

ATC EUTECTICS LLC

ATC Automated Flows

Let your AI bot work with Outlook email and Microsoft Teams chats, without ever signing in to your Microsoft account.

Outlook Inbox to Bot | Outlook Send from Bot | Outlook Get Attachments for Bot | Teams Chat to Bot | Teams Send from Bot | Outlook Get Sent

Product guide & technical setup · Version 1.4.1 · 19 October 2026

atceutectics.com

Contents

> **About this guide (version 1.4.1).** Part 1 explains each flow in plain language. It describes the flows with the changes listed in Part 2, Appendix A; Part 2, section 1 (“Fix status”) shows which packages include them. This guide is not an offer and does not set prices, terms or support commitments.

Part 1 · Product guide (User Guide v1.4.0)

1. What these flows do
2. The six flows at a glance
3. Each flow in plain language
4. What to expect
5. How bots use these files
6. Frequently asked questions
7. Getting started

Part 2 · Technical setup (Technical Setup & Deployment Guide v1.4.2)

1. Overview
2. Flow reference (from the exported definitions)
3. Prerequisites
4. Import each package
5. Configure folders and paths
6. Turn on and test
7. Run-failure alerts
8. Security notes
9. Troubleshooting
10. Updating and versioning
11. Uninstall
12. Known limits and customization

Appendix A. Flow fixes (step by step)

PART 1

Product guide

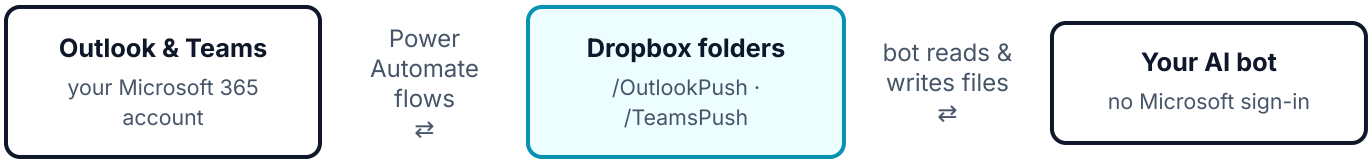
For the person whose email and Teams chats the bot will work with.

1. What these flows do

ATC Automated Flows are six small automations that run in Microsoft Power Automate on your behalf. They let an AI bot work with your Outlook email and Microsoft Teams chats **without the bot ever logging in to your Microsoft account**. Instead, everything passes through a set of folders in Dropbox:

- **Incoming:** when an email or chat message arrives, a flow saves a copy into Dropbox, where your bot can read it.
- **Outgoing:** when your bot wants to send an email or a chat message, it drops a small instruction file into a Dropbox "outbox" folder. A flow picks it up and sends the message from your account.

You do not need to do anything day to day. Your IT contact sets the flows up by importing ATC's ready-made packages (all fixes are included and tested), then switches them on. After that they run in the background.



2. The six flows at a glance

#	Flow	Direction	What it does	Where files appear in Dropbox
1	Outlook Inbox to Bot	Outlook → bot	Saves each new Inbox email as a small text file (no attachments)	/OutlookPush/Inbox
2	Outlook Send from Bot	Bot → Outlook	Sends a new email, or replies to one, from your mailbox	Bot writes to /OutlookPush/Outbox (attachments in /OutlookPush/Outfiles)
3	Outlook Get Attachments for Bot	Outlook → bot (on request)	When the bot asks, saves the attachments of a specific email, plus a full copy of that email	/OutlookPush/Files (attachments and email copy) and /OutlookPush/Fetched (list)
4	Teams Chat to Bot	Teams → bot	Saves each new Teams chat message as a small text file	/TeamsPush/Inbox
5	Teams Send from Bot	Bot → Teams	Posts a message into a Teams chat as you	Bot writes to /TeamsPush/Outbox
6	Outlook Get Sent	Outlook → bot	Saves a full copy of each email you send, including attachments	/OutlookPush/Sent

The numbering matches the Technical Setup Guide. Folder names are not case-sensitive in Dropbox, so /OutlookPush/Inbox and /outlookpush/inbox are the same folder.

3. Each flow in plain language

3.1 Outlook Inbox to Bot

1. A new email arrives in your Outlook **Inbox**.
2. The flow converts the email body to plain text.
3. It saves a small file (ending in **.json**) into **/OutlookPush/Inbox**. The file holds the sender, recipients, subject, time received, the plain-text body, a conversation ID (so replies can be grouped), and a yes/no flag for whether the email has attachments.
4. Attachments are **not** copied by this flow. If the bot needs them, it uses *Outlook Get Attachments for Bot* (3.3).

The file name is a coded version of Outlook's internal message ID. It looks like a long string of letters and symbols. That is normal.

3.2 Outlook Send from Bot

1. The bot writes an instruction file into **/OutlookPush/Outbox**. The file says either "send a new email" (with To, Cc, Bcc, subject and text) or "reply" to a particular email (optionally reply-all).
2. If the bot wants to attach files, it first places them in **/OutlookPush/Outfiles**. For safety, the flow only accepts attachments from that folder. Any other location makes the flow stop without sending.
3. The flow sends the email from **your** mailbox. It appears in your Sent Items just as if you had sent it yourself, so *Outlook Get Sent* will also archive it.
4. After sending, the flow deletes the instruction file and the attachment files it used.
5. If sending fails because of a temporary problem, the flow tries again up to two more times, a minute apart. Very occasionally a retry can send a duplicate, if Microsoft actually sent the first attempt but didn't confirm it in time. If it still can't send, it moves the instruction file to **/OutlookPush/Outbox/Failed** and sends an alert email. The alert is sent **from your own mailbox** to the address chosen at setup (normally your own mailbox, or your IT contact), so a copy also appears in your Sent Items and in /OutlookPush/Sent. A request in that folder is not sent again automatically; it can be checked and re-sent.

3.3 Outlook Get Attachments for Bot

This flow only runs when your bot asks for it.

1. The bot drops a small request file naming one email into **/OutlookPush/Fetch**.
2. The flow looks up that email. If it has any attachments, the flow first saves a **full copy of the whole email** into **/OutlookPush/Files** (ending in `_parent.eml`).
3. It then saves each attachment into **/OutlookPush/Files**. The files are renamed with the email's coded ID and a number, keeping the original file type (for example `…_1.pdf`, `…_2.xlsx`).
4. Finally it writes a list into **/OutlookPush/Fetched** showing the full copy first, then each attachment's original name, its type, its size and the saved file name. The list also says whether the job is "complete" or "incomplete".

5. The request file is always removed from /OutlookPush/Fetch, even if something went wrong (the list then says "incomplete").
6. If the bot asks for the same email twice, the saved files (including the full copy) are simply replaced.
7. An email with no attachments gets an empty list and no full copy.

Size note: attachments larger than 20 MB are skipped. They still appear in the list, but with no saved file. Small pictures inside the email body (for example logos in signatures) count as attachments too and are marked as "inline".

Attached emails: Outlook does not hand over an attached email (or calendar item) as a file of its own. It is inside the full copy (`_parent.eml`), so the bot opens it from there. Because the full copy is saved for every email that has attachments, /OutlookPush/Files holds one extra file per such email, about the size of the email.

3.4 Teams Chat to Bot

1. A new message is posted in one of your Teams **chats**.
2. The flow saves the message as a small .json file in **/TeamsPush/Inbox**. The file holds who it was from, when it was sent, the text, a link to the message, and the chat's ID so the bot can answer in the same chat.
3. Only the text is saved. Files and images shared in the chat are not copied.
4. **Your own messages are saved too**, as well as the bot's replies, so the bot can follow the whole conversation. This is intended. If you'd rather the bot only saw other people's messages, ATC can change it during setup.
5. In a very busy chat (a quick burst of many messages), the odd message can be missed. This is a known limit.

3.5 Teams Send from Bot

1. The bot writes a small instruction file (which chat, and what to say) into **/TeamsPush/Outbox**.
2. The flow posts the message into that Teams chat **as you**. This works in group chats and, in ATC's testing, in one-to-one chats. Your IT contact re-checks one-to-one chats during setup in your organisation.
3. The instruction file is deleted once the message has been posted.

3.6 Outlook Get Sent

1. You send an email (from Outlook on any device, or via the bot).
2. When it lands in your **Sent Items** folder, the flow exports the complete message, including attachments.
3. It saves it into **/OutlookPush/Sent** as an **.eml** file named with the date and time it was processed (UTC) followed by a coded version of the message ID, for example `261015143005_<coded message id>.eml` (year, month, day, hour, minute, second). It uses the same coding as the Inbox and attachment files.
4. An .eml file can be opened in Outlook or most email programs if you ever need to look at it yourself.

