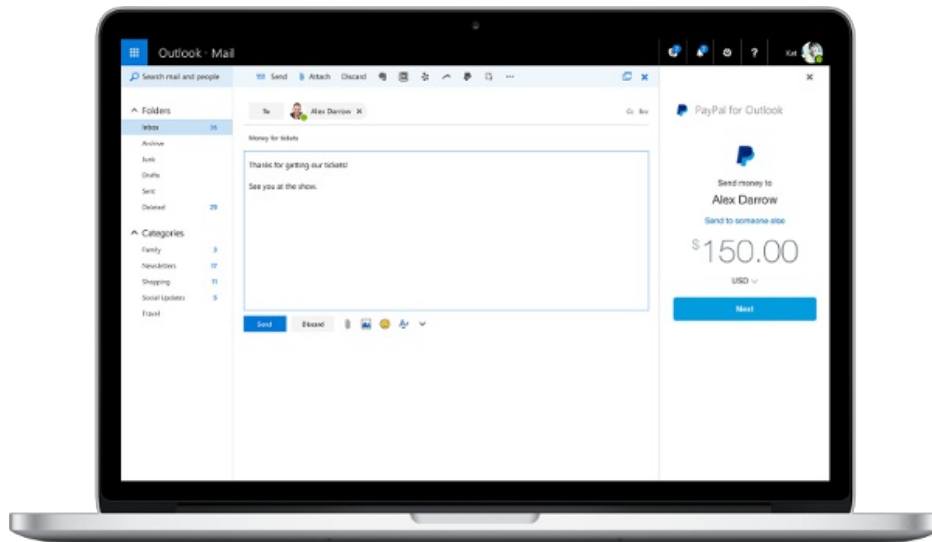
REST APIs

You can access Outlook data in Office 365 or Outlook.com via OData and REST by using OAuth2 and [Microsoft Graph](#). These APIs empower your app to perform actions like search email, post events to user's calendars, synchronize contacts, and much more. Check our [Get Started](#) section for guided walkthroughs on a number of popular platforms.



Outlook add-ins

Outlook add-ins provide a way for you to bring your app or experience right into Outlook. Outlook add-ins work across desktop, web, and mobile versions of Outlook and allow users to take actions on the message or appointment they are currently reading or composing. Check out our [quick start](#) to get started with your first add-in.



Actionable Messages

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 and Teams. Developers can now embed actions in their emails or notifications, elevating user engagement with their services and increasing organizational productivity. Check out our [Get Started](#) section to try it out.



Need help?

Ask the developer community questions about your code on the [Microsoft Q&A forum](#).

You can also join the Outlook developer community on Stack Overflow. We use the following tags for each of the integration types:

- REST APIs: [StackOverflow outlook-restapi tag](#)
- Outlook add-ins: [StackOverflow outlook-web-addins tag](#)
- Actionable messages: [StackOverflow office365connectors tag](#)

Suggestions

We want to hear from you! If there are features that you'd like us to consider, please visit [Microsoft 365 Developer Platform Ideas](#) and [Submit an Idea](#).

One Outlook REST API - your favorite platform - 400+ million users

4/18/2022 • 2 minutes to read • [Edit Online](#)

With the simplicity of REST, you can use your favorite language and IDE, write your app once, and capture 400 million monthly active Outlook.com users, and tens of millions active Office 365 users.

Start with choosing a language for your app—Node, Python, Ruby, Swift—just to name a few. Write the code, take advantage of new, streamlined services to register and authorize the app, and access user's mail, calendar, and contacts data on Outlook.com or Office 365. You can use the same Outlook REST API for Android, iOS, Windows, on the web, mobile, and desktop. There's no need for any specialized Exchange knowledge!

Get started

1. [Register](#) your app. Registration takes only a minute.
2. [Start](#) coding and implement REST API calls.

Streamlined services

There's just one simple process for Outlook.com and Office 365: register and get dynamic user authorization to access users' mail, calendar, and contacts.

Take an early look at the following services in preview status:

- [Azure Active Directory admin center](#)

Use a Microsoft account or Microsoft 365 subscription account to register your app. It takes only a few steps to identify your app for Outlook.com and Office 365.

- v2 endpoints:

[Use these common authentication endpoints](#) to sign users in with their personal Outlook.com accounts, or business or school credentials, and request authorization for access. Instantly expand the reach of your app!

- `https://login.microsoftonline.com/common/oauth2/v2.0/authorize`
- `https://login.microsoftonline.com/common/oauth2/v2.0/token`

Use these preview services when:

- Rewriting existing Outlook.com apps that use the Windows Live API. Windows Live API has been deprecated for Outlook.com. Plan to accommodate changes to these services over their preview period.
- Creating new Outlook.com and Office 365 apps that access mailbox data.

Plan to upgrade in-production Office 365 apps after the preview period is over.

Outlook REST API via Microsoft Graph

Use one common REST endpoint...

```
https://graph.microsoft.com/{version}
```

...to make all Outlook REST API calls in the following APIs:

- [Outlook Mail REST API](#)
- [Outlook Calendar REST API](#)
- [Outlook Contacts REST API](#)
- [Outlook Notifications REST API](#)
- [Outlook Photo REST API](#)
- [Outlook Settings REST API](#)

For more information and a comparison between the Graph endpoints and the Outlook API endpoints, see [Compare Microsoft Graph and Outlook endpoints](#).

Overview of using the Outlook REST APIs

4/18/2022 • 3 minutes to read • [Edit Online](#)

TIP

Try out sample REST calls in the [Graph Explorer](#). You can use your own account, or one of our test accounts. Once you're done exploring the API, come back here and select your favorite platform on the left. We'll guide you through the steps to write a simple application to retrieve messages from your inbox.

If your preferred platform isn't listed yet, continue reading on this page. We'll go through the same set of steps using raw HTTP requests.

The purpose of this guide is to walk through the process of calling the [Outlook Mail API](#) to retrieve messages in Office 365 and Outlook.com. Unlike the platform-specific getting started guides, this guide focuses on the OAuth and REST requests and responses. It will cover the sequence of requests and responses that an app uses to authenticate and retrieve messages.

This tutorial will use [Microsoft Graph](#) to call the Mail API. Microsoft recommends using Microsoft Graph to access Outlook mail, calendar, and contacts. You should use the Outlook APIs directly (via `https://outlook.office.com/api`) only if you require a feature that is not available on the Graph endpoints.

With the information in this guide, you can implement this in any language or platform capable of sending HTTP requests.

Use OAuth2 to authenticate

In order to call the Mail API, the app requires an access token from the Microsoft identity platform. Use one of the [supported OAuth 2.0 flows](#) to obtain an access token.

Calling the Mail API

Once the app has an access token, it's ready to call the Mail API. The [Mail API Reference](#) has all of the details. Since the app is retrieving messages, it will use an HTTP GET request to the

`https://graph.microsoft.com/v1.0/me/mailfolders/inbox/messages` URL. This will retrieve messages from the inbox.

Refining the request

Apps can control the behavior of GET requests by using [OData query parameters](#). It is recommended that apps use these parameters to limit the number of results that are returned and to limit the fields that are returned for each item. Let's look at an example.

Consider an app that displays messages in a table. The table only displays the subject, sender, and the date and time the message was received. The table displays a maximum of 25 rows, and should be sorted so that the most recently received message is at the top.

To achieve this, the app uses the following query parameters:

- The `$select` parameter is used to specify only the `subject`, `from`, and `receivedDateTime` fields.
- The `$top` parameter is used to specify a maximum of 25 items.
- The `$orderby` parameter is used to sort the results by the `receivedDateTime` field.

This results in the following request.

Mail API request for messages in the inbox

```
GET https://graph.microsoft.com/v1.0/me/mailfolders/inbox/messages?
$select=subject,from,receivedDateTime&$top=25&$orderby=receivedDateTime%20DESC

Accept: application/json
Authorization: Bearer eyJ0eXAi...b66LoPVA
```

Mail API Response

```

HTTP/1.1 200 OK
Content-Type:
application/json;odata.metadata=minimal;odata.streaming=true;IEEE754Compatible=false;charset=utf-8

{
  "@odata.context":
  "https://graph.microsoft.com/v1.0/$metadata#users(...)/mailfolders('inbox')messages(subject,from,receivedDate
  Time)",
  "value": [
    {
      "@odata.etag": "W/\"CQAAABYAAAAoPBSqxXQOT6tuE0pxCMrtAABufX4i\\\"\"",
      "id": "AAMkADRMMDExYzhjLWYyNGMtNDZmMC1iZDU4LTRkMjk4YTdjMjU5OABGAAAAAABp4MZ-
      5xP3TJnNAPmjsRslBwAoPBSqxXQOT6tuE0pxCMrtAAAAAEMAAoPBSqxXQOT6tuE0pxCMrtAABufW1UAAA=",
      "subject": "Ruby on Rails tutorial",
      "from": {
        "emailAddress": {
          "address": "jason@contoso.onmicrosoft.com",
          "name": "Jason Johnston"
        }
      },
      "receivedDateTime": "2015-01-29T20:44:53Z"
    },
    {
      "@odata.etag": "W/\"CQAAABYAAAAoPBSqxXQOT6tuE0pxCMrtAABSzmz4\\\"\"",
      "id": "AAMkADRMMDExYzhjLWYyNGMtNDZmMC1iZDU4LTRkMjk4YTdjMjU5OABGAAAAAABp4MZ-
      5xP3TJnNAPmjsRslBwAoPBSqxXQOT6tuE0pxCMrtAAAAAEMAAoPBSqxXQOT6tuE0pxCMrtAABMirSeAAA=",
      "subject": "Trip Information",
      "from": {
        "emailAddress": {
          "address": "jason@contoso.onmicrosoft.com",
          "name": "Jason Johnston"
        }
      },
      "receivedDateTime": "2014-12-09T21:55:41Z"
    },
    {
      "@odata.etag": "W/\"CQAAABYAAAAoPBSqxXQOT6tuE0pxCMrtAABzx1LG\\\"\"",
      "id": "AAMkADRMMDExYzhjLWYyNGMtNDZmMC1iZDU4LTRkMjk4YTdjMjU5OABGAAAAAABp4MZ-
      5xP3TJnNAPmjsRslBwAoPBSqxXQOT6tuE0pxCMrtAAAAAEMAAoPBSqxXQOT6tuE0pxCMrtAABAb1ZoAAA=",
      "subject": "Multiple attachments",
      "from": {
        "emailAddress": {
          "address": "jason@contoso.onmicrosoft.com",
          "name": "Jason Johnston"
        }
      },
      "receivedDateTime": "2014-11-19T20:35:59Z"
    },
    {
      "@odata.etag": "W/\"CQAAABYAAAAoPBSqxXQOT6tuE0pxCMrtAAA9yBBa\\\"\"",
      "id": "AAMkADRMMDExYzhjLWYyNGMtNDZmMC1iZDU4LTRkMjk4YTdjMjU5OABGAAAAAABp4MZ-
      5xP3TJnNAPmjsRslBwAoPBSqxXQOT6tuE0pxCMrtAAAAAEMAAoPBSqxXQOT6tuE0pxCMrtAAA9x_8YAAA=",
      "subject": "Attachments",
      "from": {
        "emailAddress": {
          "address": "jason@contoso.onmicrosoft.com",
          "name": "Jason Johnston"
        }
      },
      "receivedDateTime": "2014-11-18T20:38:43Z"
    }
  ]
}

```

Now that you've seen how to make calls to the Mail API, you can use the API reference to construct any other kinds of calls your app needs to make. However, bear in mind that your app needs to have the appropriate

permissions configured on the app registration for the calls it makes.

Outlook REST API conceptual documentation

4/18/2022 • 2 minutes to read • [Edit Online](#)

The [Microsoft Graph documentation](#) includes the following Outlook-related conceptual documentation.

Calendar API

- [Outlook calendar API overview](#)
- [Find possible meeting times on the Outlook calendar](#)
- [Schedule repeating appointments as recurring events in Outlook](#)

Contacts API

- [Outlook personal contacts API overview](#)

Mail API

- [Outlook mail API overview](#)
- [Create and send Outlook messages](#)
- [Organize Outlook messages](#)
- [Share Outlook message folders between users](#)

Compare Microsoft Graph and Outlook REST API endpoints

4/18/2022 • 5 minutes to read • [Edit Online](#)

The Outlook REST APIs are available in both [Microsoft Graph](#) and the Outlook API endpoint (`https://outlook.office.com/api`). The APIs generally provide the same functionality and use the same resource types.

NOTE

The Outlook REST APIs are deprecated.

The Outlook REST endpoints will be fully decommissioned in November 2022. Migrate existing apps to use Microsoft Graph.

Which endpoint should I use?

Use Microsoft Graph whenever possible. The Microsoft Graph endpoint lets you access Outlook and many more [services and features](#), including other Office 365 services, Enterprise Mobility + Security, and Windows 10. Choosing the Microsoft Graph endpoint allows your app to get an access token that can provide access to both Outlook data and other resources, without having to make multiple token requests.

Feature differences

There are some features that are currently either only available on the Outlook endpoint, or are only in beta in Microsoft Graph. If your app needs these features, you should access them via the Outlook endpoint.

NOTE

We are constantly working to incorporate all of the features currently available on the Outlook endpoint into Microsoft Graph. Be sure to check back periodically as this list is updated.

FEATURE	DIFFERENCE BETWEEN ENDPOINTS
Outlook tasks	Access to users' tasks in Microsoft Graph is available through the To Do API
Rich notifications	The Outlook API allows developers to request specific fields to be included with the notification payload by using the <code>\$select</code> parameter. Microsoft Graph does not support this feature.
Streaming notifications	The Outlook API supports streaming notifications in preview on the beta endpoint. Microsoft Graph does not support this feature.

Moving from Outlook endpoint to Microsoft Graph

The APIs are very similar on the Microsoft Graph endpoint and the Outlook endpoint. However, there are some

differences to be aware of, especially if you are migrating to Microsoft Graph or using both endpoints in the same application.

API versions

The Microsoft Graph API offers two versions: `v1.0` and `beta`, while Outlook offers `v1.0`, `v2.0`, and `beta`. Microsoft Graph `v1.0` matches Outlook `v2.0`, and Microsoft Graph `beta` matches Outlook `beta`.

OAuth scopes

While the Microsoft Graph and Outlook endpoints both rely on Azure AD-issued tokens, and the [permissions](#) used are the same, the way that your application requests those permissions is slightly different for each endpoint.

Azure v2 OAuth2 endpoint

Apps that use the [Azure AD v2.0 endpoint](#) for OAuth2 request permission scopes in the `scope` parameter in an authentication or token request.

- For Microsoft Graph, apps specify permissions prefixed with `https://graph.microsoft.com/`. For example, an app can request the `Mail.Read` permission by including `https://graph.microsoft.com/Mail.Read` in the `scope` parameter.
- For the Outlook endpoint, apps specify permissions prefixed with `https://outlook.office.com/`. For example, an app can request the `Mail.Read` permission by including `https://outlook.office.com/Mail.Read` in the `scope` parameter.

NOTE

If a domain is not included in a scope, Microsoft Graph is assumed.

Tokens issued for one endpoint are not valid for the other. Additionally, you cannot mix permissions for one endpoint with permissions for the other in a single request.

The Azure AD v2.0 endpoint only supports the [client credentials flow](#) for the Microsoft Graph endpoint. Apps that need to use an app-only token with the Outlook endpoint must use the Azure AD v1.0 endpoint.

Azure v1 OAuth2 endpoint

Apps that use the [Azure AD v1.0 endpoint](#) specify their required permissions during app registration in the Azure Portal.

- For Microsoft Graph, choose the **Microsoft Graph** API when adding required permissions.
- For the Outlook endpoint, choose the **Office 365 Exchange Online** API when adding required permissions.

Resource property names

The resources are largely the same between Microsoft Graph and Outlook. However, the two endpoints handle casing of the property names differently. Microsoft Graph uses camelCase for property names, while Outlook uses PascalCase. Translating between the two simply requires converting the case. Property names that are changed are specified in the table below.

For example, the Microsoft Graph [message resource](#) defines properties such as `subject`, `from`, and `receivedDateTime`. On the Outlook endpoint, these properties are named `Subject`, `From`, and `ReceivedDateTime`.

Changed property names

The following property names are different between Microsoft Graph and Outlook.

RESOURCE TYPE	MICROSOFT GRAPH PROPERTY	OUTLOOK PROPERTY
contact	mobilePhone	MobilePhone1

Tracking changes (synchronization)

Both endpoints support querying collections for changes relative to a synchronization state. While the functionality is the same, the methods are slightly different.

On the Microsoft Graph endpoint, changes are queried by using [delta queries](#). This is implemented as a `delta` function on the collection.

On the Outlook endpoint, changes are queried by [adding a header](#) to normal resource collection queries.

Batching

Both endpoints support batching up to 20 separate requests into one HTTP request.

[Microsoft Graph batching](#) encodes multiple API requests into a JSON body with a content type of `application/json`.

In addition to the JSON body format, [Outlook endpoint batching](#) also supports a multi-part body format with a content type of `multipart/mixed`.

Example: retrieving a message

Let's take a look at a simple example. In this scenario, a web app requests a list of messages in the user's inbox.

Microsoft Graph

First, the app has the user sign in to authorize the application. Because the app uses the Microsoft Graph scope `Mail.Read`, the authorization URL looks like the following:

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?
scope=openid+Mail.Read&response_type=code&client_id=<SOME GUID>&redirect_uri=<REDIRECT URL>
```

Once the app has an access token, it sends the following request:

```
GET https://graph.microsoft.com/v1.0/me/mailfolders/inbox/messages?
$top=1&$select=subject,from,receivedDateTime,isRead
Accept: application/json
Authorization: Bearer <token>
```

The server returns the following response:

```
{
  "@odata.context": "https://graph.microsoft.com/v1.0/$metadata#users('b63d5fb9-4f43-44c4-8f9d-fd0727842876')/mailFolders('inbox')/messages(subject,from,receivedDateTime,isRead)",
  "@odata.nextLink": "https://graph.microsoft.com/v1.0/me/mailfolders/inbox/messages?$top=1&$select=subject%2cfrom%2creceivedDateTime%2cisRead&$skip=1",
  "value": [
    {
      "@odata.etag": "W/\"CwAAABYAAACd9nJ/tVysQos2hTfspawRAAD8ujHV\"",
      "id": "AAMkAGI2...",
      "receivedDateTime": "2015-11-03T03:21:04Z",
      "subject": "Scrum",
      "isRead": false,
      "from": {
        "emailAddress": {
          "name": "user0TestUser",
          "address": "user0@contoso.com"
        }
      }
    }
  ]
}
```

Outlook

First, the app has the user sign in to authorize the application. Because the app uses the Outlook scope

`https://outlook.office.com/Mail.Read`, the authorization URL looks like the following:

```
https://login.microsoftonline.com/common/oauth2/v2.0/authorize?
scope=openid+https%3A%2F%2Foutlook.office.com%2Fmail.read&response_type=code&client_id=<SOME
GUID>&redirect_uri=<REDIRECT URL>
```

Once the app has an access token, it sends the following request:

```
GET https://outlook.office.com/api/v2.0/me/mailfolders/inbox/messages?
$top=1&$select=Subject,From,ReceivedDateTime,IsRead
Accept: application/json
Authorization: Bearer <token>
```

The server returns the following response:


