

DEXPRO AWF External Approvals — Power Automate Developer Guide

“ **Audience:** DEXPRO developers / maintainers of the Power Automate solution package.

Customers / admins: see [SETUP-GUIDE.md](#). The customer flow is "import the solution package, configure connections and environment variables" — they do not need to read this document.

This guide describes how the `DEXPROApprovalFlow` Power Platform solution package is constructed: the structure of the cloud flow, the two adaptive cards, and the steps to rebuild / re-export the package after changes.

Solution package overview

The shipped solution contains **one cloud flow** and **two connection references**:

Component	Type	Notes
<code>DEXPRO AWF Externe Genehmigung</code>	Cloud flow (modern flow, type 1, category 5)	The entire end-to-end automation
<code>dxp_sharedsharepointonline_*</code>	Connection reference	Used by the SharePoint trigger and every list / item action
<code>dxp_sharedteams_*</code>	Connection reference	Used by <i>Post adaptive card and wait</i> and <i>Post message in a chat or channel</i>

“ **One Teams connection reference**, not two. Earlier exports had a duplicate because the *Post message* actions had been authored against a different Teams connection from the *Post adaptive card and wait* actions. Always reuse the same connection across all Teams actions before exporting.

The trigger is *When an item is created* on the SharePoint approval list. Site Address and List Name are configured as **environment variables** (`dxp_SharepointSiteAddress` and `dxp_SharepointListName`), which the import wizard prompts for during installation.

Flow structure

Trigger: When an item is created (SharePoint)

|

▼

Compose_Actions (build Card A buttons)

|

▼

Post adaptive card and wait for a response ← Card A (full body, actions from Compose_Actions)

|

▼

Switch on body(...)?['data']?['action']

|

└─ case "claim" → Get item → Condition (already claimed?)

|

└─ Yes → Teams message + Terminate

|

└─ No → Update item (Claimed + ClaimedBy)

|

→ Get peer items

|

→ Apply to each (Update item: Cancel peer)

|

→ Compose_Actions2 (build Card B buttons)

|

→ Post adaptive card and wait ← Card B

|

→ Update item (final decision)

|

→ Post message (Teams) ← Result card (□/□)

|

└─ case "approve" → Update item (Approved + ResponseComment + ResponseDateTime)

|

→ Post message (Teams) ← Result card (□)

|

└─ case "reject" → Update item (Rejected + ResponseComment + ResponseDateTime)

|

→ Post message (Teams) ← Result card (□)

|

▼

Handle_error (Scope, run after Switch: Failed | TimedOut)

└─ Post message (Teams) – "already handled" template

The Switch is what makes the flow safe for the three possible card actions. The `claim` case must:

1. Re-check the SP item's current `ClaimedBy` (Get item) before proceeding, to catch peer claimers who beat us in the BC poll-interval window.
 2. Pre-cancel peer items (Get peer items + Apply to each) so other members see "already claimed by ..." immediately, without waiting for BC's polling cycle.
 3. Post a second card (Card B) to the claimer for the actual Approve/Reject decision.
-

Peer-card handling when one member decides (v2 — info card)

“ **Status: Shipped in solution v1.0.0.7.** When one group member claims/decides, every other member's still-open card is followed by a neutral **"no longer open" info card**, and a click on the now-stale card is rejected by BC's `GuardActionable`. This is the **non-premium** solution. It does **not** delete/replace the original card in place — see the finding below for why that needs premium.

Finding: in-place closing of peer cards is not possible on Standard licensing

We investigated truly *replacing* each peer's open card (so the buttons vanish). It can't be done reliably without premium, for a chain of connector constraints:

1. The peer cards are sent with **Post adaptive card and wait for a response**, which **does not expose the card's message ID while it waits** (and none at all on timeout). ([Microsoft Q&A](#))
2. Each peer card runs in its **own flow instance** (the SharePoint trigger uses `splitOn`), so the claimer's run can't reach into a peer's run to update its card anyway.
3. The obvious redesign — *Post card in a chat or channel* (returns a message ID) + a separate *When someone responds to an adaptive card* trigger — is **documented by Microsoft as not combinable**: that trigger "cannot be combined with Post adaptive card in a chat or channel ... use Post adaptive card and wait for a response instead." In practice the response trigger never fires for those cards. ([Teams connector reference](#) · [community report](#))

So in-place closing would require a **premium** path (a custom bot / HTTP action with Graph, or a Power Apps component). That is tracked as a future option; it is **not** in this package.