Size note: the .eml file is larger than the attachments it contains. In ATC testing, an email with a 16 MB attachment produced a 22.5 MB file, and an email with a small picture produced a 358 KB file. 16 MB attachments are the verified size; much larger emails may not be copied.

Only new mail: this flow starts from the moment it is switched on. It does not copy emails you sent before then.

4. What to expect

- **Timing:** most items show up within a minute or two. The outgoing flows check their Dropbox outbox about once a minute. Microsoft's email and Teams triggers can occasionally take a few minutes longer at busy times.
- **Messages are sent as you.** Anything the bot sends by email or Teams comes from your account, with your name.
- **Files build up.** Copies saved into /OutlookPush/Inbox, /OutlookPush/Sent, /OutlookPush/Files, /OutlookPush/Fetched, /OutlookPush/Outbox/Failed and /TeamsPush/Inbox are not deleted automatically. Your bot or your IT contact should tidy them up from time to time.
- **Large items:** very large emails or attachments may take longer, and the largest may not be copied (see the size notes above).

5. How bots use these files

A typical bot works like this:

1. It watches **/OutlookPush/Inbox** and **/TeamsPush/Inbox** for new files and reads them.
2. If an email has attachments it needs, it puts a request in **/OutlookPush/Fetch**, then reads the results from **/OutlookPush/Fetched** and **/OutlookPush/Files**.
3. It can read **/OutlookPush/Sent** to see what you have already sent, for context or follow-up.
4. To answer, it writes an instruction file to **/OutlookPush/Outbox** (email) or **/TeamsPush/Outbox** (Teams).

Your bot provider configures this. You do not need to create or edit these files yourself.

6. Frequently asked questions

Does the bot get my Microsoft password? No. The flows run inside your own Microsoft 365 account. The bot only sees what the flows place in Dropbox.

Will the bot see all my old email? Not automatically. The automatic copies (Inbox, Sent and Teams) only include email and chat messages that arrive, or that you send, after the flows are switched on. However, if the bot has the ID of an older email that is still in your mailbox, it can ask *Outlook Get Attachments for Bot* for that email's attachments (and, if it has attachments, a full copy of the email). It can also reply to that email. The bot normally gets IDs from the copies it already has, but these two flows are not limited to new mail. Your bot's settings decide what it is allowed to request.

An email I expected isn't in Dropbox. What should I do? Wait a couple of minutes, then check the folder again. If it still hasn't appeared, contact your IT person. They can check the flow's run history.

Why are the file names so long and strange? They are based on Outlook's and Teams' internal message IDs, so each file is unique and can be matched to the original message.

Can I open the files myself? Yes. The .json files are plain text. The .eml files in /OutlookPush/Sent, and the `_parent.eml` copies in /OutlookPush/Files, open in Outlook or most email programs.

Can I delete files from these folders? You can delete old copies in /OutlookPush/Inbox, /OutlookPush/Sent, /OutlookPush/Files, /OutlookPush/Fetched and /TeamsPush/Inbox, and checked requests in /OutlookPush/Outbox/Failed. Please don't touch the Outbox, Outfiles or Fetch folders while the bot is working, because the flows use those as work queues.

I got an "ATC flow failed" or "ATC flow blocked" email. What now? Both alerts mean an email the bot asked to send did **not** go out, and the request was moved to /OutlookPush/Outbox/Failed. The alert comes from your own mailbox (Outlook Send from Bot sends it with your account) to the address chosen at setup, so you will also see it in your Sent Items. The first time, check your Junk folder and mark it "Not junk". If the flow's Outlook connection itself is broken, no alert can be sent; the request still moves to the Failed folder, and the failure still shows in the flow's run history. - **"ATC flow failed: Outlook Send from Bot"** means Microsoft refused or couldn't complete the send, even after retries (for example, a bad address, an email that no longer exists, or a service outage). - **"ATC flow blocked: attachment path outside Outfiles"** means the bot tried to attach a file from a folder it isn't allowed to use. This is a safety check, and nothing was sent.

In both cases, forward the alert to your IT contact or bot provider. The link in the alert opens the run details. Once the cause is fixed, the request can be uploaded to /OutlookPush/Outbox again. Don't re-send it yourself if you aren't sure whether it went out.

Do I need a special Microsoft licence? No premium licence should be needed. All the connectors are Microsoft "Standard" connectors. Your IT contact confirms this against your organisation's licence during setup, and ATC checks it as part of assisted setup.

Does the bot send emails without my approval? That depends on how your bot is set up. The flows send whatever instruction files appear in the outbox folders, so approval rules belong in the bot's settings. Ask your bot provider.

How do I stop everything? Ask your IT person to turn the flows off in Power Automate. Nothing is sent or copied while they are off.

Who do I contact for help? Your IT contact first. For product questions, contact ATC Eutectics LLC via atceutectics.com.

7. Getting started

Start by contacting ATC. ATC can set the flows up with you and your IT contact (assisted setup), or your IT administrator can set them up using Part 2 of this guide (self setup).

Pricing and availability: contact us at **atceutectics.com/contact**.

PART 2

Technical setup & deployment

For the customer’s IT administrator.

> Changes in this edition (v1.4.1, 19 Oct 2026): the package names in section 1 no longer carry a build tag (flows 2, 3 and 5; same packages). Changes in 1.4.0 (19 Oct 2026): Fixes 4, 6, 7 and 8 are in the packages and tested; package table updated to the 19 Oct packages, with flows 2 and 5 renamed; Fix 8 redesigned (a full copy of the email whenever it has attachments, new `kind:"parent"` manifest entry); the Fix 7 alert uses Office 365 Outlook Send an email (V2) to the flow's own mailbox, so the Mail connector is no longer needed; Safeld decoding corrected (only a trailing `-` becomes `=`); test notes added.

1. Overview

ATC Automated Flows are six Microsoft Power Automate cloud flows, delivered as legacy export packages (.zip). They connect one user's Outlook mailbox and Microsoft Teams chats to an AI bot, using Dropbox as the exchange layer. The bot never holds Microsoft 365 credentials. It only reads and writes files in agreed Dropbox folders.

#	Package (latest build)	Display name inside that package	Customer name	Trigger	Dropbox folders
1	OutlookInboxtoBot template_20260930133301.zip	Outlook inbox to bot	Outlook Inbox to Bot	Office 365 Outlook — When a new email arrives (V3), Inbox	writes /OutlookPush/Inbox
2	OutlookSendfromBot_20261019132957.zip	Outlook Send from Bot	Outlook Send from Bot	Dropbox — When a file is created, /OutlookPush/Outbox (polls every 1 min)	reads /OutlookPush/Outbox, /OutlookPush/Outfiles; writes /OutlookPush/Outbox/Failed
3	OutlookGetAttachmentsforBot_20261019141223.zip	Outlook Get Attachments for Bot	Outlook Get Attachments for Bot	Dropbox — When a file is created, /OutlookPush/Fetch (polls every 1 min)	reads /OutlookPush/Fetch; writes /OutlookPush/Files, /OutlookPush/Fetched
4	TeamsChattoBotTemplate_20260929151131.zip	Teams chat to bot	Teams Chat to Bot	Microsoft Teams — When a new chat message is added	writes /TeamsPush/Inbox
5	TeamsSendfromBot_20261019131006.zip	Teams Send from Bot	Teams Send from Bot	Dropbox — When a file is created, /TeamsPush/Outbox (polls every 1 min)	reads /TeamsPush/Outbox

#	Package (latest build)	Display name inside that package	Customer name	Trigger	Dropbox folders
6	OutlookGetSent_20261019123853.zip	Outlook Get Sent	Outlook Get Sent	Office 365 Outlook — When a new email arrives (V3), Sent Items	writes /OutlookPush/Sent

Paths are stored in lowercase in the flow definitions (for example `/outlookpush/inbox`). Dropbox paths are case-insensitive, so the mixed-case names in this guide refer to the same folders.

Package names. The six customer package names are **Outlook Inbox to Bot**, **Outlook Send from Bot**, **Outlook Get Attachments for Bot**, **Teams Chat to Bot**, **Teams Send from Bot** and **Outlook Get Sent**. The packages for flows 1 and 4 still carry an older internal display name; give them the customer name in the import's **Create as new** step (section 4).

Renamed packages. `OutlookSendfromBot_…` and `TeamsSendfromBot_…` packages exist from 19 October 2026 (listed above). The older zips (`OutlookSendfromBottemplate_20261001131905.zip`, `TeamsSendTemplate_20260929141502.zip` and earlier builds of flows 3 and 6) stay in place and are not deleted, but deploy the packages in the table. Zips with a build tag between the flow name and the timestamp are ATC build copies or superseded builds. **Do not deploy** them: one earlier 19 Oct build of Outlook Send from Bot sends its alert with the Mail connector and is superseded by the package in the table.

