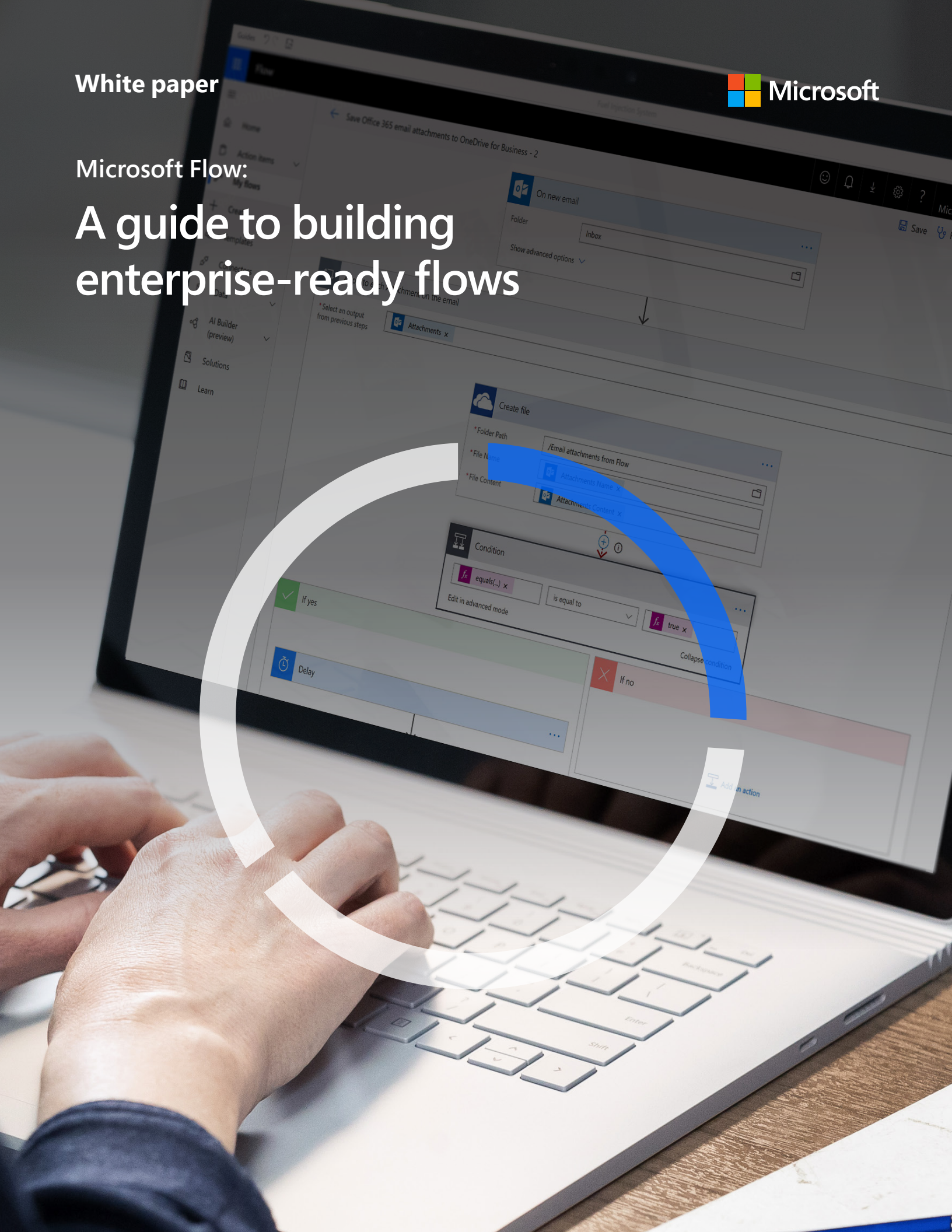
White paper

Microsoft

Microsoft Flow:

A guide to building enterprise-ready flows



Summary:

Microsoft Flow is a cloud-based service that makes it practical and simple for line-of-business users to build workflows that automate time-consuming business tasks and processes across applications and services. In addition to personal productivity uses, Microsoft Flow can also be used for automation as part of line-of-business applications built by a team of app makers that follows a more formal application life cycle. These can range from small departmental applications to applications that are mission critical for the enterprise. Often, a central IT group is involved in the management and promotion of these applications from their development into test and production environments. This document recommends patterns and practices targeted at that audience and anyone else that wants to create flows that are efficient, effective, and perform reliably at scale.

Author:
Jerry Weinstock, CRM Innovation

Technical Contributors:
Wim Coorevits, Microsoft

Audrie Gordon, Microsoft

Stephen Siciliano, Microsoft

Jussi Roine

David Yack, Colorado Technology Consultants

Julie Yack, Colorado Technology Consultants

Jessica Tse, CRM Innovation

Prerequisites

This document assumes a basic understanding of Microsoft Flow, and that readers have built several flows. If you are new to Microsoft Flow, you should consider visiting https://docs.microsoft.com/flow/ to become familiar with some of the core concepts prior to proceeding. This foundational knowledge will help you better understand the concepts presented in this document.

Objectives

Microsoft is modernizing business processes automation with Microsoft Flow. Microsoft Flow offers rich workflow and business process capabilities for building line-of-business applications. It is the connective glue that will be used by end users and app developers alike for digital transformation, regardless of what application they are using.

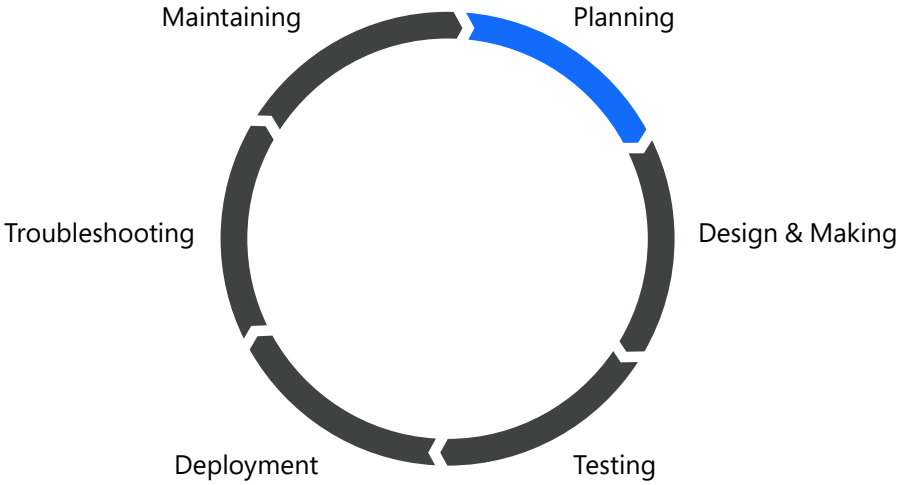
In this document we outline practices, principles, methods, and patterns for designing such Microsoft Flow solutions. We have set out to assist every flow maker to do their best to create flows that are efficient, effective, and reliably perform their intended business function. As adoption of Microsoft Flow increases, building flows that are scalable to the requirements of the organization will become essential.

Note: Microsoft Flow is increasingly seen as the successor to classic Common Data Service (CDS) workflow and SharePoint Designer workflow. Organizations should start using Microsoft Flow today to take advantage of its superior capabilities that include a flow checker, run history, analytics, rich connectivity to 200+ services, a visual designer, and much more.

And while there is some commonality in design practices and patterns for those of you that have experience with the classic CDS workflows or the SharePoint Designer, Microsoft Flow offers new and different capabilities that benefit from a new set of design patterns and practices.

Approach

Our approach to the organization of the topics in the document follows the life cycle of a flow.



- Planning: Proper planning is the foundation of all worthy projects; the simplicity of this step should not be glossed over. Understanding how to provide a sound basis for every single flow and the approach to using flows in your organization will have significant benefits.
- Design and Making: This section dives into the specific methods and approaches for utilizing triggers and actions efficiently and effectively.
- Testing: We review what you should know about the different testing methods and the pros and cons of each. We go over the considerations that you should know about for each of these methods.
- Deployment: Learn how to include flow in your application life-cyle management process for your organization.
- Troubleshooting: When things go wrong, you need to know where and what to look for and the best ways to get it working again.
- Maintaining: Flows have been built and are enabled. You expect them to run with minimal management. Learn the best methods for achieving that objective.

Throughout this document each recommended practice describes an approach and includes a practical example. We have specifically deferred from referring to them as "best practices" to avoid the perspective that there is only one way to do something right. There may be multiple choices on how to build a flow "right", but they can be situationally specific. Therefore, consider the recommendations in this document just that, a preferred way to approach building Flows.

When possible, the examples are generic (not connector specific). However, when necessary for clarity, an Office 365 family product or the Common Data Service (CDS) is used in the example. Additionally, we also outline product-specific practices related to CDS or SharePoint.

Microsoft Flow Version

The information in this document is based on the released version as of March 1, 2019, with some consideration of the pending enhancements outlined in the April '19 Release Notes: https://docs.microsoft.com/business-applications-release-notes/april19/.

Recommend Practices Recap

For those that are not inclined to read this document in its entirety, we have identified and recapped some of the most important points to help ensure your flows successfully complete every single time.

Planning

- Keep it simple; you can create some very powerful flows that only have a few steps.
- Adopt a naming convention for use by your organization.
- Document the version changes in the flow description area.
- Build documentation into the flow by personalizing the trigger and action card names and using the comments fields.

