

PowerShell Teams Message Script Documentation

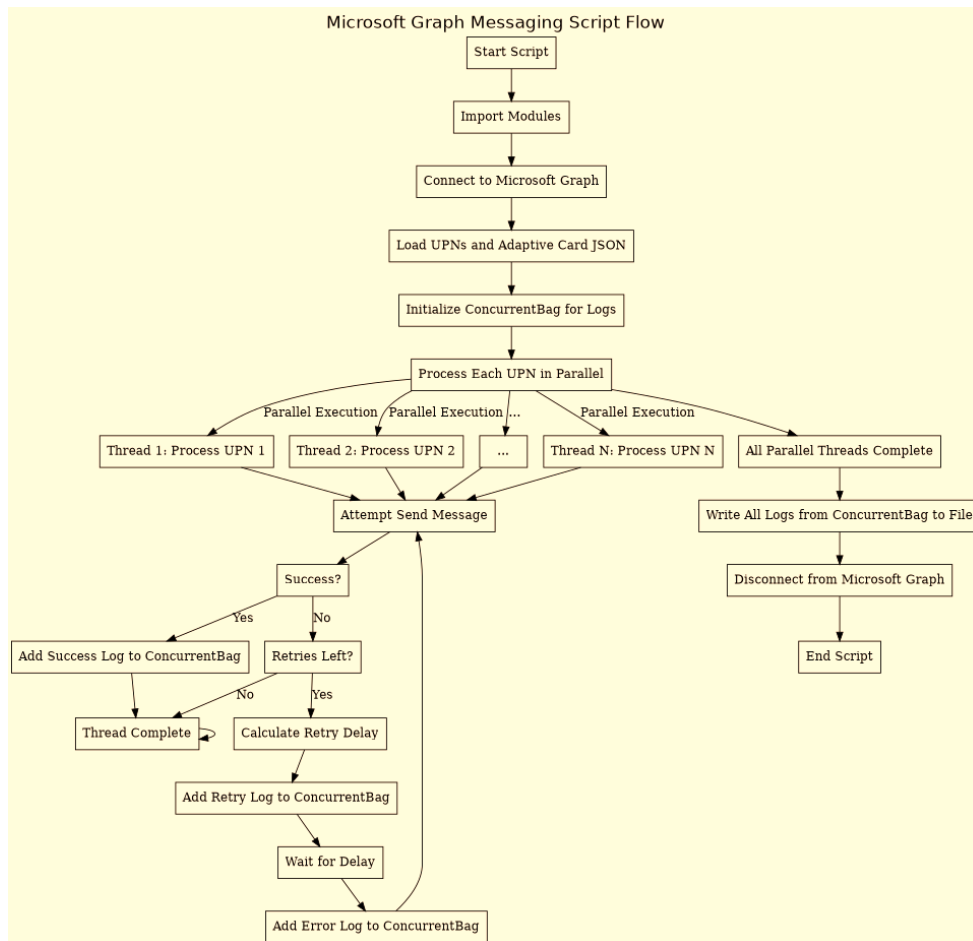
This document provides guidance on deploying a PowerShell script that sends Adaptive Card messages to users in Microsoft Teams. The script uses the Microsoft Graph API for communication, and it employs parallel processing to work efficiently while reducing file input and output operations.

The Adaptive Card, which is central to the script, contains six buttons. Each button corresponds to a specific prompt. When a user clicks on a button, the card triggers an action that opens a new browser tab. The chosen prompt is automatically copied into a dialog box in the browser. The user can then review the prompt, make edits to modify it if needed, and click a "Send" button to execute the prompt. This interactive design allows users flexibility to tailor the prompt before sending.