Fix status. The packages listed above include all the Appendix A fixes (Fixes 4, 6, 7 and 8), tested by ATC on 19 October 2026. Sections 2.2, 2.3 and 2.6 describe these packages. Customers who deploy them don't make the Appendix A edits; they only set the alert recipient (Fix 7, section 12.1).

2. Flow reference (from the exported definitions)

2.1 Outlook Inbox to Bot

- **Trigger:** Office 365 Outlook *When a new email arrives (V3)* on the Inbox folder. Split-on is enabled, so each email gets its own run. No other trigger filters are set.
- **Actions:**
  1. *Html to text* (Content Conversion) on the email body.
  2. *Select + Compose (SafeId)*: builds a Dropbox-safe, case-preserving name from the Outlook message ID. Each lowercase character is prefixed with `~` (Dropbox names are case-insensitive), `/` becomes `_` and `=` becomes `-`.
  3. *Compose (MessageJson)* with these fields: `id` (SafeId), `messageId`, `at` (receivedDateTime), `from`, `to`, `subject`, `text` (converted text, or Outlook's bodyPreview if the conversion is empty), `conversationId`, `hasAttachments`.
  4. Dropbox *Create file* at `/outlookpush/inbox`, named `<SafeId>.json`.
  5. *Condition (AlreadyThere)*, which runs only if Create file fails. If the failure is HTTP 409 or "already exists" (a duplicate), the run ends normally. Otherwise the run is terminated as **Failed**.
- **Attachments:** not copied. `hasAttachments` tells the bot to use flow 3, Outlook Get Attachments for Bot (section 2.3).

2.2 Outlook Send from Bot

- **Trigger:** Dropbox *When a file is created* in `/outlookpush/outbox`, polling every 1 minute.
- **Request file (JSON) schema** as defined in Parse JSON:

```
{
    "action": "send" | "reply",
    "to": "<address 1>;<address 2>",
    "cc": "", "bcc": "",
    "subject": "…",
    "text": "…",
    "messageId": "<Outlook message id, for replies>",
    "replyAll": false,
    "attachments": [ { "Name": "report.pdf", "path": "/outlookpush/outfiles/report.pdf" } ]
}
```

- **Actions:**
  1. *Get file content* → *Parse JSON*.
  2. Initialise variables `MailFiles` (array), `FileIds` (array) and `BadPath` (boolean).
  3. *Apply to each* attachment, one at a time. If the path starts with `/outlookpush/outfiles/` (case-insensitive) and does not contain `..`, the flow runs *Get file metadata using path* and *Get file content using path*, then appends `{Name, ContentBytes}` to MailFiles and the Dropbox file ID to FileIds. Any other path sets `BadPath = true`.
  4. *Condition (PathOk)*: if BadPath is true, the run is terminated as **Failed** and nothing is sent.
  5. *Condition (IsReply)*: if `action` = `"reply"`, the flow runs Office 365 Outlook *Reply to email (V3)* with messageId, Cc, Bcc, body, ReplyAll and attachments. Otherwise it runs *Send an email (V2)* with To, Subject, body, Cc, Bcc, attachments and Importance Normal. The body is `text` wrapped in an HTML paragraph.
  6. Deletes each used attachment from Dropbox, then deletes the request file.
- **Notes:** for replies, the `to` and `subject` fields are not passed to the reply action. Any field not listed in the schema is ignored. Mail is sent as the owner of the Office 365 Outlook connection.
- **Failures (Fix 7, Appendix A.4):** Send and Reply retry up to 2 times, 1 minute apart, for temporary errors; permanent errors such as a bad message ID are not retried. A request that fails, or is blocked by the path check, is copied to /OutlookPush/Outbox/Failed, deleted from /OutlookPush/Outbox, and an alert email is sent with Office 365 Outlook *Send an email (V2)* to the flow's own mailbox; the run then ends as **Failed**. The alert needs the Outlook connection to be healthy, and its copy appears in /OutlookPush/Sent through flow 6. Retries can't rule out a duplicate: if Outlook sends the email but the response times out, the retry sends it again. This is uncommon.

2.3 Outlook Get Attachments for Bot

- **Trigger:** Dropbox *When a file is created* in `/outlookpush/fetch`, polling every 1 minute.
- **Request file:** `{ "messageId": "<Outlook message id>" }`. The `messageId` field is required.
- **Actions:**
  1. *Get file content* → *Parse JSON* → *Select/Compose SafeId*, using the same scheme as 2.1.
  2. *Get email (V2)* without attachment content, to read the attachment list. This list contains file attachments only; attached Outlook items (emails, calendar items) are not in it.

3. *Condition (HasParent)*: if the email has any attachment, *Export email (V2)* saves a full copy of the email as `<SafeId>_parent.eml` in `/outlookpush/files` (replacing an existing copy), and the manifest gets a first entry for it: `{name: "original-message.eml", contentType: "message/rfc822", size: 0 (not reported), file: "<SafeId>_parent.eml", kind: "parent"}`. Attached emails are inside that copy.
4. *Apply to each* attachment, one at a time (it runs whatever the outcome of HasParent):
   - If size ≤ 20,971,520 bytes (20 MB): the flow increments a counter and works out the extension from the attachment name. First it drops everything from the first `@` onward, because some inline images have names like `image001.png@01DA1234.5678`. Then it takes the text after the last dot, trimmed and lowercased. Examples: `Report.Final.PDF` gives `pdf`, and `image001.png@01DA1234.5678` gives `png`. If the name has no dot, or the result is longer than 8 characters or contains a space or a character not allowed in Windows file names, `bin` is used. The flow then runs *Get Attachment (V2)*. If content bytes are returned, it runs Dropbox *Create file* in `/outlookpush/files` as `<SafeId>_<n>.<ext>`.
   - Each attachment is recorded as `{name, contentId, isInline, contentType, size, file}`. `file` is empty if the attachment was over 20 MB. A listed attachment that returns no bytes is recorded with `kind: "embedded"` and `file` pointing to `<SafeId>_parent.eml`. Existing files are replaced (Fix 6).
5. *Compose (Manifest)*: `{id, messageId, status: "complete" | "incomplete", files: [...]}`. The status is "complete" only if both the loop and HasParent succeeded.
6. The manifest is saved as `/outlookpush/fetched/<SafeId>.json` (replacing an existing one), then the request file is always deleted, even if the run fails.
7. If the status is "incomplete", the run is terminated as **Failed**.

- **Reruns and attached items (Fixes 6 and 8, Appendix A.2, A.3):** fetching the same message again succeeds and replaces the files in place, and the request is always cleared. Attached emails and other Outlook items are inside `<SafeId>_parent.eml`. A full copy is saved for every email that has any attachment, which adds one .eml (about the size of the email) per such email. Emails without attachments get no copy and `"files": []`.

2.4 Teams Chat to Bot

- **Trigger:** Microsoft Teams *When a new chat message is added* (webhook trigger) for chats that the connection's user belongs to. No filters are set.
- **Actions:**
  1. Office 365 Users *Get my profile (V2)*.
  2. For each trigger item: Teams *Get messages in a chat* (top 20) for the chat, then *Filter array* for the triggering message ID.
  3. If the message is found: *Html to text* on the body (or summary), then *Compose* `{id (Teams messageId), at (createdDateTime), from (user or app display name, else "unknown"), text, link (linkToMessage), conversationId}`.
  4. Dropbox *Create file* `/teamspush/inbox/<messageId>.json`.
- **Connections:** two Teams connection references (one for the trigger and one for Get messages). Both can use the same account. Office 365 Users and Content Conversion are also required.
- **Own messages are captured by design.** The "NotMine" condition is deliberately left open (always true), so the user's own chat messages, and the bot's own posts made through Teams Send from Bot, are

written to /TeamsPush/Inbox too. The bot sees the whole conversation. This can be customized per customer (section 12.1).

2.5 Teams Send from Bot

- **Trigger:** Dropbox *When a file is created* in `/teamspush/outbox`, polling every 1 minute.
- **Request file:** `{ "conversationId": "<Teams chat id>", "text": "…" }`.
- **Actions:** *Get file content* → *Parse JSON* → Teams *Post message in a chat or channel* (Post as: User; Post in: Group chat; Recipient: conversationId; Message: text in an HTML paragraph) → Dropbox *Delete file* (the request file), only if the post succeeded.
- **1:1 chats:** although the setting reads "Group chat", posting into one-to-one chats works (verified by ATC). Re-confirm in each new tenant using the section 6 checklist.