```
{
  "@odata.context":
  "https://outlook.office.com/api/v2.0/$metadata#Me/MailFolders('inbox')/Messages(Subject,From,ReceivedDateTim
e,IsRead)",
  "@odata.nextLink":
  "https://outlook.office.com/api/v2.0/$metadata#Me/MailFolders('inbox')/Messages(Subject,From,ReceivedDateTim
e,IsRead)",
  "value": [
    {
      "@odata.etag": "W/\"CwAAABYAAACd9nJ/tVysQos2hTfspawRAAD8ujHV\"",
      "Id": "AAMkAGI2...",
      "ReceivedDateTime": "2015-11-03T03:21:04Z",
      "Subject": "Scrum",
      "IsRead": false,
      "From": {
        "EmailAddress": {
          "Name": "user0TestUser",
          "Address": "user0@contoso.com"
        }
      }
    }
  ]
}
```

Outlook API reference documentation

4/18/2022 • 2 minutes to read • [Edit Online](#)

The Outlook REST APIs are a part of [Microsoft Graph](#). Microsoft recommends using Microsoft Graph to access Outlook mail, calendar, and contacts. You should use the Outlook API endpoints directly (via

`https://outlook.office.com/api`) only if you require a feature that is not available on the Graph endpoints.

For more information about the differences between Graph and the Outlook endpoints, see [Compare Microsoft Graph and Outlook endpoints](#).

Released APIs

The following APIs are released and ready for production use in both the Graph and Outlook endpoints.

- [Mail API](#)
- [Calendar API](#)
- [Contacts API](#)
- [Groups API](#)
- [Push Notifications API](#)
- [User Photo API](#)
- [Open Extensions API](#)
- [Extended Properties API](#)
- [FindMeetingTimes API](#)
- [Sync messages, mail folders, events, contacts, and contact folders API](#)
- [People API](#)
- [Batch REST Requests](#)

Transitioning APIs

The following APIs are released and ready for production in the Outlook endpoint, but are in preview status in the Graph endpoint.

- [Tasks API](#)

Actionable messages in Outlook and Office 365 Groups

4/18/2022 • 5 minutes to read • [Edit Online](#)

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 now embed actions in their emails or notifications, elevating user engagement with their services and increasing organizational productivity.

Office 365 provides two solutions to enhance productivity with Outlook Actionable Messages: actionable messages via email, and actionable messages via Office connectors.

NOTE

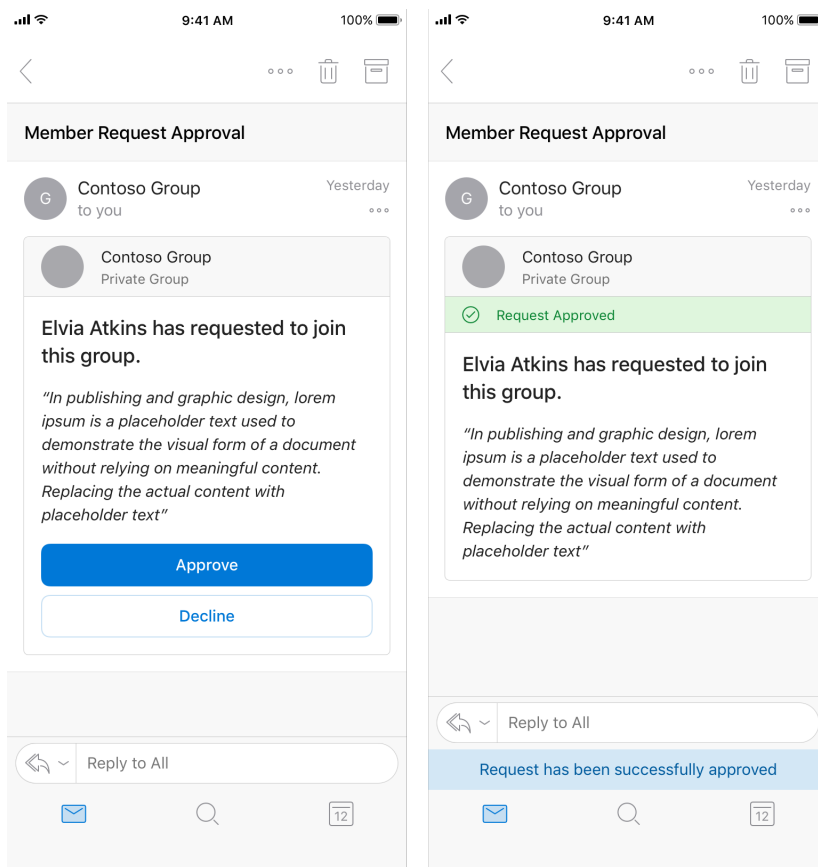
Actionable messages are also available in Microsoft Teams. See [Office connectors for Teams](#) for more information.

User experience

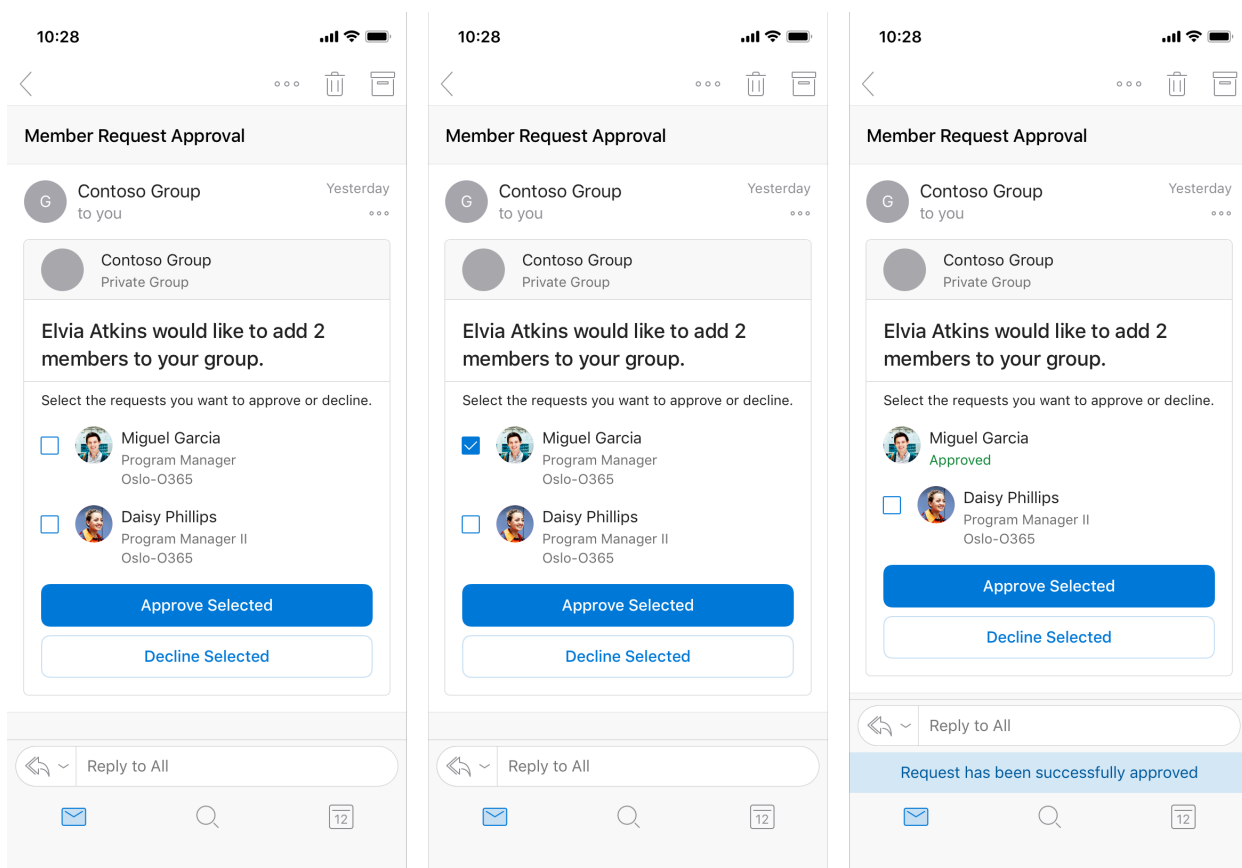
Let's take a look at the end-to-end user experience for both an email-based and a connectors-based actionable message scenario.

Actionable messages via email: group membership request approval scenario

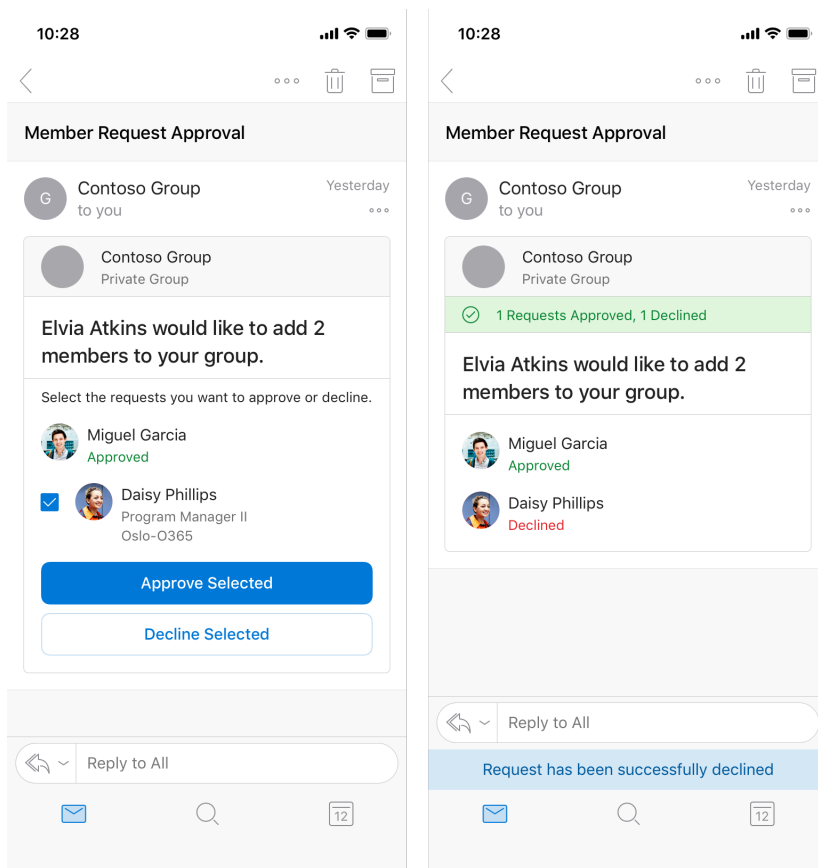
A Contoso employee submits a request to join a private Office 365 group. Office 365 sends an Actionable Message to the person who owns the group to approve or decline the request. The card included in the message contains all the information the approver might need to quickly understand who submitted the request and any message they included to explain their request. It also includes **Approve** and **Decline** actions that can be taken right from Outlook. The owner approves the request, and the card updates to indicate the outcome.



The new member of the group submits a second request to add her team members to the group. Office 365 sends an Actionable Message to the owner with clear information about who submitted the request and the new members to add. The recipient can approve all, some, or none of the proposed new members. The owner approves one new member, and the card updates to indicate the outcome. The approved member is no longer selectable, while the remaining member remains selectable.



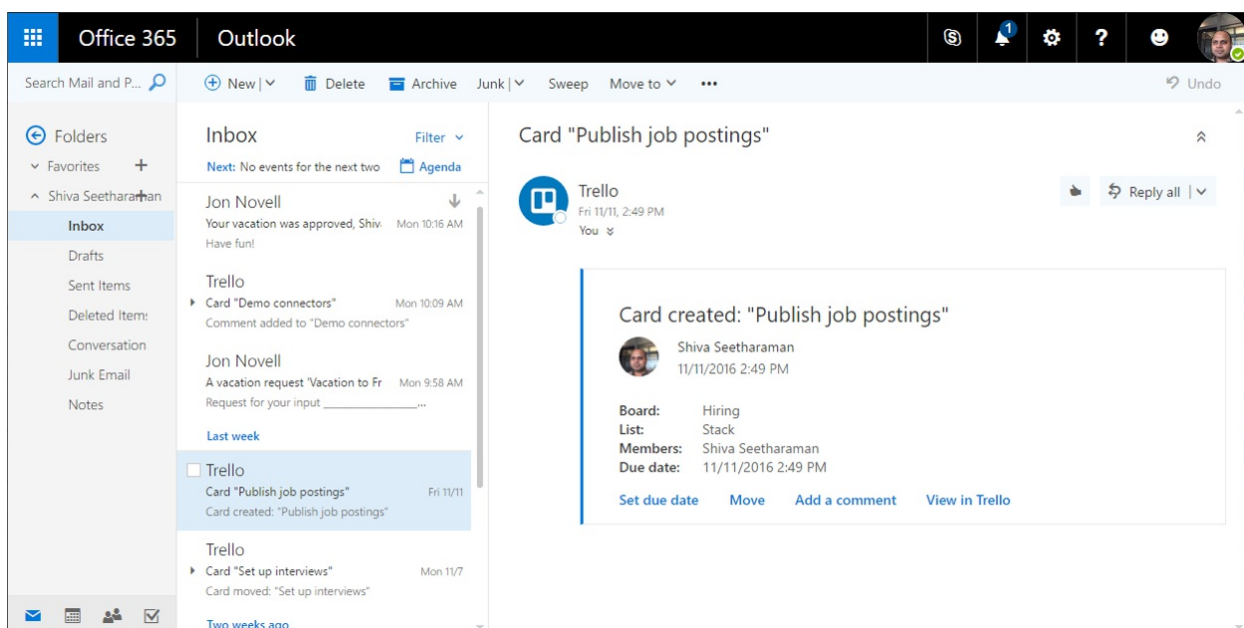
The owner declines the other requested new member, and the card updates to indicate the outcome. Both members are no longer selectable, and the action buttons are removed.



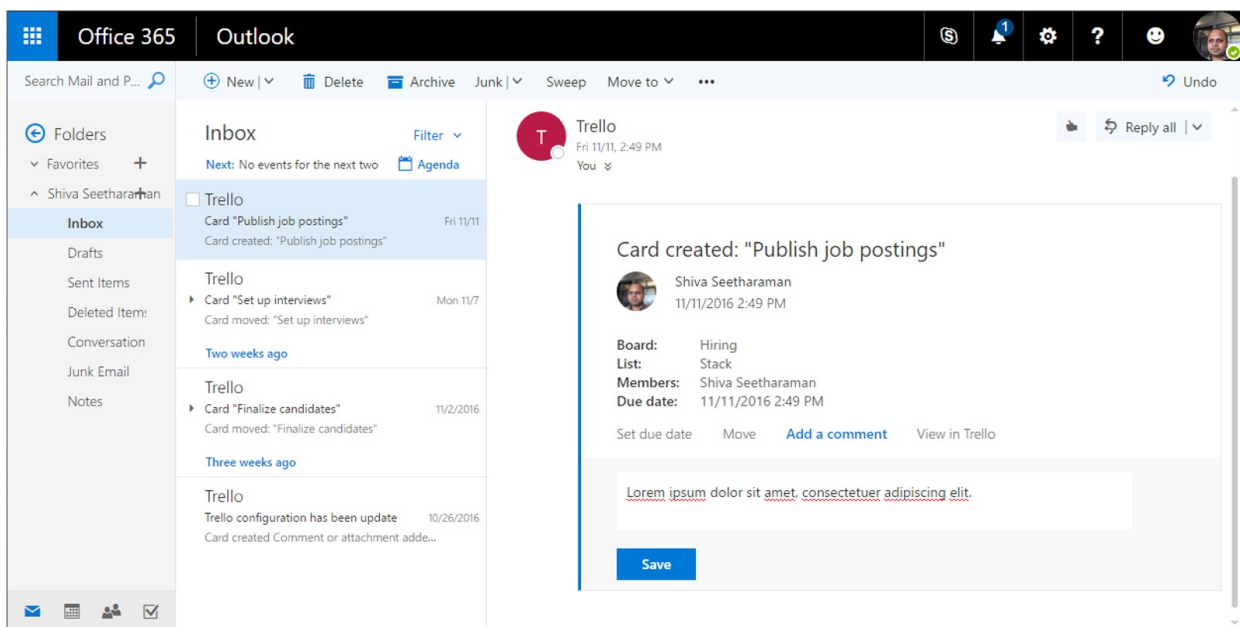
Actionable messages via Office connectors: task management scenario

Adele Vance and her team use Trello as their task management system. Adele has configured the Trello connector in her account, and will receive granular notifications as activity occurs in the Trello boards she is interested in.

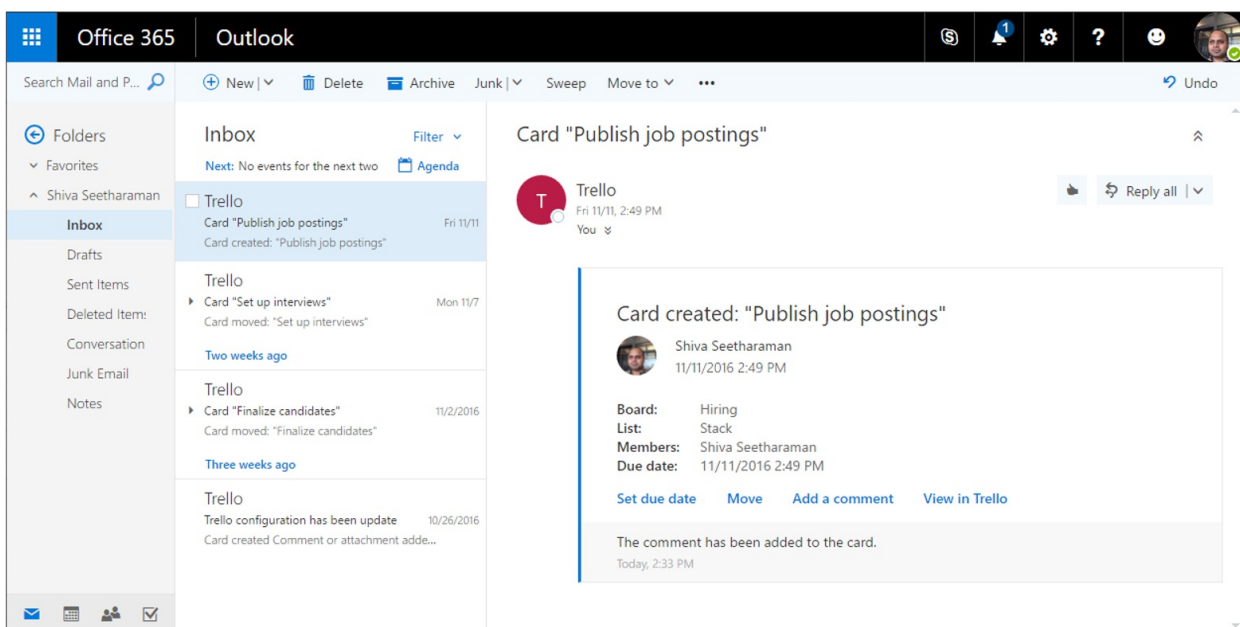
Shiva, in Adele's team, creates a new Trello card in the "Hiring" board. He needs the latest job postings to be published. Adele receives an actionable message that tells her all about the new card and the task it represents: who created it, in which list, what the due date is, and more.



Adele has a few notes she recently took on a piece of paper with important things that should be mentioned in the job postings. She decides to add these as a comment to the Trello card. She clicks the **Add a comment** action, and is presented with a text input field in which she can type her notes:



Adele then clicks the **Save** button, and the notes are immediately saved to the Trello card. A confirmation appears at the bottom of the message:



Office connectors

Office connectors are a great way to get useful information and content into your Office 365 Groups in Outlook or Microsoft Teams. Any user can connect their group or team to services like Trello, Bing News, Twitter, etc., and get notified of activity from that service. From tracking a team's progress in Trello, to following important hashtags in Twitter, Office connectors make it easier for an Office 365 group in Outlook, Microsoft Teams, or Yammer to stay in sync and get more done.

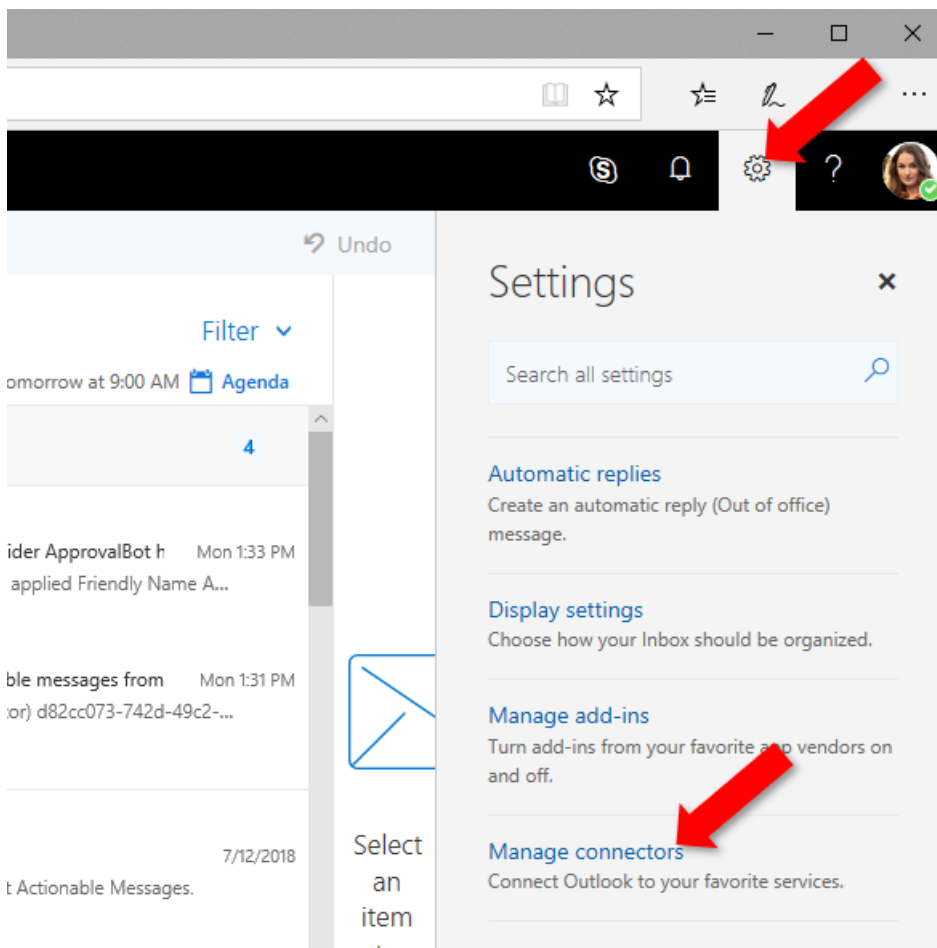
Accessing Office connectors from Outlook

Office connectors are available for both the inbox and Groups for any Office 365 Mail user. Connectors can be managed in either Outlook on the web, or Outlook 2016 or later on Windows.

Accessing connectors in Outlook on the web

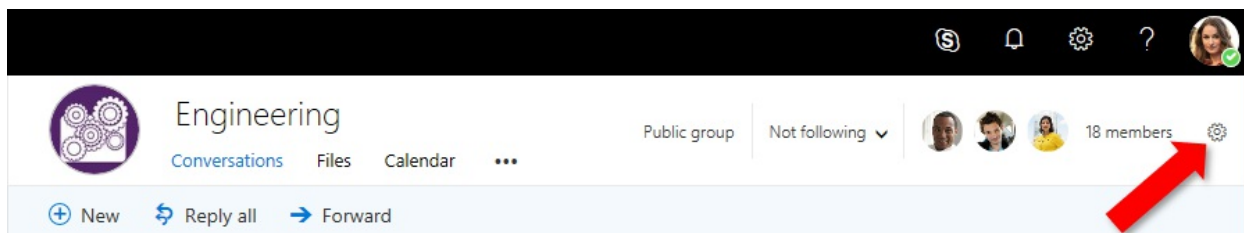
Inbox connectors

Users access inbox connectors from the **Settings** menu, accessed from the gear icon in the top-right corner.



Groups connectors

Users access group connectors from the **Group settings** menu, accessed from the gear icon in the group title bar.



Group settings

Engineering

Manage group email

Choose which group messages to receive in your inbox.

Connectors

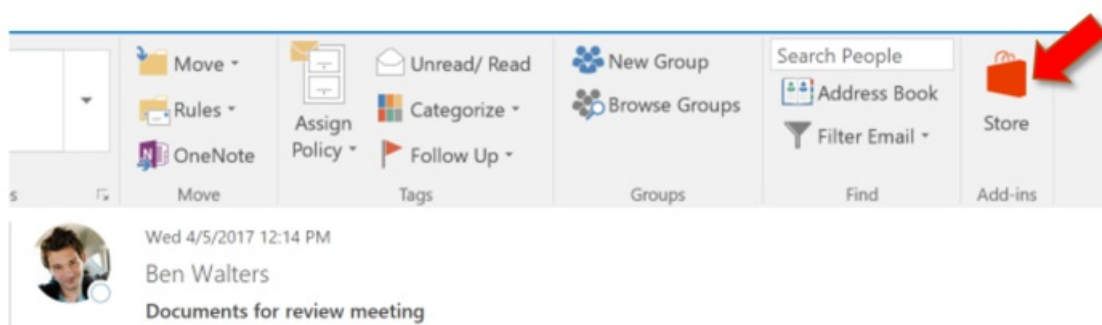
Manage or add your favorite services to this group.

[Remove from Favorites](#)

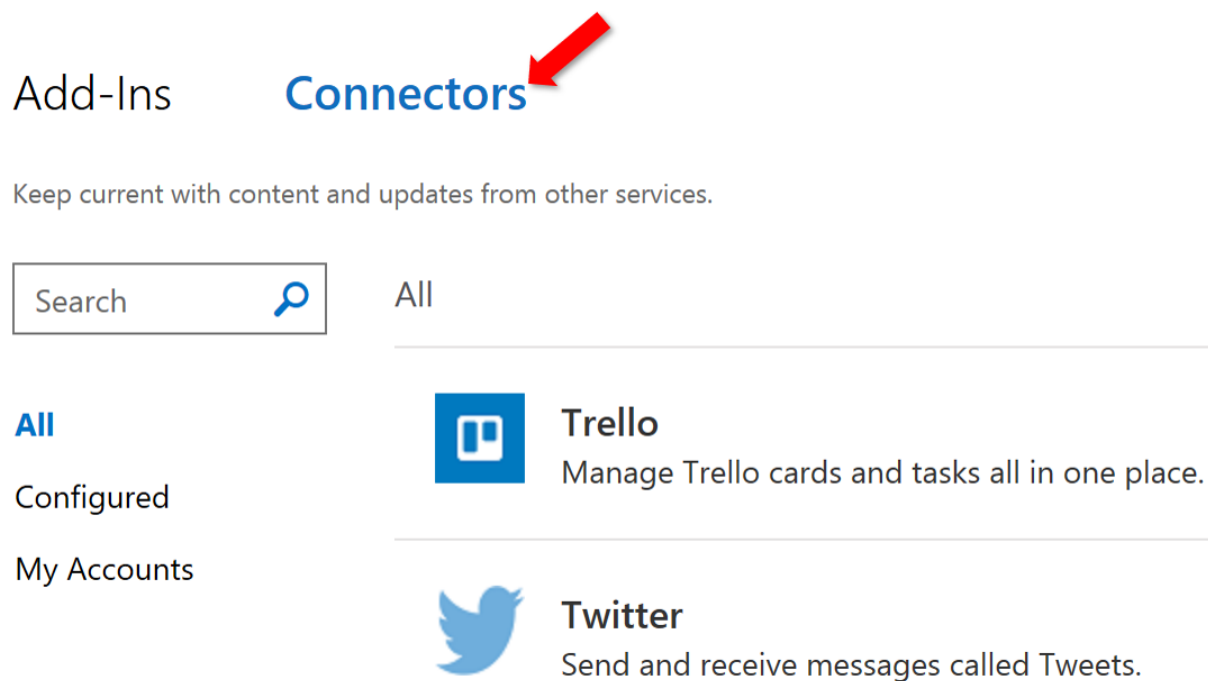
[Invite others](#)

[Leave group](#)

Users access both inbox and group connectors from the **Store** button. The Store button loads connectors for the inbox if Outlook is currently displaying a mail folder in the user's mailbox, and loads connectors for the currently selected group if displaying a group.



In the inbox case, the Store dialog displays the **Add-ins** tab by default. Connectors are available in the **Connectors** tab.



Release Notes

Currently, you can only configure connectors from Outlook on the web, or Outlook 2016 or later on Windows, but you can view information posted by Connectors to your Group or inbox in multiple clients such as Outlook on the web, Outlook 2016 or later, and the Office365 Groups Mobile app.

For information on admin controls, look at the FAQ section of this [support article](#).

Outlook version requirements for actionable messages

Actionable messages are available to all customer mailboxes on Exchange Online in Office 365 or Outlook.com with a supported client. The following table lists the availability of actionable messages for current Outlook clients. For information on the Office 365 release channels, see [Overview of update channels for Office 365 ProPlus](#).

NOTE

Currently actionable message cards do not change the way that they render when Outlook is in dark mode. Support for dark mode for actionable messages is coming soon.

CLIENT	ACTIONABLE MESSAGES SUPPORTED?	ADAPTIVE CARD SUPPORTED?
Outlook on the web for Office 365	Yes	Yes
Office 365 ProPlus Monthly Channel	Yes, in version 1705, Build 8201	Yes, in version 1805, Build 9330
Office 365 ProPlus Semi-Annual Channel (Targeted)	Yes, in version 1708, Build 8431	Yes, in version 1808, Build 10730.20262
Office 365 ProPlus Semi-Annual Channel	Yes, in version 1708, Build 8431.2153	Yes, in version 1808, Build 10730.20264
Outlook 2016 on Mac	Yes, in version 16.38, Build 20060702	Yes (Legacy MessageCard format is not supported)
Outlook on iOS	Yes	Yes (Legacy MessageCard format is not supported)
Outlook on Android	Yes	Yes (Legacy MessageCard format is not supported)
Office Professional Plus 2016 (Click-to-Run only)	Actionable Messages are available for Office 365 only	No
Exchange 2016 On-Premises Outlook on the web	Actionable Messages are available for Office 365 only	No

Submit feedback

There are multiple ways you can send us feedback.

- The in-product **Send Feedback** link (preferred)
- If you have a question, need help, or are experiencing an issue with your code, ask the developer community on [Microsoft Q&A](#).
- You can also post questions to [Stack Overflow](#) - Tag your questions with `Office365Connectors`.
- If you have a feature suggestion, please post your idea on our [Microsoft 365 Developer Platform Ideas](#) forum, and vote for your suggestions there.

Get started with actionable messages in Office 365

4/18/2022 • 2 minutes to read • [Edit Online](#)

Actionable messages can be posted via a group or inbox connector, or can be sent directly over email. Choosing the right delivery mechanism depends on your scenario.

NOTE

Office 365 administrators can disable actionable messages via the [Set-OrganizationConfig cmdlet](#). If actionable messages do not render, check with your administrator to make sure the feature is enabled in your organization.

Connectors vs Email: Choosing a delivery mechanism

With Office connectors, any user can choose to connect to services like Trello, Bing News, Twitter, etc., from Outlook and get notified of activity from that service into their Office 365 inbox or Group. With actionable messages for Office connectors, user can now act on these notifications to complete routine tasks without the hassle of context switching or signing in. For example the Trello connector allow users to subscribe to the boards and notifications they care about and lets them take actions such as set a due date or add a comment without ever leaving Outlook.

On the other hand, 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.

The two approaches may not be necessarily mutually exclusive, as you may choose to use both solutions for enabling actionable messages from your service. For example, a CRM system may generate an approval email for authorizing a sales discount by a manager, thus enabling an actionable message via email. Consider other scenarios where users or teams want to be notified of important changes to opportunities, leads or accounts they manage. In these cases, a user may update the stage of an opportunity or lead, or set the close date depending on the notification they get. Office connectors are a great way to accomplish these scenarios.

Once you've decided which delivery method is best for your scenario, you're ready to start trying some examples.

- [Designing Actions and Inputs](#)
- [Actionable Messages via Email](#)
- [Actionable Messages via Connectors](#)

Questions

For questions prior to signing up or for help, please send email to onboardoam@microsoft.com.

Send an actionable message via email in Office 365

4/18/2022 • 4 minutes to read • [Edit Online](#)

Supported scenarios

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.

Create an actionable message card

Let's start by creating an actionable message card. We'll start with something simple, just a basic card with an

`Action.Http`

action and an

`Action.OpenUrl`

action. We'll use the [Card Playground](#) to design the card.

IMPORTANT

The sample card markup in this topic omits the `originator` property. This works in a testing scenario, where the recipient is the same as the sender. When sending actionable messages to anyone else, the `originator` property must be set to a valid provider ID generated by the [Actionable Email Developer Dashboard](#). Leaving this property empty when sending to others results in the card being removed.

Go to the Card Playground and paste in the following JSON:

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "Visit the Outlook Dev Portal",
      "size": "large"
    },
    {
      "type": "TextBlock",
      "text": "Click **Learn More** to learn more about Actionable Messages!"
    },
    {
      "type": "Input.Text",
      "id": "feedbackText",
      "placeholder": "Let us know what you think about Actionable Messages"
    }
  ],
  "actions": [
    {
      "type": "Action.Http",
      "title": "Send Feedback",
      "method": "POST",
      "url": "https://...",
      "body": "{{feedbackText.value}}"
    },
    {
      "type": "Action.OpenUrl",
      "title": "Learn More",
      "url": "https://docs.microsoft.com/outlook/actionable-messages"
    }
  ]
}
```