Design and Making

- Do all your condition checks outside of looping actions.
- Configure "run after" to get immediate notification for flows that aren't completing.
- Iterate—build your flow one piece at a time, validate, and then expand.
- Use recurrence to control flow runs to match business requirements.
- Plan for unexpected inputs in instant flows.
- Understand the flow execution limits, and design the flow to stay within those limits.
- Avoid using hard-coded values and use dynamic content to make them portable.
- Use variables and compose to simplify checks and conditions.
- Be on the lookout for flows that would create an endless loop.
- Limit the number of records returned from the data source by building filters that only return the minimum number of records from the source service.

Testing

- Use the built-in testing capabilities along with the flow checker to test, review, and modify the flows.
- Speed up testing by using button triggers when possible, then use the actual trigger.

Deployment

- Use solutions to transport flows from development to test, to production as part of your application life-cycle management process.
- Educate your users about different methods for sharing flows and the permissions granted by doing so.

Troubleshooting

- Process failed flow runs using the resubmit function to rerun the flow with the exact same data.

Maintaining

- When flows aren't part of an ALM process use the *Export the flow as a package* feature to create your own versioning history and backups.

Planning

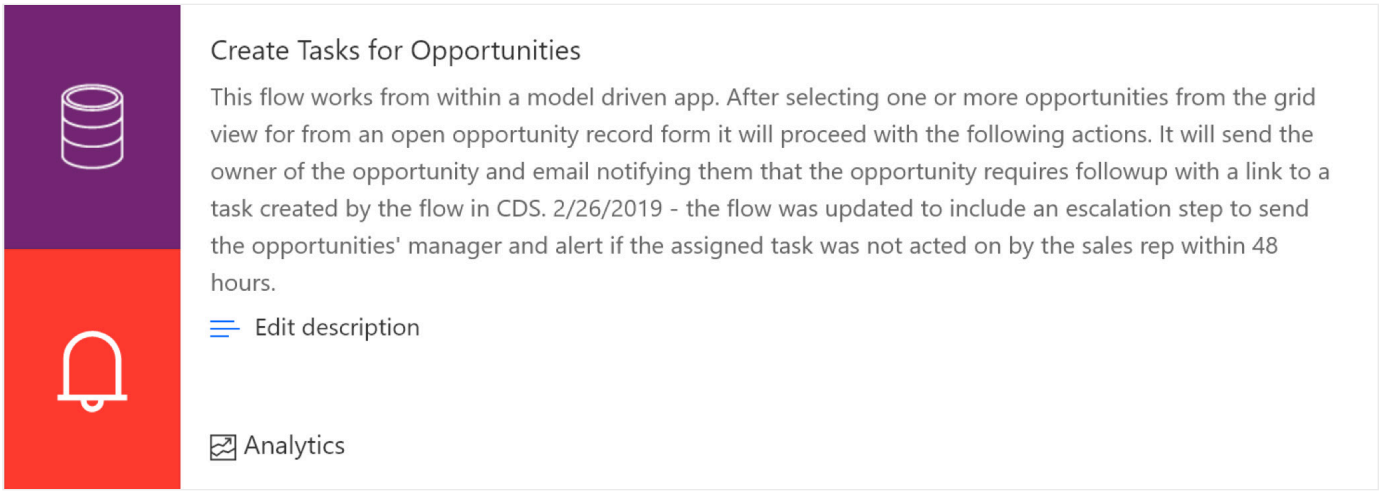
Flow can be a surprisingly easy application you can use to develop powerful solutions. It can be tempting to just jump into the user interface (UI) and start clicking on triggers and actions, developing as you go. This can affect the quality and effectiveness of your flow later. We are not suggesting that every flow you develop must go through a rigorous development process. But some appropriate amount of preparatory work you do will be beneficial.

The following sections introduce some important documentation steps that will ultimately save you and your team a lot of time (and grief).

Document Your Work

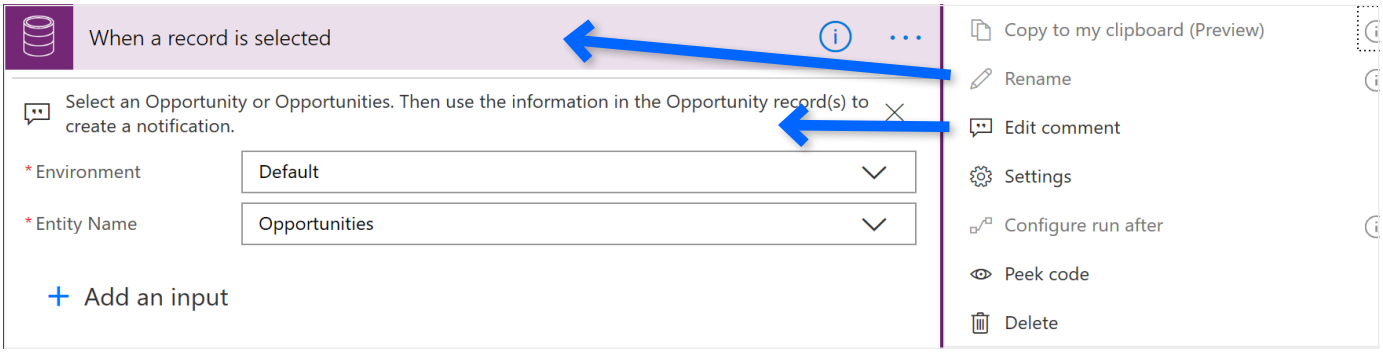
Use the *Edit flow* feature to customize the name of the flow and insert a description of the purpose of the flow. Use a name that is informative to you and to other members of your team. It is recommended to put a version history in the description area too. This is where you can maintain the change log. The description area doesn't support carriage return / line feeds, so everything will appear as one paragraph regardless of the formatting you insert.

For example, this flow has been given a descriptive name along with an explanation of its use and purpose along with versioning dates.

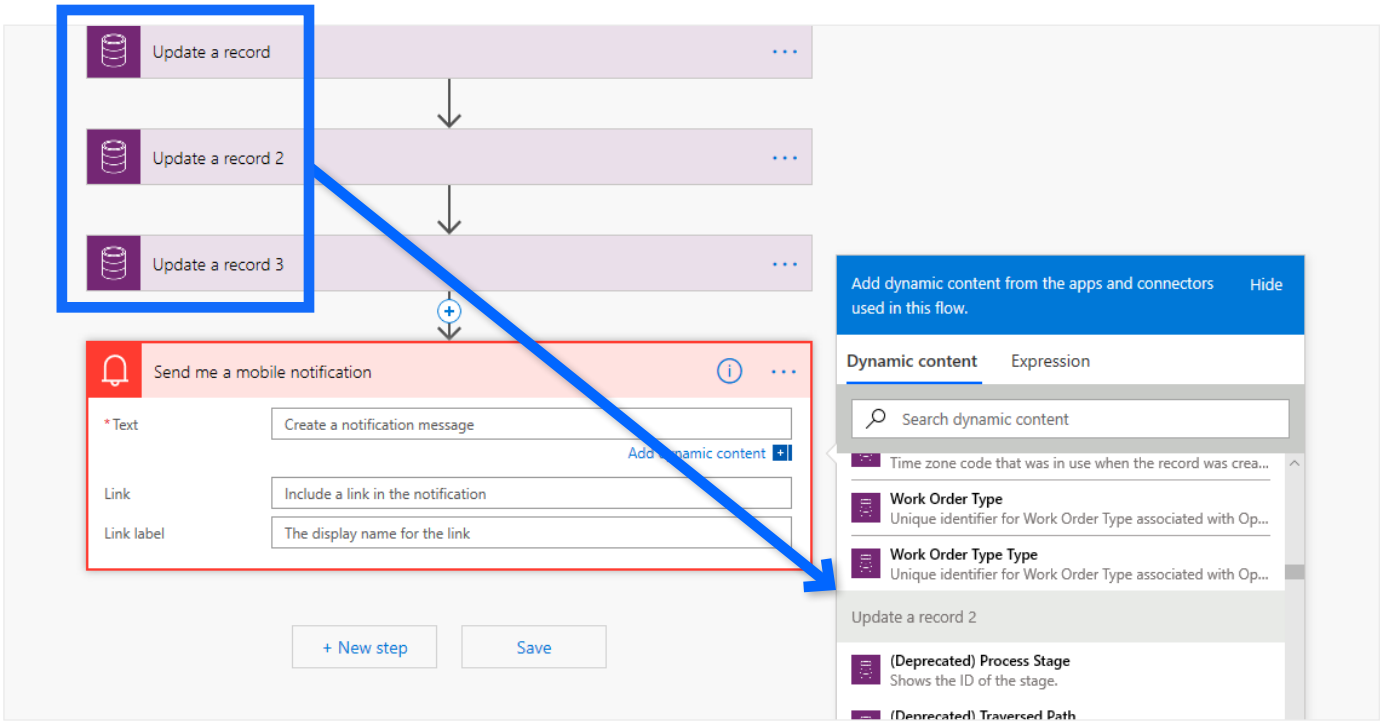


Step Names and Comments

Step names default to the connector trigger or action name, for example: update a record. If you don't update the names of the triggers and actions, you will have a hard time later constructing the downstream actions as the default naming doesn't provide much insight. Within the step there is also a setting where you can insert a comment. This is the best place to put the description of what the step's purpose is versus trying to use the name of the step to describe its entire function.