2.6 Outlook Get Sent

- **Trigger:** Office 365 Outlook *When a new email arrives (V3)* on **Sent Items**. Include Attachments = Yes, Importance = Any, Only with Attachments = No, split-on enabled.
- **Actions:**
  1. *Export email (V2)* by the trigger's message ID (full MIME message).
  2. *Select* + *Compose (SafeId)*, using the same scheme as 2.1, then Dropbox *Create file* in `/outlookpush/sent` named `concat(formatDateTime(utcNow(),'yyMMddHHmmss'),'_',outputs('SafeId'),'.eml')` (Fix 4, Appendix A.1). The timestamp is processing time in UTC. Real names look like `261019121141_AAM~kAD~d~h…~f~gE2AAA-.eml`: 12 digits, `_`, the SafeId, `.eml`. Mail sent from older packages used `yyyy-MM-dd_HHmmss_<raw message id>.eml`, so both forms can be in the folder.
- **Matching a file to its message:** take the part after the **first** `_` (the SafeId can itself contain `_`), remove `.eml`, then remove each `~`, turn `_` back into `/`, and turn **only a trailing** `-` back into `=`. Outlook IDs can contain `-` natively, so any other `-` stays. The `Message-ID:` header inside the .eml also identifies the message, so it can be matched against the original email or replies.
- **Size, verified (ATC, 15 Oct 2026):** an email with a PNG produced a 358 KB .eml, and an email with a 16 MB attachment produced a 22.5 MB .eml (MIME/base64 overhead). 16 MB attachments are the verified size. Larger messages have not been tested; the Dropbox connector's file limit is 50 MB.
- **No backfill:** only mail that reaches Sent Items after the flow is turned on is captured.
- **No error handling** is built in. Power Automate's own failure emails to the flow owner still apply. A dedicated alert is optional (section 7).

3. Prerequisites

1. **Microsoft 365 licence** that includes Power Automate for cloud flows using standard connectors (most Microsoft 365 business and enterprise plans do). The flow owner must be able to sign in at make.powerautomate.com.
2. **Exchange Online mailbox** for the user (Outlook flows) and **Microsoft Teams** for the user (Teams flows).
3. **Dropbox account** (personal or Business). The Dropbox connector supports personal and shared folders. Team folders are not supported.
4. **Connectors used** (all Standard):

Connector	Used by	Class*
Office 365 Outlook	Flows 1, 2, 3, 6	Standard
Dropbox	All six	Standard
Content Conversion	Flows 1, 4	Standard
Microsoft Teams	Flows 4, 5	Standard
Office 365 Users	Flow 4	Standard
Mail (optional, for alerts you add, section 7)	None by default	Standard

*The Dropbox and Mail classes are confirmed in Microsoft's connector reference ("Power Automate: Standard"). The others are Microsoft first-party connectors classed as Standard.

Licensing: all connectors used by these flows are Standard. Each customer confirms this against their own Microsoft 365 / Power Platform licence. ATC checks this as part of assisted setup, and the self-setup checklist covers it too (section 12.3).

5. **Data Loss Prevention (DLP):** if your tenant has Power Platform DLP policies, Dropbox, Office 365 Outlook, Microsoft Teams, Office 365 Users, Content Conversion and Mail (if you add Mail alerts, section 7) must all be in the same data group (for example "Business"). Otherwise the flows will be suspended.
6. **Dropbox folders** created in advance (Dropbox triggers need an existing folder to select):

```
/OutlookPush/Inbox      /OutlookPush/Sent
/OutlookPush/Outbox     /OutlookPush/Outfiles
/OutlookPush/Outbox/Failed   (used by flow 2)
/OutlookPush/Fetch      /OutlookPush/Fetched
/OutlookPush/Files
/TeamsPush/Inbox        /TeamsPush/Outbox
```

4. Import each package

Repeat for each of the six zips. Import one at a time, and use the latest timestamped zip of each flow.

1. Sign in to **make.powerautomate.com** as the user whose mailbox and Teams will be connected, and select the correct environment (top right).
2. Go to **My flows › Import › Import Package (Legacy)**.
3. Click **Upload**, choose the .zip, and wait for **Review package content**.
4. Under **Resource type: Flow**, click the Action/Setup cell (it shows *Update*). Choose **Create as new** and enter a clear name, for example `ATC – Outlook Inbox to Bot`. Click **Save**.
5. For each **Related resource (connection)** in the package, click **Select during import**. Each package lists these:

Package	Connections to select
1 Outlook Inbox to Bot	Office 365 Outlook, Content Conversion, Dropbox

Package	Connections to select
2 Outlook Send from Bot	Office 365 Outlook, Dropbox (the Fix 7 alert uses the same Outlook connection)
3 Outlook Get Attachments for Bot	Office 365 Outlook, Dropbox
4 Teams Chat to Bot	Microsoft Teams (listed twice; the same connection can be used for both), Office 365 Users, Content Conversion, Dropbox
5 Teams Send from Bot	Microsoft Teams, Dropbox
6 Outlook Get Sent	Office 365 Outlook, Dropbox

For each connection, either pick an existing connection or choose **+ Create new**, sign in, return to the import page, click **Refresh list**, select the new connection, and click **Save**.

- Office 365 Outlook / Teams / Office 365 Users / Mail: sign in as the **target user**.
- Dropbox: sign in to the Dropbox account that will hold the bot data.

6. When every resource shows a selection, click **Import**. Wait for the confirmation message.
7. Open the imported flow and continue with section 5. Do not turn it on yet.

5. Configure folders and paths

The packages contain mailbox folder IDs and Dropbox folder IDs from ATC's build environment. These must be re-selected in your environment.

1. Open the flow › **Edit**.
2. **Outlook triggers (flows 1 and 6):** open the trigger and set **Folder** to **Inbox** (flow 1) or **Sent Items** (flow 6) using the folder picker. For flow 6, check that *Include Attachments* = Yes.
3. **Dropbox triggers (flows 2, 3, 5):** open *When a file is created* and use the folder picker to select `/OutlookPush/Outbox` (flow 2), `/OutlookPush/Fetch` (flow 3) or `/TeamsPush/Outbox` (flow 5).
4. **Dropbox Create file actions:** check the Folder Path:
   - Flow 1: `/outlookpush/inbox`
   - Flow 3: `/outlookpush/files` and `/outlookpush/fetched`
   - Flow 4: `/teamspush/inbox`
   - Flow 6: `/outlookpush/sent`
5. **If you change any folder names**, also update:
   - Flow 2: the condition **InOutfiles**, which hard-codes `startsWith(toLower(path), '/outlookpush/outfiles/')`.
   - The bot's configuration, so it reads from and writes to the same paths.
6. Click **Save**, then fix any errors the Flow Checker reports.

6. Turn on and test

To turn a flow on, open it and click **Turn on**, or use the toggle in My flows. After each test, open the flow's **28-day run history** and confirm the run Succeeded.

Test procedures

Flow	Test	Expected result
1 Inbox to Bot	Send an email to the user's mailbox from another account	A `<SafeId>.json` file in /OutlookPush/Inbox within a minute or two, with the correct from, subject and text
6 Get Sent	From the user's mailbox, send an email with a small attachment (for example a PNG)	A `<yyMMddHHmmss>_<SafeId>.eml` file in /OutlookPush/Sent that opens in Outlook with the attachment intact
3 Get Attachments	Copy a `messageId` from a file in /OutlookPush/Inbox (for an email with attachments). Upload `test.json` to /OutlookPush/Fetch containing `{"messageId":"<id>"}`	Attachments and a `<SafeId>_parent.eml` copy of the email in /OutlookPush/Files, a manifest in /OutlookPush/Fetched with `"status":"complete"` (first entry `"kind":"parent"`), and the request file removed. Repeat the same request: the run still Succeeds (judge by the run status; Dropbox keeps the old modified time when an identical file is overwritten)
2 Send from Bot (send)	Upload `test.json` to /OutlookPush/Outbox: `{"action":"send","to":"<your test address>","subject":"ATC test","text":"Hello"}`	Email received at the test address; request file deleted; a copy appears in /OutlookPush/Sent if flow 6 is on
2 Send from Bot (reply + attachment)	Upload a small file to /OutlookPush/Outfiles. Upload a request with `"action":"reply"`, a valid `messageId` and `"attachments":[{"Name":"a.pdf","path":"/outlookpush/outfiles/a.pdf"}]`	Reply sent with the attachment; the attachment and request files deleted
2 Path guard	Upload a request with a **fake** `to` address and an attachment path outside /OutlookPush/Outfiles	Nothing is sent; the request moves to /OutlookPush/Outbox/Failed; an "ATC flow blocked" alert arrives; the run shows Failed, and its run history shows the PathOk False branch (BadName chain) ran
4 Teams Chat to Bot	Have a colleague send the user a Teams chat message	A `<messageId>.json` file in /TeamsPush/Inbox containing the text and conversationId
5 Teams Send from Bot (group chat)	Copy a `conversationId` from a /TeamsPush/Inbox file for a group chat. Upload `test.json` to /TeamsPush/Outbox: `{"conversationId":"<id>","text":"ATC test"}`	Message posted as the user in that chat; request file deleted

Flow	Test	Expected result
5 Teams Send from Bot (1:1 chat)	Repeat with the `conversationId` of a one-to-one chat	Message posted in the 1:1 chat (re-confirm in every tenant)

How to upload a request file: save the JSON in a text editor (for example as `test.json`), then upload or drag it into the folder at dropbox.com or in the Dropbox desktop app. Don't create the file in place and type into it, because the trigger fires as soon as the file appears and may read an empty file. The trigger only fires for files in the folder itself, not in subfolders.