Feel free to experiment with this simple example in the playground. You can see the [adaptive card reference](#) for details on the available fields. Once you have a card you're happy with, you can move on to sending it.

Sending actionable messages via email

IMPORTANT

You can design and test actionable messages by using the [Card Playground](#), which allows you to send actionable messages to yourself. You can also send actionable messages to yourself using the [Office 365 SMTP server](#). You will be unable to send actionable messages to any other user until you have registered using the [actionable messages developer dashboard](#).

To embed an actionable message card in an email message, we need to wrap the card in a `<script>` tag. The `<script>` tag is then inserted into the `<head>` of the email's HTML body.

NOTE

Because the card JSON must be wrapped in a `<script>` tag, the body of the actionable message email MUST be HTML. Plain-text messages are not supported.

1. Add the `hideOriginalBody` attribute to control what happens with the body of the email. In this case we'll set the attribute to `true` so that the body will not be shown.

```

{
  "type": "AdaptiveCard",
  "version": "1.0",
  "hideOriginalBody": true,
  "body": [
    {
      "type": "TextBlock",
      "text": "Visit the Outlook Dev Portal",
      "size": "large"
    },
    {
      "type": "TextBlock",
      "text": "Click Learn More to learn more about Actionable Messages!"
    },
    {
      "type": "Input.Text",
      "id": "feedbackText",
      "placeholder": "Let us know what you think about Actionable Messages"
    }
  ],
  "actions": [
    {
      "type": "Action.Http",
      "title": "Send Feedback",
      "method": "POST",
      "url": "https://...",
      "body": "{{feedbackText.value}}"
    },
    {
      "type": "Action.OpenUrl",
      "title": "Learn More",
      "url": "https://docs.microsoft.com/outlook/actionable-messages"
    }
  ]
}

```

2. Wrap the resulting JSON in a `<script>` tag of type `application/adaptivecard+json`.

NOTE

If you are using the [legacy message card format](#) rather than the Adaptive card format, the `<script>` tag type MUST be `application/ld+json`.

```

<script type="application/adaptivecard+json">{
  "type": "AdaptiveCard",
  "version": "1.0",
  "hideOriginalBody": true,
  "body": [
    {
      "type": "TextBlock",
      "text": "Visit the Outlook Dev Portal",
      "size": "large"
    },
    {
      "type": "TextBlock",
      "text": "Click **Learn More** to learn more about Actionable Messages!"
    },
    {
      "type": "Input.Text",
      "id": "feedbackText",
      "placeholder": "Let us know what you think about Actionable Messages"
    }
  ],
  "actions": [
    {
      "type": "Action.Http",
      "title": "Send Feedback",
      "method": "POST",
      "url": "https://...",
      "body": "{{feedbackText.value}}"
    },
    {
      "type": "Action.OpenUrl",
      "title": "Learn More",
      "url": "https://docs.microsoft.com/outlook/actionable-messages"
    }
  ]
}
</script>

```

3. Generate an HTML document to represent the email body and include the `<script>` tag in the `<head>` .

```

<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <script type="application/adaptivecard+json">{
    "type": "AdaptiveCard",
    "version": "1.0",
    "hideOriginalBody": true,
    "body": [
      {
        "type": "TextBlock",
        "text": "Visit the Outlook Dev Portal",
        "size": "large"
      },
      {
        "type": "TextBlock",
        "text": "Click **Learn More** to learn more about Actionable Messages!"
      },
      {
        "type": "Input.Text",
        "id": "feedbackText",
        "placeholder": "Let us know what you think about Actionable Messages"
      }
    ],
    "actions": [
      {
        "type": "Action.Http",
        "title": "Send Feedback",
        "method": "POST",
        "url": "https://...",
        "body": "{{feedbackText.value}}"
      },
      {
        "type": "Action.OpenUrl",
        "title": "Learn More",
        "url": "https://docs.microsoft.com/outlook/actionable-messages"
      }
    ]
  }
</script>
</head>
<body>
  Visit the <a href="https://docs.microsoft.com/outlook/actionable-messages">Outlook Dev Portal</a> to
  learn more about Actionable Messages.
</body>
</html>

```

4. Send a message via SMTP with the HTML as the body.

Sending the message

For examples of sending messages, see the following.

- [Send Actionable Message via Microsoft Graph](#): A sample console app written in C# that sends an actionable message using [Microsoft Graph](#).
- [Send Actionable Message via SMTP](#): A sample Python script that sends an actionable message using the Office 365 SMTP server. It also includes a sample HTML payload for the actionable message email body.

Perform actions

For examples of performing actions, see the following.

- [Hello Actionable Messages](#): A sample project with one-click button Azure deployment. This sample is a simple end-to-end actionable message solution that can be up and working within 10 minutes, and serves as a reference for building a production action endpoint.

Troubleshooting tools

- [Actionable Messages Debugger](#): an Outlook add-in that allows developers to inspect the card payload in their actionable messages and identify why the card is not rendering.

Post an actionable message card to an Office 365 group

4/18/2022 • 2 minutes to read • [Edit Online](#)

Create an actionable message card

Let's start by creating an actionable message card. We'll start with something simple, just a basic card with an `HttpPost` action and an `OpenUri` action. We'll use the [Card Playground](#) to design the card.

IMPORTANT

Office connectors only support the [legacy MessageCard format](#) for cards. They do not support the Adaptive Card format.

Go to [Card Playground](#) and paste in the following JSON:

```
{
  "@context": "https://schema.org/extensions",
  "@type": "MessageCard",
  "themeColor": "0072C6",
  "title": "Visit the Outlook Dev Portal",
  "text": "Click Learn More to learn more about Actionable Messages!",
  "potentialAction": [
    {
      "@type": "ActionCard",
      "name": "Send Feedback",
      "inputs": [
        {
          "@type": "TextInput",
          "id": "feedback",
          "isMultiline": true,
          "title": "Let us know what you think about Actionable Messages"
        }
      ],
      "actions": [
        {
          "@type": "HttpPost",
          "name": "Send Feedback",
          "isPrimary": true,
          "target": "http://..."
        }
      ]
    },
    {
      "@type": "OpenUri",
      "name": "Learn More",
      "targets": [
        { "os": "default", "uri": "https://docs.microsoft.com/outlook/actionable-messages" }
      ]
    }
  ]
}
```

Feel free to experiment with this simple example in the playground. You can see the [message card reference](#) for details on the available fields. Once you have a card you're happy with, you can move on to sending it.

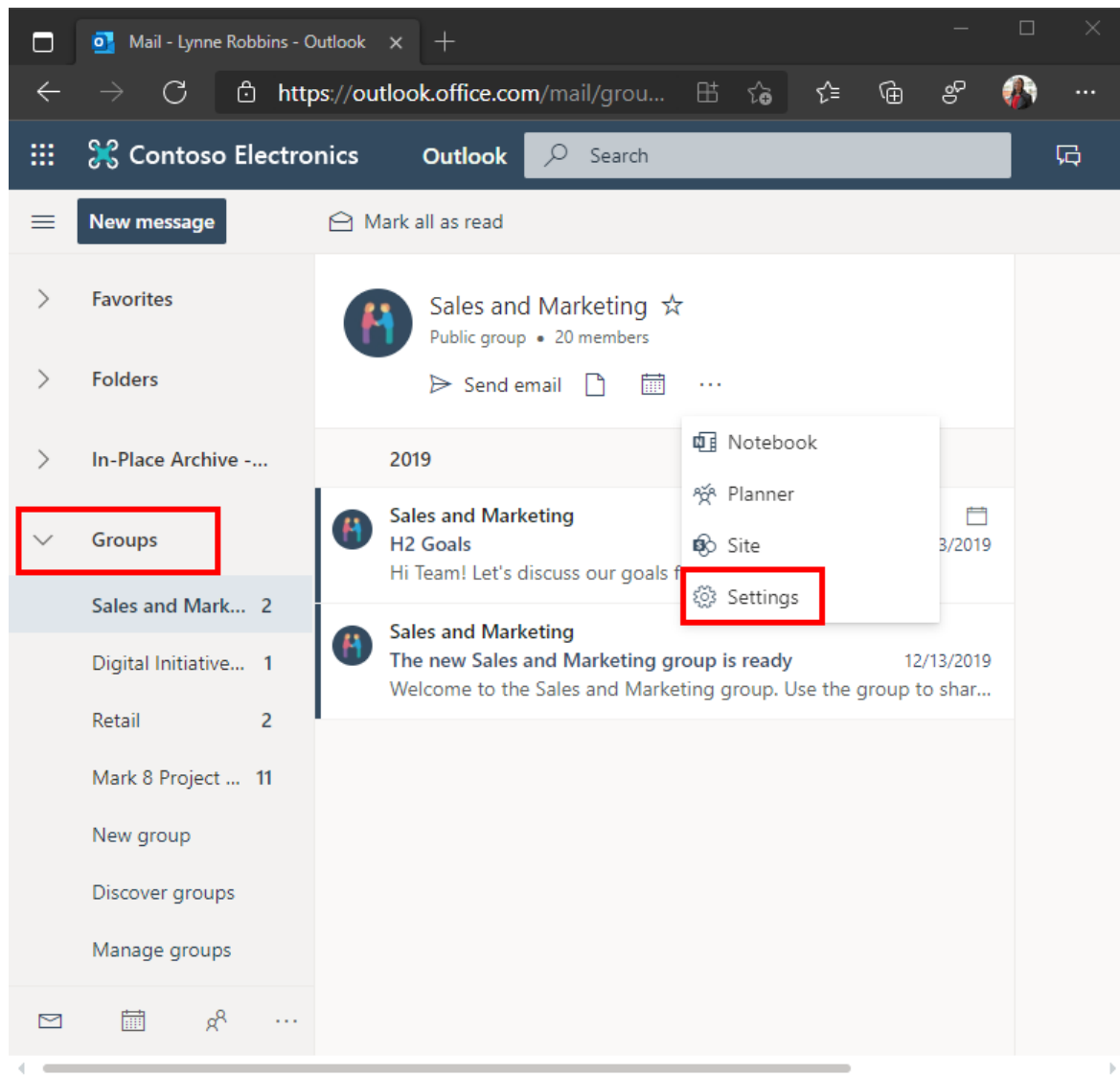
Sending actionable messages via Office connectors

Connectors use webhooks to create Connector Card messages within an Office 365 group. Developers can create these cards by sending an HTTP request with a simple JSON payload to an Office 365 group webhook address. Let's try posting some basic cards to a group.

You'll need an Microsoft 365 subscription to proceed. If you do not have an Microsoft 365 subscription, you can get a Microsoft 365 developer subscription from the [Microsoft 365 Developer Program](#).

Get a connector webhook URL for a Microsoft 365 Group

1. Log on to Outlook on the web at <https://outlook.office.com>.
2. In the folder list, select a group under the **Groups** section. In the group's menu, select the ellipses (...), then select **Settings**.



3. In the **Group Settings** pane, select **Connectors**.
4. Locate and select the **Incoming Webhook** connector in the list of available connectors.

Connectors for 'Sales and M...

[Build a Connector](#)
[Send feedback](#)

Keep your group current with content and updates from other services.

Incoming


Search Results

Sort by: Popularity ▾

MANAGE

Configured

My Accounts



Incoming Webhook

Send data from a service to your Office 365 group in real time.

Add

5. Enter a name for this connector and choose **Create**.

6. Copy the webhook URL that is displayed and save it. Choose **Done**.

Copy the URL below to save it to the clipboard, then select Save. You'll need this URL when you go to the service that you want to send data to your group.