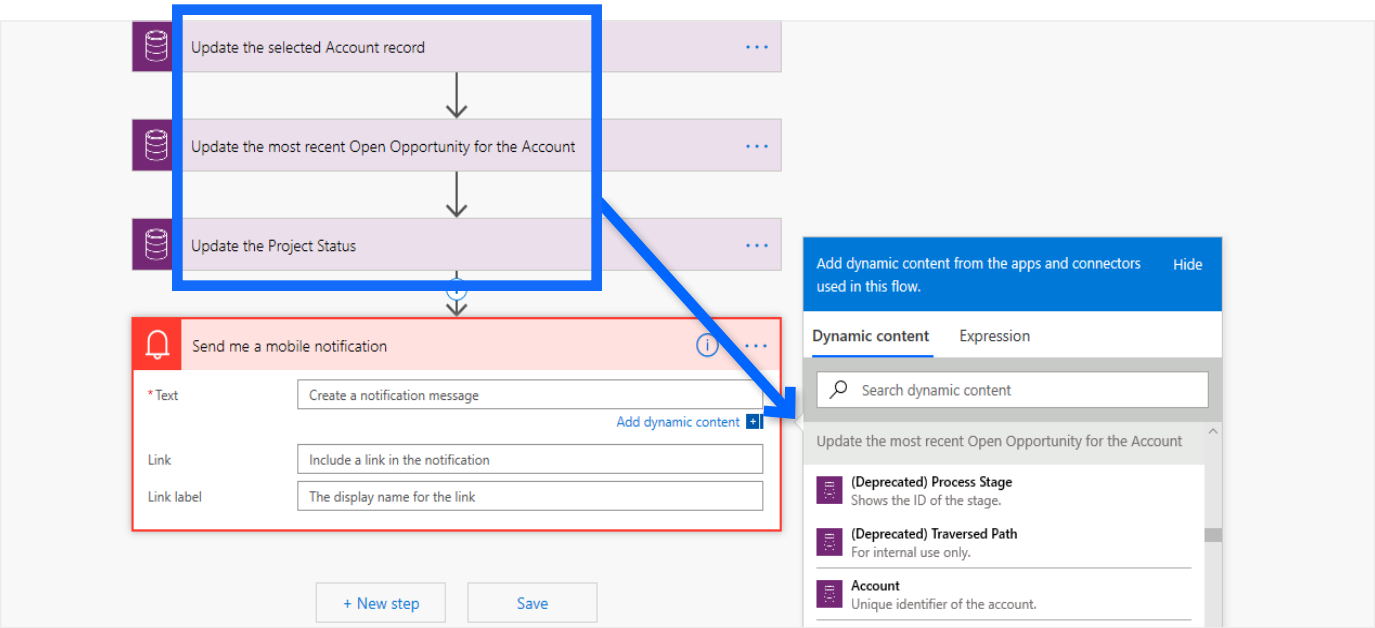


The default naming will look as follows when subsequent *Update a record* CDS actions are used. Then you are stuck having to pick the correct dynamic content from a generic listing, which makes the flow design and management that much more difficult.



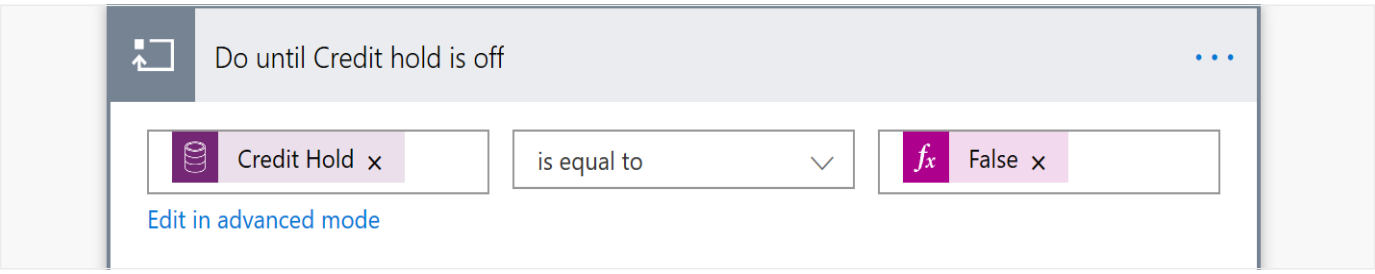
A superior approach is to rename the actions or triggers to a descriptive explanation of what that activity does in the flow.

Look at the same flow components, but with the actions named for what they do.



Renaming is also very important for conditions and loops. Naming conditions and loops makes it easier when you are inspecting flows and understanding why a flow took one path versus another or what a certain loop is doing. Without relevant naming, you will only see a generic name that does not provide any insight into what that step does.

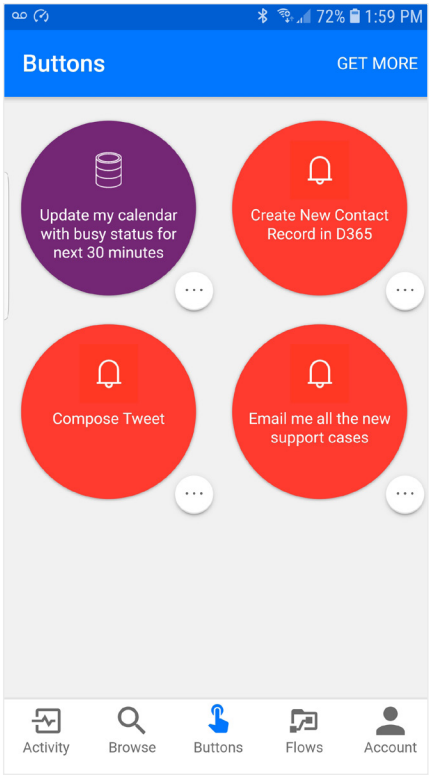
For example, you might rename it something more informative like: *Do Until Credit hold is off.* That makes it much easier to understand what is happening.



When you take the time to pick good step names, your flow will essentially become self-documenting.

Readability

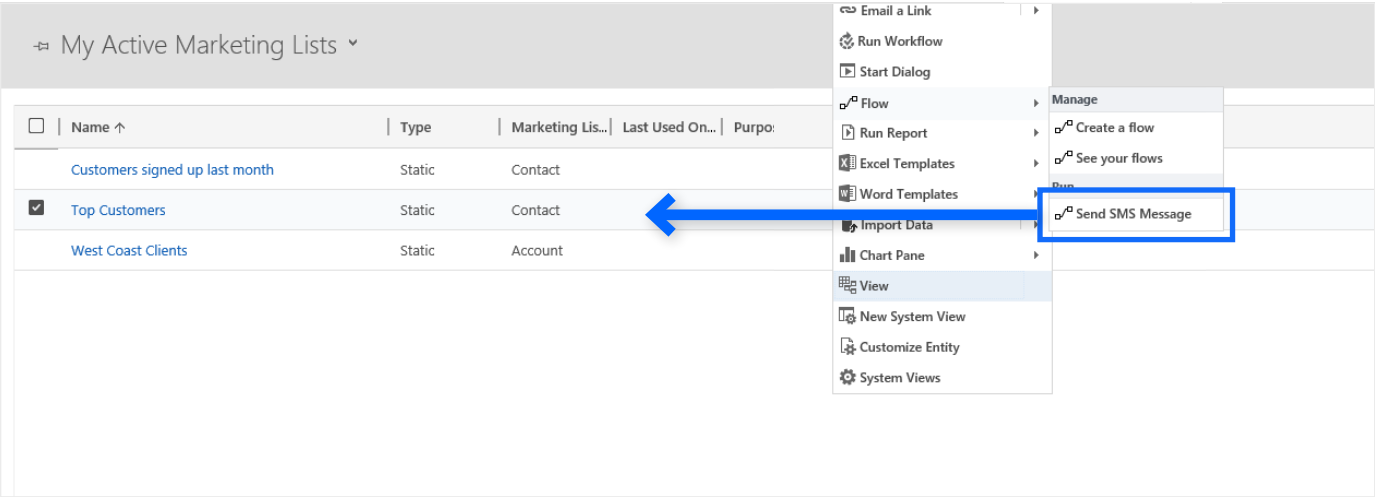
Teams using flows should consider naming for both consistency and ease of readability. This is especially true for button flows. Button flows are instant flows that appear to everyone in your tenant with whom they have been shared. For these flows it is most important to have a descriptive name assigned, because the name given to them is the name of the button as it appears to the user in the mobile app.



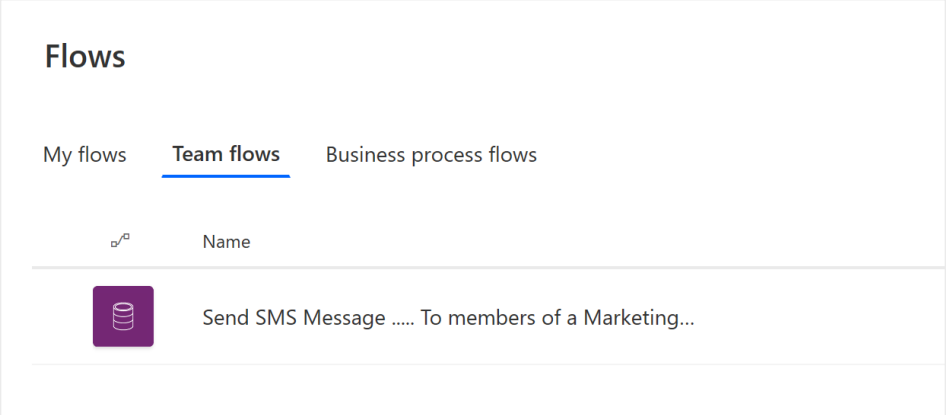
Note: The maximum length of a button name is 27 characters when inside the app. However, if you are using the pinning function for a button you will have substantially fewer characters displayed. Understand how your users will be using the buttons and plan accordingly.