7. Run-failure alerts

Power Automate already emails the flow owner about some failures, but those emails can be delayed and are not guaranteed. Flow 2 has a built-in alert (Fix 7, Appendix A.4), sent with Office 365 Outlook *Send an email (V2)* to the flow's own mailbox; it needs the Outlook connection to be healthy, and its copy appears in /OutlookPush/Sent. For other flows, add one where you need it.

Use: the **Mail** connector's **Send an email notification (V3)** (Standard). It is sent by Microsoft, so it doesn't depend on the user's Outlook connection and does not land in Sent Items. **Do not use** the Notifications connector ("Send me an email notification" / "Send me a mobile notification"). Microsoft's connector reference marks the connector and both actions as DEPRECATED: the mobile action because the mobile app was deprecated on 31 August 2026, and the email action with a note to migrate to a supported email action (Microsoft Learn, "Notifications [DEPRECATED]", https://learn.microsoft.com/en-us/connectors/flowpush/, checked 15 Oct 2026). The Mail connector and its Standard class are documented at https://learn.microsoft.com/en-us/connectors/sendmail/. Office 365 Outlook *Send an email (V2)* also works, but it fails when the Outlook connection is broken, and in flow 6 it would create a mail loop (the alert lands in Sent Items and triggers Get Sent again).

7.1 Pattern (any flow)

1. Edit the flow. Put the actions to watch inside a **Scope** named `Try`.
2. After the Scope, add **Send an email notification (V3)** with a subject such as `ATC flow failed: <flow name>`. In the body, include the run link expression:
   `concat('https://make.powerautomate.com/environments/', workflow()?['tags']?['environmentName'], '/flows/', workflow()?['name'], '/runs/', workflow()?['run']?['name'])`.
3. Set its **Run after** to `Try` › **has failed** and **has timed out** only.
4. Add a **Terminate** (Status: Failed) after the alert, so the run still shows as Failed in run history.

7.2 Flows that already terminate as Failed

Flows 1, 2 and 3 use **Terminate (Failed)** inside conditions. A Terminate stops the run immediately, so an alert placed after it never runs. Put the alert **before** the Terminate in that branch, as Fix 7 does for the PathOk branch.

8. Security notes

1. **One user per deployment.** Every connection runs as the account that created it. Import the flows under, and connect them to, the specific user whose mailbox and Teams the bot should serve. Do not use an admin or shared account unless that is the intended mailbox.
2. **Least privilege:**
   - Use a **dedicated Dropbox account or folder tree** for bot data. The Dropbox connection can reach the whole connected account, not just these folders.
   - Grant the bot's own Dropbox access only to /OutlookPush and /TeamsPush where possible.
3. **Outboxes are send-as.** Anyone or anything that can write files to /OutlookPush/Outbox or /TeamsPush/Outbox can send email or Teams messages as the user. Restrict write access to these folders to the bot.
4. **Attachment path guard.** Flow 2 only attaches files under /OutlookPush/Outfiles and rejects `..`. Keep this check if you customise the flow.
5. **Data at rest in Dropbox.** Email content, Sent Items exports (including attachments), attachment files and Teams messages are stored in Dropbox in plain form. Apply your retention, eDiscovery, compliance and classification policies. The flows never delete the copies in /OutlookPush/Inbox, /OutlookPush/Sent, /OutlookPush/Files, /OutlookPush/Fetched, /OutlookPush/Outbox/Failed or /TeamsPush/Inbox, so set up a cleanup routine (bot-side or manual).
6. **Older mail is reachable by ID.** Outlook Get Attachments for Bot (and, after Fix 8, its parent-email export) and the reply action in Outlook Send from Bot work on **any** message still in the mailbox if the request includes its message ID, not only on mail that arrived after setup. Restrict write access to /OutlookPush/Fetch and /OutlookPush/Outbox for this reason.
7. **Secrets.** The packages contain no passwords or tokens. Connections are created in your tenant during import. Never put credentials in request files or flow inputs.
8. **Sharing.** Do not share the flows or connections with other users unless required. Co-owners can view run history, which includes message content.

9. Troubleshooting

Symptom	Checks
Nothing appears in Dropbox	Flow is **On**; open **Run history** (28 days). No runs means a trigger issue (wrong folder selected, connection broken, flow suspended by DLP). A failed run means the failing action shows the error.
Items arrive late	Outlook and Teams triggers are event-driven but can lag by a few minutes. Dropbox triggers poll every minute, and throttling can stretch this. Check for "Delayed" or throttling warnings on the flow page.
Connection errors / "Unauthorized"	Connections can expire (password change, MFA or Conditional Access changes, Dropbox token revoked). Go to **Data › Connections**, choose **Fix connection** or re-authenticate, then resubmit failed runs.

Symptom	Checks
Flow turned off by itself	Flows can be suspended after repeated failures, long inactivity, licence changes or DLP policy changes. Fix the cause and turn the flow back on.
Large emails/attachments fail	Connector limits apply (for example, Microsoft documents a 50 MB maximum file size for the Dropbox connector). Flow 3 skips attachments over 20 MB by design. Very large Sent Items exports in flow 6 may fail. Add the alert in section 7.
Outbox request not processed	Check the JSON is valid and matches the schema in 2.2 or 2.5, and that the file was created directly in the outbox folder (not a subfolder). Flow 2: failed requests are in /OutlookPush/Outbox/Failed and an alert was sent; fix the request and upload it to /OutlookPush/Outbox again. If a request is in Failed but no alert arrived, check the Office 365 Outlook connection (the alert is sent through it). Flow 5: failed requests stay in the outbox.
Duplicate-name failures	Flow 1 tolerates duplicates. Flow 3 replaces existing files after Fix 6. Flows 4 and 6 fail on an existing name (rare, because names are unique per message and, for flow 6, per second). Review the run.
Resubmitting	In Run history, select a failed run and click **Resubmit**. For outbox flows, make sure the request file still exists, so you don't send twice.

10. Updating and versioning

1. ATC packages are versioned by the timestamp in the file name (`<FlowName>_YYYYMMDDHHMMSS.zip`, the UTC date and time the package was exported or built, for example `OutlookGetAttachmentsforBot_20261019141223.zip`). Always deploy the latest, except a package marked superseded in section 1.
2. Before updating, export the current flow as a backup: **My flows › ... › Export › Package (.zip)**.
3. To update in place: **Import Package (Legacy)** with the new zip, and under the Flow resource choose **Update** and select the existing flow. Re-select connections. Then re-check the trigger folder selections and paths (section 5) and any alert steps you added, because an update replaces the definition and removes local customisations. An update to a flow that is on stays on; a package imported with **Create as new** comes in turned off.
4. Or import as **Create as new**, test, then turn off and delete the old flow. Only one version of each flow should be on at a time, to avoid duplicate files or duplicate sends.
5. Record the deployed package name and date for each flow.

11. Uninstall

1. Turn off all six flows (My flows › toggle off).
2. Delete each flow (... › **Delete**).
3. In **Data › Connections**, delete the Office 365 Outlook, Dropbox, Teams, Office 365 Users, Content Conversion and Mail connections if nothing else uses them. Optionally, revoke the Power Automate app in Dropbox (Settings › Connected apps).

4. Remove or archive the /OutlookPush and /TeamsPush folders in Dropbox, according to your retention policy.
5. Disconnect or reconfigure the bot so it stops writing to the outbox folders.

12. Known limits and customization

ATC tailors these flows for each customer, either during assisted setup or with the self-setup checklist below.