`https://outlook.office365.com/webhook/.`



Done

The webhook URL should look similar to the following:

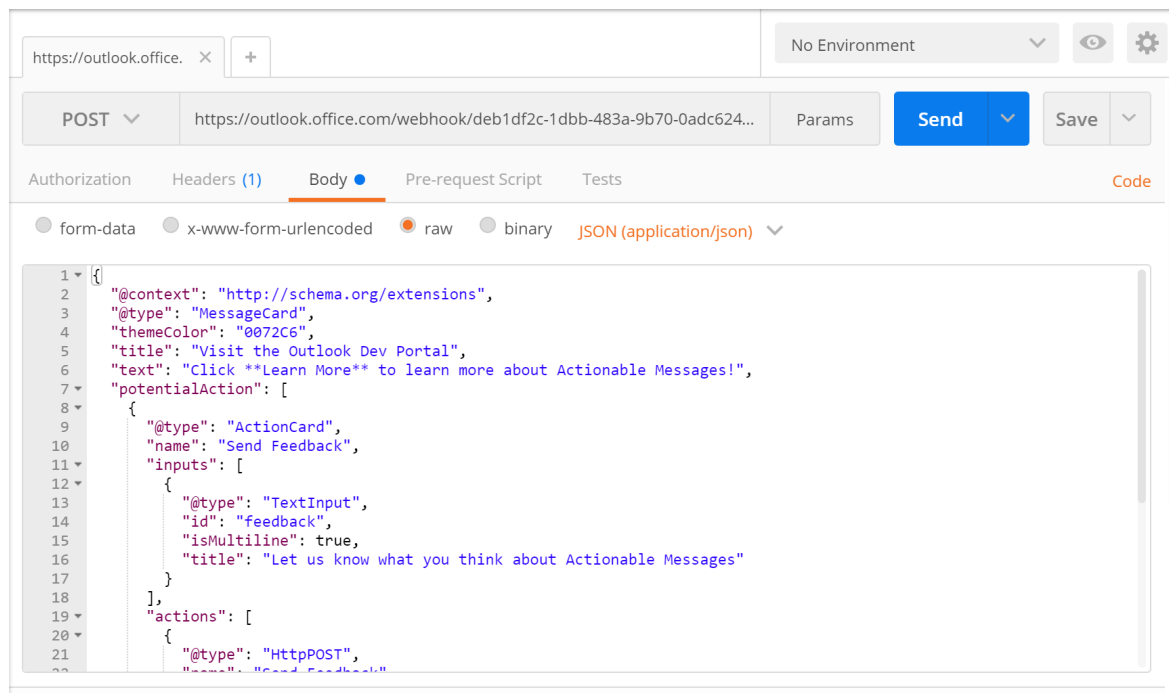
```
https://outlook.office365.com/webhook/a1269812-6d10-44b1-abc5-b84f93580ba0@9e7b80c7-d1eb-4b52-8582-76f921e416d9/IncomingWebhook/3fdd6767bae44ac58e5995547d66a4e4/f332c8d9-3397-4ac5-957b-b8e3fc465a8c
```

Send the message

Use [Postman](#) to post an actionable message payload to the webhook URL. Open Postman. Create a new tab if needed and configure the tab as follows:

1. Click the **GET** and change to **POST**.
2. In the text box labeled `Enter request URL` paste the webhook URL.
3. Click **Body** underneath the URL, then select the **raw** option.
4. Click **Text** and change to **JSON (application/json)**.
5. Enter the message card JSON in the text area below.

The Postman window should look like this when you are done:



6. Click **Send** to post the message.

Link Azure Active Directory identity with your own identity provider (Preview)

4/18/2022 • 2 minutes to read • [Edit Online](#)

Action.Http actions in actionable messages include an Azure AD-issued token in the **Authorization** header, which provides information about the user's identity. However, this information may not be sufficient to authenticate the user to your service. With identity linking, you can signal the Outlook client to present UI to allow the user to authenticate with your service. Once the user authenticates, you can associate their Azure AD identity with your own to allow for seamless authentication for future requests.

Using identity linking

Your service can trigger authentication on any **Action.Http** action endpoint by returning a **401 Unauthorized** response with a **ACTION-AUTHENTICATE** header. The header contains the authentication URL for your service.

Once authentication is completed, redirect the request to the URL specified in the **Identity-Linking-Redirect-Url** header sent in the original request.

Identity linking flow

Initial request to your action endpoint

Microsoft servers send an initial POST request to your action endpoint.

```
POST https://api.contoso.com/myEndpoint
Authorization: Bearer eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiIsImtpZCI6...
Identity-Linking-Redirect-Url: https://outlook.office.com/connectors/adelev@contoso.com/723a1c49-f8dc-4063-843e-d4c2b7180b8b/postAuthenticate
Content-Type: application/json

{
  // action body
}
```

Your service [validates the JWT token](#) and extracts the user's identity from the **sub** claim.

```
{
  ...
  "sub": "AdeleV@contoso.com",
  "aud": "https://api.contoso.com"
}
```

Your service finds no user with a linked identity of **AdeleV@contoso.com** so it persists **Identity-Linking-Redirect-Url** header value and returns a **401** response.

NOTE

The exact method you use to persist the redirect URL from the **Identity-Linking-Redirect-Url** header is dependent on your implementation. If your service uses OAuth, you may save it in the **state** parameter, for example.

```
HTTP/1.1 401 Unauthorized
ACTION-AUTHENTICATE: https://identity.contoso.com/authenticate?
state=https://outlook.office.com/connectors/adelev@contoso.com/723a1c49-f8dc-4063-843e-
d4c2b7180b8b/postAuthenticate
```

Authentication request

After Outlook receives the `401` with the `ACTION-AUTHENTICATE` header, it will open a task pane and navigate to the URL from the header.

```
GET https://identity.contoso.com/authenticate?
state=https://outlook.office.com/connectors/adelev@contoso.com/723a1c49-f8dc-4063-843e-
d4c2b7180b8b/postAuthenticate
```

Your service authenticates the user and associates the identity provided by the Azure AD-issued token with the user in your system. Once complete, the service redirects the request to the URL from the `Identity-Linking-Redirect-Url` header.

```
HTTP/1.1 302 Found
Location: https://outlook.office.com/connectors/adelev@contoso.com/723a1c49-f8dc-4063-843e-
d4c2b7180b8b/postAuthenticate
```

Retry action

After Outlook receives the redirect back from your authentication server, it immediately retries the original request. This time, because you've associated the Azure AD identity with your own, your endpoint processes the request normally.

Example

You can use the following sample card in the [Card Playground](#) to see this in action. The endpoint in this card will prompt you to login to the Microsoft identity platform and (with your consent) will make a Graph request to [get your profile](#). The code for this endpoint is available as a sample on [GitHub](#).

```

{
  "hideOriginalBody": true,
  "type": "AdaptiveCard",
  "padding": "none",
  "body": [
    {
      "type": "TextBlock",
      "text": "Identity Linking Demo"
    },
    {
      "type": "ActionSet",
      "actions": [
        {
          "type": "Action.Http",
          "method": "POST",
          "url": "https://amidentitylinking.azurewebsites.net/action",
          "body": "{}",
          "title": "Get User Details",
          "isPrimary": true
        }
      ]
    }
  ],
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "version": "1.0"
}

```

Client support roadmap

Identity linking is available to a limited set of clients, with support for the feature being added in the future. The following table provides the approximate timeline.

CLIENT	AVAILABILITY
Office 365 ProPlus	Available
Outlook on the web for Office 365	Coming soon
Outlook on iOS	Coming soon
Outlook on Android	Coming soon
Outlook on Mac	TBD

Resources

- [Security requirements for actionable messages in Office 365](#)
- [Designing Outlook Actionable Message cards with the Adaptive Card format](#)
- [Sample identity linking service](#)

Invoke an Outlook add-in from an actionable message

4/18/2022 • 3 minutes to read • [Edit Online](#)

Actionable messages allow the user to take quick actions on an email message or connector card, and Outlook add-ins allow you to extend Outlook to add new features and interactions. Now, with the

`Action.InvokeAddInCommand` action type, you can combine these two types of integrations to create more powerful and compelling experiences. For example, you could:

- Send a welcome message as an actionable email message to new users after they sign up for your service, with an action that allows them to quickly install and start using your add-in.
- Use an add-in for more complex actions (i.e. to present a form to the user), for scenarios where a simple action input would not suffice.
- Pre-populate UI elements in your add-in before presenting UI to the user.

`Action.InvokeAddInCommand` actions can work with add-ins that are already installed by the user, or they can work with add-ins that are not installed. If the required add-in is not installed, the user is prompted to install the add-in with a single click.

NOTE

Single-click installation of the required add-in is only supported if the add-in is published in [AppSource](#).

The following example shows the prompt users see if the add-in is not installed.



Turn Add-in on?

This action requires an add-in to be turned on:



Message Header Analyzer

By [Stephen Griffin](#)

This Add-in may access your personal data. By turning on this add-in, you agree to its [License terms](#) and [Privacy statement](#). You can turn it off at any time from the Add-in Store.

Turn On

Not now

Invoking the add-in

Actionable messages invoke add-ins by specifying an [Action.InvokeAddInCommand](#) action in the message. This action specifies the add-in to invoke, along with the identifier of the add-in button that opens the appropriate task pane.

The required information is found in the [add-in's manifest](#). First, you'll need the add-in's identifier, which is specified in the [Id element](#).

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<OfficeApp
  xmlns="http://schemas.microsoft.com/office/appforoffice/1.1"
  xmlns:xsi="https://www.w3.org/2001/XMLSchema-instance"
  xmlns:bt="http://schemas.microsoft.com/office/officeappbasictypes/1.0"
  xsi:type="MailApp">
  <Id>527104a1-f1a5-475a-9199-7a968161c870</Id>
  <Version>1.0.0.0</Version>
  ...
</OfficeApp>
```

For this add-in, the add-in identifier is `527104a1-f1a5-475a-9199-7a968161c870`.

Next, you'll need the `id` attribute of the [Control element](#) that defines the add-in button that opens the appropriate task pane. Keep in mind that the `Control` element MUST:

- Be defined inside a [MessageReadCommandSurface extension point](#)
- Have its `xsi:type` attribute set to `Button`

- Contain an [Action element](#) of type `ShowTaskpane`

```
<ExtensionPoint xsi:type="MessageReadCommandSurface">
  <OfficeTab id="TabDefault">
    <Group id="msgReadCmdGroup">
      <Label resid="groupLabel"/>
      <Control xsi:type="Button" id="showInitContext">
        <Label resid="readButtonLabel"/>
        <Supertip>
          <Title resid="readButtonTitle"/>
          <Description resid="readButtonDesc"/>
        </Supertip>
        <Icon>
          <bt:Image size="16" resid="icon-16"/>
          <bt:Image size="32" resid="icon-32"/>
          <bt:Image size="80" resid="icon-80"/>
        </Icon>
        <Action xsi:type="ShowTaskpane">
          <SourceLocation resid="readPaneUrl"/>
          <SupportsPinning>true</SupportsPinning>
        </Action>
      </Control>
    </Group>
  </OfficeTab>
</ExtensionPoint>
```

For this add-in button, the ID is `showInitContext`.

With these two pieces of information, we can create a basic `Action.InvokeAddInCommand` action as follows:

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "Invoking an add-in command from an Actionable Message card",
      "size": "large"
    }
  ],
  "actions": [
    {
      "type": "Action.InvokeAddInCommand",
      "title": "Invoke \"View Initialization Context\"",
      "addInId": "527104a1-f1a5-475a-9199-7a968161c870",
      "desktopCommandId": "showInitContext"
    }
  ]
}
```

Passing initialization data to the add-in

The `Action.InvokeAddInCommand` action can also provide additional context to the add-in, allowing the add-in to do more than just simply activate. For example, the action could provide initial values for a form, or provide information that allows the add-in to "deep link" to a specific item in your back-end service.

In order to pass initialization data, include an `initializationContext` property in the `Action.InvokeAddInCommand` action. There is no set schema for the `initializationContext` property, you can include any valid JSON object.

For example, to extend the sample action from above, we could modify the action as follows:

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "Invoking an add-in command from an Actionable Message card",
      "size": "large"
    }
  ],
  "actions": [
    {
      "type": "Action.InvokeAddInCommand",
      "title": "Invoke \\\"View Initialization Context\\\"",
      "addInId": "527104a1-f1a5-475a-9199-7a968161c870",
      "desktopCommandId": "showInitContext",
      "initializationContext": {
        "property1": "Hello world",
        "property2": 5,
        "property3": true
      }
    }
  ]
}
```

Receiving initialization data in the add-in

If your action passes initialization data, the add-in must be prepared to receive it. Add-ins can retrieve initialization data by calling the [Office.context.mailbox.item.getInitializationContextAsync](#) method. This should be done whenever the task pane opens or loads a new message.

```
// Get the initialization context (if present)
Office.context.mailbox.item.getInitializationContextAsync(
function(asyncResult) {
  if (asyncResult.status == Office.AsyncResultStatus.Succeeded) {
    if (asyncResult.value != null && asyncResult.value.length > 0) {
      // The value is a string, parse to an object
      var context = JSON.parse(asyncResult.value);
      // Do something with context
    } else {
      // Empty context, treat as no context
    }
  } else {
    if (asyncResult.error.code == 9020) {
      // GenericResponseError returned when there is
      // no context
      // Treat as no context
    } else {
      // Handle the error
    }
  }
}
);
```

Resources

- [Outlook-Add-In-Actionable-Message sample add-in](#)

Refresh an actionable message when the user opens it

4/18/2022 • 3 minutes to read • [Edit Online](#)

Actionable messages allow users to take quick actions on an email message, often based on data presented in the card. However, sometimes data changes after the actionable message has been sent. For example, your service might send an actionable message to multiple approvers asking them to approve or reject a request. One approver approves the request, but the actionable message in the other approver's mailbox still asks for approval. Now, with the `autoInvokeAction` property on actionable messages, you can provide an HTTP endpoint to retrieve an up-to-date Adaptive Card payload with the latest information when the user opens the email in Outlook.

IMPORTANT

Refreshing the actionable message when the user opens it has a direct impact on the perceived performance of your actionable message solution. It is crucial that your service that supplies the updated card meet the performance requirements described in [Implementing the Web API](#).

Registration requirements

Actionable Messages services registered in the [developer dashboard](#) with the **Test Users** or **Organization** scope can use this feature as soon as they are approved. If your service is registered with the **Global** scope, you must contact onboardoam@microsoft.com to enable this feature.

Using autoInvokeAction

In order to use this feature, your card must use the [Adaptive Card](#) format. The `autoInvokeAction` property is an Outlook-specific property added to the `AdaptiveCard` type. The value of this property is an `Action.Http` action with the `method` set to `POST`. The `url` property specifies a Web API endpoint in your service that will provide the updated Adaptive Card payload.

```
{
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [...],
  "actions": [...],
  "autoInvokeAction": {
    "method": "POST",
    "url": "https://actionablemessages.contoso.com/api/getupdatedcard",
    "body": "",
    "type": "Action.Http"
  }
}
```

Crafting the initial Adaptive Card

When using `autoInvokeAction`, it is very important that the initial card included with the message still be valuable to the end user and, ideally, be actionable. The `autoInvokeAction` might fail, or network conditions might slow it down, in which case the initial card is all the end user will see.

- **Do not** send an empty initial card with just an `autoInvokeAction`. Such cards will be rejected by the Actionable Message platform.
- **Do not** send an initial card that is just a placeholder asking the user to wait.
- **Do** send an initial card that presents the information in its current state, even if that means the user might see outdated data by the time they open your message.

Implementing the Web API

The `Action.Http` action specified in the `autoInvokeAction` property works exactly the same as any other `Action.Http` action. For details on implementing the endpoint for this action, see [Implementing the Web API](#).

The endpoint for an `autoInvokeAction` must also meet the following additional requirements.

- Requests **must return within 2 seconds**.
- Requests that take longer will be ignored by the client, and the original card will continue to display. The message will still be updated on the server.
- Successful responses should include a `CARD-UPDATE-IN-BODY` header with value `true` and an Adaptive Card JSON payload.

On success, the Adaptive Card returned will completely replace the existing card in the email message. If the URL returns an error or times out, the existing card will continue to display.

Example approval scenario

Consider this example vacation request card generated by a leave request tool and sent to the employee's manager.

TIP

You can access the JSON and modify this card sample by selecting the **Vacation Approval** sample in the [Actionable Message Designer](#).

Vacation Approval

Vacation request by Collin Ballinger is pending for your approval



Collin Ballinger
Design Lead
Bangalore - O365

Leave Reason:

I need to visit my aunt in the hospital and so I need 2 days off.

From:

23.09.2019

To:

25.09.2019

Type of leave:

Paid Holiday

Employee's Leave Balance:

13

Approve

Decline

The card includes information to help the manager make a decision, including the dates requested and how much leave the employee has remaining. This information was accurate when the message was generated and sent. However, the data could change before the manager checks his email. For example, the employee might edit his request in the leave request tool to change the requested dates.

By including an `autoInvokeAction` property on the actionable message generated by the tool, your card with the original data is replaced by a new card with new data when the manager opens it. The URL specified in the `autoInvokeAction` returns the same JSON payload, with new values for the requested vacation dates.

Vacation Approval

Vacation request by Collin Ballinger is pending for your approval



Collin Ballinger
Design Lead
Bangalore - O365

Leave Reason:

I need to visit my aunt in the hospital and so I need 2 days off.

From:

25.09.2019

To:

27.09.2019

Type of leave:

Paid Holiday

Employee's Leave Balance:

13

Approve

Decline

Security requirements for actionable messages in Office 365

4/18/2022 • 6 minutes to read • [Edit Online](#)

Securing actionable email is simple and easy. There are two phases within the end-to-end experience that impose security requirements on your service when supporting actionable messages with Office 365. The phases and their corresponding requirements are as follows.

1. Send phase: The pre-requisites for your service to send actionable messages are as follows:
 - If you're using actionable email, you'll need to enable [sender verification](#). Note that this does not apply to connector messages.
 - Your service must be registered with Microsoft.
 - The Action URL must support HTTPS.
2. Action processing phase: When processing an action, your service should:
 - Verify the bearer token (a JSON Web token) included in the header of the HTTP POST request. Verification can also be done leveraging the sample libraries provided by Microsoft.
 - Include Limited Purpose Token from your service as part of the target URL, which can be used by your service to correlate the service URL with the intended request & user. This is optional, but highly recommended.

Sender verification

Office 365 requires sender verification in order to enable actionable messages via email. Your actionable message emails must either originate from servers that implement DomainKeys Identified Mail (DKIM) and Sender Policy Framework (SPF), or you must implement [signed cards](#).

While DKIM and SPF are sufficient for some scenarios, that solution will not work in some situations where emails are sent via an external provider, which can lead to recipients not experiencing the enhanced actionable message. For this reason, we recommend always implementing signed cards which work in all cases and are fundamentally more secure since they do not rely on DNS records.

Implementing DKIM and SPF

DKIM and SPF are industry standard ways to prove a sender's identity when sending emails over SMTP. Many companies already implement these standards to secure the emails they are already sending. To learn more about SPF/DKIM and how to implement them, see:

- [DomainKeys Identified Mail \(DKIM\)](#)
- [Sender Policy Framework](#)

Signed card payloads

Actionable messages [sent via email](#) support an alternative verification method: signing the card payload with an RSA key or X509 certificate. This method is required in the following scenarios:

- SPF/DKIM failure caused by sender setup or recipient tenant set custom security services in front of Office 365 services.
- Your scenario for actionable messages requires sending from multiple email accounts.

To use signed cards, you must register your public key in the [email developer dashboard](#), and use the corresponding private key to sign the card.

SignedCard

Signed actionable message cards are available when sending via email. Use this format to include a signed card in the HTML body of an email. This payload is serialized in Microdata format appended in the end of HTML body.