What v2 (info card) actually does

Two existing fire-and-forget messages become neutral **info cards** ([Close Card JSON](#)) — same width and look as the result card, grey `emphasis` header, no buttons:

Where	Trigger	Card body text
<code>claim</code> case → Condition Yes (a peer already claimed)	the second member clicks anything	<code>LblAlreadyClaimedByMsg</code> (with <code>{claimer}</code> from <code>Get item</code>)
<code>Handle_error</code> scope (stale card after BC cancelled the peer)	the peer clicks a button on an already-cancelled entry	<code>LblAlreadyHandledMsg</code>

Both use `PostCardToConversation` (Standard, *Post as Flow bot / Chat with Flow bot*, **recipient** = the approver email) — fire-and-forget, no response needed, so none of the trigger-combination limits apply. The card reads its localized strings (`LblClosedTitle`, `LblAlready...Msg`) from the SharePoint trigger body, so it renders in the approver's language.

“ **Why the BC plumbing is still in place.** The BC side (field `Teams Message ID`, bound action `setTeamsMessageId`, SP column `TeamsMessageId`) was built before this finding and is **kept**: it is harmless, and it is exactly what a future premium in-place-close path would need. The flow simply doesn't call `setTeamsMessageId` today, so `TeamsMessageId` stays empty.

Step-by-step build (rebuild from scratch)

This is the procedure for re-creating the flow from an empty Power Automate environment. Use it when the source flow is lost, when you need to upgrade to a newer template, or when on-boarding a new developer.

Build pre-requisites

- An environment with the **SharePoint** and **Microsoft Teams** standard connectors authorised under your DEXPRO account.

- A SharePoint list to bind to. Easiest: run the BC **Setup External Approvals** wizard once in a sandbox; it creates `DXP AWF Approvals` for you.
- The Compose expressions and Card A / Card B JSON in [Reference: Compose expressions & Card JSONs](#) below.

1. Create the flow

1. <<https://make.powerautomate.com>> → **Create** → **Automated cloud flow**.
2. **Flow name:** `DEXPRO AWF Externe Genehmigung`.
3. Trigger: search for *When an item is created* under **SharePoint**. Click **Create**.
4. Before binding the trigger, add the two **environment variable parameters** the flow uses for the SharePoint coordinates:
 - In the flow designer toolbar, click **Parameters** → **New parameter**.
 - Add parameter **1**:
 - **Name:** `Sharepoint Site Address (dxd_SharepointSiteAddress)`
 - **Type:** `String`
 - **Default value:** `https://dexprosolutions.sharepoint.com/sites/MSBCDev` (DEXPRO sandbox — customers update this during import)
 - Add parameter **2**:
 - **Name:** `Sharepoint List Name (dxd_SharepointListName)`
 - **Type:** `String`
 - **Default value:** `DXP AWF Approvals` (must match the list name BC creates — see `DefaultListNameLbl` in AWFSPIntegration; the SETUP-GUIDE uses the same name)
5. Back in the **When an item is created** trigger, set:
 - **Site Address** → switch to the Expression tab → `parameters('Sharepoint Site Address (dxd_SharepointSiteAddress)')`
 - **List Name** → switch to the Expression tab → `parameters('Sharepoint List Name (dxd_SharepointListName)')`

All subsequent SharePoint actions in the flow must reference the same parameters in the same way.

2. Send the first adaptive card (Card A)

1. + **New step** → *Compose* (Data Operation). Rename it to **Compose_Actions**. Paste the [Compose_Actions expression](#) into its **Inputs**. This builds Card A's action buttons dynamically (claim vs. approve/reject, plus the attachment/Open-in-BC links) so the card body does not need per-action `isVisible` bindings — the Teams renderer ignores those on actions, which is why the buttons are assembled here instead.
2. + **New step** → *Post adaptive card and wait for a response* (Microsoft Teams).
3. Configure:

Setting	Value
---------	-------

Post as	Flow bot
Post in	Chat with Flow bot
Recipient	trigger ApproverEmail
Adaptive Card	Paste the Card A JSON below — its <code>actions</code> reference <code>@{outputs('Compose_Actions')}</code>
Update message (Expression tab)	<code>@{triggerOutputs()?['body/LblResponseSent']}</code>

“ This action **blocks** until the approver responds (default 30 days). The **Update message** is the plain text Teams collapses the card into once the approver clicks a button. Keep it **short and status-neutral** — `LblResponseSent` ("Response sent.") — for two reasons: (1) Power Automate rejects any expression here that references this action's *own* `body(...)` (`InvalidTemplate: the action cannot reference itself`), so it cannot carry the `□ / □` decision; (2) the decision and record details are shown in full on the [result card](#) posted right below, so a long collapse line is just wasted space. A `□` glyph here would also be misleading on a reject.

3. Branch on the action value

1. + **New step** → **Switch** (Control).
2. **On** (expression): `body('Post_adaptive_card_and_wait_for_a_response')?['data']?['action']`
3. Add three cases. The **Equals** field takes a plain lowercase literal — no quotes, no `@{}`:

Case	Equals
Claim	<code>claim</code>
Approve	<code>approve</code>
Reject	<code>reject</code>

3a. `claim` case

The decision tree is: re-check current `ClaimedBy` → either tell the approver someone else got there first, or write the claim, cancel peers, and post Card B.