12.1 Customization

1. **Teams Chat to Bot, your own messages (by design).** The user's own Teams chat messages, and the bot's posts made through Teams Send from Bot, are captured too, so the bot sees the whole conversation. The bot should recognise its own posts so it doesn't answer itself. To capture only other people's messages, edit the flow's **NotMine** condition: left side (expression) `not(equals(coalesce(first(body('Filter_array'))?['from/user/id'], ''), body('Get_my_profile_(V2)')?['id']))`, **is equal to**, right side `true`. Test it during setup.
2. **Package names.** The customer names are Outlook Inbox to Bot, Outlook Send from Bot, Outlook Get Attachments for Bot, Teams Chat to Bot, Teams Send from Bot and Outlook Get Sent. A "Customer – " prefix can be added at import.
3. **Alert recipient.** Fix 7 sends its alerts with Office 365 Outlook *Send an email (V2)* from the flow's mailbox. In the ATC package the **To** of `AlertFailed` and `AlertBad` is ATC's own build mailbox, so at setup change both to the customer's address (normally the flow's own mailbox, or the IT contact).
4. **Folder names.** These can be changed (section 5). Update the bot's configuration to match.

12.2 Known limits

1. **Teams Chat to Bot checks the latest 20 messages.** If the triggering message is not among the latest 20 in that chat (for example, during a burst of messages), it is not written, and the run still succeeds. Accepted as a known limit.
2. **Outlook Get Sent size.** Verified with 16 MB attachments (a 22.5 MB .eml). Larger messages have not been tested, and the Dropbox connector's file limit is 50 MB. Left as is.
3. **Get Attachments skips attachments over 20 MB.** They are listed in the manifest with an empty `file`.
4. **Get Attachments saves a full email copy whenever an email has any attachment.** Office 365 Outlook does not list attached emails as attachments, so the full copy (`<SafeId>_parent.eml`) is the only way to capture them. It adds one .eml per such email; its manifest entry reports size 0.
5. **Teams 1:1 chats.** Posting works even though the action is set to "Group chat" (verified by ATC). Re-confirm in each tenant (section 6).
6. **No backfill.** Only mail and messages that arrive after a flow is turned on are captured.

12.3 Licensing and setup checklist

1. **Licensing:** every connector used (Office 365 Outlook, Dropbox, Microsoft Teams, Office 365 Users, Content Conversion, and Mail if you add section 7 alerts) is Standard. The customer confirms this against their own licence. ATC checks this during assisted setup, and self-setup customers confirm it at this step.
2. DLP: all of the connectors above are in the same data group (section 3).

3. Dropbox folders created, including /OutlookPush/Outbox/Failed (section 3).
4. All six flows imported, folders re-selected, flows turned on (sections 4–6).
5. Every test in section 6 has passed, **including Teams Send from Bot in a 1:1 chat**.
6. NotMine choice made (12.1), and the alert recipient (To of AlertFailed and AlertBad in flow 2) set (12.1).
7. The deployed package name and date are recorded for each flow (section 10).

Appendix A. Flow fixes (step by step)

These are the exact edits for the fixes listed in sections 2.2, 2.3 and 2.6, in the order they should be made. Each one ends with a test. The 19 October 2026 packages in section 1 already include them, so customers who deploy those packages don't need to make these edits; just set the alert recipient in Fix 7 (section 12.1).

A.0 Before you start (applies to every fix)

1. **Back up the flow first.** In make.powerautomate.com go to **My flows**, click **...** next to the flow, then **Export › Package (.zip)**. Save the zip. If anything goes wrong you can re-import it.
2. **Typing an expression.** Click in the field, then click the **fx** button (or "Insert expression"). Paste the expression exactly as shown, without a leading `@`, and click **Add** or **OK**.
3. **Renaming an action.** Click the action's title bar and type the new name. Use the exact names in these steps, because later expressions refer to them (spaces in a name become `_` in expressions).
4. **Configure run after.** Click the action, open **Settings** (or **...** › **Configure run after** in the classic designer), find **Run after**, choose the action named in the step, and tick exactly the boxes listed. Untick any others.
5. **Adding a Scope.** Click **+** where you want it, search for **Scope** (built-in "Control"), add it, rename it, then drag the named actions into it. If dragging does not work, add a new action of the same type inside the Scope, copy the settings across, and delete the old one.
6. **Save** after each fix, and fix anything the Flow checker reports before testing.
7. Make one fix, test it, then move on. The tests use small request files uploaded into Dropbox.

A.1 Fix 4: Outlook Get Sent, Dropbox-safe file names

What changes: the .eml name becomes a short UTC timestamp (`yyMMddHHmmss`) followed by the same SafeId as flows 1 and 3, so the Outlook message ID stays in the name. In the SafeId, every lowercase letter gets a `~` in front (Dropbox names ignore case), `/` becomes `_` and `=` becomes `-`. New names look like `261015142233_AAM~kAD~d~h…AAA-.eml`. **Checked against real files (19 Oct 2026):** names in /OutlookPush/Sent look like `261019121141_AAM~kAD~d~hNWM…~f~gE2AAA-.eml`: 12 digits, one `_`, the SafeId (which ends in `-` and here also contains a native `-` and, in other files, `_`), then `.eml`.

Matching a file to its message: the name is 12 digits (UTC `yyMMddHHmmss`), then `_`, then the SafeId, then `.eml`. Split on the **first** `_` only, because the SafeId itself can contain `_` (from a `/` in the ID). To decode the SafeId back to the Outlook message ID: remove each `~`, turn every `_` back into `/`, and turn **only a trailing** `-` back into `=`. Outlook IDs can contain `-` natively (for example `…jH3-TL…`), so a `-` anywhere else stays as it is. A bot can also read the `Message-ID:` header inside the .eml, which is the standard internet ID of the email, and match that against the original email or replies.

Path length (checked 15 Oct 2026). Dropbox says a full file path must be fewer than 260 characters (Dropbox Help, "Naming Dropbox files and folders", https://help.dropbox.com/organize/file-names). These figures use ATC's real IDs: 3 Sent Items IDs, 152 characters each and 199 as SafeId, plus 447 Inbox files

whose SafeIds run up to 200 characters. - **New name:** 12 + 1 + 199 + 4 = **216 characters** (217 with the longest SafeId seen). - **In Dropbox:** `/OutlookPush/Sent/<name>` comes to **234 to 235 characters**. - **On a synced Windows PC:** `C:\Users\<user>\Dropbox\OutlookPush\Sent\<name>` (41 + name) comes to **257 to 258 characters**. That's **under 260, but with only 2 to 3 characters to spare**. - **Other PCs:** a longer Windows user name, a "Dropbox (Company)" folder name, or an ID with more lowercase letters can push the path to 260 or more. Dropbox still syncs such files, but some programs may not open them from that PC (according to Dropbox's community support answers). Bots that read through the Dropbox API are not affected. - **Smallest safe fallback:** drop the timestamp and use `concat(outputs('SafeId'),'.eml')`. That gives about 203 characters (244 to 245 local on this PC), and files still sort by Dropbox's modified time. Only use it if a PC with a longer path actually has trouble.

Steps (flow: Outlook Get Sent):

1. Open the flow and click **Edit**.
2. Click **+** between **Export email (V2)** and **Create file**, then add **Data Operation › Select**. Leave its name as `Select`.
   - **From** (expression): `range(0, length(triggerOutputs()?['body/id']))`
   - **Map:** click **Switch Map to text mode** (the small "T" icon next to Map), then enter this expression:

```
if(equals(substring(triggerOutputs()?['body/id'], item(), 1), toUpper(substring(triggerOutputs()?['body/id'], item(), 1))), substring(triggerOutputs()?['body/id'], item(), 1), concat('~', substring(triggerOutputs()?['body/id'], item(), 1)))
```

3. Click **+** below Select and add **Data Operation › Compose**. Rename it `SafeId`. **Inputs** (expression):

```
replace(replace(join(body('Select'), ''), '/', '_'), '=', '-')
```

4. Open **Create file**. Clear **File Name** and enter this expression:

```
concat(formatDateTime(utcNow(),'yyMMddHHmmss'),'_',outputs('SafeId'),'.eml')
```

5. Leave Folder Path (`/outlookpush/sent`) and File Content (Body of Export email) unchanged. Click **Save**.

Test 4:

1. From the mailbox, send a short email with subject `ATC fix 4 test` to any address (yourself is fine).
2. Within about 2 minutes, a new file appears in /OutlookPush/Sent. Its name starts with 12 digits (for example `261015…`, the UTC date and time), then `_`, then the coded ID with `~` characters, and it ends in `-.eml`, not `=.eml`.
3. Download it and open it in Outlook. The message is intact. Optional: open it in a text editor and check that it has a `Message-ID:` line.
4. In the flow's run history, the run shows **Succeeded**.

A.2 Fix 6: Outlook Get Attachments for Bot, safe to run again on the same message

What changes: saving an attachment or manifest that already exists replaces the old file instead of failing. The request file in /OutlookPush/Fetch is always removed, even when the run fails, and a manifest is written even if the email can't be found (status "incomplete"). File names stay the same as today, so nothing changes for the bot.