The other type of flows where the naming is critical are instant flows, such as those that start with the CDS trigger *when a record is selected* within a PowerApps or Dynamics 365 app (other examples include OneDrive and SharePoint). These flows are shown in a menu in the context of the entity type with which they interact. There is a limited amount of space for the name, so the most descriptive part of the name should be first. And since the flow runs from within the entity, it is not necessary to include the entity name in the visible part of the flow name.

When creating the flow name, think more about what the flow does or accomplishes rather than where and how it runs. In the following example we are using a flow to send SMS messages to members of a marketing list. The name of the flow says what it does, and because it is running against a marketing list, we don't need to repeat that in the flow name.

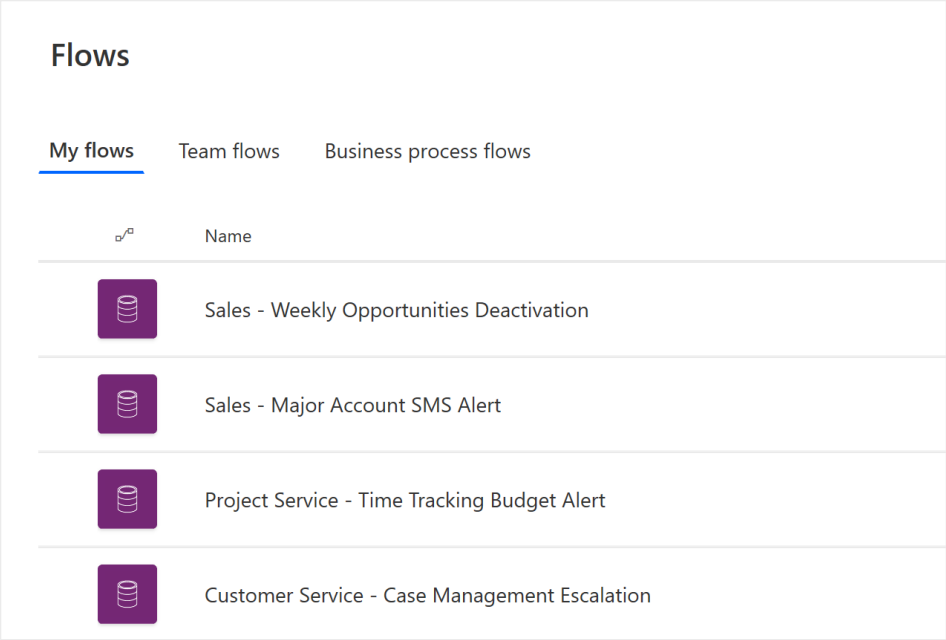


If for any reason you need a more complete description, just add the descriptive information at the end of the name.



Other types of flows, both recurring and event driven, are not seen by the end user. They are asynchronous flows that run in the background, hidden from the user, and execute a predefined set of actions. They are only visible in the Microsoft Flow interface to individuals managing the solution. However, that doesn't diminish the need for them to be named clearly so an(y) administrator can identify what flow(s) have run or failed.

For example, naming something "Deactivate old records" would not be as clear as "Sales—Weekly Opportunities Deactivation." From quickly looking at the latter, you can tell that the flow is part of the Sales app and not the Customer Service app, it runs weekly, and cleans up opportunity records.



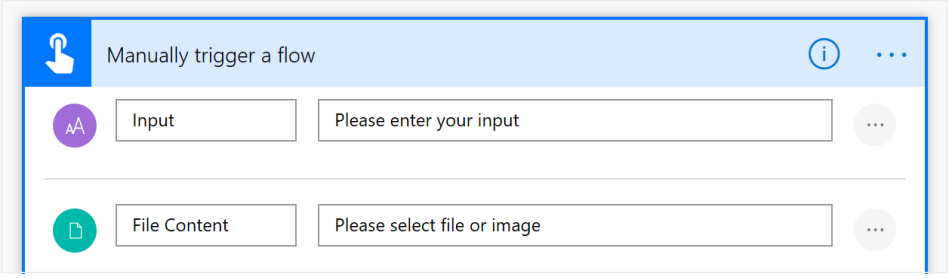
There are multiple ways to approach flow naming structures, so get your team to develop a convention and stick to it.

Naming seems like such a minor thing to spend time on; however, a little pre-work can make the names of flows more usable to all those that use and manage them.

Flow Button User Input

Instant flows have the capability to prompt the user for input parameters that will be available to the flow at execution. These flows can be an alternative to both on-demand classic CDS workflows as well as some CDS dialogs that you might want to replace with a more flexible workflow automation. They can also serve as structured form input for SharePoint and dozens of other connected services. Make sure to provide a valid name for them as that name will be displayed to the user that triggers the flow.

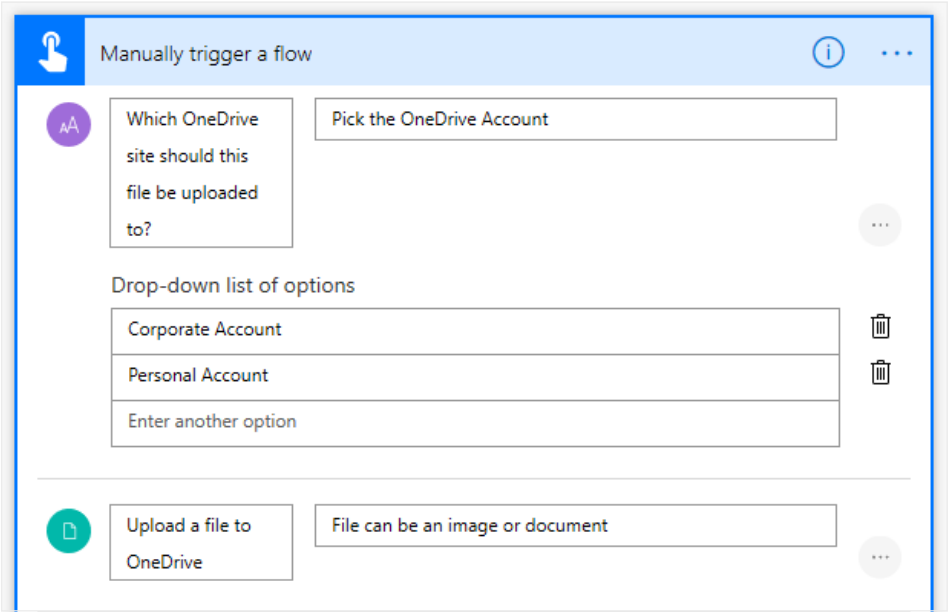
In the choose the type of user input area you can add one or more fields that will be presented to the user to enter. The following is an example of the default values when adding a text input and a file prompt.



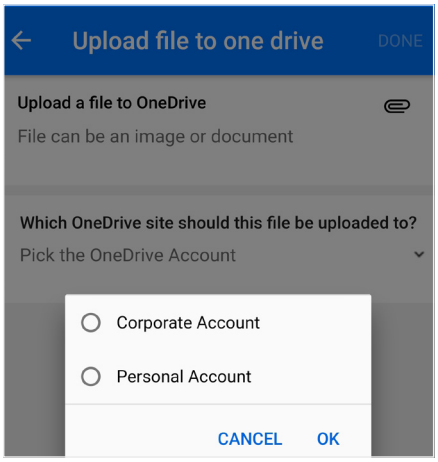
The first text box is the label that will be shown to the end user, and it is also the name that appears in the dynamic content in the flow designer. The value you provide should therefore mean something to the end user. The second text box is an input hint that will be displayed to the end user when running the instant flow.

The following is an illustration of how you might change the defaults if you were prompting the user to upload a file to OneDrive. As you can see, it gives clear guidance to the user on how their response will be used.

This is the view from the designer for the maker:



And here is how it will appear to the user in the mobile app:



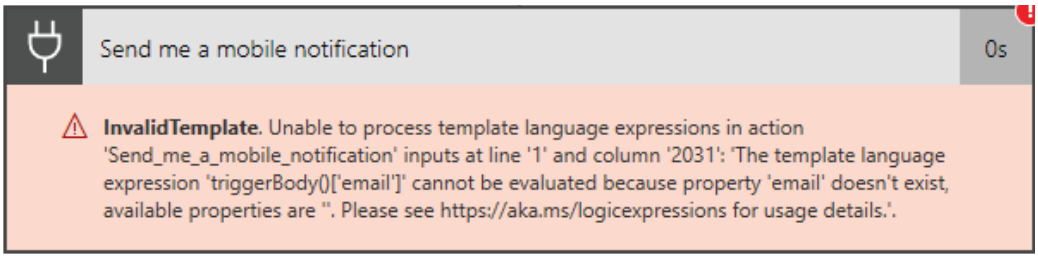
Input fields will appear in the order they are created. The trigger doesn't offer the capability to change the sequence. The only option is to remove and re-add in the right order. Planning here is very important to minimize rework.

Changing the input field names will automatically update the label for the end users as well as the names used by the dynamic content fields in your flow.

> TIP: If you delete the field, it will not delete it downstream where it has been used. The flow can be saved, and the button will run; however, it will fail since the field will have a null value.