High-Level Design

The script operates based on the following high-level design principles:

1. **Authentication:** Establishes a connection to the Microsoft Graph API using modern authentication with specified scopes (Chat.ReadWrite, User.Read).
2. **Input Data:** Reads a list of User Principal Names (UPNs) from a specified Excel file using the ImportExcel module.
3. **Adaptive Card:** Loads the content of an Adaptive Card from a JSON file. This card will be sent as the message payload in Teams chats.
4. **Parallel Processing:** Utilizes the **ForEach-Object -Parallel** cmdlet (available in PowerShell 7+) to process the list of UPNs concurrently. This significantly speeds up the message sending process compared to sequential execution.
5. **Concurrent Chat Creation and Message Sending:** Within each parallel thread, the script performs the following steps for a given UPN:
 - Creates a one-on-one chat conversation with the target user and the script's executing user.
 - Constructs the message body, embedding the Adaptive Card as an attachment.
 - Sends the message containing the Adaptive Card to the newly created chat.
6. **Retry Mechanism:** Implements a retry loop with exponential backoff and jitter to handle transient errors during chat creation or message sending. It attempts to respect the Retry-After header from Graph API responses if present.
7. **Concurrent Logging:** Employs a [System.Collections.Concurrent.ConcurrentBag] to collect log messages (success, error, retry attempts) from all parallel threads in a thread-safe manner. This avoids multiple threads writing to the log file simultaneously, which can cause performance issues and data corruption.
8. **Single Log File Output:** After all parallel processing is complete, the script writes all the collected log messages from the ConcurrentBag to a single timestamped CSV log file. This minimizes file I/O operations during the execution.
9. **Cleanup:** Disconnects the Microsoft Graph session upon completion.



Prerequisites

Before deploying and running this script, ensure you have the following:

1. **PowerShell 7.0 or later:** ForEach-Object -Parallel is required.
2. **Required PowerShell Modules:**
 - **ImportExcel:** Used to read UPNs from an Excel file. Install using `Install-Module -Name ImportExcel`.
 - **Microsoft.Graph.Teams:** Contains cmdlets for interacting with Teams chats and messages. Install using `Install-Module -Name Microsoft.Graph.Teams`.
 - **Microsoft.Graph.Authentication:** Required for connecting to Microsoft Graph. Install using `Install-Module -Name Microsoft.Graph.Authentication`.
3. **Microsoft Graph Permissions:** The user account running the script must have sufficient permissions to create chats and send messages. The script requests **Chat.ReadWrite** and **User.Read** scopes, which are necessary. You may need to grant admin consent for these permissions in your Azure Active Directory/Microsoft Entra ID tenant.
4. **Input Excel File:** A .xlsx or .xls file containing a column named UPN with the User Principal Names of the target users.
5. **Adaptive Card JSON File:** A valid JSON file containing the structure of the Adaptive Card you wish to send.

6. **Account/ID for sending message:** One account to run the script which should have Teams license as the messages would be sent using this account

Deployment and Configuration

1. **Save the Script:** Save the provided PowerShell code as a .ps1 file (e.g., SendTeamsAdaptiveCard.ps1).
2. **Place Input Files:**
 - Place your Excel file containing UPNs in a known location.
 - Place your Adaptive Card JSON file in a known location.
3. **Configure Script Variables:** Open the .ps1 file in a text editor and modify the following variables in the "Configuration" section:

- **\$excelFilePath:** Set this to the full path of your input Excel file (e.g., "C:\Scripts\Users.xlsx").

```
# Input Excel file containing UPNs
$excelFilePath = "C:\Users\avichandra\Downloads\CopilotChat-Nudges\Users.xlsx"
```

- **\$adaptiveCardPath:** Set this to the full path of your Adaptive Card JSON file (e.g., "C:\Scripts\MyAdaptiveCard.json").

```
# Path to the Adaptive Card JSON file
$adaptiveCardPath = "C:\Users\avichandra\Downloads\CopilotChat-Nudges\adaptiveCardActions.JSON"
```