1. **Get item** (SharePoint) — read the current row.

Setting	Value
Site Address	same as trigger
List Name	same as trigger

Setting	Value
Id	trigger ID

1. **Condition** — has someone else already claimed?

Both values **must** be entered via the **Expression** tab; otherwise PA stores the literal strings and the comparison fails.

Field	Tab	Value
Left value	Expression	<code>not(empty(body('Get_item')?['ClaimedBy']))</code>
Operator	—	is equal to
Right value	Expression	<code>true</code>

“ **Why not** `ClaimedBy is not equal to ''` ? SharePoint returns `null` for an unset text column. `null != ""` evaluates to `true`, which would route every first-ever claim into the "already claimed" branch. `empty()` correctly treats `null` and `""` the same.

- **Yes** branch:

1. **Post message in a chat or channel** (Teams)

- **Recipient:** trigger ApproverEmail
- **Message** (Expression tab): ``
`replace(replace(triggerOutputs()?['body/LblAlreadyClaimedByMsg'], '{title}',
triggerOutputs()?['body/Title']), '{claimer}', body('Get_item')?['ClaimedBy'])`
``

2. **Terminate** (Control), Status = `Succeeded`. Stops the flow cleanly without tripping the stale-card scope.

- **No** branch — continue with steps 3 to 7 below.

Setting	Value
Id	trigger ID
Title	trigger Title
Approval Status	Claimed
ClaimedBy	Expression tab: <code>triggerOutputs()?['body/ApproverEmail']</code>

“ ⚠ The `ClaimedBy` field **must** be entered via the Expression tab. If you type the value as text, PA stores the literal string and SharePoint writes that literal into the column. Every subsequent claim attempt would then read it as "already

claimed" and the message would say "claimed by `triggerOutputs()?...`".

1. **Get items** (SharePoint) — find peer SP items in the same group.

Setting	Value
Site Address	same as trigger
List Name	same as trigger
Filter Query	<code>BCCompanyName eq '@{triggerOutputs()?['body/BCCompanyName']}' and BCInstanceID eq '@{triggerOutputs()?['body/BCInstanceID']}' and BCApprovalEntryNo eq '@{triggerOutputs()?['body/BCApprovalEntryNo']}' and GroupCode eq '@{triggerOutputs()?['body/GroupCode']}' and Id ne @{triggerOutputs()?['body/ID']}' and Processed ne 1</code>

The five `eq` clauses scope the result to peer items belonging to **the same BC standard Approval Entry** in the same workflow instance, group, and BC company — excluding the current item. The trailing `Processed ne 1` skips items the BC poller has already finalised. Including `BCApprovalEntryNo` matters for the (rare) case of two parallel stages of the same workflow assigned to the same group.

Rename this action to `Get peer items`.

1. **Apply to each** over `value` from `Get peer items`. Inside the loop, **a single Update item** action:

Setting	Value
Id	Expression: <code>items('Apply_to_each')?['ID']</code>
Title	Expression: <code>items('Apply_to_each')?['Title']</code>
Approval Status	<code>Cancelled</code>
ClaimedBy	Expression: <code>triggerOutputs()?['body/ApproverEmail']</code>
Processed	<code>Yes</code>

“ **One** loop, not three. Power Automate's designer sometimes auto-nests `Apply to each` when you reference an item from one foreach inside another action — verify after saving that the JSON has only one `Foreach` action here. A nested triple loop runs the Update item N^3 times for N peers.

1. **Compose** — add a *Compose* (Data Operation) action and rename it to **Compose_Actions2**. Paste the [Compose_Actions2 expression](#) into its **Inputs**. Card B always shows Approve/Reject (the claimer has already taken the entry), so this variant

omits the claim branch.

2. Post adaptive card and wait for a response — send Card B to the claimer.

Setting	Value
Post as	Flow bot
Post in	Chat with Flow bot
Recipient	Expression: outputs('Update_item_record_the_claim_on_the_SP_item')?['body/ApproverEmail']
Adaptive Card	Paste the Card B JSON — its actions reference @{outputs('Compose_Actions2')}
Update message (Expression tab)	@{triggerOutputs()?['body/LblResponseSent']}

“ The **Update message** is the text Teams replaces Card B with once the claimer submits Approve/Reject (the card collapses). Same as Card A: keep it short and status-neutral (LblResponseSent) — it cannot reference this action's own body(...) (PA InvalidTemplate self-reference error), and the outcome is shown on the [result card](#) posted right below. (This supersedes the earlier LblClaimedSuccessMsg text; that label is still written to the SP item for backward compatibility but is no longer the Card B update message.) The visible result card follows from [step 9](#).