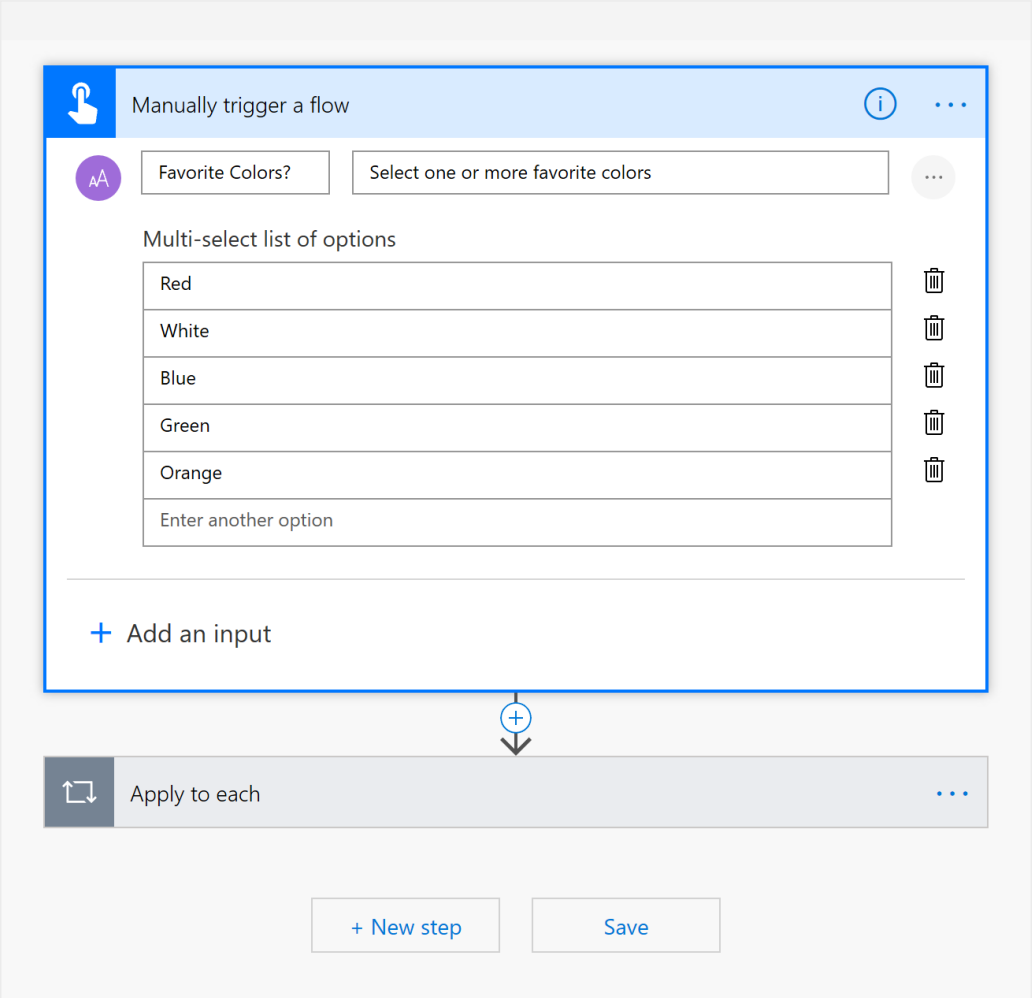
Button Input Settings

For each of the button input types there is a settings feature. All of them except input have only one option: *Make the field optional*. If you enable this setting, you will need to properly code for it downstream because when flow is presented with a null value in an action step, it will cause the flow to fail as in the following example:



See the section Null Values in this document for design patterns for handling null values.

In addition to the optional field setting for every input field, a single text input field can be turned into either a single-select drop-down list of options or a multi-select list of options. For those of you coming from CDS, note that a multi-select list doesn't function the way a multi-select option set behaves in CDS. In flow each selection is treated as a unique response with no option to concatenate the selections into one output field. Therefore, Microsoft Flow will automatically add an **Apply to each** loop to your flow.



If this is not what you want, you will need to rearchitect your process or business logic. To concatenate the multiple responses into one field, you would use the join action.

Designing and Making

In this section we cover the technical aspects of making a flow, so it executes the intended business logic or fails gracefully if something unanticipated occurs.

General

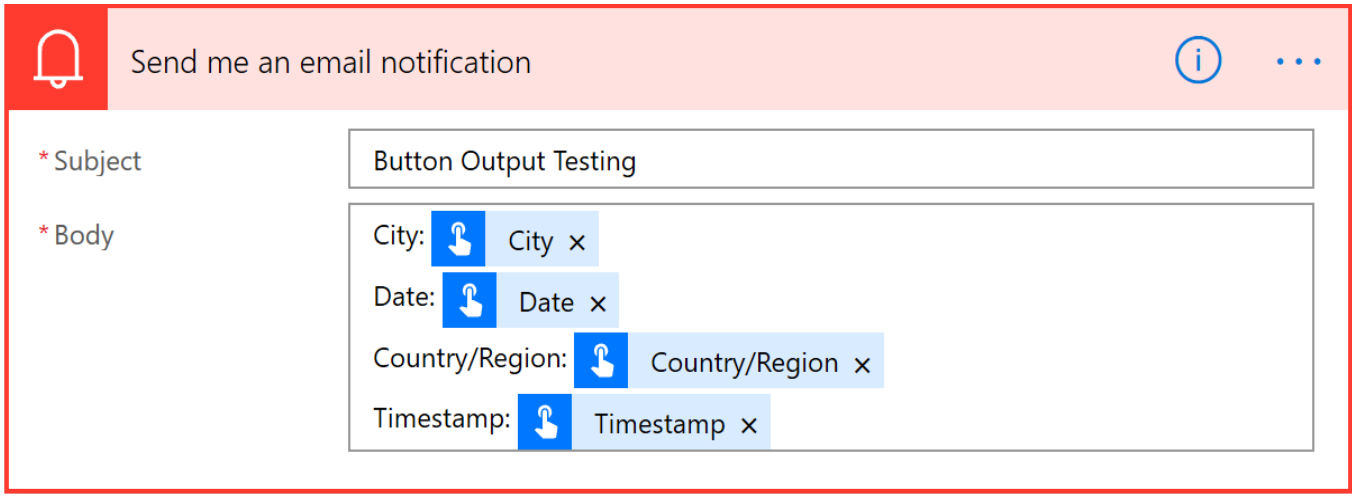
Let's look at several design practices that fit into just about every business application that might become a flow.

It is obvious that the number one approach to building great flows is to keep them simple. It is quite possible to create flows that perform very valuable personal productivity or enterprise tasks that have only a trigger and one action. That being said, the reality is that an enterprise-level flow will likely have multiple steps and various output actions to fully fulfill its business purpose.

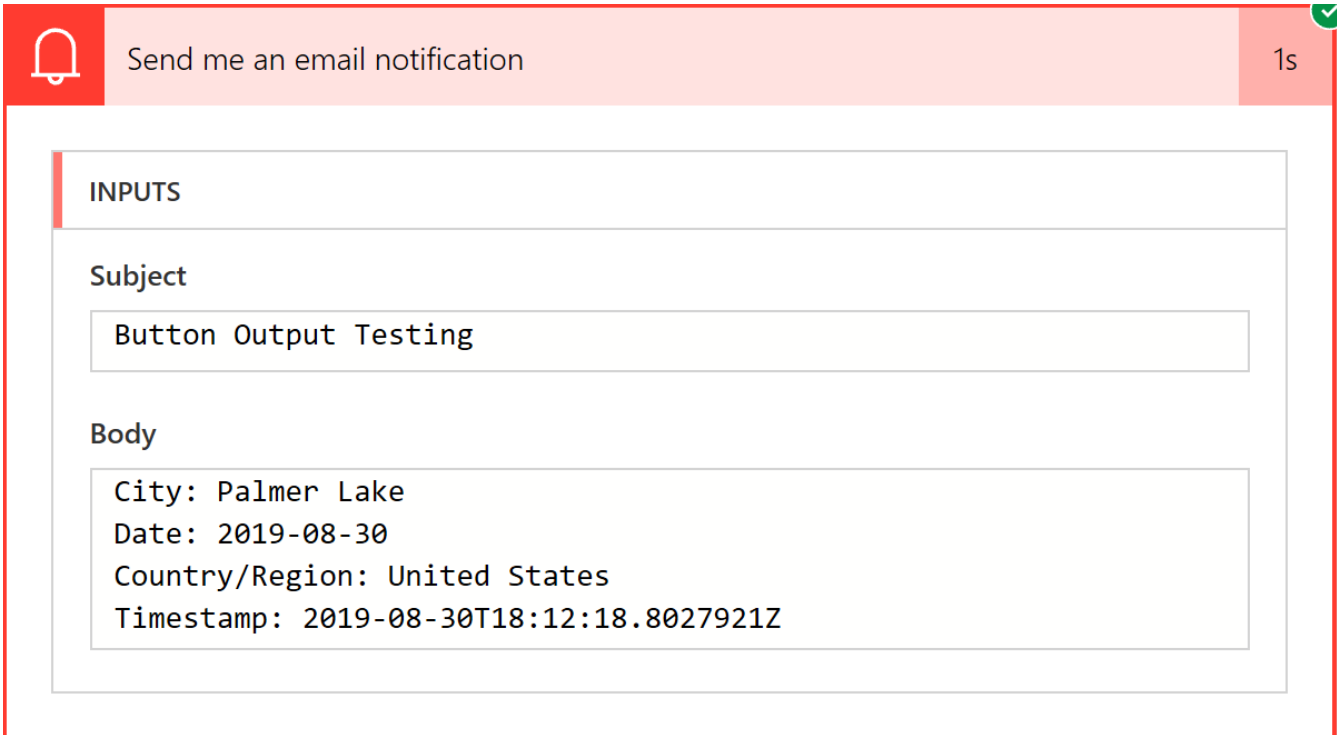
The first step in creating a flow is defining the starting event, the trigger. If you aren't quite sure what you want the trigger to be, or if you need to test or prototype a design pattern before building out the entire flow, use a button for the trigger event. This can make it easy to quickly run your prototype on demand as needed.