- **\$logfile:** The script automatically generates a timestamped log file name in the specified directory. Ensure the directory path ("C:\Users\avichandra\Downloads\CopilotChat-Nudges\" in the example) exists or can be created by the script. You can change this base directory.

```
# Generate timestamped log file name
$timestamp = Get-Date -Format "yyyyMMdd_HH:mm:ss"
$logFile = "C:\Users\avichandra\Downloads\CopilotChat-Nudges\CopilotMessageLog_$timestamp.csv"
```

- **\$ThrottleLimit:** Adjust this value based on your system's resources and the desired level of concurrency. A higher number means more threads run simultaneously. Start with a lower number (like 5 or 10) and increase if your system can handle it.

```
} -ThrottleLimit 5 # Adjust ThrottleLimit as needed
```

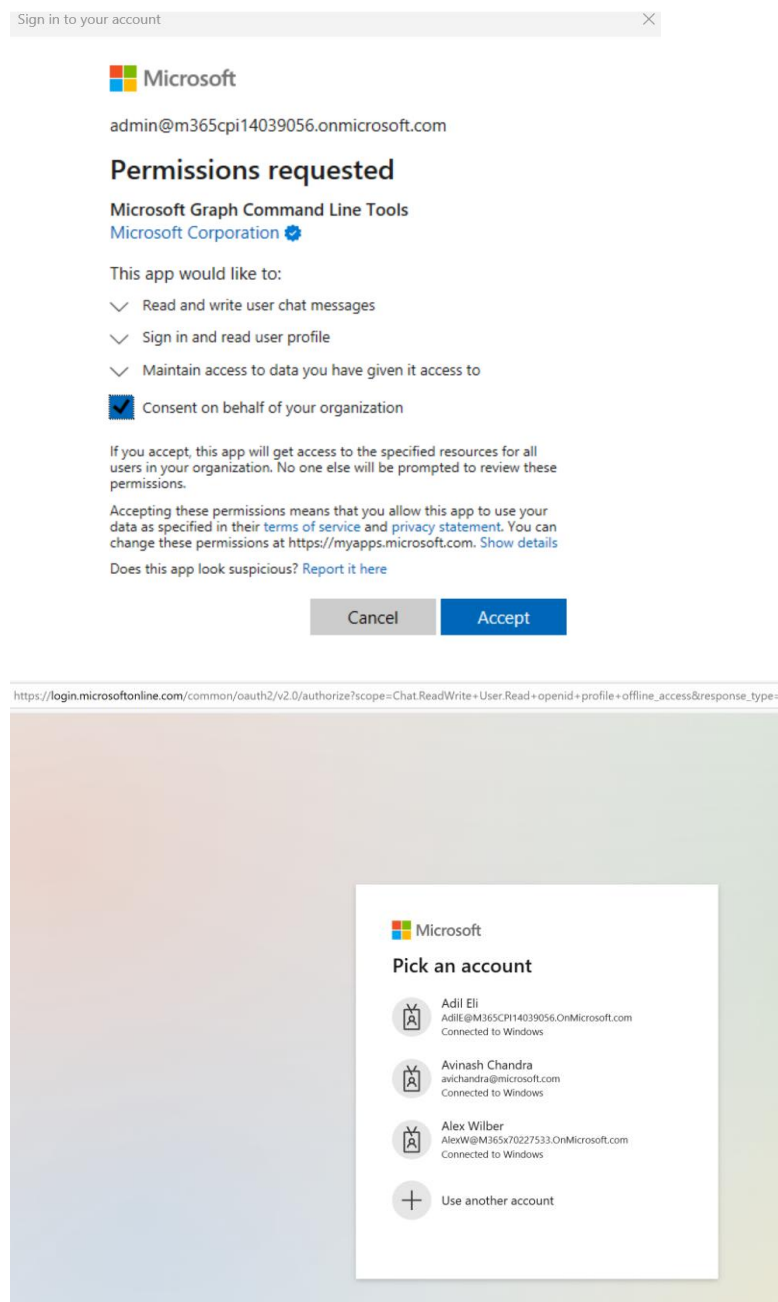
- **\$retryLimit:** Set the maximum number of times the script should retry sending a message to a user if an error occurs.

```
# --- Retry Settings ---
$retryLimit = 3
```

How to Run

1. Open PowerShell 7 or later.
2. Navigate to the directory where you saved the script using `cd <path_to_script_directory>`.
3. Run the script: `.\SendTeamsAdaptiveCard.ps1`
4. You will be prompted to authenticate with Microsoft Graph. Follow the on-screen instructions to sign in with an account that has the necessary permissions.