How the replace works: each save is wrapped in a Scope. *Create file* is tried first. If it fails because the file is already there, *Get file metadata using path* finds the existing file and *Update file* overwrites it. In run history, a red *Create file* inside a green Scope is normal for a rerun.

Steps (flow: Outlook Get Attachments for Bot):

1. Open the flow, click **Edit**, and expand **Apply to each › SmallEnough (True) › HasBytes (True)**. You'll see *Create file*, *OneFile* and *Append to array variable*.
2. At the top of the HasBytes **True** branch, add a **Scope** and rename it `SaveAttachment`. Drag **Create file** into it.
3. Inside SaveAttachment, below Create file, add **Dropbox › Get file metadata using path**. Rename it `AttExisting`.
   - **File Path** (expression): `concat('/outlookpush/files/', outputs('FileName'))`
   - **Run after:** Create file › tick **has failed** only.
4. Below AttExisting, still inside the Scope, add **Dropbox › Update file**. Rename it `AttReplace`.
   - **File** (expression): `body('AttExisting')?['Id']`
   - **File Content** (expression): `base64ToBinary(body('Get_Attachment_(V2)')?['contentBytes'])`
   - Run after: leave the default (AttExisting is successful).
5. Click **OneFile**. **Run after:** SaveAttachment › tick **is successful** only. Check that OneFile now sits after the Scope, not inside it.
6. Scroll down to **Manifest**. **Run after:** Apply to each › tick **is successful**, **has failed**, **is skipped** and **has timed out**.
7. Below Manifest, add a **Scope** and rename it `SaveManifest`. Drag **Create file 1** (the one that writes to `/outlookpush/fetched`) into it.
8. Inside SaveManifest, below Create file 1, add **Dropbox › Get file metadata using path**. Rename it `ManExisting`.
   - **File Path** (expression): `concat('/outlookpush/fetched/', outputs('SafeId'), '.json')`
   - **Run after:** Create file 1 › tick **has failed** only.
9. Below ManExisting, add **Dropbox › Update file**. Rename it `ManReplace`.
   - **File** (expression): `body('ManExisting')?['Id']`
   - **File Content** (expression): `outputs('Manifest')`
10. Click **Delete file** (the one that deletes the request). **Run after:** SaveManifest › tick **is successful**, **has failed**, **is skipped** and **has timed out**. This is the step that always clears /OutlookPush/Fetch.
11. Open **IsComplete**. Replace the left side of the condition with this expression, keep **is equal to**, and set the right side to the expression `true`:

```
and(equals(actions('Manifest')?['outputs']?['status'], 'complete'), equals(actions('SaveManifest')?['status'], 'Succeeded'))
```

12. Click **Save**.

Test 6:

1. In /OutlookPush/Inbox, open a .json file whose `hasAttachments` is `True` and copy its `messageId`.

2. Upload `fix6-a.json` containing `{"messageId":"<paste id>"}` to /OutlookPush/Fetch. Within about 2 minutes you should see the attachments in /OutlookPush/Files, a manifest in /OutlookPush/Fetched with `"status":"complete"`, and fix6-a.json gone. The run shows **Succeeded**.
3. **Rerun:** upload `fix6-b.json` with the **same** messageId. Expected: the run shows **Succeeded** (before the fix it failed), the existing files and the manifest are replaced in place (no duplicates), and fix6-b.json is gone. **Judge this by the run status, not by modified times:** when Dropbox receives an overwrite with identical content it keeps the old modified time and revision, so an unchanged file does not look refreshed even though the replace path ran.
4. **Bad ID:** upload `fix6-c.json` containing `{"messageId":"NOT-A-REAL-ID"}`. Expected: the run shows **Failed** (correct), fix6-c.json is **removed** from /OutlookPush/Fetch, and /OutlookPush/Fetched has `NOT-A-REAL-ID.json` with `"status":"incomplete"` and no files. Delete that test manifest afterwards.

A.3 Fix 8: Outlook Get Attachments for Bot, attached emails (redesigned 19 Oct 2026)

Why it was redesigned: testing on 19 Oct 2026 showed that Office 365 Outlook *Get email (V2)* does **not** list attached Outlook items (attached emails, calendar items) in its `attachments` array at all. Only file attachments appear there. The 15 Oct design (a `NeedParent` condition inside the attachment loop, run when an attachment had no bytes) therefore never ran, and an attached email was dropped without any error. That design is replaced by the one below.

What changes: whenever the email has any attachment (`hasAttachments` is true), the flow exports the **whole original email** once, before the attachment loop, as `<SafeId>_parent.eml` in /OutlookPush/Files. Any attached email or other Outlook item is inside that .eml, so the bot opens it from there. The manifest's `files` list then **starts** with an entry for that copy:

```
{ "name": "original-message.eml", "contentId": "", "isInline": false, "contentType": "message/rfc822", "size": 0, "file": "<SafeId>_parent.eml", "kind": "parent", "note": "Full copy of the original email, saved whenever it has attachments. Attached Outlook items are inside it. size is not reported (0)." }
```

- `size` is always 0 for this entry, because the size is not reported. Use the Dropbox file size if you need it.
- Normal file attachments follow, unchanged, with no `kind` field. Attachments over 20 MB are still listed with an empty `file`.
- A listed attachment that returns no content bytes (for example a cloud link attachment) is still recorded, with `"kind":"embedded"`, `odataType` (if Outlook supplies it), and `file` pointing to the same `<SafeId>_parent.eml`.
- An email with no attachments gets no copy, and its manifest has `"files": []`.
- **Cost:** a full email copy (about the size of the email) is saved for **every** email that has any attachment, not only for those that contain an attached email.

Structure as built (flow: Outlook Get Attachments for Bot):

1. **Initialize ParentEml** (Variables › Initialize variable, Name `ParentEml`, Type String, Value empty), between *Initialize variable 1* and *Get email (V2)*.
2. **HasParent** (Control › Condition), straight after *Get email (V2)* and **outside** the loop. Run after: Get email (V2) › **is successful**. Left side (expression), **is equal to**, right side (expression) `true`:

```
equals(body('Get_email_(V2)')?['hasAttachments'], true)
```

3. In HasParent's **True** branch, in this order:
   - **ExportParent**: Office 365 Outlook › Export email (V2). Message Id (expression): `body('Parse_JSON')?['messageId']`
   - **SaveParent**: a Scope, run after ExportParent › is successful. Inside it:
     - **ParCreate**: Dropbox › Create file. Folder Path `/outlookpush/files`. File Name (expression): `concat(outputs('SafeId'), '_parent.eml')`. File Content (expression): `body('ExportParent')`
     - **ParExisting**: Dropbox › Get file metadata using path. File Path (expression): `concat('/outlookpush/files/', outputs('SafeId'), '_parent.eml')`. Run after: ParCreate › **has failed** only.
     - **ParReplace**: Dropbox › Update file. File (expression): `body('ParExisting')?['Id']`. File Content (expression): `body('ExportParent')`. Run after: ParExisting › is successful.
   - **Set ParentEml**: Variables › Set variable, run after SaveParent › is successful. Name `ParentEml`, Value (expression): `concat(outputs('SafeId'), '_parent.eml')`
   - **ParentEntry**: Data Operation › Compose, run after Set ParentEml › is successful. Inputs (expression):

```
setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(json('{}'), 'name', 'original-message.eml'), 'contentId', ''), 'isInline', false), 'contentType', 'message/rfc822'), 'size', 0), 'file', variables('ParentEml')), 'kind', 'parent'), 'note', 'Full copy of the original email, saved whenever it has attachments. Attached Outlook items are inside it. size is not reported (0).')
```

- **Append ParentEntry**: Variables › Append to array variable, run after ParentEntry › is successful. Name `Saved`, Value (expression): `outputs('ParentEntry')`
- Leave the **False** branch empty (no attachments, no copy). 4. **Apply to each** (the attachment loop): Run after: HasParent › **is successful**, **has failed**, **is skipped** and **has timed out**, so the file attachments are still saved if the copy fails. 5. **NoBytes** (in Apply to each › SmallEnough (True) › HasBytes (False)): there is no NeedParent condition any more, so NoBytes is the first action in that branch with the default run after. Its Inputs (expression) are unchanged:

```
setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(setProperty(json('{}'), 'name', items('Apply_to_each')?['name']), 'contentId', coalesce(items('Apply_to_each')?['contentId'], '')), 'isInline', coalesce(items('Apply_to_each')?['isInline'], false)), 'contentType', coalesce(items('Apply_to_each')?['contentType'], '')), 'size', coalesce(items('Apply_to_each')?['size'], 0)), 'file', variables('ParentEml')), 'kind', 'embedded'), 'odataType', coalesce(items('Apply_to_each')?['@odata.type'], '')), 'note', 'No direct bytes (attached Outlook item or link). The attached item is inside the parent email saved as file.')
```

6. **Manifest**: the status is "complete" only if both the loop and HasParent succeeded, otherwise "incomplete". The status part of its Inputs (expression) is:

```
if(and(equals(actions('Apply_to_each')?['status'], 'Succeeded'), equals(actions('HasParent')?['status'], 'Succeeded')), 'complete', 'incomplete')
```

If the status is "incomplete" (for example, the whole email is over the Dropbox connector's 50 MB limit and the copy fails), **IsComplete** fails the run. The request file is still cleared (Fix 6). Do Fix 6 first, because this fix reuses its replace pattern.