If, on the other hand, you want to test the trigger but haven't figured out what actions should follow, then you can simply use the notification action. Flows must always have at least one step in addition to the trigger.

The following example illustrates a quick way to see the results of a button input without building out a connection to a service and figuring out the field mappings. Once you are satisfied by testing the upstream process, you can then add the appropriate connection and action in place of the notification. This gives you the ability to work on components of a larger flow one component at a time.



This action produces a nicely formatted output that you can use to easily inspect the format of the data output, which can be critical when working with dates and other specialty fields.

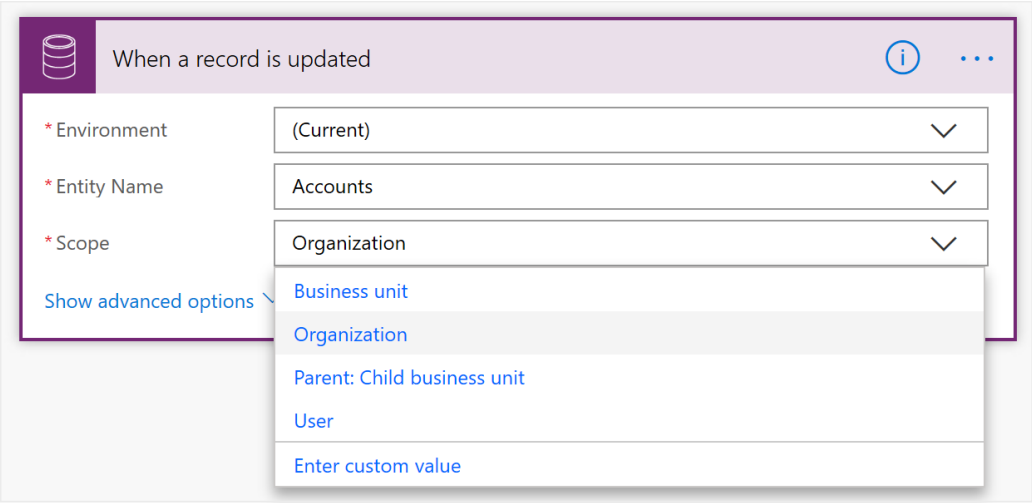


Triggers

The trigger is the component that listens for events to kick off the flow. To make sure that triggers fire when you want, for the types of events that you want, and within the constraints of the organization's security model, there are some points to consider.

CDS Triggers

When using CDS triggers, it's important to properly set the options on the trigger to ensure it only fires when the desired events occur. The create, update, and delete triggers all support a scope setting to help minimize unnecessary triggering. Scope should always be set to the appropriate level to minimize unnecessary triggering.

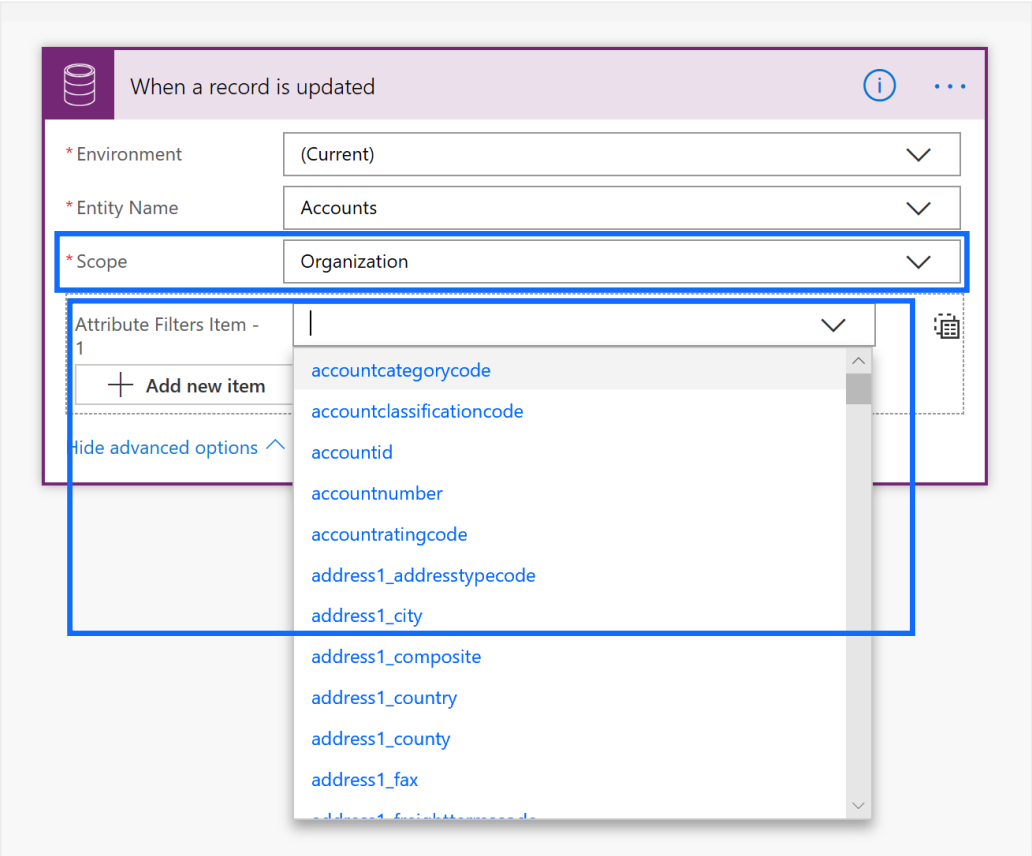


When you are creating flows for the entire organization, be sure to select *Organization* for the scope. If not, the flow will likely fail to perform as expected when running under a Service Principal account. Alternatively, if you are a flow power user and you are creating a personal productivity flow, then the opposite is true. You should set the scope to be more restrictive and triggered only by events that are relevant to the business purpose of your flow.

Note: If you are unfamiliar with the boundaries of *Scope* then refer to this page for further information: https://docs.microsoft.com/flow/connection-cds.

The update trigger has an additional attribute filter setting allowing you to control which records qualify based on field/attribute level detection. By default, any update to the entity will trigger the flow. Attribute filters allow you to specify which attributes your flow cares about and limits the trigger to execute only when those fields change. In general, you should always set attribute filters unless you really are interested in *any* change to the record.

Setting the scope and attribute filtering are essential. Do not take the easy way out with the intention of cleaning it up later.

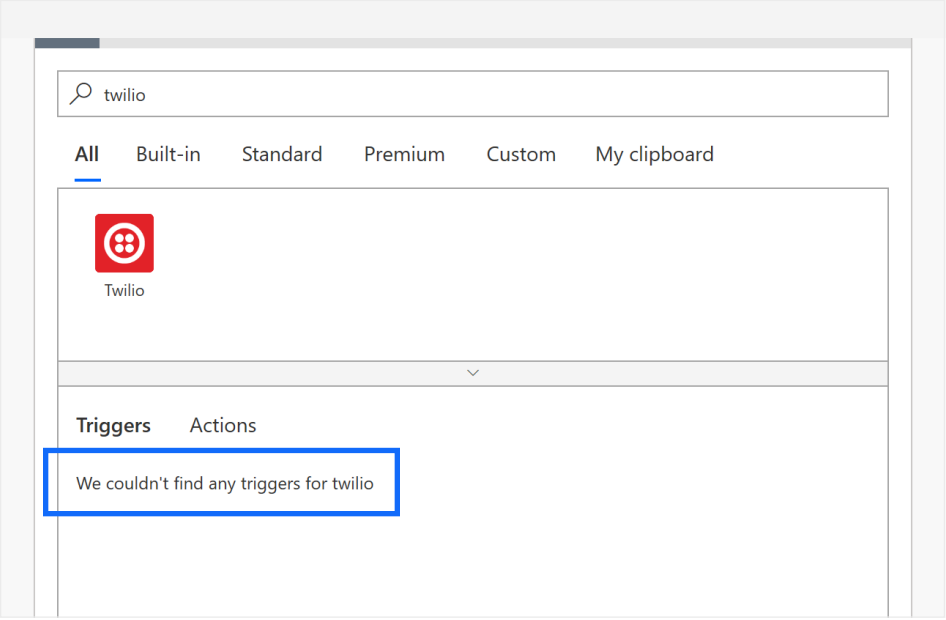


TIP: Using filtering is one of the best methods for minimizing the possibility that you create an endless loop flow due to actions later in the flow that update the same record that triggered the flow.