One Time consent is prompted for Graph permission:



5. The script will start processing the UPNs in parallel, displaying progress and status messages in the console.

```
PowerShell 7 (x64)
Welcome to Microsoft Graph!

Connected via delegated access using 14d82eec-204b-4c2f-b7e8-296a70dab67e
Readme: https://aka.ms/graph/sdk/powershell
SDK Docs: https://aka.ms/graph/sdk/powershell/docs
API Docs: https://aka.ms/graph/docs

NOTE: You can use the -NoWelcome parameter to suppress this message.

Successfully connected to Microsoft Graph.
Successfully loaded UPNs from Excel file.
Successfully loaded Adaptive Card JSON.
Starting parallel processing for 31 users...
[ ] Sending to Robink@M365CPI14039056.OnMicrosoft.com...
[ ] Sending to AmberR@M365CPI14039056.OnMicrosoft.com...
[ ] Sending to SashaO@M365CPI14039056.OnMicrosoft.com...
[ ] Sending to TeresaS@M365CPI14039056.OnMicrosoft.com...
[ ] Sending to Stevens@M365CPI14039056.OnMicrosoft.com...
[ ] Message sent to TeresaS@M365CPI14039056.OnMicrosoft.com
[ ] Sending to EkaS@M365CPI14039056.OnMicrosoft.com...
[ ] Message sent to Stevens@M365CPI14039056.OnMicrosoft.com
[ ] Sending to SoniaR@M365CPI14039056.OnMicrosoft.com...
[ ] Message sent to Robink@M365CPI14039056.OnMicrosoft.com
[ ] Sending to SydneyM@M365CPI14039056.OnMicrosoft.com...
[ ] Message sent to AmberR@M365CPI14039056.OnMicrosoft.com
[ ] Sending to VanceD@M365CPI14039056.OnMicrosoft.com...
[ ] Message sent to EkaS@M365CPI14039056.OnMicrosoft.com
[ ] Sending to Adams@M365CPI14039056.OnMicrosoft.com...
[ ] Message sent to SoniaR@M365CPI14039056.OnMicrosoft.com
[ ] Sending to CoreyG@M365CPI14039056.OnMicrosoft.com...
```

-
-
-
-
-
- 6.
7. Upon completion, the script will disconnect from Microsoft Graph and indicate the total time taken.

```
PowerShell 7 (x64)
[ ] Message sent to Rainier@M365CPI14039056.OnMicrosoft.com
[ ] Message sent to BillieV@M365CPI14039056.OnMicrosoft.com
[ ] Message sent to MarioR@M365CPI14039056.OnMicrosoft.com
[ ] Message sent to HadarC@M365CPI14039056.OnMicrosoft.com
Parallel processing finished.
Total Time Taken: 41.38 seconds
Writing collected logs to file: C:\Users\avichandra\Downloads\CopilotChat-Nudges\CopilotMessageLog_20250519_114605.csv
Log file updated successfully with collected messages.
Disconnecting from Microsoft Graph.

ClientId                : 14d82eec-204b-4c2f-b7e8-296a70dab67e
TenantId                : 2a892635-7a0a-4aa2-aea9-f8e1d12786a1
Scopes                   : {Chat.ReadWrite, openid, profile, User.Read...}
AuthType                 : Delegated
TokenCredentialType      : InteractiveBrowser
CertificateThumbprint    :
CertificateSubjectName   :
SendCertificateChain     : False
Account                  : admin@M365CPI14039056.onmicrosoft.com
AppName                  : Microsoft Graph Command Line Tools
ContextScope             : CurrentUser
Certificate               :
PSHostVersion            : 7.5.1
ManagedIdentityId       :
ClientSecret              :
Environment               : Global

Disconnected from Microsoft Graph.
PS C:\Users\avichandra\Downloads\CopilotChat-Nudges>
```