```
<section itemscope itemtype="http://schema.org/SignedAdaptiveCard">
  <meta itemprop="@context" content="http://schema.org/extensions" />
  <meta itemprop="@type" content="SignedAdaptiveCard" />
  <div itemprop="signedAdaptiveCard" style="mso-hide:all;display:none;max-height:0px;overflow:hidden;">
[SignedCardPayload]</div>
</section>
```

Note: Partners who prefer to use the legacy `MessageCard` entity may create a `SignedMessageCard` entity in place of a `SignedAdaptiveCard`.

SignedCardPayload

SignedCardPayload is a string encoded by JSON Web Signature (JWS) standard. [RFC7515](#) describes JWS, and [RFC7519](#) describes JSON Web Token (JWT). Given no claim is required in JWT, JWT libraries can be used to build JWS signature.

Note: The term "JWT" can be used interchangeably in practice. However, we prefer the term "JWS" here.

Here is an example of SignedCardPayload. The encoded Adaptive Card appears in the form of [header].[payload].[signature] as per JWS specification.

eyJhbGciOiJSUzI1NiIsInR5cCI6IkpXVCJ9.eyJzZw5kZXI0IjZzXzJ2aWNLWfJyY291bnRAY29udG9zby5jb20iLCJvcmlnaW5hdG9yIjoInJvJnJgWZyTmZzNhi00YTFiLWI4NGMTYTDiNWMTMk4NzkyIiwicmVjaXBpZW50cCI6ImlhbmhG16ZwQ0i0iJXBXCjQb2huQGvNbnRvc28uY29tXCIsXCjQyW5lQGNvbnRvc28uY29tXCjDIiwiYWRhchRpdmdVDYXJkU2VyaWFSaXplZCI6IntcIiRzY2h1bWFCIjpcImh0dHA6Ly9hZGFwdG12ZWNhcmRzLm1vL3NjaGVTYXMyVWRhchRpdmdUyY2FyZS5cQc29uXCIsXCj0eXB1XCi6XCjBZGFwdG12ZUNhcmRcIixcInZ1cnNpb25cIjpcIjEumFwiLWFiYm9keWV0I0t7XCjZaXplXCi6XCjSYXJnZWVwLWFiWdGV4dFwi0lwiSGVsbG8gQWw0aW9uYWJsZSBtZXNzYWdlXCiScXCj3cmFwXCI6dHJ1ZSxcInR5cGVcIjpcIiRleHRChG9ja1wiFv0sXCjY3Rpb25cXCi6W3tcInR5cGVcIjpcIkFjdGlvbi5JbnZva2VBZGRjRkbnVwbW1hbmRcIixcInRpdGx1XCi6XCjPcGVuIEFjdGlvbmFibGUGTWVzc2FnZXMGrgRgidwdnZXJcIixcImFkZEluSWRcIjpcIjNkMTQ0yWOGY2LWFiYjMtNGJhZi1hYW5kLWUyY2Q0NjdiYjBmYVwiLWFiZGVza3RvcENvbnB1hbmRjZWFwi0lwiYW1EZWJ1Z2d1ck9wZW5QYW5lQnV0dG9uXCj9XX0iLCJpYXQ0IjE1NDUzZDg0NTN9.9P9mk33S1VZyJtZgdZd-
LNTTjvucyeeoitygwp9b117TeQfTudh9K5c3bF7BeZyQs6Iiwa1VGRdiiR4q9EKAB1qDsmIcJnw6aYWdUZ1KY41NoYgQCH__FxEPHViGiDN6qt1vA6ED0w0iFaTUWtA5cF5MfiRBIhPQ530mbRnNa0QsRBYtyB54EDJxjBF1vNSKOeVHA12d4gqcGxsytQA0PA7XmbrZ8B7fEU2uNjsiLQpoh6A1tevp1a2C7W6h-Weksgsmjpw2YT0A0X67VZ1TcS50zAHmjv2RhqsfXSDlN-ZsTRERu4Hs5d92NY9ijjPduNSLyUFNCw7HLNPFqaPmZsw

The header in above JWS is:

```
{
  "alg": "RS256",
  "typ": "JWT"
}
```

The payload in above JWS is:


```
{
  "sender": "service-account@contoso.com",
  "originator": "65c680ef-36a6-4a1b-b84c-a7b5c6198792",
  "recipientsSerialized": "[\\\"john@contoso.com\\\", \\\"jane@contoso.com\\\"]",
  "adaptiveCardSerialized": "{\\\"$schema\\\": \\\"http://adaptivecards.io/schemas/adaptive-card.json\\\", \\\"type\\\": \\\"AdaptiveCard\\\", \\\"version\\\": \\\"1.0\\\", \\\"body\\\": [{\\\"size\\\": \\\"large\\\", \\\"text\\\": \\\"Hello Actionable message\\\", \\\"wrap\\\": true, \\\"type\\\": \\\"TextBlock\\\"}], \\\"actions\\\": [{\\\"type\\\": \\\"Action.InvokeAddInCommand\\\", \\\"title\\\": \\\"Open Actionable Messages Debugger\\\", \\\"addInId\\\": \\\"3d1408f6-afb3-4baf-aacd-55cd867bb0fa\\\", \\\"desktopCommandId\\\": \\\"amDebuggerOpenPaneButton\\\"}]}\",
  \"iat\": 1545348153
}
```

Required Claims

CLAIM	DESCRIPTION
<code>originator</code>	MUST be set to the ID provided by Microsoft during onboarding.
<code>iat</code>	The time that the payload was signed.
<code>sender</code>	The email address used to send this actionable message.
<code>recipientsSerialized</code>	The stringified list of the recipients of the email. This should include all the To/CC recipients of the email.
<code>adaptiveCardSerialized</code>	The stringified Adaptive Card.

Sample code generating signed card:

- [.NET Sample](#)
- [Node.js Sample](#)

Verifying that requests come from Microsoft

All action requests from Microsoft have a bearer token in the HTTP `Authorization` header. This token is a [JSON Web Token](#) (JWT) token signed by Microsoft, and it includes important claims that we strongly recommend should be verified by the service handling the associated request.

CLAIM NAME	VALUE
<code>aud</code>	The base URL of the target service, e.g. <code>https://api.contoso.com</code>
<code>iss</code>	The issuer of the token. This should be <code>https://substrate.office.com/sts/</code>
<code>sub</code>	The identity of the user who took the action. For Actionable Messages sent over email, <code>sub</code> would be the email address of the user. For connectors, <code>sub</code> will be the objectID of the user who took the action.
<code>sender</code>	The identity of sender of the message containing the action. This value is only present if the actionable message was sent via email . Actionable messages sent via connectors do not include this claim in their bearer token.

Typically, a service will perform the following verifications.

1. The token is signed by Microsoft. OpenID metadata is located at `https://substrate.office.com/sts/common/.well-known/openid-configuration`, which includes information regarding signing keys.
2. The `aud` claim corresponds to the service's base URL.

With all the above verifications done, the service can trust the `sender` and `sub` claims to be the identity of the sender and the user taking the action. The service can validate that the `sender` and `sub` claims match the sender and user it is expecting.

Please refer to the Microsoft code samples provided below, which show how to do these validations on the JWT token.

- [.NET Sample](#)
- [Node.js Sample](#)
- [Java Sample](#)
- [Python Sample](#)

Action-Authorization header

The use of `Authorization` header by Actionable messages may interfere with existing authentication/authorization mechanism for the target endpoint. In this case, developers can set the `Authorization` header to `null` or an empty string in the `headers` property of an `Action.Http` action. Actionable messages will then send the same bearer token via `Action-Authorization` header instead of using `Authorization` header.

TIP

The Azure Logic App service returns `HTTP 401 Unauthorized` if the `Authorization` header contains the bearer token set by actionable messages.

Example Action.Http with Action-Authorization header

```
{
  "type": "Action.Http",
  "title": "Say hello",
  "method": "POST",
  "url": "https://api.contoso.com/sayhello",
  "body": "{{nameInput.value}}",
  "headers": [
    { "name": "Authorization", "value": "" }
  ]
}
```

Verifying the identity of the user

The bearer token included with all requests includes the Azure AD identity of the Office 365 user who took the action. If necessary, you can include your own token, specific to your service, in the URLs of your HTTP actions to represent the identity of the user in your system. Microsoft does not prescribe how the limited-access tokens should be designed or used by the service. This token is opaque to Microsoft, and is simply echoed back to the service.

Service-specific tokens can be used to correlate service URLs with specific messages and users. Microsoft recommends that if you use your own service-specific token, you include it as part of the action target URL, or in the body of the request coming back to the service. For example:

```
https://contoso.com/approve?requestId=abc&serviceToken=a1b2c3d4e5...
```

Service-specific tokens act as correlation IDs (for e.g. a hashed token using the userID, requestId, and salt). This allows the service provider to keep track of the action URLs it generates and sends out and match it with action requests coming in. In addition to correlation, the service provider may use the service-specific token to protect itself from replay attacks. For example, the service provider may choose to reject the request, if the action was already performed previously with the same token.

Designing Outlook Actionable Message cards with the Adaptive Card format

4/18/2022 • 24 minutes to read • [Edit Online](#)

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. In this topic we'll cover the Outlook-specific features of the Adaptive Card format.

IMPORTANT

The Adaptive Card format is only available for Actionable Messages sent via email, and is **required** to support Outlook on iOS and Android. The MessageCard format is still supported but is now de-emphasized. Office connectors and Microsoft Teams connectors do not currently support the Adaptive Card format. If you are implementing an Office 365 or Microsoft Teams connector, please refer to the [MessageCard format reference](#).

For information on which Outlook versions support the Adaptive Card format, see [Outlook version requirements for actionable messages](#).

Card Playground

Our [Card Playground tool](#) has been updated to support the Adaptive Card format. There you will find Adaptive Card samples (including the one below) that can help you get started crafting your own cards and also allows you to send those cards to your own Microsoft 365 email account to see how they look in Outlook.

Adaptive Cards Designer (preview)

The [Adaptive Cards Designer](#) provides a drag-and-drop experience to quickly build and tweak adaptive cards.

A simple Adaptive Card example

FLIGHT ITINERARY - CONTOSO AIR

Passenger: David Claux



2 Stops

Fri, October 10 8:30 AM

San Francisco

SFO



Amsterdam

AMS

Non-Stop

Fri, October 18 9:50 PM

Amsterdam

AMS



San Francisco

SFO

The above card illustrates some of the core and most powerful capabilities of the Adaptive Card format:

- The ability to stack elements of various types in any order
- The ability to control the amount of space between those elements
- The ability to layout elements in multiple columns
- The ability to align elements horizontally and vertically

Here is how this card is crafted:

```
{
  "$schema": "https://adaptivecards.io/schemas/adaptive-card.json",
  "version": "1.0",
  "type": "AdaptiveCard",
  "speak": "<s>Your flight is confirmed for you and 3 other passengers from San Francisco to Amsterdam on Friday, October 10 8:30 AM</s>",
  "body": [
    {
      "type": "ColumnSet",
      "columns": [
        {
          "width": "stretch",
          "verticalContentAlignment": "center",
          "items": [
            {
              "type": "TextBlock",
              "size": "medium",
              "text": "***FLIGHT ITINERARY - CONTOSO AIR**"
            },
            {
              "type": "TextBlock",
              "spacing": "none",
              "text": "Passenger: David Claux",
              "isSubtle": true
            }
          ]
        },
        {
          "width": "auto",
          "items": [
            {
              "type": "Image",
              "width": "48px",
              "url": "http://lh3.googleusercontent.com/ik5VKcUE5U7qGSpU3XWwAwe_zeOnHU5x_79o-VXf-C_EGrFHP4-NcKRctblrJM5i061=w300"
            }
          ]
        }
      ]
    },
    {
      "type": "TextBlock",
      "text": "2 Stops",
      "weight": "bolder",
      "spacing": "medium"
    },
    {
      "type": "TextBlock",
      "text": "Fri, October 10 8:30 AM",
      "weight": "bolder",
      "spacing": "none"
    },
    {
      "type": "ColumnSet",
      "separator": true,
      "columns": [
        {
```

```

        "type": "Column",
        "width": 1,
        "items": [
            {
                "type": "TextBlock",
                "text": "San Francisco",
                "isSubtle": true
            },
            {
                "type": "TextBlock",
                "size": "extraLarge",
                "color": "accent",
                "text": "SFO",
                "spacing": "none"
            }
        ]
    },
    {
        "type": "Column",
        "width": "auto",
        "items": [
            {
                "type": "TextBlock",
                "text": "&nbsp;"
            },
            {
                "type": "Image",
                "url": "https://messagecardplayground.azurewebsites.net/assets/airplane.png",
                "size": "small",
                "spacing": "none"
            }
        ]
    },
    {
        "type": "Column",
        "width": 1,
        "items": [
            {
                "type": "TextBlock",
                "horizontalAlignment": "right",
                "text": "Amsterdam",
                "isSubtle": true
            },
            {
                "type": "TextBlock",
                "horizontalAlignment": "right",
                "size": "extraLarge",
                "color": "accent",
                "text": "AMS",
                "spacing": "none"
            }
        ]
    }
],
},
{
    "type": "TextBlock",
    "text": "Non-Stop",
    "weight": "bolder",
    "spacing": "medium"
},
{
    "type": "TextBlock",
    "text": "Fri, October 18 9:50 PM",
    "weight": "bolder",
    "spacing": "none"
},
{
    "type": "ColumnSet",

```

```

"separator": true,
"columns": [
  {
    "type": "Column",
    "width": 1,
    "items": [
      {
        "type": "TextBlock",
        "text": "Amsterdam",
        "isSubtle": true
      },
      {
        "type": "TextBlock",
        "size": "extraLarge",
        "color": "accent",
        "text": "AMS",
        "spacing": "none"
      }
    ]
  },
  {
    "type": "Column",
    "width": "auto",
    "items": [
      {
        "type": "TextBlock",
        "text": "&nbsp;"
      },
      {
        "type": "Image",
        "url": "https://messagecardplayground.azurewebsites.net/assets/airplane.png",
        "size": "small",
        "spacing": "none"
      }
    ]
  },
  {
    "type": "Column",
    "width": 1,
    "items": [
      {
        "type": "TextBlock",
        "horizontalAlignment": "right",
        "text": "San Francisco",
        "isSubtle": true
      },
      {
        "type": "TextBlock",
        "horizontalAlignment": "right",
        "size": "extraLarge",
        "color": "accent",
        "text": "SFO",
        "spacing": "none"
      }
    ]
  }
]
}
]
}
}

```

Adaptive Card design tips

An Adaptive Card can be very simple or quite complex depending on the layout you wish to achieve. It is always a good idea to plan your design ahead of writing the Adaptive Card payload, using a paint tool for instance or even just pen and paper; this will make it a lot easier to translate visuals into the appropriate Adaptive Card

constructs. Below are a few design tips to help you get started.

Text formatting

All `TextBlock` elements in a card can be formatted using Markdown. Outlook supports basic Markdown.

IMPORTANT

Since all `TextBlock` elements are processed as Markdown, be sure to escape Markdown special characters (such as `*` or `#`) if needed.

EFFECT	MARKDOWN SYNTAX
Italics	<code>*This text is in italics*</code>
Bold	<code>**This text is bold**</code>
Bold + italics	<code>***This text is bold and in italics***</code>
Strike-through	<code>~~This text is struck through~~</code>
Link	<code>[Microsoft](http://www.microsoft.com)</code>
Headings (level 1 through 6)	<code># Heading</code> through <code>##### Heading</code>
Bulleted list	<code>* List item</code> or <code>- List item</code>

TIP

- Do use Markdown to format text.
- Don't use HTML markup in your cards. HTML is ignored and treated as plain text.

Design for a narrow screen

Just like when designing the HTML body of an email, you have to assume that your Adaptive Card might be displayed on both wide and narrow screens (e.g. a desktop and a mobile phone.)

TIP

- Do design your Adaptive Card in such a way that it looks great on a narrow screen. Typically, a card designed for a narrow screen will scale well to a wide screen. The opposite is however not true.
- Don't design your Adaptive Card assuming that only users of Outlook on the desktop will see it.

Craft your images with high DPI screens in mind

Not so long ago, most screens had a somewhat low resolution (1024x768 pixels for instance) and were operating at 96 DPI (Dots Per Inch), meaning that 96 pixels would fit within an actual inch of the screen. But in the past few years, screens have grown considerably in terms of resolution and DPI, especially on mobile devices, and it is now very common for a screen to operate at 192 DPI or even more.

When designing your Adaptive Cards, you need to make sure your images will look good on any screen regardless of its DPI.

TIP

- **Do** design your images assuming they will be displayed on a high DPI screen. An image designed for a low (96) DPI screen will be blown up when displayed on a higher DPI screen and will therefore look pixelated. An image designed for a high DPI screen will be shrunk on a lower DPI screen which usually yields good results. In other words, it is better to design a 100x100 pixels image and display it at 50x50 pixels than to design a 50x50 pixels image and display it at 100x100 pixels.
- **Do** use the `width` and `height` properties of the **Image** element if you need to precisely control the actual size of the images in your card.
- **Don't** design your images with a fixed background color, like white, unless that background color is supposed to be visible to the user. In Outlook, your Adaptive Cards will not necessarily be displayed on top of a white background, and your images should be able to superimpose themselves on top of any background color. For that reason, **do** make the background of your images transparent.
- **Don't** craft your images with built-in paddings. Such paddings usually interfere with the overall layout by introducing undesirable spacing on the side of your image.

Use of containers

Use the `Container` element only when necessary. The `Container` element makes it possible to group a set of elements together.

TIP

- **Do** use a `Container` to **emphasize a group of elements**: by setting the `style` property of the `Container` to `emphasis` you can make that `Container`, and the elements it contains, stand out.
- **Do** use a `Container` to **associate an action with group of elements**: by setting the `selectAction` property of a `Container`, the `Container` and its content become a single clickable area that triggers the specified action.
- **Do** use a `Container` to make a portion of your card collapsible: by using an `Action.ToggleVisibility` targeted to a `Container`, you can easily make a group of elements collapsible.
- **Don't** use `Container` for any other reason.

Use of columns

Use `ColumnSet` only when you need to align several elements on a single horizontal line.

TIP

- **Do** use `ColumnSet` for table-like layouts in general.
- **Do** use `ColumnSet` if you need to, for example, display an image of the far left of the card and some text on the same line at the far right of the card.
- **Do** use the appropriate sizing approach for columns:
 - Use `"width": "auto"` for a `Column` to use as much width as is necessary to fit its content.
 - Use `"width": "stretch"` for a `Column` to use the remaining width in the `ColumnSet`. When multiple `Columns` have `"width": "stretch"`, they all equally share the remaining width.
 - Use `"width": <number>` for a `Column` to use a proportion of the available width in the `ColumnSet`. If you have three columns with their `width` property set to `1`, `4` and `5` respectively, they will end up using 10%, 40% and 50% of the available width, respectively.
 - Use `"width": "<number>px"` to have a specific pixel width. This is particularly useful (and necessary) when creating table layouts.
- **Don't** use `ColumnSet` if all you need is stack elements vertically.

Outlook-specific Adaptive Card properties and features

Outlook introduces a set of additional Adaptive Card properties and features for use in the context of Actionable Messages.

IMPORTANT

Outlook-specific Adaptive Card properties and features **only work in the context of Actionable Messages**. They will NOT work in other Adaptive Card enabled applications and are therefore not documented on the [official Adaptive Cards site](#).

Adaptive Card features not supported with Outlook Actionable Messages

Action.Submit

The `Action.Submit` action type is **NOT** supported with Outlook Actionable Messages. If you include an `Action.Submit` in your card, it will not be displayed.

Input.Time

The `Input.Time` element type is **NOT** supported with Outlook Actionable Messages. If you include an `Input.Time` element in your card, it will not be displayed. If you need to allow users to input a time, use an `Input.Text` instead and validate its value server-side.

Action.Http

Outlook Actionable Messages use an HTTP-based action model via the `Action.Http` type. `Action.Http` makes it possible to make a GET or POST request to a specific target url as a result of a user taking an action in a card.

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>type</code>	String	Yes	Must be set to <code>Action.Http</code> .
<code>title</code>	String	No	The title of the action as it will appear on screen on a button control, for instance.
<code>method</code>	String	Yes	Valid values are <code>GET</code> and <code>POST</code> . When <code>method</code> is set to <code>POST</code> the <code>body</code> property must be specified.
<code>url</code>	String	Yes	The url of the request's target endpoint. The <code>url</code> property supports input value substitution . Note: this URL must be accessible from the internet, you cannot use <code>localhost</code> .
<code>headers</code>	Array of HttpHeader objects	No	An optional list of headers that should be sent to the target endpoint.
<code>body</code>	String	Only if <code>method</code> is set to <code>POST</code>	The body of the POST request. The <code>body</code> property supports input value substitution .

HttpHeader

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>name</code>	String	Yes	The name of the HTTP header. For example, <code>Content-Type</code> .
<code>value</code>	String	Yes	The value of the HTTP header. For example, <code>application/json</code> . The <code>value</code> property supports input value substitution .

Action.Http example

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "Hello world!"
    }
  ],
  "actions": [
    {
      "type": "Action.Http",
      "title": "Click me!",
      "method": "POST",
      "url": "https://contoso.com/api/...",
      "body": "<body of the POST request>",
      "headers": [
        { "name": "Content-Type", "value": "application/json" }
      ]
    }
  ]
}
```

Hello world!

Click me!

Implementing the Web API

The URL specified in the `url` property must conform to the following requirements.

- The endpoint must accept POST requests.
- The endpoint should accept the contents of the `body` property.
- The endpoint should use the JWT sent in the `Authorization` header to [verify that requests come from Microsoft](#).

Input value substitution

Adaptive Cards may contain inputs, and it may be necessary to pass the values of these inputs to the target endpoint via an `Action.Http` action. This is done using input value substitution. Consider the following example:

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "What's your name?"
    },
    {
      "type": "Input.Text",
      "id": "nameInput",
      "placeholder": "Type your name"
    }
  ],
  "actions": [
    {
      "type": "Action.Http",
      "title": "Say hello",
      "method": "GET",
      "url": "https://contoso.com/sayhello?name={{nameInput.value}}"
    }
  ]
}
```

What's your name?

Type your name

Say hello

The above card defines a text input and sets its `id` property to `nameInput`. It also defines an `Action.Http` action that makes a GET call to an endpoint on domain `contoso.com`. With the inclusion of `?name={{nameInput.value}}` on the target URL, the value of the input with `id` `nameInput` will be dynamically substituted at the time the action is taken by the user. So if the user had entered the name David in the text input, the target URL after substitution would be `https://contoso.com/sayhello?name=David`

Input value substitution also works in the body property of an `Action.Http` action. For example:

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "What's your name?"
    },
    {
      "type": "Input.Text",
      "id": "nameInput",
      "placeholder": "Type your name"
    }
  ],
  "actions": [
    {
      "type": "Action.Http",
      "title": "Say hello",
      "method": "POST",
      "url": "https://contoso.com/sayhello",
      "body": "{{nameInput.value}}"
    }
  ]
}
```

Reporting Action.Http execution success or failure

Your service should return an HTTP 200 status code when it successfully executes an `Action.Http` action. If the action execution fails, your service should return an HTTP 4xx status code, and it should also include the `CARD-ACTION-STATUS` HTTP header in its response to specify a custom error message. The value of that header will be displayed to the end-user in case the `Action.Http` fails to execute.

TIP

Follow these guidelines when returning a response to `Action.Http` actions.