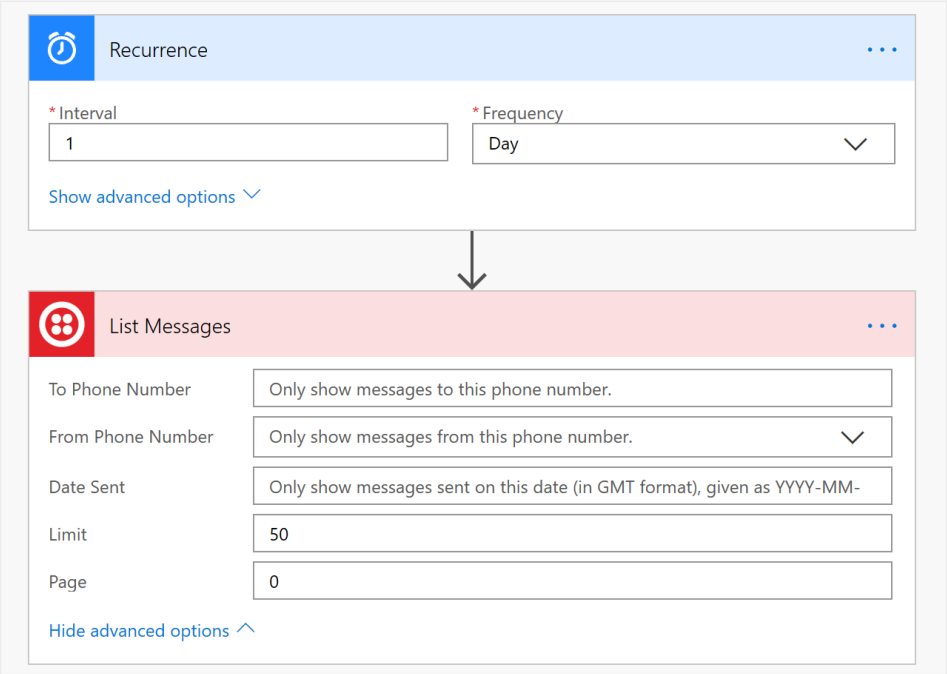
Recurrence

The recurrence trigger allows you to schedule the frequency of flow runs to match your business requirements. Not all flows need to run 24/7, continuously checking to see if an event has occurred. Additionally, recurrence is a good method to preserve the use of the flow runs allotment for your tenant while still providing the business logic support required. It can also be used to minimize the risk of exceeding the API calls allowed by the connected services.

Scheduling flows using recurrence is best for non-real-time event handling. Also, there are some connectors that don't have any triggers. It is necessary to use recurrence scheduling as the trigger for those services. For example, when looking for the Twilio connector it will tell you that there are no triggers for this connector. You will need to set a recurrence and then use a Twilio action.



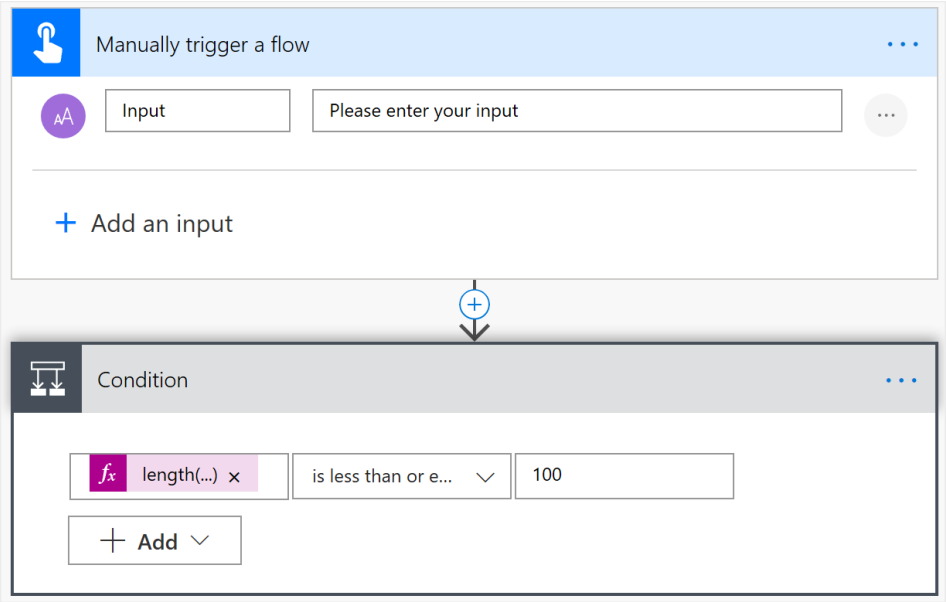
Here is an example of using recurrence with the Twilio action to check for new messages.



User Input Limits

User inputs in instant flows allow a very large amount of data to be entered. In fact, there is no set limit. The limit is ultimately determined by the size of the overall limit for a flow package. Therefore, when an action takes that input and tries to update a field in the destination, you may find that it will fail because the input exceeded the allowed length of the target. To prevent the flow from failing, you should check for its length in a condition and take an alternative path if it exceeds the allowed value. Limits and configuration details can be found at: https://docs.microsoft.com/flow/limits-and-config.

This is an example of a design pattern that checks the length of the input field and would take a certain action depending on the results. This could be used when it is not appropriate to just shorten the input and a notification needs to be sent to the form submitter that the input was not processed.



Another approach, without relying on a condition check, just uses substring to limit the input to the maximum length allowed.

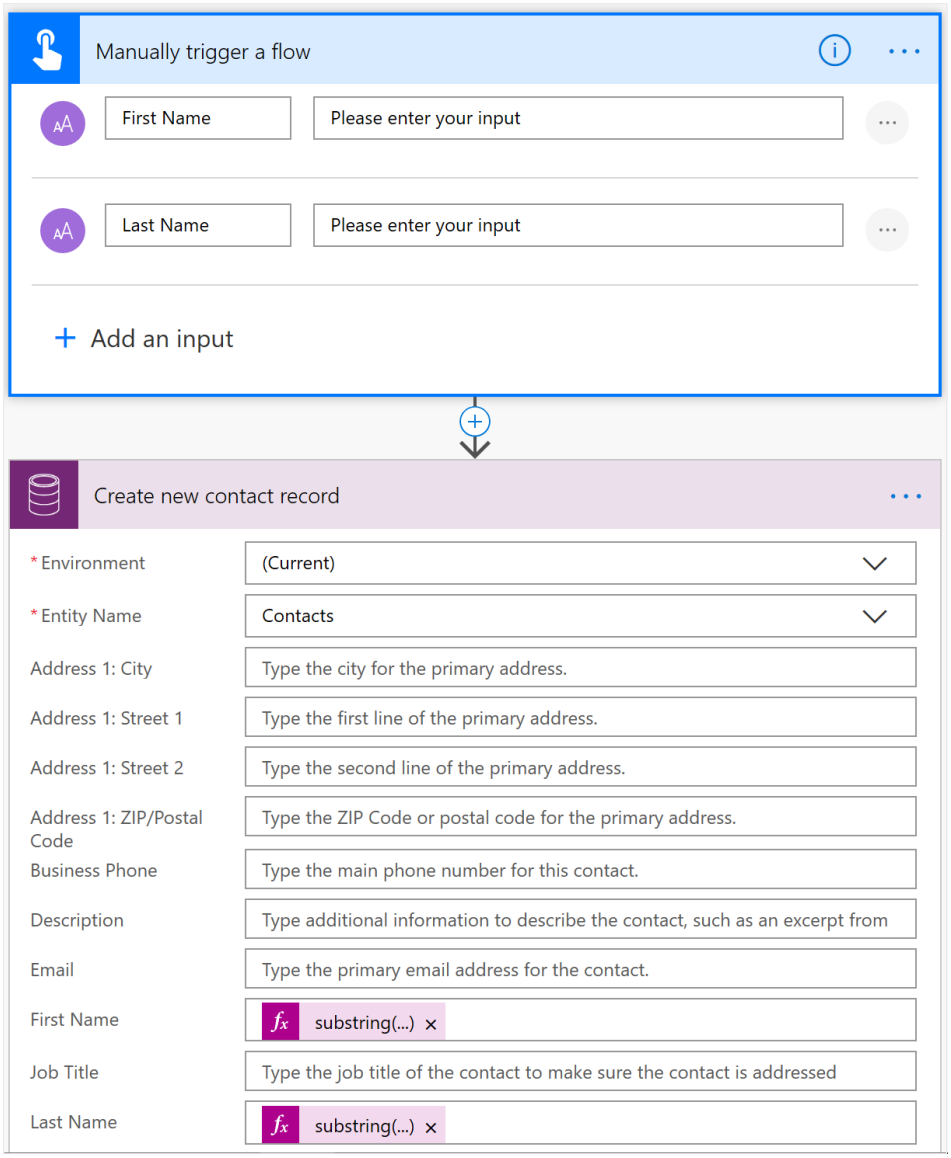
In the following example, we are using the substring expression to ensure that the resultant value does not exceed the default length (50 characters) allowed for the first and last name fields in CDS, nor does it return a null if the initial text input is less than 50.

For First Name:

```
substring(triggerBody()['text'],0,min(length(triggerBody()['text']),50))
```

For Last Name:

```
substring(triggerBody()['text_1'],0,min(length(triggerBody()['text_1']),50))
```



Actions

Actions allow you to perform various operations on records, such as *Create*, *Update*, *Delete*, *List*, *Upload*, *Append*, and *Get* as well as many actions on connected services like sending a SMS, translating text, uploading an image, getting approval, or sending an email, to highlight just a few.