1. Update item — stamp the final decision.

Setting	Value
Id	Expression: outputs('Update_item_record_the_claim_on_the_SP_item')?['body/ID']
Title	Expression: outputs('Update_item_record_the_claim_on_the_SP_item')?['body/Title']
Approval Status	Expression: if(equals(body('Post_adaptive_card_and_wait_for_a_response_2')?['data']?['action'], 'approve'), 'Approved', 'Rejected')
ResponseComment	@{body('Post_adaptive_card_and_wait_for_a_response_2')?['data']?['comment']}
ResponseDateTime	@{utcNow()}

Do **not** add a `ClaimedBy` value here. Leaving it unset preserves the value written in step 3. An empty expression would clear it.

“ **Note (existing package behaviour):** In the exported flow the Update item in this step sits inside a `For_each` loop over the peer items returned in step 4 (`value` from `Get peer items`). The loop iterates over peers but writes the same claimed-item ID and data on every iteration — so the Update item runs N times for N peers but always touches the claimed item, not the peers. The result is functionally identical to a single Update item outside the loop. This quirk is documented here so that a sanity check doesn't mistake it for a nested-loop bug.

1. **Post the result card** — add a *Post message in a chat or channel* action as described in [step 3d](#), reading the decision from the **Card B** response (`body('Post_adaptive_card_and_wait_for_a_response_2')['data']['action']`). Place it after step 8, outside the peer loop if the designer didn't nest it.

3b. `approve` case

1. **Update item:**

Setting	Value
Id	trigger <code>ID</code>
Title	trigger <code>Title</code>
Approval Status	<code>Approved</code>
ResponseComment	<code>@{body('Post_adaptive_card_and_wait_for_a_response')['data']['comment']}</code>
ResponseDateTime	<code>@{utcNow()}</code>

1. **Post the result card** — add a *Post message in a chat or channel* action as described in [step 3d](#), passing `approve` as the decision.

3c. `reject` case

1. Same Update item as `approve`, just `Approval Status = Rejected`.

“ Don't omit `ResponseComment` / `ResponseDateTime` on Reject — the approver's reason and timestamp belong in the BC audit trail.

1. **Post the result card** — same *Post message* action as [step 3d](#), passing `reject` as the decision.

3d. Result card after every decision

This is the visible "lite" card the approver sees after responding. The collapsed *wait* card shows only a short neutral "Response sent." line; **this** card carries the actual outcome — a clear coloured header (`</>`) and the record details, using the **full chat width**.

Add a **Post message in a chat or channel** (Teams) action — *Post as Flow bot, Post in Chat with Flow bot*, **Recipient** = trigger `ApproverEmail`. Switch the **Message** to the rich-card editor (the `</>` Code view / adaptive-card editor) and paste the [Result Card JSON](#).

The card needs to know which decision was made. The three call sites differ only in **how they read the action value**, so bind the card's decision fact and header to one of:

Call site	Decision expression to substitute for <code>@{DECISION}</code> in the Result Card JSON
<code>approve</code> case (3b)	<code>body('Post_adaptive_card_and_wait_for_a_response')['data']['action']</code>
<code>reject</code> case (3c)	<code>body('Post_adaptive_card_and_wait_for_a_response')['data']['action']</code>
<code>claim</code> case (3a, after Card B)	<code>body('Post_adaptive_card_and_wait_for_a_response_2')['data']['action']</code>

“ Practically, the `approve` and `reject` cases share the exact same Result Card (same `_2`-less expression). The `claim` case uses an otherwise identical card that reads the action from the **Card B** response (`..._2`). Keep them as two copies of the same JSON differing only in that one expression — see the note in the [Result Card JSON](#) section.

4. Handle stale cards from cancelled entries

When BC cancels a peer's entry (after another member's claim), the peer's still-rendered Teams card is now stale. If they click anything, the Switch's Update item fails with `404 itemNotFound`. Catch it with a Scope:

1. Below the Switch, + **New step** → **Scope** (Control). Rename it `Handle_error`.
2. ... on the scope → **Configure run after** → uncheck `is successful`, check `has failed` and `has timed out`. The dialog targets the Switch — that's what we want.
3. Inside, **Add an action** → *Post message in a chat or channel* (Teams):
 - **Recipient:** trigger `ApproverEmail`

- **Message** (Expression tab): `replace(triggerOutputs()?['body/LblAlreadyHandledMsg'], '{title}', triggerOutputs()?['body/Title'])`

5. Save the flow

Click **Save** in the top right. Verify it appears under **My flows** with status *On*.

Reference: Compose expressions & Card JSONs

Both card JSONs target Adaptive Cards **1.5** — the version Microsoft Teams' Power Automate connector renderer supports. Bumping to 1.6 (e.g. to use the `Icon` element and the `ApprovalsApp` catalog icon) makes Teams refuse the card with *"We're sorry, this card couldn't be displayed"*.