- Do return the `CARD-ACTION-STATUS` header in your error responses.
- Do make the message in that header as informative and meaningful as possible.
- Don't mention either the name of the person taking the action nor the time the action is being taken in your `CARD-ACTION-STATUS` header.

Refresh 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. 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.

To refresh a card as a result of an `Action.Http` action, a service needs to do the following:

- Include the JSON payload of the new card in the body of the response to the HTTP POST request it received.
- Add the `CARD-UPDATE-IN-BODY: true` HTTP header to the response, in order to let the receiving client know that it should parse the response body and extract a new card (this is to avoid unnecessary processing when no refresh card is included.)

TIP

Follow these guidelines when returning refresh cards.

- **Do** use 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.
- **Do** 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.
- **Don't** use refresh cards to lead a conversation with the user. For instance, don't use refresh cards for a multi-step "wizard".
- **Do** include at least an `Action.OpenUrl` action to view the entity in the external app it comes from.

Action.InvokeAddInCommand

The `Action.InvokeAddInCommand` action opens an Outlook add-in task pane. If the add-in is not installed, the user is prompted to install the add-in with a single click.

When an `Action.InvokeAddInCommand` action is executed, Outlook first checks if the requested add-in is installed and turned on for the user. If it is not, the user is notified that the action requires the add-in, and is able to install and enable the add-in with a single click. Outlook opens the requested task pane, making any initialization context specified by the action available to the add-in.

For more information, see [Invoke an Outlook add-in from an actionable message](#).

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>type</code>	String	Yes	Must be set to <code>Action.InvokeAddInCommand</code> .
<code>title</code>	String	No	The title of the action as it will appear on screen on a button control, for instance.
<code>addInId</code>	String	Yes	Specifies the add-in ID of the required add-in. The add-in ID is found in the <code>Id</code> element in the add-in's manifest.

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>desktopCommandId</code>	String	Yes	Specifies the ID of the add-in command button that opens the required task pane. The command button ID is found in the <code>id</code> attribute of the Control element that defines the button in the add-in's manifest. The specified <code>Control</code> element MUST be defined inside a MessageReadCommandSurface extension point , be of type <code>Button</code> , and the control's <code>Action</code> must be of type <code>ShowTaskPane</code> .
<code>initializationContext</code>	Object	Yes	Developers may specify any valid JSON object in this field. The value is serialized into a string and made available to the add-in when the action is executed. This allows the action to pass initialization data to the add-in.

Action.DisplayMessageForm

The `Action.DisplayMessageForm` action opens the read form of a message given that message's ID. Message IDs can be retrieved via the [Outlook REST APIs](#).

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>type</code>	String	Yes	Must be set to <code>Action.DisplayMessageForm</code> .
<code>title</code>	String	No	The title of the action as it will appear on screen on a button control, for instance.
<code>itemId</code>	String	Yes	Specifies the ID of the message to open.

Action.DisplayAppointmentForm

The `Action.DisplayAppointmentForm` action opens the read form of a calendar item given that calendar item's ID. Calendar item IDs can be retrieved via the [Outlook REST APIs](#).

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>type</code>	String	Yes	Must be set to <code>Action.DisplayAppointmentForm</code> .

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>title</code>	String	No	The title of the action as it will appear on screen on a button control, for instance.
<code>itemId</code>	String	Yes	Specifies the ID of the calendar item to open.

Action.ToggleVisibility

The `Action.ToggleVisibility` action makes it possible to show and/or hide specific elements of a card as a result of a user clicking on a button or other actionable element. Coupled with the `isVisible` property, `Action.ToggleVisibility` allows for an extra degree of interactivity within a single card.

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>type</code>	String	Yes	Must be set to <code>Action.ToggleVisibility</code> .
<code>title</code>	String	No	The title of the action as it will appear on screen on a button control, for instance.
<code>targetElements</code>	Array of String or TargetElement	Yes	The list of elements that should have their visibility toggled. When elements of the <code>targetElements</code> array are specified as strings, they must represent the Id of an element in the card; when the action is executed, these elements become visible if they were not, and invisible otherwise. When elements of the array are specified as <code>TargetElement</code> objects, the visibility of each targeted element is defined by the <code>isVisible</code> property of the <code>TargetElement</code> object.

TargetElement

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>elementId</code>	String	Yes	The Id of the target element.
<code>isVisible</code>	Boolean	Yes	Specifies whether the target element should be visible once the action has completed.

Action.ToggleVisibility example


```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "***Action.ToggleVisibility example**": click the button to show or hide a welcome message"
    },
    {
      "type": "TextBlock",
      "id": "helloWorld",
      "isVisible": false,
      "text": "***Hello World!***",
      "size": "extraLarge"
    }
  ],
  "actions": [
    {
      "type": "Action.ToggleVisibility",
      "title": "Click me!",
      "targetElements": [ "helloWorld" ]
    }
  ]
}
```

The example card renders similar to the following before the button is clicked:

Action.ToggleVisibility example: click the button to show or hide a welcome message

Click me!

The example card renders similar to the following after the button is clicked:

Action.ToggleVisibility example: click the button to show or hide a welcome message

Hello World!

Click me!

ActionSet element

Outlook Actionable Messages add support for the `ActionSet` element that makes it possible to add action buttons anywhere in a card.

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>type</code>	String	Yes	Must be set to <code>ActionSet</code> .
<code>id</code>	String	No	The unique ID of the element.

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>spacing</code>	String	No	Controls the amount of spacing between this element and the previous element.
<code>separator</code>	Boolean	No	Controls whether or not a separator line should be displayed between this element and the previous element. The separator line is displayed in the middle of the space defined by the <code>spacing</code> property.
<code>horizontalAlignment</code>	String	No	Controls the horizontal alignment of this element within its container.
<code>actions</code>	Array of <code>Action</code> objects	No	The actions to be displayed in the set.

Aside from the fact that `ActionSet` can be placed anywhere in the card, it behaves exactly like the actions property of an `AdaptiveCard`.

ActionSet example

```
{
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "ActionSet",
      "actions": [
        {
          "type": "Action.ToggleVisibility",
          "title": "Click me!",
          "targetElements": [ "helloWorld" ]
        }
      ]
    },
    {
      "type": "TextBlock",
      "text": "***Action.ToggleVisibility example**: click the button above to show or hide a welcome message"
    },
    {
      "type": "TextBlock",
      "id": "helloWorld",
      "isVisible": false,
      "text": "***Hello World!**",
      "size": "extraLarge"
    }
  ]
}
```

Click me!

Action.ToggleVisibility example: click the button above to show or hide a welcome message

Additional properties on all Adaptive Card element types

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>isVisible</code>	Boolean	No. Defaults to true .	The initial visibility state of the element. When <code>isVisible</code> is set to <code>false</code> , the element is initially not visible in the card. It can be made visible using an <code>Action.ToggleVisibility</code> action, as documented above.

Please refer to the previous example for an illustration of how to use `isVisible`.

Additional properties on the AdaptiveCard type

The following additional properties can be specified on an [AdaptiveCard object](#) in the context of Outlook Actionable Messages:

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>autoInvokeAction</code>	Action.Http	No	The <code>autoInvokeAction</code> property specifies a URL that supplies an updated Adaptive Card payload to replace the existing payload in the message. The <code>method</code> of the <code>Action.Http</code> action MUST be <code>POST</code> . This allows your service to supply up-to-date information in the actionable message. For more information, see Refresh an actionable message when the user opens it .

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>correlationId</code>	String	No	<p>The <code>correlationId</code> property simplifies the process of locating logs for troubleshooting issues. We recommend that when sending an actionable card, your service should set and log a unique UUID in this property. When the user invokes an <code>Action.Http</code> action on the card, Office 365 sends the <code>Card-Correlation-Id</code> and <code>Action-Request-Id</code> headers in the POST request to your service. <code>Card-Correlation-Id</code> contains the same value as the <code>correlationId</code> property in the card. <code>Action-Request-Id</code> is a unique UUID generated by Office 365 to help locate specific action performed by a user. Your service should log both of these values when receiving action POST requests.</p>
<code>expectedActors</code>	Array of String	No	<p><code>expectedActors</code> contains a list of expected email addresses of the users that may take <code>Action.Http</code> actions on the card. A user can have multiple email addresses and the <code>Action.Http</code> target endpoint might not be expecting the particular email address presented in the <code>sub</code> claim of the bearer token. For example, a user could have both the <code>john.doe@contoso.com</code> or <code>john@contoso.com</code> email address, but the <code>Action.Http</code> target endpoint expects to receive <code>john@contoso.com</code> in the <code>sub</code> claim of the bearer token. By setting the <code>expectedActors</code> property to <code>["john@contoso.com"]</code>, the <code>sub</code> claim will have the expected email address.</p>

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>hideOriginalBody</code>	Boolean	No. Defaults to false .	<p>When set to true, causes the HTML body of the message to be hidden. This is very useful in scenarios where the card is a better or more useful representation of the content than the HTML body itself, which is especially true when the card contains actions.</p> <p>Consider hiding the original HTML body if the card itself contains all the information a user would need, or if the content of the card is redundant with the content of the body. Do always include a nice, meaningful HTML body, even if it is going to be hidden. The HTML body is the only thing an email client that doesn't support cards will be able to display. Furthermore, cards are not included when replying to or forwarding emails, only the HTML body. Don't hide the body when it is complementary to the information presented in the card. For example, the body of an expense report approval might describe the report in great details while the card just presents a quick summary along with approve/decline actions.</p>
<code>originator</code>	String	Yes	<p>For actionable email, MUST be set to the provider ID generated by the Actionable Email Developer Dashboard.</p>

Additional properties on the Column type

The following additional properties can be specified on a [Column object](#) in the context of Outlook Actionable Messages:

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>width</code>	Number or String	No (defaults to <code>auto</code>)	<p>This property allows for precise control of the width of a <code>Column</code> within its <code>ColumnSet</code>. See Column width values for details.</p>

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>verticalContentAlignment</code>	String. Valid values are <code>top</code> , <code>center</code> and <code>bottom</code> .	No. Defaults to <code>top</code> .	The <code>verticalContentAlignment</code> property makes it possible to vertically position the content of the column (e.g. all its elements.) This is particularly useful for table-like layouts.
<code>backgroundImage</code>	String	No	The <code>backgroundImage</code> property represents the URL to an image that is to be used as the background of the <code>Column</code> . The background image covers the entirety of the <code>Column</code> 's surface and is scaled so as to preserve its original aspect ratio.

Column width values

If `width` is expressed as a number, it represents the relative weight of the `Column` within its `ColumnSet`. For a weighted `Column` to really be useful, there should be at least one more weighted `Column` in the set. For example, if column A has its `width` set to `1` and column B has its `width` set to `2`, then column A will use a third of the available space in the set, while column B will use the remaining two-thirds.

If `width` is expressed as a string, it can have the following values:

- `auto`: The `Column` will use as much of the available space as is required to fit its content.
- `stretch`: The `Column` will use whatever space remains in the set. If multiple columns have their `width` property set to `stretch`, they all share the remaining space equally.
- `<number>px` (ex. `50px`): The column will spread across the specified number of pixels.
- `<number>*`, (ex. `1*`): This is equivalent to specifying `width` as a number.

Column example

```
{
  "$schema": "https://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "ColumnSet",
      "columns": [
        {
          "width": "50px",
          "items": [
            {
              "type": "Image",
              "url": "https://adaptivecards.io/content/cats/1.png",
              "size": "stretch"
            }
          ]
        },
        {
          "width": "stretch",
          "verticalContentAlignment": "center",
          "items": [
            {
              "type": "TextBlock",
              "text": "This card has two ColumnSets on top of each other. In each, the left column is explicitly sized to be 50 pixels wide.",
              "wrap": true
            }
          ]
        }
      ]
    },
    {
      "type": "ColumnSet",
      "columns": [
        {
          "width": "50px"
        },
        {
          "width": "stretch",
          "verticalContentAlignment": "center",
          "items": [
            {
              "type": "TextBlock",
              "text": "In this second ColumnSet, columns align perfectly even though there is nothing in the left column.",
              "wrap": true
            }
          ]
        }
      ]
    }
  ]
}
```



This card has two ColumnSets on top of each other. In each, the left column is explicitly sized to be 50 pixels wide.

In this second ColumnSet, columns align perfectly even though there is nothing in the left column.

Additional properties on the Container type

The following additional properties can be specified on a [Container object](#) in the context of Outlook Actionable Messages:

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>verticalContentAlignment</code>	String. Valid values are <code>top</code> , <code>center</code> and <code>bottom</code> .	No. Defaults to <code>top</code> .	The <code>verticalContentAlignment</code> property makes it possible to vertically position the content of the column (e.g. all its elements.) This is particularly useful for table-like layouts.
<code>backgroundImage</code>	String	No	The <code>backgroundImage</code> property represents the URL to an image that is to be used as the background of the <code>Container</code> . The background image covers the entirety of the <code>Container</code> 's surface and is scaled so as to preserve its original aspect ratio.

These properties behave exactly like their counterpart on the `Column` type. Please refer to the above example.

Additional properties on the Image type

The following additional properties can be specified on an [Image object](#) in the context of Outlook Actionable Messages:

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>width</code>	String	No	This property allows for precise control over the width of an image, in pixels. The allowed format is <code><number>px</code> where <code><number></code> is an integer. When <code>width</code> is specified, the <code>size</code> property is ignored. If <code>width</code> is specified but <code>height</code> isn't, the height of the image is automatically computed so as to respect its aspect ratio.

PROPERTY NAME	TYPE	REQUIRED	DESCRIPTION
<code>height</code>	String	No	<p>This property allows for precise control over the height of an image, in pixels. The allowed format is <code><number>px</code> where <code><number></code> is an integer.</p> <p>When <code>height</code> is specified, the <code>size</code> property is ignored. If <code>height</code> is specified but <code>width</code> isn't, the width of the image is automatically computed so as to respect its aspect ratio.</p>
<code>backgroundColor</code>	String	No	<p>The <code>backgroundColor</code> property specifies the color the image should be rendered on top of. <code>backgroundColor</code> is particularly useful in cases where a single image should be used on top of a variety of background colors, as it removes the need to craft multiple versions of a single image. The format of the <code>backgroundColor</code> property is <code>#RRGGBB</code> where <code>RR</code>, <code>GG</code> and <code>BB</code> are the hexadecimal values of the red, green and blue components of the color, respectively.</p>

Image properties example

```
{
  "$schema": "https://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.0",
  "body": [
    {
      "type": "TextBlock",
      "text": "Below, the same image is presented on top of two different background colors."
    },
    {
      "type": "Image",
      "width": "64px",
      "url": "https://messagecardplayground.azurewebsites.net/assets/circleontransparentbackground.png",
      "backgroundColor": "#FF0000"
    },
    {
      "type": "Image",
      "style": "person",
      "width": "64px",
      "url": "https://messagecardplayground.azurewebsites.net/assets/circleontransparentbackground.png",
      "backgroundColor": "#0000FF"
    }
  ]
}
```

Below, the same image is presented on top of two different background colors.



See also

- [Adaptive Card portal](#)

Register your service with the actionable email developer dashboard

4/18/2022 • 7 minutes to read • [Edit Online](#)

To test and publish actionable messages from your service, you need to provide certain information to Microsoft to enable this functionality for emails from your service. The [developer dashboard](#) helps you submit and track status of your submission via the web portal.

NOTE

You can easily try out actionable messages via email by sending email to yourself with the required markup without any intervention from Microsoft. This would typically be the first step you try out as you dip your toes into this capability. Check out [these samples](#) to send an actionable message to your mailbox, or use the [Card Playground](#) to send an actionable message to yourself.

If you are a developer working with actionable messages via email, you will use the portal for the following cases:

- To test actionable messages from your service to your own mail box
- To publish actionable message from your service so any email user within your organization using Office 365 can receive these specially formatted message (this is typically used for enabling actionable messages from a service that is specific to your organization, like a line-of-business app)
- To publish actionable messages from your service so any email user in Office 365 using your service can receive these specially formatted messages

For all the above cases, you will be submitting certain details to Microsoft, which after being reviewed and approved, will enable actionable messages for your service.

Dashboard sections

The developer dashboard is divided into a few logical sections you need to fill out based on the scope you'd like to request Microsoft to enable actionable message from your service.

Details of your provider

In this section, you need to supply key details that will allow Office 365 to accept emails with markup from your service as well as URLs that can be invoked via the action buttons from those emails.

The key fields are:

- **Sender email address:** This is one or more static email addresses corresponding to the service that will send out emails with action markup. Example: `myservice@contoso.com`.
- **Target URLs:** This is one or more domains corresponding to URLs that will process the actions. Your target URL can correspond to the top level domain or the sub-domain of the TLD. They need to be https enabled URLs. Example: `https://api.myservice.com`.
- **Public Key:** If you plan to send actionable messages as Signed Card, then you need to specify the public key corresponding to the private key you will use for signing the card. The format for this field is an [RSAKeyValue](#) element.

```
<RSAKeyValue>
  <Modulus>xA7SEU+e0yQ...</Modulus>
  <Exponent>AQAB</Exponent>
</RSAKeyValue>
```

For an example of how to get public key XML from a .cert file, see [PublicKey Class](#).

NOTE

Once your submission is approved, it may take some time to take effect. If you encounter the error below when sending signed cards, and you're sure that your payload is correct, please try again after a few hours.

Adaptive card signature validation failed - Failed to validate signature

Scope of your submission

In this section, you need to specify at what scope you want to enable actionable message for your service. The applicable scopes are:

- **Test Users:** This enables actionable emails from your service to some of the O365 email users in your organization. This scope is generally used for testing actionable messages integration with few test users that you have specified.
- **Organization:** This enables actionable message from your service to any Microsoft 365 email user within your organization. This scope is typically used for enabling actionable messages from a service that is specific to your organization, like a line- of-business application internal to your organization.
- **Global:** This enables actionable message from your service for any email user in Office 365.

Each of the above are independent steps. i.e. you can pick only one scope for each submission and will be subject to the approval process by Microsoft.

NOTE

Remember, you can easily try out actionable messages by sending an email to yourself with the required markup without any intervention from Microsoft. You can use the [Card Playground](#) to send to yourself without writing any code. This would typically be the first step to try out actionable messages.

Self-service registration

Self-service of registrations is available for registrations that use the following scopes.

- **Test Users:** The registration request is auto-approved for your test users you specify. This will enable actionable emails from your service sent to test users.
- **Organization:** This registration request will be sent to your organization's administrators with **Exchange administrator** permissions. Any administrator with those permissions receive an email with submission details and will be able to review and approve your request. If no users have the **Exchange administrator** role assigned, users with the **Global administrator** role will receive this email instead.

Once the submission is approved, whether auto-approved or by your administrator, it will take up to 24 hours for the registration to take effect.

For **My organization** registrations, the administrator accounts will receive an email and the submitter will also be copied on those emails. This will allow you to reach out to your administrator if you need to provide further clarifications or details. Once the request is approved, the submitter and the administrators will be notified with another email.

After 24 hours have passed, you can verify if the registration has taken into effect by sending an actionable message from your service to your mailbox or specified test users (for **Test Users** scope), or any user mailbox in your organization (for **Organization** scope). If 24 hours have passed and the registration is still not in effect, please contact us by using the feedback link at the top the registration dashboard labeled **Registration not working?**.

Test user email addresses

This section is only applicable when your scope of submission to enable actionable messages is **Test Users**.

In this section provide a list of Microsoft 365 email users in your organization, separated by a semi-colon (;).

This will help you to test your actionable messages integration on a few users, before creating an **Organization** or **Global** scope submission.

Contact info

This section is only applicable when your scope of submission to enable actionable messages is **Global**.

In this section, you need to provide contact details, so we can reach out to you if we have further questions regarding your submission. All information provided must be valid and accurate.

Publisher information

This section is only applicable when your scope of submission to enable actionable messages is **Global**.

In this section, you need to provide details about your service that will be sending Actionable Messages and related support information, so we can reach out to you or direct customers to your support site. This information will also be used as part of the approval process by Microsoft for your provider.

Scenario details

This section is only applicable when your scope of submission to enable actionable messages is **Global**.

In this section, you need to provide details on the scenario for which users will consume actionable messages from your service and other relevant details. This is to help Microsoft determine that validity and usefulness of the solution provided by your service.

Verification details

This section is only applicable when your scope of submission to enable actionable messages is **Global**.

In this section, you need to provide details for Microsoft to verify the actionable message and the corresponding actions that are invoked from the email sent by your provider/service.

Additionally, send a valid email coming from your production servers (or a server with similar DKIM/SPF/From:/Return-Path: headers) including the markup `toonboardoam@microsoft.com`. This procedure will enable Microsoft to determine that the solution complies with all the guidelines and requirements listed in Registration Criteria.

- Make sure that the markup is correct prior to sending the email.
- Office 365 removes all markup when forwarding an email. Do not forward the email but send it directly.

Registration criteria

There are some things you need to keep in mind when you submit your solution for approval for **Global** scope since it can have broad impact to users in Office 365.

Email sender quality guidelines

- Emails must be authenticated via DKIM or SPF.
- The top-level domain (TLD) of the SPF check or DKIM signature must match the TLD of your `From:` email address. For example, if you use `From: myservice@contoso.com` the DKIM or SPF must be for `contoso.com` or

- .contoso.com .

- Emails must come from a static email address, e.g. `myservice@contoso.com` .
- Emails must follow the email sender guidelines.
 - See [Sending mail to Office 365](#) for Office 365.
 - See [Policies, Practices, and Guidelines](#) for Outlook.com.
 - See [M3AAWG Sender Best Practices](#) and [ReturnPath Sending Best Practices](#) for industry guidelines.
- Consistent history of sending a high volume of mail from your domain (order of hundred emails a day minimum to Office 365) for a few weeks at least.
- A very low rate of spam complaints from users.
- High-fidelity, routine and simple actions available for your service should be used. For more complex interactions, `OpenURI` actions can be used.
- Actions should be used for transactional mail where a high interaction rate is expected. They should not be used on promotional bulk mail.