Test 8:

1. In Outlook, create a new email and attach one PDF and one existing email as an item (Outlook desktop: **Attach File › Outlook Item**, or drag an email into the new message; Outlook on the web: drag an email from the list into the message). Send it so that it **lands in the mailbox the flows watch** (for example, to yourself), so that Outlook Inbox to Bot writes its file to /OutlookPush/Inbox. If it only exists in Sent Items, use the message ID of the Sent copy instead (decode it from the file name in /OutlookPush/Sent, see A.1).
2. Copy its `messageId` from the new file in /OutlookPush/Inbox (or use the Sent copy's ID). Upload `fix8-a.json` containing `{"messageId":"<id>"}` to /OutlookPush/Fetch.
3. Expected: the manifest has `"status":"complete"`. Its first `files` entry has `"kind":"parent"` and `"file":"<SafeId>_parent.eml"`, and the PDF entry has a normal `…_n.pdf` file. The .eml exists in /OutlookPush/Files, and opening it shows the attached email inside. The run shows **Succeeded**.
4. Upload the same request again as `fix8-b.json`. The run shows **Succeeded** (inside SaveParent, a red ParCreate followed by green ParExisting and ParReplace is normal), and no second copy is created. Judge this by the run status, not by the modified time (see Test 6).
5. Fetch an email that has **no** attachments. The manifest is `"complete"` with `"files": []`, and no `_parent.eml` is created.

A.4 Fix 7: Outlook Send from Bot, retries, failed-request folder and alert

What changes:

- Send and Reply each retry up to **2 times, 1 minute apart**, for temporary errors (timeouts, throttling, server errors). Permanent errors such as a bad address or message ID are not retried, so a bad message ID fails and alerts straight away. A retry after a timeout can occasionally send a duplicate if the first attempt actually went through. Two retries keep that risk small; it can't be removed completely.
- If a request can't be sent, or is blocked by the attachment path check, the request is **copied to /OutlookPush/Outbox/Failed** (named `<UTC time>_<run id>.json`), **deleted from /OutlookPush/Outbox**, and an **alert email** is sent. The run then ends as **Failed**, so it also shows in run history. The Dropbox trigger ignores subfolders, so a file in Failed is never re-sent automatically.
- **Alert:** Office 365 Outlook **Send an email (V2)**, sent from the flow's own mailbox (the mailbox of the flow's Office 365 Outlook connection) to that same mailbox. It uses the connection the flow already has, so no extra connector is needed. Consequences: the alert **needs the Outlook connection to be healthy** (if that connection is broken, the request is still copied to Failed and the run still shows Failed, but no alert arrives), and a copy of every alert lands in Sent Items, so Outlook Get Sent saves it to /OutlookPush/Sent. That is not a loop, because Get Sent never sends mail. Do not use the Notifications connector ("Send me an email/mobile notification"). Microsoft marks it DEPRECATED (https://learn.microsoft.com/en-us/connectors/flowpush/, checked 15 Oct 2026). The Mail connector's *Send an email notification (V3)* is no longer used here.

Steps (flow: Outlook Send from Bot):

1. In Dropbox, create the folder **/OutlookPush/Outbox/Failed**.
2. Open the flow and click **Edit**. Expand **IsReply**.
3. **Retries:** click **Reply to email (V3)** (True branch) › **Settings** › **Retry policy**. Set **Type** to Fixed interval, **Count** to `2` and **Interval** to `PT1M`. Do the same for **Send an email (V2)** (False branch).

4. **Failure path:** hover over the arrow below **IsReply** (above *Apply to each 1*), click **+ › Add a parallel branch**, then add **Data Operation › Compose**. Rename it `FailedName`.
   - Inputs (expression): `concat(formatDateTime(utcNow(),'yyyyMMdd_HHmmss'), '_', workflow()?['run']?['name'], '.json')`
   - **Run after:** IsReply › tick **has failed**, **is skipped** and **has timed out**. Untick is successful.
5. Below FailedName, add **Dropbox › Create file**. Rename it `SaveFailed`. Folder Path `/outlookpush/outbox/failed`. File Name (expression) `outputs('FailedName')`. File Content (expression) `body('Get_file_content')`.
6. Below SaveFailed, add **Dropbox › Delete file**. Rename it `DeleteFailed`. File (expression): `triggerOutputs()?['headers/x-ms-file-id']`. Leave run after as the default (SaveFailed is successful), so the request is deleted only once its copy is safe.
7. Below DeleteFailed, add **Office 365 Outlook › Send an email (V2)**, using the flow's existing Outlook connection. Rename it `AlertFailed`.
   - **To:** the address that should get alerts, normally the flow's own mailbox (or the customer's IT contact, chosen at setup).
   - **Subject:** `ATC flow failed: Outlook Send from Bot`
   - **Body** (expression):

```
concat('A bot request could not be sent. A copy of the request was saved as
/OutlookPush/Outbox/Failed/', outputs('FailedName'), ' (if it is not there, the request is still in
/OutlookPush/Outbox). Run details: https://make.powerautomate.com/environments/', workflow()?
['tags']?['environmentName'], '/flows/', workflow()?['name'], '/runs/', workflow()?['run']?['name'])
```

- **Run after:** DeleteFailed › tick **is successful**, **has failed** and **is skipped**. 8. Below AlertFailed, add **Control › Terminate**. Rename it `StopFailed`. **Status** Failed. **Run after:** AlertFailed › tick **is successful** and **has failed**. 9. **Blocked path:** expand **PathOk › False**, which contains *Terminate*. Above that Terminate, add the same four-step chain with these names: `BadName` (Compose, same expression as FailedName), `SaveBad` (Create file, File Name `outputs('BadName')`), `DeleteBad` (Delete file) and `AlertBad` (Office 365 Outlook Send an email (V2), same To, subject `ATC flow blocked: attachment path outside Outfiles`, with the body expression using `outputs('BadName')`). Set AlertBad's run after to DeleteBad › **is successful**, **has failed** and **is skipped**. Set the existing **Terminate**'s run after to AlertBad › **is successful** and **has failed**. Shortcut: on each action of the first chain, use **...** › **Copy action**, then **+** › **Paste an action** in the False branch, and rename the copies. 10. Click **Save**.

Test 7:

1. **Normal send still works:** run the section 6 Send from Bot test. The email arrives, the request is deleted, and no alert is sent.
2. **Send failure:** upload `fix7-a.json` containing `{"action":"reply","messageId":"BAD-ID","text":"ATC fix 7 test, should fail"}` to /OutlookPush/Outbox. Expected: nothing is sent, fix7-a.json leaves the Outbox, a copy appears in /OutlookPush/Outbox/Failed, an alert email arrives (check Junk the first time and mark it Not junk), and the run shows **Failed**.
3. **Blocked path:** upload `fix7-b.json` containing `{"action":"send","to":"<a fake address>","subject":"ATC fix 7 path test","text":"x","attachments":[{"Name":"x.txt","path":"/OutlookPush/x.txt"}]}`. Use a **fake** `to` address (one that cannot reach a real person), so that nothing real goes out if the path check ever let it through. Expected: nothing is sent, the request moves to Failed, an "ATC flow blocked: attachment

path outside Outfiles" alert arrives, and the run shows **Failed**. The copy in Failed looks the same whichever chain made it, so **check run history** to confirm that the PathOk False branch ran (BadName, SaveBad, DeleteBad, AlertBad, Terminate) and not FailedName.

4. **Retry settings:** on Reply to email (V3) and Send an email (V2), open **...** › **Peek code**. Each shows `"retryPolicy": {"type": "fixed", "count": 2, "interval": "PT1M"}`. A real retry can't easily be forced; this check confirms the setting.
5. Delete the test files from /OutlookPush/Outbox/Failed.

Prepared from ATC Automated Flows User Guide v1.4.0 and Technical Setup & Deployment Guide v1.4.2 (19 October 2026); section 12.4 of the technical guide covers an ATC-internal tool and is not included. Microsoft 365, Outlook, Teams and Power Automate are trademarks of the Microsoft group of companies. Dropbox is a trademark of Dropbox, Inc. © 2026 ATC Eutectics LLC · atceutectics.com