Each card's `actions` array is **not** inlined in the card JSON anymore — it is produced by a `Compose` action that runs immediately before the *Post adaptive card* step, and the card references it via `"actions": @outputs('Compose_Actions')` (Card A) / `@outputs('Compose_Actions2')` (Card B). This replaces the old per-action `isVisible` bindings, which the Teams card renderer does not honour on actions. Each `if(...)` branch returns either a one/two-element action array or `json('[]')`, and `union(...)` concatenates them — so a button is present only when its condition holds.

Compose_Actions (Card A)

Builds Card A's buttons: the **Claim** button only for an unclaimed group approval; **Approve/Reject** for an individual approval or an already-claimed group entry; plus **Attachments** / **Open in BC** links when those URLs are present.

```
union(
  if(and(equals(triggerOutputs()?['body/IsGroupApproval'], true),
    empty(triggerOutputs()?['body/ClaimedBy'])),
    createArray(json(concat('{"type":"Action.Submit","title":"","data":{"action":"claim"}}')),
    json('[]')),
  if(or(not(equals(triggerOutputs()?['body/IsGroupApproval'], true)),
    not(empty(triggerOutputs()?['body/ClaimedBy']))),
    createArray(
      json(concat('{"type":"Action.Submit","title":"","data":{"action":"approve"}}')),
      json(concat('{"type":"Action.Submit","title":"","data":{"action":"reject"}}')),
      json(concat('{"type":"Action.Submit","title":"","data":{"action":"openInBC"}}')),
      json(concat('{"type":"Action.Submit","title":"","data":{"action":"attachments"}}'))
    )
  )
)
```

```

'', "style": "positive", "data": {"action": "approve"}}')),
    json(concat('{"type": "Action.Submit", "title": "", triggerOutputs()?["body/LblReject"],
'', "style": "destructive", "data": {"action": "reject"}}')),
    json('[]')),
    if(not(empty(triggerOutputs()?["body/AttachmentsUrl"])),
        createArray(json(concat('{"type": "Action.OpenUrl", "title": "",
triggerOutputs()?["body/LblAttachments"], "", "url": "",
triggerOutputs()?["body/AttachmentsUrl"], ""}'))),
        json('[]')),
    if(not(empty(triggerOutputs()?["body/BCDocumentUrl"])),
        createArray(json(concat('{"type": "Action.OpenUrl", "title": "",
triggerOutputs()?["body/LblOpenInBC"], "", "url": "", triggerOutputs()?["body/BCDocumentUrl"],
""}'))),
        json('[]'))
)

```

Compose_Actions2 (Card B)

Card B is posted only after a claim, so it always offers **Approve/Reject** (no claim branch) plus the **Attachments / Open in BC** links when present.

```

union(
    createArray(
        json(concat('{"type": "Action.Submit", "title": "", triggerOutputs()?["body/LblApprove"],
'', "tooltip": "", triggerOutputs()?["body/LblApprove"],
'', "style": "positive", "data": {"action": "approve"}}')),
        json(concat('{"type": "Action.Submit", "title": "", triggerOutputs()?["body/LblReject"],
'', "tooltip": "", triggerOutputs()?["body/LblReject"],
'', "style": "destructive", "data": {"action": "reject"}}'))
    ),
    if(not(empty(triggerOutputs()?["body/AttachmentsUrl"])),
        createArray(json(concat('{"type": "Action.OpenUrl", "title": "",
triggerOutputs()?["body/LblAttachments"], "", "tooltip": "",
triggerOutputs()?["body/LblAttachments"], "", "url": "",
triggerOutputs()?["body/AttachmentsUrl"], ""}'))),
        json('[]')),
    if(not(empty(triggerOutputs()?["body/BCDocumentUrl"])),
        createArray(json(concat('{"type": "Action.OpenUrl", "title": "",
triggerOutputs()?["body/LblOpenInBC"], "", "tooltip": "", triggerOutputs()?["body/LblOpenInBC"],
'', "url": "", triggerOutputs()?["body/BCDocumentUrl"], ""}'))),

```

```
json('[]'))
)
```

“ **Note:** Card B references `@{outputs('Compose_Actions2')}`. If your exported flow instead points Card B at `Compose_Actions`, update either the card or this guide so the two agree — Card B must use the claim-less variant.

Card A JSON

Sent first to every approver. The `actions` come from **Compose_Actions**, which covers the three group-approval scenarios (individual, single-member group auto-claimed, multi-member group claim flow). The body still uses `isVisible` on its **containers** (record details, comment box) — that binding *is* honoured on body elements.