Actions guidelines

- Label of the button needs to reflect clear action to be taken.
- `Action.OpenUrl` action must deep link into the specific page associated with the entity/information presented in the actionable message.
- Low failure rate and fast response for services handling action requests.
- Please see [Designing Outlook actionable message cards with the Adaptive Card format](#) for additional guidelines on designing actionable messages.

Approval of your submission

We will notify you on the email address you provided during your submission, so please ensure you provide the right contact information.

Register your connector with the Office connectors developer dashboard

4/18/2022 • 3 minutes to read • [Edit Online](#)

Building an Office connector for your application is easy. All you need to do is register your connector in our developer portal, add an integrated configuration experience to your application, and implement your connector. You can make it easy for your users to discover the connector by publishing to our catalog.

Build your own Connector

Registering your Connector

Visit the [Connector Developer Portal](#) and login with your Microsoft 365 credentials. If you do not have an Microsoft 365 subscription you can get a one year [FREE Microsoft 365 subscription](#) under the Microsoft 365 Developer Program.

Choose **New Connector** and fill out the form. Once you choose **Save**, new options appear on the page.

- The **Sideload to Outlook** button will temporarily add your connector into the logged on user's Outlook experience. That user will be able to configure the connector either in their inbox or on any group that user is a member of.
- The **Publish to Store** button will start the publish process, including review by Microsoft for listing in the store.

Adding an integrated configuration experience

An integrated configuration experience allows the user to configure your connector without leaving Outlook. Your application exposes the configuration as a web page that utilizes the Microsoft Teams JavaScript library to communicate configuration status back to Outlook.

For details, see [Integrating the configuration experience](#).

NOTE

The [Integrating the configuration experience](#) document is Microsoft Teams-specific, but the documented methods work the same way in Outlook.

Outlook-specific configuration requirements

If your configuration experience requires authentication, there are additional requirements to enable the [authentication flow](#) in Outlook on Windows.

Outlook on Windows passes an additional query parameter to your connectors authentication start page.

```
callbackUrl=<connectors url>
```

Your app must preserve this value and pass it as an additional parameter to the

`microsoftTeams.authentication.notifySuccess` OR `microsoftTeams.authentication.notifyFailure` methods once authentication is complete.

```
microsoftTeams.authentication.notifySuccess(result, callbackUrl);
```

Publish your Connector to the Store

Once you have thoroughly tested your connector and it is ready to be listed in the Office connector catalog, you can use the **Publish to Store** button to submit it for review. Once reviewed and approved your connector would be added to the connector catalog.

Connector submission checklist

- Ensure that your connector is fully functional and thoroughly tested before submitting it to the Store.
- Test your connector cards in various clients where your users would use it: Outlook on the Web, Outlook 2016 or later, and Outlook Groups mobile apps.
- Ensure that you strictly use Markdown for text decoration and not send HTML in your connector card payload.
- Maintain a balance between adding value and generating too much noise. Ensure that the user is not bogged down with too many notifications.
- Identify the right events to send connector cards for. Ensure that the information you send to the group is valuable to the members of the group.
- When sending reports or summaries, use a digest format and allow the user to choose the time and frequency of the reports.
- When sending connector cards make the best use of Markdown to highlight important parts of the card.
- Make your connector cards actionable by providing relevant actions whenever possible.
- Actions invoked should be really low in failure rate and should have fast response by your endpoint.
- Ensure that you have provisions for the user to pause or remove the configuration.
- Have clear user-facing documentation on the capabilities your connector offers.
- When registering your connector:
 - Ensure that the name and logo of your connector does not infringe upon a trademark or copyright of any other product or service.
 - Provide a high quality logo of type jpg, jpeg, png or gif that is under 60KB in size.
 - Provide a short description of your application (e.g. 'Contoso Help Desk brings companies and customers together').
 - Provide a detailed description of your connector (e.g. 'The Contoso Help Desk connector notifies your Office 365 group about activity on your customer's tickets').
- When publishing your connector to Store:
 - Make sure to fill out step by step instructions and share test account information to let us test your connector.

Next steps

- If you have a question, need help, or are experiencing an issue with your code, ask the developer community on [Microsoft Q&A](#).
- You can also post your question or issue on **Stack Overflow**. Tag your question according to the technology involved:
 - REST APIs: [microsoftgraph](#) or [outlook-rest-api](#)
 - Add-ins: [outlook-web-addins](#)
 - Actionable messages or Outlook connectors: [office365connectors](#)
- If you have a feature suggestion, please post your idea on our [Microsoft 365 Developer Platform Ideas](#) forum, and vote for your suggestions there.

Legacy actionable message card reference

4/18/2022 • 25 minutes to read • [Edit Online](#)

NOTE

This document describes the original JSON format for the actionable message card format. For actionable messages sent via email, this has been replaced with the [Adaptive Card format](#). Microsoft recommends that new actionable message integrations use the Adaptive Card format, and existing integrations consider updating to Adaptive Card format. The Adaptive Card format is **required** to support Outlook on iOS and Android. However, if you are sending actionable messages via an Office connector, or to a Microsoft Teams connector, you must continue to use the message card format.

Cards are meant to provide easy to read, at-a-glance information that users can very quickly decipher and act upon when appropriate. As such, the guiding principle for designing great card is "content over chrome," which means cards are straight to the point and minimize the use of anything that would be distracting such as icons or custom colors.

Card playground

Ready to experiment with your card design? Head to the [Card Playground](#) which allows you to see what your card will look like as you edit the associated JSON payload.

NOTE

The Card Playground loads Adaptive Card examples by default. You can find message card format examples by selecting the **Select a sample** dropdown in the playground.

Design guidelines

Text formatting

All text fields in a card and its section can be formatted using Markdown. We support basic Markdown.

IMPORTANT

Since all fields are processed as Markdown, be sure to escape Markdown special characters (such as `*` or `#`) if needed.

EFFECT	MARKDOWN
Italics	<code>*Italic*</code>
Bold	<code>**Bold**</code>
Bold italics	<code>***Bold Italic***</code>
Strike-through	<code>~~Strike-through~~</code>
Links	<code>[Microsoft](https://www.microsoft.com)</code>

EFFECT	MARKDOWN
Headings (<code><h1></code> through <code><h6></code>)	<code># Heading</code> through <code>##### Heading</code>
Bulleted lists	<code>* List item</code> or <code>- List item</code>

TIP

Follow these guidelines when formatting text fields.

- Do use Markdown to format text.
- Don't use HTML markup in your cards. HTML is ignored and treated as plain text.

Using sections

If your card represents a single "entity", you may be able to get away with not using any section. That said, sections support the concept of an "activity" which is often a good way to represent data in a card.

If your card represents multiple "entities" or is, for instance, a digest for a particular news source, you will definitely want to use multiple sections, one per "entity."

TIP

Follow these guidelines when planning the layout of your card.

- Do use sections to logically group data together.
- Sometimes, multiple sections MAY be used to represent a single logical group of data; this allows for more flexibility on ordering the information presented in the card. For example, it makes it possible to display a list of facts before an activity.
- Don't include more than 10 sections. Cards are meant to be easy to read; if there is too much information in a card, it will be lost on the user.
- For digest-like cards, consider adding a "View full digest" action at the end of the card.

Card fields

FIELD	TYPE	DESCRIPTION
<code>@type</code>	String	Required. Must be set to <code>MessageCard</code> .
<code>@context</code>	String	Required. Must be set to <code>https://schema.org/extensions</code> .

FIELD	TYPE	DESCRIPTION
<code>correlationId</code>	UUID	<p>The <code>correlationId</code> property simplifies the process of locating logs for troubleshooting issues. We recommend that when sending an actionable card, your service should set and log a unique UUID in this property.</p> <p>When the user invokes an action on the card, Office 365 sends the <code>Card-Correlation-Id</code> and <code>Action-Request-Id</code> headers in the POST request to your service. <code>Card-Correlation-Id</code> contains the same value as the <code>correlationId</code> property in the card. <code>Action-Request-Id</code> is a unique UUID generated by Office 365 to help locate specific action performed by a user. Your service should log both of these values when receiving action POST requests.</p>
<code>expectedActors</code>	Array of String	<p>Optional. This contains a list of expected email addresses of the recipient for the action endpoint.</p> <p>A user can have multiple email addresses and the action endpoint might not be expecting the particular email address presented in the <code>sub</code> claim of the bearer token. For example, a user could have both the <code>john.doe@contoso.com</code> or <code>john@contoso.com</code> email address, but the action endpoint expects to receive <code>john@contoso.com</code> in the <code>sub</code> claim of the bearer token. By setting this field to <code>["john@contoso.com"]</code>, the <code>sub</code> claim will have the expected email address.</p>
<code>originator</code>	String	<p>Required when sent via email, not applicable when sent via connector. For actionable email, MUST be set to the provider ID generated by the Actionable Email Developer Dashboard.</p>


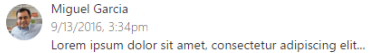
FIELD	TYPE	DESCRIPTION
<code>summary</code>	String	<p>Required if the card does not contain a <code>text</code> property, otherwise optional. The <code>summary</code> property is typically displayed in the list view in Outlook, as a way to quickly determine what the card is all about.</p> <p>Do always include a summary. Don't include details in the summary. For example, for a Twitter post, a summary might simply read "New tweet from @someuser" without mentioning the content of the tweet itself.</p>
<code>themeColor</code>	String	<p>Specifies a custom brand color for the card. The color will be displayed in a non-obtrusive manner.</p> <p>Do use <code>themeColor</code> to brand cards to your color. Don't use <code>themeColor</code> to indicate status.</p>

FIELD	TYPE	DESCRIPTION
<code>hideOriginalBody</code>	Boolean	<p><i>Only applies to cards in email messages</i></p> <p>When set to true, causes the HTML body of the message to be hidden. This is very useful in scenarios where the card is a better or more useful representation of the content than the HTML body itself, which is especially true when the card contains actions (see below.)</p> <p>Consider hiding the original HTML body:</p> <ul style="list-style-type: none"> • If the card itself contains all the information a user would need • If the content of the card is redundant with the content of the body <p>Do always include a nice HTML body, even if it is going to be hidden. The HTML body is the only thing an email client that doesn't support cards will be able to display. Furthermore, cards are not included when replying to or forwarding emails, only the HTML body.</p> <p>Don't hide the body when it is complementary to the information presented in the card. For example, the body of an expense report approval might describe the report in great details while the card just presents a quick summary along with approve/decline actions.</p>
<code>title</code>	String	<p>The <code>title</code> property is meant to be rendered in a prominent way, at the very top of the card. Use it to introduce the content of the card in such a way users will immediately know what to expect.</p> <p>Examples:</p> <ul style="list-style-type: none"> • Daily news • New bug opened • Task <code><name of task></code> assigned <p>Do keep title short, don't make it a long sentence.</p> <p>Do mention the name of the entity being referenced in the title.</p> <p>Don't use hyperlinks (via Markdown) in the title.</p>


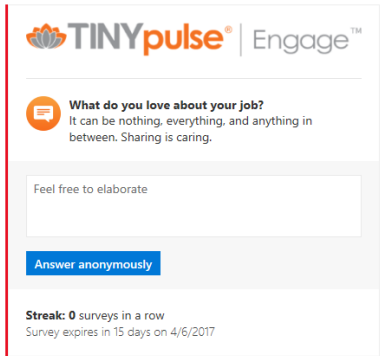
FIELD	TYPE	DESCRIPTION
<code>text</code>	String	<p>Required if the card does not contain a <code>summary</code> property, otherwise optional. The <code>text</code> property is meant to be displayed in a normal font below the card's title. Use it to display content, such as the description of the entity being referenced, or an abstract of a news article.</p> <p>Do use simple Markdown, such as bold or italics to emphasize words, and links to external resources. Don't include any call to action in the text property. Users should be able to not read it and still understand what the card is all about.</p>
<code>sections</code>	Array of <code>Section</code>	A collection of sections to include in the card. See Section fields .
<code>potentialAction</code>	Array of <code>Actions</code>	A collection of actions that can be invoked on this card. See Actions .

Section fields

FIELD	TYPE	DESCRIPTION
<code>title</code>	String	<p>The <code>title</code> property of a section is displayed in a font that stands out while not as prominent as the card's title. It is meant to introduce the section and summarize its content, similarly to how the card's title property is meant to summarize the whole card.</p> <p>Do keep title short, don't make it a long sentence. Do mention the name of the entity being referenced in the title. Don't use hyperlinks (via Markdown) in the title.</p>

FIELD	TYPE	DESCRIPTION
<div>startGroup</div>	Boolean	<p>When set to <code>true</code>, the <code>startGroup</code> property marks the start of a logical group of information. Typically, sections with <code>startGroup</code> set to <code>true</code> will be visually separated from previous card elements. For example, Outlook uses a subtle horizontal separation line.</p>  <p>Do use <code>startGroup</code> to separate sections that represent different objects; for example, multiple tweets in a digest.</p>
<div>activityImage</div> <div>activityTitle</div> <div>activitySubtitle</div> <div>activityText</div>	String	<p>These four properties form a logical group. <code>activityTitle</code>, <code>activitySubtitle</code> and <code>activityText</code> will be displayed alongside <code>activityImage</code>, using a layout appropriate for the form factor of the device the card is being viewed on. For instance, in Outlook on the Web, <code>activityTitle</code>, <code>activitySubtitle</code> and <code>activityText</code> are displayed on the right of <code>activityImage</code>, using a two-column layout:</p>  <p>Use the activity fields for scenarios such as:</p> <ul style="list-style-type: none"> Someone did something <ul style="list-style-type: none"> Use <code>activityImage</code> to display the picture of that person. Use <code>activityTitle</code> to summarize what they did. Make it short and to the point. Use <code>activitySubtitle</code> to show, for instance, the date and time the action was taken, or the person's handle. <ul style="list-style-type: none"> <code>activitySubtitle</code>

FIELD	TYPE	DESCRIPTION
		<p>will be rendered in a more subdued font</p> <ul style="list-style-type: none">◦ Don't include essential information◦ Don't include calls to action◦ Avoid Markdown formatting◦ Use <code>activityText</code> to provide details about the activity.<ul style="list-style-type: none">◦ Do use simple Markdown to emphasize words or link to external sources◦ Don't include calls to action• A news article abstract<ul style="list-style-type: none">◦ Use <code>activityImage</code> to display the picture associated with the article◦ Use <code>activitySubtitle</code> to display the date and time the article was originally posted◦ Use <code>activityText</code> to display the actual abstract

FIELD	TYPE	DESCRIPTION
<code>heroImage</code>	Image	<p>Use <code>heroImage</code> to make an image the centerpiece of your card. For example, a tweet that contains an image will want to put that image front and center:</p>  <p><code>heroImage</code> can also be used to add a banner to your card, like the "TINYPulse – Engage" banner below:</p> 
<code>text</code>	String	<p>The section's <code>text</code> property is very similar to the <code>text</code> property of the card. It can be used for the same purpose.</p>

FIELD	TYPE	DESCRIPTION
facts	Array of name/value pairs	<p>Facts are a very important component of a section. They often contain the information that really matters to the user.</p> <p>Facts are displayed in such a way that they can be read quickly and efficiently. For example, in Outlook on the Web, facts are presented in a two-column layout, with fact names rendered in a slightly more prominent font:</p> <div> <div>Board:</div> <div>TradersInc, mobile app</div> <div>List:</div> <div>Work Items</div> <div>Assigned to:</div> <div>Miguel Garcia</div> <div>Due date:</div> <div>11/15/2016</div> </div> <p>There are many uses for facts. Some scenarios:</p> <ul style="list-style-type: none"> • A bug was created <ul style="list-style-type: none"> ◦ Bug ID: 1234 ◦ Opened by: Adele Vance ◦ Assigned to: Alex Darrow • Application usage report <ul style="list-style-type: none"> ◦ Application name: Contoso CRM App ◦ Period: August 1, 2016 - September 30, 2016 ◦ Number of users: 542 ◦ Number of sessions: 2056 ◦ Average time spend in the application: 76 seconds • Expense approval <ul style="list-style-type: none"> ◦ Submitted by: Pradeep Gupta ◦ Date submitted: October 21, 2016 ◦ Total amount: \$1,426.95 <p>Do use facts instead of embedding important information inside the text property of either the card or the section.</p> <p>Do keep fact names short.</p> <p>Avoid making fact values too long.</p> <p>Avoid using Markdown formatting for both fact names and values. Let facts be rendered as intended as that is how they will have the most impact.</p> <p>Do however use Markdown for links in fact values only. For instance, if a fact references an external document, make the value of that fact a link to the document.</p> <p>Don't add a fact without a real purpose. For instance, a fact that would always have the same value across all cards is not interesting and a</p>

FIELD	TYPE	DESCRIPTION
<code>images</code>	Array of Image objects	The <code>images</code> property allows for the inclusion of a photo gallery inside a section. That photo gallery will always be displayed in a way that is easy to consume regardless of the form factor of the device it is being viewed on. For instance, in Outlook on the Web, images might be displayed as a horizontal strip of thumbnails with controls allowing to scroll through the collection if it doesn't all fit on the screen. On mobile, images might be displayed as a single thumbnail, with the user able to swipe through the collection with their finger.
<code>potentialAction</code>	Array of <code>Actions</code>	A collection of actions that can be invoked on this section. See Actions .

Image object

Defines an image as used by the `heroImage` and `images` property of a section.

FIELD	TYPE	DESCRIPTION
<code>image</code>	String	The URL to the image.
<code>title</code>	String	A short description of the image. Typically, <code>title</code> is displayed in a tooltip as the user hovers their mouse over the image.

Actions

Cards are very powerful in the sense that they allow users to take quick actions without leaving their email client. When designing cards, consider making them actionable, as that will increase user engagement and productivity.

Actions are specified using the `potentialAction` property which is available both on the card itself and on each section. There are four types of actions:

- [OpenUri](#)
- [HttpPost](#)
- [ActionCard](#)
- [InvokeAddInCommand](#)

There can be a maximum of 4 actions (whatever their type) in a `potentialAction` collection.

- **Do** include actions that will make the biggest impact for the end user, like the most repetitive ones.
- **Don't** add 4 actions just because you can. In many cases, fewer actions will lead to a better experience.
- **Don't** craft your cards in an effort to replace an external application. Cards are meant to complement such applications, not to replace them.

OpenUri action

Opens a URI in a separate browser or app.

Although links can be achieved through Markdown, an `OpenUri` action has the advantage of allowing you to specify different URIs for different operating systems, which makes it possible to open the link in an app on mobile devices.

- **Consider** using an `OpenUri` action rather than a link in Markdown if there is a clear advantage for your users in their ability to open the link in an app on their mobile device.
- **Do** include at least an `OpenUri` action to view the entity in the external app it comes from.
- **Do** make the `OpenUri` action the last one in the `potentialAction` collection.

NOTE

Microsoft Teams and Outlook on the web only support HTTP/HTTPS URLs in the `targets` array for an `OpenUri` action.

FIELD	TYPE	DESCRIPTION
<code>name</code>	String	<p>The <code>name</code> property defines the text that will be displayed on screen for the action.</p> <p>Do use verbs. For instance, use "Set due date" instead of "Due date" or "Add note" instead of "Note." In some cases, the noun itself just works because it is also a verb: "Comment"</p> <p>Don't name an <code>OpenUri</code> action in such a way that it suggests the action can be taken right from the client. Instead, name the action "View in <name of site/app>" or "Open in <name of site/app>"</p>
<code>targets</code>	Array	<p>The <code>targets</code> property is a collection of name/value pairs that defines one URI per target operating system.</p> <p>Supported operating system values are <code>default</code>, <code>windows</code>, <code>iOS</code> and <code>android</code>. The <code>default</code> operating system will in most cases simply open the URI in a web browser, regardless of the actual operating system.</p> <p>Example targets property:</p> <pre>"targets": [{ "os": "default", "uri": "https://yammer.com/.../123" }, { "os": "iOS", "uri": "yammer://u/123" }, { "os": "android", "uri": "yammer://u/123" }, { "os": "windows", "uri": "yammer://u/123" }]</pre>

HttpPOST action

Makes a call to an external Web service.

When an `HttpPOST` action is executed, a POST request is made to the URL in the `target` field, and the target service needs to authenticate the caller. This can be done in a variety of ways, including via a Limited Purpose Token embedded in the target URL. For more information and help on choosing the security mechanism that works best for your particular scenario, please see [Security requirements for actionable messages](#).

FIELD	TYPE	DESCRIPTION
<code>name</code>	String	The <code>name</code> property defines the text that will be displayed on screen for the action. Do use verbs. For instance, use "Set due date" instead of "Due date" or "Add note" instead of "Note." In some cases, the noun itself just works because it is also a verb: "Comment"
<code>target</code>	String	Defines the URL endpoint of the service that implements the action. Note: this URL must be accessible from the internet, you cannot use <code>localhost</code> .
<code>headers</code>	Array of <code>Header</code>	A collection of <code>Header</code> objects representing a set of HTTP headers that will be emitted when sending the POST request to the target URL. See Header .
<code>body</code>	String	The body of the POST request.
<code>bodyContentType</code>	String	The <code>bodyContentType</code> is optional and specifies the MIME type of the body in the POST request. Some services require that a content type be specified. Valid values are <code>application/json</code> and <code>application/x-www-form-urlencoded</code> . If not specified, <code>application/json</code> is assumed.