8. A log file named CopilotMessageLog_YYYYMMDD_HHMMSS.csv (where MMDD_HHMMSS is the timestamp) will be created in the specified log directory, containing the results of each message sending attempt.

Logging

The script uses a ConcurrentBag to collect log entries during parallel execution. This ensures that logging from multiple threads is handled safely.

- Each log entry includes a timestamp, the UPN of the target user, the status (Success, Error, or Retry), and a relevant message.
- These entries are stored in memory until all parallel tasks are finished.
- Finally, all collected logs are written to the timestamped CSV file in a single operation.

Error Handling and Retries

The script includes a try/catch block within the parallel loop to handle potential errors during chat creation or message sending.

- If an error occurs, the script will log the error and attempt to retry sending the message up to the number specified by \$retryLimit.
- It implements an exponential backoff strategy with jitter for retries, waiting for a longer duration after each failed attempt.
- If the error response from the Graph API includes a Retry-After header, the script will wait for the duration specified by that header before retrying.
- After reaching the \$retryLimit, the script logs a final error message for that user.

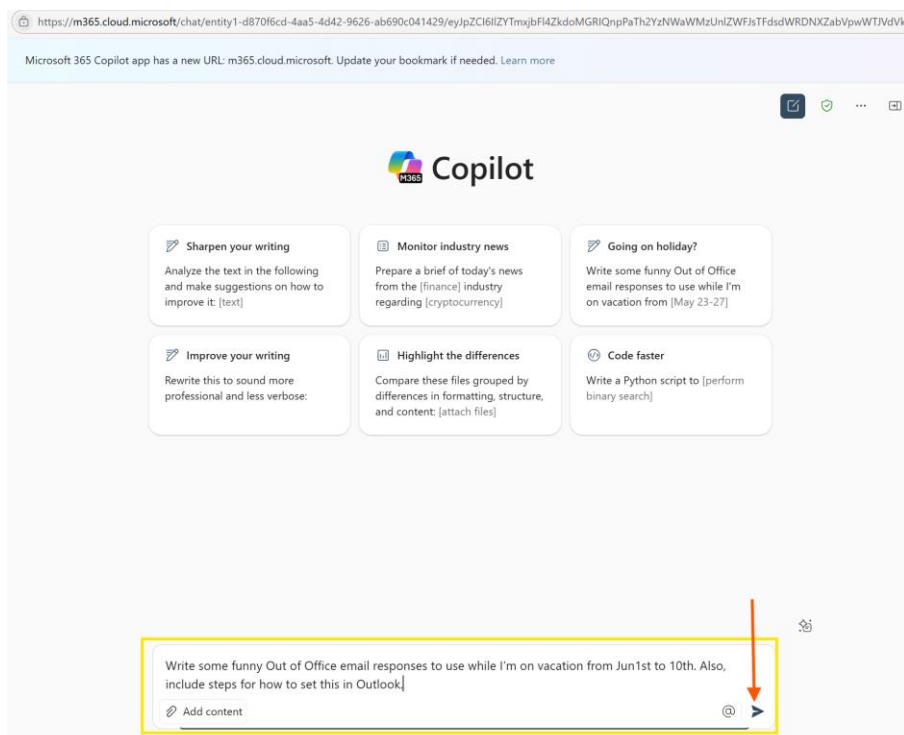
End user experience

End user will receive the one of the following adaptive card message. If script uses (**adaptiveCardActions.JSON**)



When the user click any of the button:

When the user clicks any button from the adaptive card, say he/she clicked a button labelled "Going on Holiday," the respective adaptive card action triggers, opening a new tab for Copilot chat with the prompt copied in the Prompt Dialog box. The user can click send or modify the prompt as needed. See the screenshot below for reference.