“ The FactSet's "Description" row shows `coalesce(Description, Title)`. BC writes `Description` as the localized RecordId text (`Format(RecId, 0, 1)` — translated table/field captions), so this renders the record identity for any table, in the approver's language.

```
{
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.5",
  "body": [
    {
      "type": "TextBlock",
      "text": "@{triggerOutputs()?['body/LblTitle']}",
      "style": "heading",
      "weight": "Bolder",
      "size": "Large",
      "wrap": true
    },
    {
      "type": "TextBlock",
      "text": "@{triggerOutputs()?['body/StageName']}",
      "spacing": "None",
      "isSubtle": true,
    }
  ]
}
```

```
        "wrap": true
    },
    {
        "type": "FactSet",
        "spacing": "Medium",
        "facts": [
            { "title": "@{triggerOutputs()?['body/LblDescription']}", "value":
"@{coalesce(triggerOutputs()?['body/Description'], triggerOutputs()?['body/Title'])}" },
            { "title": "@{triggerOutputs()?['body/LblAmount']}", "value":
"@{triggerOutputs()?['body/Amount']}" },
            { "title": "@{triggerOutputs()?['body/LblDueDate']}", "value":
"@{triggerOutputs()?['body/DueDate']}" },
            { "title": "@{triggerOutputs()?['body/LblSender']}", "value":
"@{triggerOutputs()?['body/SenderName']}" }
        ]
    },
    {
        "type": "Container",
        "spacing": "Medium",
        "isVisible": "@{not(empty(triggerOutputs()?['body/RecordDetails']))}",
        "items": [
            { "type": "TextBlock", "text": "@{triggerOutputs()?['body/LblDetails']}",
"weight": "Bolder", "wrap": true },
            { "type": "TextBlock", "text": "@{triggerOutputs()?['body/RecordDetails']}",
"wrap": true, "spacing": "Small" }
        ]
    },
    {
        "type": "Container",
        "spacing": "Medium",
        "isVisible": "@{or(not(equals(triggerOutputs()?['body/IsGroupApproval'], true)),
not(empty(triggerOutputs()?['body/ClaimedBy'])))}",
        "items": [
            {
                "type": "Input.Text",
                "id": "comment",
                "label": "@{triggerOutputs()?['body/LblComment']}",
                "placeholder": "@{triggerOutputs()?['body/LblComment']}...",
                "isMultiline": true,
```

```

        "maxLength": 250
      }
    ]
  }
},
"actions": @{{outputs('Compose_Actions')}}
}

```

Card B (after claim) JSON

Sent only by the `claim` branch after Card A's Claim button is pressed. Same body content as Card A, except the comment box is always visible (no `isVisible` on its container). The `actions` come from **Compose_Actions2** (always Approve/Reject, no Claim button).

```

{
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.5",
  "body": [
    {
      "type": "TextBlock",
      "text": "@{{triggerOutputs()?['body/LblTitle']}}",
      "style": "heading",
      "weight": "Bolder",
      "size": "Large",
      "wrap": true
    },
    {
      "type": "TextBlock",
      "text": "@{{triggerOutputs()?['body/StageName']}}",
      "spacing": "None",
      "isSubtle": true,
      "wrap": true
    },
    {
      "type": "FactSet",
      "spacing": "Medium",
      "facts": [
        { "title": "@{{triggerOutputs()?['body/LblDescription']}}", "value":

```

```

"@{coalesce(triggerOutputs()?['body/Description'], triggerOutputs()?['body/Title'])}" },
    { "title": "@{triggerOutputs()?['body/LblAmount']}", "value":
"@{triggerOutputs()?['body/Amount']}" },
    { "title": "@{triggerOutputs()?['body/LblDueDate']}", "value":
"@{triggerOutputs()?['body/DueDate']}" },
    { "title": "@{triggerOutputs()?['body/LblSender']}", "value":
"@{triggerOutputs()?['body/SenderName']}" }
    ]
},
{
    "type": "Container",
    "spacing": "Medium",
    "isVisible": "@{not(empty(triggerOutputs()?['body/RecordDetails']))}",
    "items": [
        { "type": "TextBlock", "text": "@{triggerOutputs()?['body/LblDetails']}",
"weight": "Bolder", "wrap": true },
        { "type": "TextBlock", "text": "@{triggerOutputs()?['body/RecordDetails']}",
"wrap": true, "spacing": "Small" }
    ]
},
{
    "type": "Container",
    "spacing": "Medium",
    "items": [
        {
            "type": "Input.Text",
            "id": "comment",
            "label": "@{triggerOutputs()?['body/LblComment']}",
            "placeholder": "@{triggerOutputs()?['body/LblComment']}...",
            "isMultiline": true,
            "maxLength": 250
        }
    ]
}
],
"actions": @{outputs('Compose_Actions2')}
}

```

Result Card JSON

The compact "lite" card posted **after** a decision (steps 3b / 3c / 3a-step-9) via a *Post message in a chat or channel* action. It is **action-less** — it confirms the outcome, it doesn't ask for input. The header is a coloured `Container` whose `style` (`good` = green for approved, `attention` = red for rejected) and glyph + word are chosen from the decision value, so the outcome is unmistakable at a glance and the card uses the full chat width.