The `Header` object is a name/value pair that represents an HTTP header.

FIELD	TYPE	DESCRIPTION
<code>name</code>	String	The header name
<code>value</code>	String	The header value

Reporting an action's execution success or failure

`HttpPOST` actions can include the `CARD-ACTION-STATUS` HTTP header in their response. This header is meant to contain text that indicates the outcome of the action's execution, whether it has succeeded or failed.

The value of the header will be displayed in a consistent way in a reserved area of the card. It is also saved with

the card so it can be displayed later on, so users can be reminded of the actions that have already been executed on a given card.

TIP

Follow these guidelines when returning a response to `HttpPost` actions.

- Do return the `CARD-ACTION-STATUS` header in your responses.
- Do make the message in that header as informative and meaningful as possible. For instance, for an "approve" action on an expense report:
 - In case of success, don't return "The action was successful", instead return "The expense was approved"
 - In case of failure, don't return "The action failed", instead return "The expense couldn't be approved at this time. Please try again later"
- Don't mention either the name of the person taking the action nor the time the action is being taken in your `CARD-ACTION-STATUS` header. Both these pieces of information will be automatically added for you and displayed in a consistent way.

Refresh cards

Refresh cards are a very powerful mechanism that allow `HttpPost` actions to fully update the card on the fly as the action successfully completes. 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 `OpenUri` action that allows the user to consult the survey online

To refresh a card as a result of an `HttpPost` action, a service needs to do the following:

- Include the JSON payload of the new card in the body of the response to the HTTP POST request it received.
- Add the `CARD-UPDATE-IN-BODY: true` HTTP header to the response, in order to let the receiving client know that it should parse the response body and extract a new card (this is to avoid unnecessary processing when no refresh card is included.)

TIP

Follow these guidelines when returning refresh cards.

- Do use 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
- Do 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
- Don't use refresh cards to lead a conversation with the user. For instance, don't use refresh cards for a multi-step "wizard"
- Do include at least an `OpenUri` action to view the entity in the external app it comes from.

ActionCard action

Presents additional UI that contains one or more [Inputs](#), along with associated actions that can be either `OpenUri` or `HttpPost` types.

Do use an `ActionCard` action if an action requires additional input from the user. Some scenarios:

- Responding to a survey
- Adding a comment to a bug
- Providing justification for declining an expense report

By default, an `ActionCard` action will be represented as a button or link in the card's UI. When clicked, that button will display an additional piece of UI containing the inputs and actions defined in the action card.

If there is a single `ActionCard` action in a `potentialAction` collection, then Outlook will represent that action "pre-expanded," e.g. its inputs and actions will be immediately visible.

FIELD	TYPE	DESCRIPTION
<code>name</code>	String	<p>The <code>name</code> property defines the text that will be displayed on screen for the action.</p> <p>Do use verbs. For instance, use "Set due date" instead of "Due date" or "Add note" instead of "Note." In some cases, the noun itself just works because it is also a verb: "Comment"</p>
<code>inputs</code>	Array of <code>Inputs</code>	<p>The <code>inputs</code> property defines the various inputs that will be displayed in the action card's UI. See Inputs</p>
<code>actions</code>	Array of <code>Actions</code>	<p>The <code>actions</code> property is an array of <code>Action</code> objects, that can be either of type <code>OpenUri</code> or <code>HttpPost</code>. The <code>actions</code> property of an <code>ActionCard</code> action cannot contain another <code>ActionCard</code> action.</p>

Example ActionCard

```
{
  "@type": "ActionCard",
  "name": "Comment",
  "inputs": [
    {
      "@type": "TextInput",
      "id": "comment",
      "isMultiline": true,
      "title": "Input's title property"
    }
  ],
  "actions": [
    {
      "@type": "HttpPost",
      "name": "Action's name prop.",
      "target": "https://yammer.com/comment?postId=123",
      "body": "comment={{comment.value}}"
    }
  ]
}
```

Inputs

Three types of inputs are supported: [TextInput](#), [DateInput](#), and [MultichoiceInput](#).

Common fields

The following fields are common to all input types.

FIELD	TYPE	DESCRIPTION
<code>id</code>	String	Uniquely identifies the input so it is possible to reference it in the URL or body of an <code>HttpPost</code> action. See Input value substitution .
<code>isRequired</code>	Boolean	<p>Indicates whether users are required to type a value before they are able to take an action that would take the value of the input as a parameter.</p> <p>Do make an input required if users MUST provide a value.</p> <p>Consider making an input required if its value is complementary to that of another required input. For instance, you could define a survey question that asks "How satisfied are you with your car" with a multi choice input followed by "Please explain your answer" as a free text input. Keep in mind that some users might not like being forced into providing such explanations, and might as a result not respond to the survey at all.</p> <p>Do make sure users know which inputs are required. Include a label in the input's title property. For example:</p> <div> <div>Comment (optional)</div> <div>or</div> <div>Please rate your experience (required)</div> </div> <p>.</p>
<code>title</code>	String	Defines a title for the input.

FIELD	TYPE	DESCRIPTION
<code>value</code>	String	Defines the initial value of the input. For multi-choice inputs, value must be equal to the value property of one of the input's choices.

TextInput

Use this input type when you need users to provide free text, such as the response to a survey question.

FIELD	TYPE	DESCRIPTION
<code>isMultiline</code>	Boolean	Indicates whether the text input should accept multiple lines of text.
<code>maxLength</code>	Number	Indicates the maximum number of characters that can be entered.

Example TextInput

```
{
  "@type": "TextInput",
  "id": "comment",
  "isMultiline": true,
  "title": "Input's title property"
}
```

DateInput

Use this input type when you need users to provide a date and or a time, such as for a task's due date.

FIELD	TYPE	DESCRIPTION
<code>includeTime</code>	Boolean	Indicates whether the date input should allow for the selection of a time in addition to the date.

Example DateInput

```
{
  "@type": "DateInput",
  "id": "dueDate",
  "title": "Input's title property"
}
```

MultichoiceInput

Use this input type when you need users to select from a list of pre-defined choices, such as a bug status, yes/no/maybe, etc.

FIELD	TYPE	DESCRIPTION
<code>choices</code>	Array of name/value pairs	Defines the values that can be selected for the multichoice input.

FIELD	TYPE	DESCRIPTION
<code>isMultiSelect</code>	Boolean	<p>If set to <code>true</code>, indicates that the user can select more than one choice. The specified choices will be displayed as a list of checkboxes. Default value is <code>false</code>.</p> <div><div>MultichoiceInput with isMultiSelect example</div><div><div>This example illustrates the use of a MultichoiceInput that allows the selection of multiple options.</div><div><div><input type="checkbox"/> Option 1</div><div><input type="checkbox"/> Option 2</div><div><input type="checkbox"/> Option 3</div></div><div><div>OK</div></div></div></div>
<code>style</code>	String (<code>normal</code> (default or <code>expanded</code>))	<p>When <code>isMultiSelect</code> is <code>false</code>, setting the <code>style</code> property to <code>expanded</code> will instruct the host application to try and display all choices on the screen, typically using a set of radio buttons.</p> <div><div>MultichoiceInput with expanded style</div><div><div>This example illustrates the use of a MultichoiceInput that displays all of its options on the screen.</div><div><div><input type="radio"/> Option 1</div><div><input type="radio"/> Option 2</div><div><input type="radio"/> Option 3</div></div><div><div>OK</div></div></div></div>

Example compact MultichoiceInput

```
{
  "@type": "MultichoiceInput",
  "id": "list",
  "title": "Pick an option",
  "choices": [
    { "display": "Choice 1", "value": "1" },
    { "display": "Choice 2", "value": "2" },
    { "display": "Choice 3", "value": "3" }
  ]
}
```

Example multi-select MultichoiceInput

```
{
  "@type": "MultichoiceInput",
  "id": "list",
  "title": "Pick an option",
  "isMultiSelect": true,
  "choices": [
    { "display": "Choice 1", "value": "1" },
    { "display": "Choice 2", "value": "2" },
    { "display": "Choice 3", "value": "3" }
  ]
}
```

Example expanded MultichoiceInput

```
{
  "@type": "MultichoiceInput",
  "id": "list",
  "title": "Pick an option",
  "style": "expanded",
  "choices": [
    { "display": "Choice 1", "value": "1" },
    { "display": "Choice 2", "value": "2" },
    { "display": "Choice 3", "value": "3" }
  ]
}
```

Input value substitution

The value of an input can be referenced in any URL of a `ViewAction` or `HttpPost` action. It can also be referenced in an `HttpPost` action's body. When an input value is referenced, it is substituted with the actual value of the input right before the action is executed.

To reference an input's value, use the following format:

```
{{<id of input>.value}}
```

Input value substitution example

```
{
  "@type": "ActionCard",
  "name": "Comment",
  "inputs": [
    {
      "@type": "TextInput",
      "id": "comment",
      "isMultiline": true,
      "title": "Input's title property"
    }
  ],
  "actions": [
    {
      "@type": "HttpPost",
      "name": "Action's name prop.",
      "target": "https://yammer.com/comment?postId=123",
      "body": "comment={{comment.value}}"
    }
  ]
}
```

InvokeAddInCommand action

Opens an Outlook add-in task pane. If the add-in is not installed, the user is prompted to install the add-in with a single click.

When an `InvokeAddInCommand` action is executed, Outlook first checks if the requested add-in is installed and turned on for the user. If it is not, the user is notified that the action requires the add-in, and is able to install and enable the add-in with a single click. Outlook opens the requested , making any initialization context specified by the action available to the add-in.

For more information, see [Invoke an Outlook add-in from an actionable message](#).

FIELD	TYPE	DESCRIPTION
-------	------	-------------

FIELD	TYPE	DESCRIPTION
<code>name</code>	String	<p>The <code>name</code> property defines the text that will be displayed on screen for the action.</p> <p>Do use verbs. For instance, use "Set due date" instead of "Due date" or "Add note" instead of "Note." In some cases, the noun itself just works because it is also a verb: "Comment"</p>
<code>addInId</code>	UUID	Specifies the add-in ID of the required add-in. The add-in ID is found in the Id element in the add-in's manifest.
<code>desktopCommandId</code>	String	<p>Specifies the ID of the add-in command button that opens the required task pane. The command button ID is found in the <code>id</code> attribute of the Control element that defines the button in the add-in's manifest. The specified <code>Control</code> element MUST be defined inside a MessageReadCommandSurface extension point, be of type <code>Button</code>, and the control's <code>Action</code> must be of type <code>ShowTaskPane</code>.</p>
<code>initializationContext</code>	Object	Optional. Developers may specify any valid JSON object in this field. The value is serialized into a string and made available to the add-in when the action is executed. This allows the action to pass initialization data to the add-in.

Example InvokeAddInCommand

```
{
  "@type": "InvokeAddInCommand",
  "name": "Invoke My Add-in",
  "addInId": "527104a1-f1a5-475a-9199-7a968161c870",
  "desktopCommandId": "show ",
  "initializationContext": {
    "property1": "Hello world",
    "property2": 5,
    "property3": true
  }
}
```

Card Examples

Trello

Card is created in a list:

Card created: "Name of card"



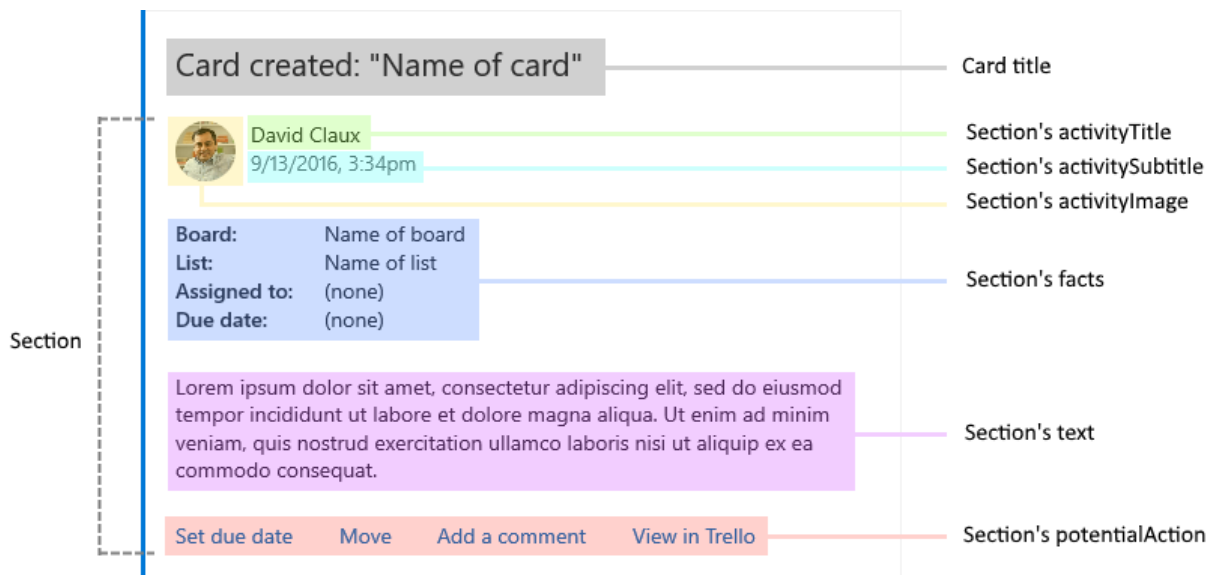
David Claux
9/13/2016, 3:34pm

Board: Name of board
List: Name of list
Assigned to: (none)
Due date: (none)

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

[Set due date](#) [Move](#) [Add a comment](#) [View in Trello](#)

Here is how that card is built:



Here's the same card with the **Add a comment** action expanded:

Card created: "Name of card"



David Claux
9/13/2016, 3:34pm

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Board: Name of board
List: Name of list
Assigned to: (none)
Due date: (none)

Set due date Move [Add a comment](#) View in Trello

Enter your comment

OK

Here's how the **Add a comment** action is built:

Card created: "Name of card"



David Claux
9/13/2016, 3:34pm

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.

Board: Name of board
List: Name of list
Assigned to: (none)
Due date: (none)

Set due date Move [Add a comment](#) View in Trello

Enter your comment

OK

Action of type ActionCard

TextInput, with isMultiline set to true

Expanded ActionCard action

ActionCard's "OK" action (HttpPost)

Trello JSON

```
{
  "summary": "Card \"Test card\"",
  "themeColor": "0078D7",
  "title": "Card created: \"Name of card\"",
  "sections": [
    {
      "activityTitle": "David Claux",
      "activitySubtitle": "9/13/2016, 3:34pm",
      "activityImage": "https://connectorsdemo.azurewebsites.net/images/MS12_Oscar_002.jpg",
      "facts": [
        {
          "name": "Board:",

```

```

        "value": "Name of board"
    },
    {
        "name": "List:",
        "value": "Name of list"
    },
    {
        "name": "Assigned to:",
        "value": "(none)"
    },
    {
        "name": "Due date:",
        "value": "(none)"
    }
],
"text": "Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat."
}
],
"potentialAction": [
    {
        "@type": "ActionCard",
        "name": "Set due date",
        "inputs": [
            {
                "@type": "DateInput",
                "id": "dueDate",
                "title": "Select a date"
            }
        ],
        "actions": [
            {
                "@type": "HttpPost",
                "name": "OK",
                "target": "https://..."
            }
        ]
    },
    {
        "@type": "ActionCard",
        "name": "Move",
        "inputs": [
            {
                "@type": "MultichoiceInput",
                "id": "move",
                "title": "Pick a list",
                "choices": [
                    { "display": "List 1", "value": "l1" },
                    { "display": "List 2", "value": "l2" }
                ]
            }
        ],
        "actions": [
            {
                "@type": "HttpPost",
                "name": "OK",
                "target": "https://..."
            }
        ]
    },
    {
        "@type": "ActionCard",
        "name": "Add a comment",
        "inputs": [
            {
                "@type": "TextInput",
                "id": "comment",
                "isMultiline": true
            }
        ],
        "actions": [
            {
                "@type": "HttpPost",
                "name": "OK",
                "target": "https://..."
            }
        ]
    }
]

```

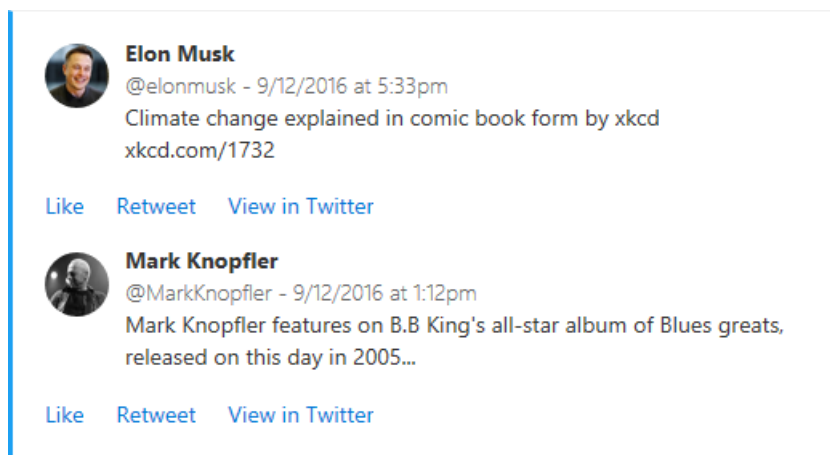
```

        "title": "Enter your comment"
    }
  ],
  "actions": [
    {
      "@type": "HttpPost",
      "name": "OK",
      "target": "https://..."
    }
  ]
},
{
  "@type": "OpenUri",
  "name": "View in Trello",
  "targets": [
    { "os": "default", "uri": "https://..." }
  ]
}
]
}
}

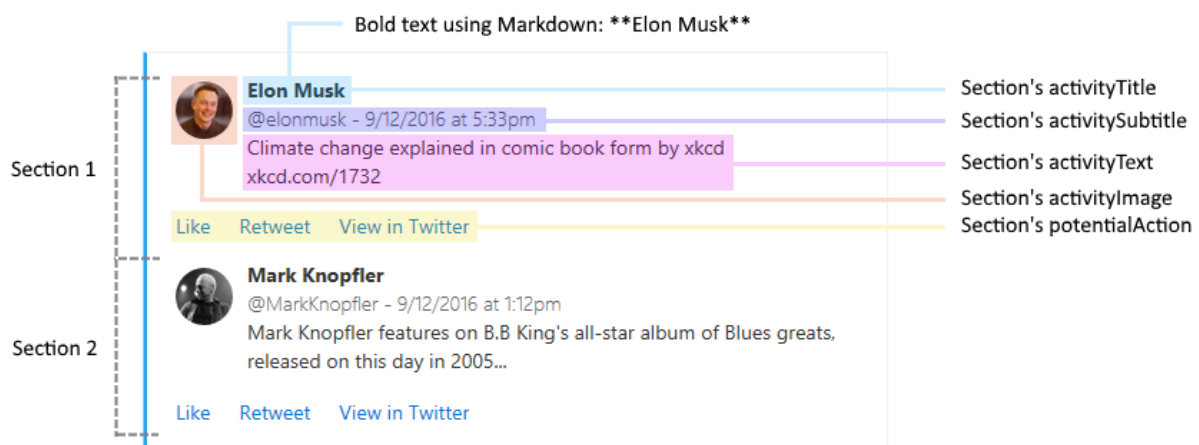
```

Twitter

Here's an example of a Twitter digest card:



Here's how that card is built:



Twitter JSON


```
{
  "themeColor": "0078D7",
  "sections": [
    {
      "activityTitle": "***Elon Musk**",
      "activitySubtitle": "@elonmusk - 9/12/2016 at 5:33pm",
      "activityImage": "https://pbs.twimg.com/profile_images/7824742260200448/zDo-gAo0.jpg",
      "activityText": "Climate change explained in comic book form by xkcd xkcd.com/1732"
    },
    {
      "activityTitle": "***Mark Knopfler**",
      "activitySubtitle": "@MarkKnopfler - 9/12/2016 at 1:12pm",
      "activityImage":
        "https://pbs.twimg.com/profile_images/378800000221985528/b2ebfafca6fd7b565fd3bf4ccdb4dc9.jpeg",
      "activityText": "Mark Knopfler features on B.B King's all-star album of Blues greats, released on this
        day in 2005..."
    }
  ]
}
```

Actionable email

Here's an example of an HTML email body with an embedded message card.

```
<html>
<head>
  <meta http-equiv="Content-Type" content="text/html; charset=utf-8">
  <script type="application/ld+json">{
    "@context": "https://schema.org/extensions",
    "@type": "MessageCard",
    "originator": "",
    "hideOriginalBody": "true",
    "themeColor": "0072C6",
    "title": "Visit the Outlook Dev Portal",
    "text": "Click **Learn More** to learn more about Actionable Messages!",
    "potentialAction": [
      {
        "@type": "ActionCard",
        "name": "Send Feedback",
        "inputs": [
          {
            "@type": "TextInput",
            "id": "feedback",
            "isMultiline": true,
            "title": "Let us know what you think about Actionable Messages"
          }
        ],
        "actions": [
          {
            "@type": "HttpPOST",
            "name": "Send Feedback",
            "isPrimary": true,
            "target": "http://..."
          }
        ]
      },
      {
        "@type": "OpenUri",
        "name": "Learn More",
        "targets": [
          { "os": "default", "uri": "https://docs.microsoft.com/outlook/actionable-messages" }
        ]
      }
    ]
  }
</script>
</head>
<body>
  Visit the <a href="https://docs.microsoft.com/outlook/actionable-messages">Outlook Dev Portal</a> to learn
  more about Actionable Messages.
</body>
</html>
```