If script uses the other adative card (**adaptiveCardMsg.JSON**) user will receive below message.

Discover the power of M365 Copilot Chat!


Microsoft 365 Copilot Chat is an approved and secure AI chat tool built specifically for work. It uses the latest AI models and data from the web to answer your questions, generate content and ideas, and find information. With Copilot Chat you can:

- Upload files to get responses based on information from your work documents, presentations, etc.
- Generate images and data visualizations.
- Refine and collaborate on chat responses using Copilot Pages.

Here are some prompts you can try in M365 Copilot Chat. Copy and use them in Copilot Chat by clicking on Try Now

Top 5 Prompts to try:

- How does [product] compare to [competitor product]? Provide the comparison in a table.?
- Draft an email to [name] that informs them that [Project X] is delayed two weeks. Make it short and casual in tone.
- Translate the following text into French: [text].
- Explain how a large language model works in simple terms.
- Write an outline for a creative brief for a video project.



Try Now

Troubleshooting

Here are some common issues you might encounter and how to resolve them:

1. ForEach-Object -Parallel is not recognized:

- **Issue:** This cmdlet parameter is only available in PowerShell 7.0 and later.
- **Solution:** Ensure you are running the script in a PowerShell 7 or newer environment. You can check your version by typing `$PSVersionTable.PSVersion` in a PowerShell console.
- If you have an older version (like Windows PowerShell 5.1), you will need to install PowerShell 7 from the official Microsoft documentation.

[Installing PowerShell on Windows - PowerShell | Microsoft Learn](#)

```
winget install --id Microsoft.PowerShell --source winget
```

PowerShell

```
winget install --id Microsoft.PowerShell --source winget
```

2. Script cannot be loaded because the execution of scripts is disabled on this system:

- **Issue: Your PowerShell execution policy is set to restrict running unsigned scripts.**
- **Solution:** You can temporarily or permanently change the execution policy to allow running local scripts.
 - Check current policy: **Get-ExecutionPolicy -list**
 - Allow local scripts (recommended for running your own scripts): **Set-ExecutionPolicy -ExecutionPolicy RemoteSigned -Scope CurrentUser**
 - Allow all scripts (use with caution): **Set-ExecutionPolicy -ExecutionPolicy Unrestricted -Scope CurrentUser**
or
 - **Set-ExecutionPolicy -Scope Process -ExecutionPolicy Bypass**

3. Permission errors when connecting to Microsoft Graph or sending messages:

- **Issue:** The user account used for authentication does not have the necessary Microsoft Graph permissions (**Chat.ReadWrite, User.Read**).
- **Solution:** Verify that the account used to Connect-MgGraph has been granted the required API permissions in Azure Active Directory/Microsoft Entra ID. If using an app registration, ensure the correct permissions are assigned and consented to. You may need administrative privileges to grant consent.

4. Module not found errors (e.g., ImportExcel, Microsoft.Graph.Teams):

- Issue: The required PowerShell modules are not installed on the system where you are running the script.
- Solution: Install the missing modules using `Install-Module -Name <ModuleName>`. For example:
- `Install-Module -Name ImportExcel`
- `Install-Module -Name Microsoft.Graph.Teams`
- `Install-Module -Name Microsoft.Graph.Authentication`

You might need to run PowerShell as an administrator to install modules globally, or you can install them for the current user using the `-Scope CurrentUser` parameter.

Cleanup

The script automatically disconnects from the Microsoft Graph session at the end of execution using `Disconnect-MgGraph`

Disclaimer

This is a sample script provided for demonstration purposes only. It is your responsibility to test and validate the script thoroughly before using it in production. Modify it to meet your organizational and compliance requirements.