Replace `@{DECISION}` below with the decision expression for the call site (see the table in [step 3d](#)) — `body('Post_adaptive_card_and_wait_for_a_response')['data']['action']` for the approve/reject cases, or `body('Post_adaptive_card_and_wait_for_a_response_2')['data']['action']` for the claim case. Everything else is identical between the two copies.

```
{
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.5",
  "body": [
    {
      "type": "Container",
      "style": "@{if(equals(@{DECISION}, 'approve'), 'good', 'attention')}",
      "bleed": true,
      "items": [
        {
          "type": "TextBlock",
          "text": "@{if(equals(@{DECISION}, 'approve'), concat('✅ ',
triggerOutputs()['body/LblApprovedResult']), concat('❌ ',
triggerOutputs()['body/LblRejectedResult']))}",
          "weight": "Bolder",
          "size": "Large",
          "wrap": true
        }
      ]
    },
    {
      "type": "TextBlock",
      "text": "@{triggerOutputs()['body/LblResultTitle']}",
      "spacing": "Small",
      "isSubtle": true,
      "wrap": true
    },
    {
      "type": "TextBlock",
```

```

        "text": "@{coalesce(triggerOutputs()?['body/Description'],
triggerOutputs()?['body/Title'])}",
        "weight": "Bolder",
        "size": "Medium",
        "wrap": true
    },
    {
        "type": "FactSet",
        "spacing": "Medium",
        "facts": [
            { "title": "@{triggerOutputs()?['body/LblAmount']}", "value":
"@{triggerOutputs()?['body/Amount']}" },
            { "title": "@{triggerOutputs()?['body/LblSender']}", "value":
"@{triggerOutputs()?['body/SenderName']}" },
            { "title": "@{triggerOutputs()?['body/LblStage']}", "value":
"@{triggerOutputs()?['body/StageName']}" }
        ]
    }
]
}

```

“ **Why `style` and not an `Icon`?** A coloured `Container` (`good` / `attention`) plus the 🟢 / 🟡 emoji in the heading gives an immediate red/green cue and stays within Adaptive Cards **1.5** — the version the Teams Power Automate renderer accepts. The `Icon` element needs 1.6, which Teams still rejects (see the version note at the top of this section).

“ **Why no Open-in-BC / Attachments buttons on the result card?** It's a confirmation, posted after the decision — the work is done. Keeping it action-less avoids a dead "respond again" affordance. If you want the source-record link to remain reachable, add an `Action.OpenUrl` driven by `BCDocumentUrl` the same way Card A does.

Close Card JSON

Used by **v2 (info card)** ([Peer-card handling](#)) — posted via *Post card in a chat or channel* (`PostCardToConversation`) as a neutral, action-less follow-up after a peer interacts with a now-stale

card. It does **not** replace the original card (that needs premium — see the finding). No red/green, because nothing was decided *by this peer*. Body uses `{title}` substitution exactly like the existing already-claimed message.

For the **claim** case use `LblAlreadyClaimedByMsg` (with `{claimer}`); for the **reassign/cancel** (`Handle_error`) case use `LblAlreadyHandledMsg`. Substitute with nested `replace(...)` the same way the v1 messages do.

```
{
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
  "type": "AdaptiveCard",
  "version": "1.5",
  "body": [
    {
      "type": "Container",
      "style": "emphasis",
      "bleed": true,
      "items": [
        {
          "type": "TextBlock",
          "text": "@{triggerOutputs()?['body/LblClosedTitle']}",
          "weight": "Bolder",
          "size": "Large",
          "isSubtle": true,
          "wrap": true
        }
      ]
    },
    {
      "type": "TextBlock",
      "text": "@{coalesce(triggerOutputs()?['body/Description'],
triggerOutputs()?['body/Title'])}",
      "weight": "Bolder",
      "wrap": true
    },
    {
      "type": "TextBlock",
      "text": "@{replace(replace(triggerOutputs()?['body/LblAlreadyClaimedByMsg'],
'{title}', triggerOutputs()?['body/Title']), '{claimer}',
triggerOutputs()?['body/ClaimedBy'])}",
      "wrap": true,
    }
  ]
}
```

```

        "spacing": "Small"
    }
}
}

```

“ The `emphasis` container style renders as a subtle grey banner — deliberately *not* green/red, because this peer didn't make the decision; their card is simply no longer actionable. `LblClosedTitle` ("No Longer Open") and the already-claimed body both arrive pre-translated in the approver's language on the SP item.

Localised message templates

The three Teams *Post message / Update message* fields use BC-translated message templates with placeholder substitution. BC writes:

- `LblAlreadyClaimedByMsg` — "This approval ({title}) has already been claimed by {claimer}. You can safely ignore this card."
- `LblAlreadyHandledMsg` — "This approval ({title}) has already been handled by another approver or cancelled in Business Central. You can safely ignore the earlier card."
- `LblClaimedSuccessMsg` — "The approval entry \"{title}\" has been successfully processed." — **legacy**. Was the Card B Update message; superseded by the neutral `LblResponseSent` collapse text. Still written to the SP item for backward compatibility, so an older flow that still binds to it keeps working.

The flow uses `replace(...)` in PA expressions to substitute `{title}` and `{claimer}` at runtime. Adding more placeholders means adding more nested `replace()` calls.

Result-card labels

These plain-word, single-line labels back the result card and the collapse text. BC writes each in the approver's configured language:

Field	EN source	Used by
<code>LblApprovedResult</code>	Approved	result-card header (prefixed <code>□</code> in the flow)
<code>LblRejectedResult</code>	Rejected	result-card header (prefixed <code>□</code> in the flow)
<code>LblResultTitle</code>	Approval Decision	result-card sub-heading

Field	EN source	Used by
LblClosedTitle	No Longer Open	Close Card header (v2 — peer card replacement)
LblResponseSent	Response sent.	wait-card collapse text (status-neutral; cannot self-reference the action)

“ The glyphs (/) and the red/green container `style` live in the **flow**, not in these labels — so the words stay translatable and the card stays renderer-safe on Adaptive Cards 1.5.

Exporting and re-publishing the solution package

After modifying the source flow, re-publish the solution package customers import.

1. Verify the source flow

Run a quick sanity check before exporting:

- Open the flow in PA → verify that the SharePoint trigger and all list / item actions reference the environment variable parameters (`parameters('Sharepoint Site Address ...')`) rather than hard-coded URLs.
- Verify the `parameters` object in the exported JSON contains `dxp_SharepointSiteAddress` and `dxp_SharepointListName`, and that their `defaultValue` fields point to the DEXPRO sandbox site (not a customer's production site).
- Confirm `Apply to each` peer-cancel loop is **single**, not nested. The designer sometimes auto-nests when you re-bind expressions.
- Confirm all Teams actions point at the **same Teams connection**. Right-click each → *My connections* → ensure the same connection is selected. This keeps the exported solution to a single Teams connection reference.

2. Export

1. Power Apps maker portal → **Solutions** → open `DEXPROApprovalFlow`.
2. **Export** → **Publish** (only if you've made changes since last publish) → **Next**.
3. **Version**: bump (e.g. `1.0.0.x` → `1.0.0.x+1`). PA appends to the package filename.

4. Choose **Managed** for customer distribution (locked, version-tracked) or **Unmanaged** for in-house testing / branching.
5. **Export** — wait for the toast → click **Download**.

3. Sanity-check the exported .zip

Unzip and look at:

- `solution.xml` — version bumped, publisher correct.
- `customizations.xml` — **one** SharePoint connection ref + **one** Teams connection ref. If you see two Teams refs, return to the source flow, fix, and re-export.
- `Workflows/<flow>.json` — open and check:
 - The `Compose_Actions` / `Compose_Actions2` actions match the [Compose expressions](#) section, and each *Post adaptive card and wait* action references the right one (`@{outputs('Compose_Actions')}` for Card A, `@{outputs('Compose_Actions2')}` for Card B).
 - Card A and Card B JSON in the *Post adaptive card and wait* actions match the [Card JSONs](#) section above.
 - Each decision branch (approve / reject / claim-after-Card-B) ends with a *Post message in a chat or channel* action carrying the [Result Card JSON](#), and its `@{DECISION}` expression reads the correct `Post_adaptive_card_and_wait_for_a_response` (approve/reject) or `..._2` (claim) action.
 - The `Apply_to_each` action contains a single `Update_item`, not nested foreaches.
 - No stale `https://dexproolutions.sharepoint.com/...` references outside of the environment variable `defaultValue` fields.

4. Publish

Drop the `.zip` into the repo's `release/` folder (or wherever the customer-facing distribution lives) and bump the customer setup guide's reference link if it points at a specific filename / version.

5. Update version references

If the import-step instructions in [SETUP-GUIDE.md](#) include a specific solution version or a download URL, update those.

Future improvements

These are tracked but not in the current package.

- **English** `LocalizedName`. The customizations.xml only has `languagecode="1031"` (de-DE). Adding `"1033"` (en-US) gives non-German tenants a sensible flow name.
 - **Separate cards into JSON files** (currently inline in the flow JSON). Easier to lint, easier to diff in code review.
 - **Adaptive Cards 1.6** if/when Teams' Power Automate connector renderer supports it. Then we can swap the title block for an `Icon` element using the `ApprovalsApp` catalog icon and bump the visual polish.
-

Cross-references

- [SETUP-GUIDE.md](#) — customer / admin install guide. Step 5 (their version) is "import the solution".
 - [SHAREPOINT-CONTRACT.md](#) — the SharePoint list schema this flow consumes. Any column rename or addition there affects the flow's expressions.
-

Revision #2

Created 2026-06-24 12:18:46 UTC by Bernd Feddersen

Updated 2026-06-24 12:18:51 UTC by Bernd Feddersen