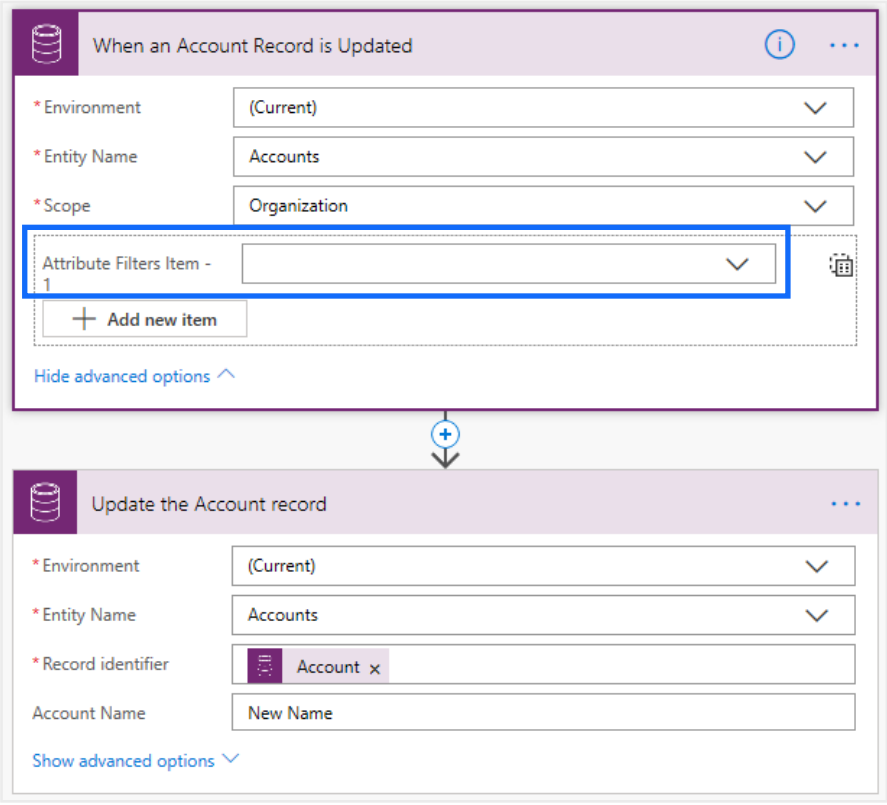
Updating Records

There are some details to be aware of when you are using the update records action.

1. Only set the fields that you intend to update with changed values when performing an update. If you specify fields that have the same value, it might be treated as change and trigger other automations. This can cause unintended consequences. Remember, keep it simple, don't do something "just to be sure."

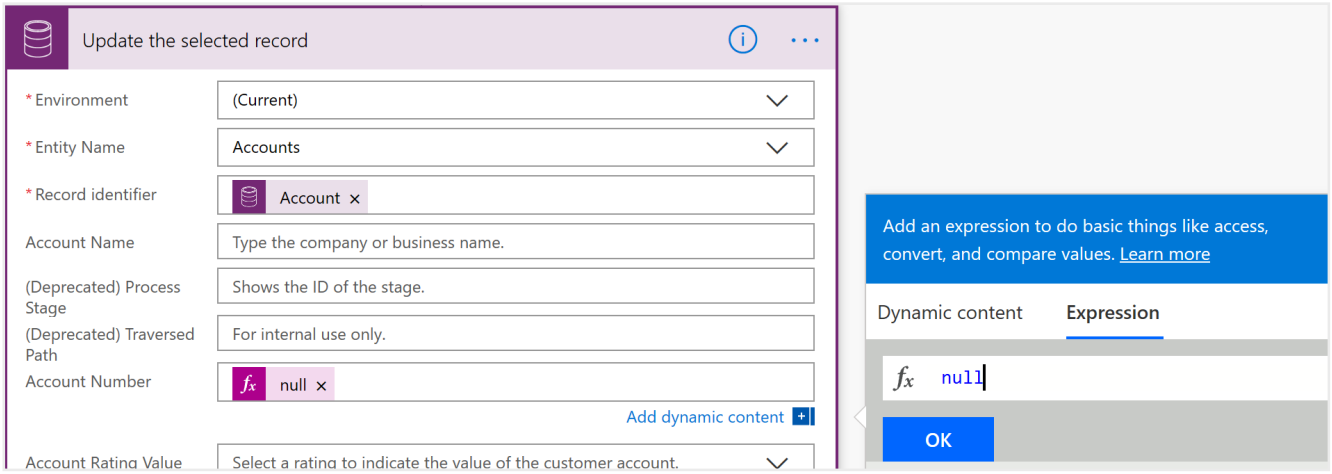
2. In SharePoint the number field columns can often have a default value of 0 (to avoid nulls), so every create/update action for that list will have the 0 number defaults in the appropriate fields. This can cause data to be overwritten if no one remembers to add the appropriate dynamic fields.

3. The following is the classic endless loop scenario for a flow. The flow is triggered when a record is updated without constraining the filtering. Then, in the downstream action the same record that triggered the flow is updated. This will cause an event that will trigger the flow again and again and again. Be sure to set an attribute filter or some condition flag to check to not get into this looping process.

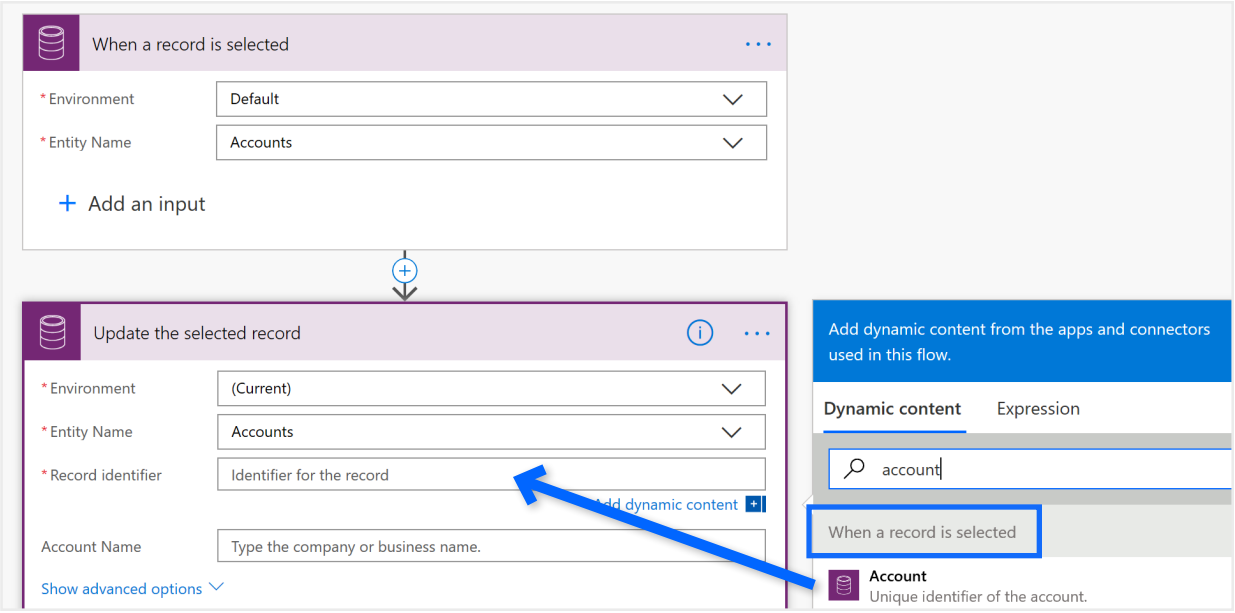


Another way you can put your flows in an endless loop scenario is when you have two flows updating a record in a field causing the other flow to trigger. When you are using the recommended techniques for naming your flows as outlined in the first section of this document, it could bring to light this type of scenario before it even kicks in.

4. When you need to clear a field during an update of a record, set the field to null. In this example the account number is cleared using the null expression.



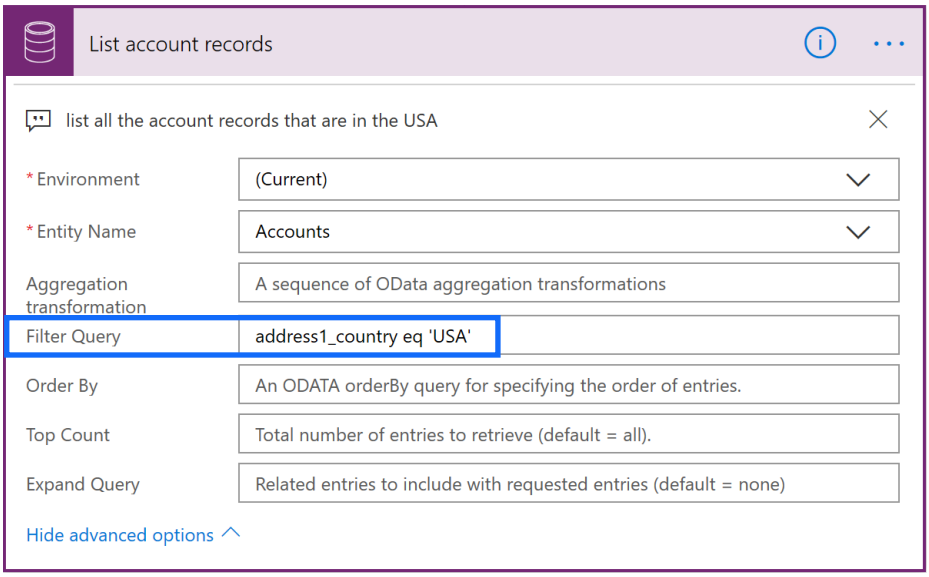
5. In the case of the record identifier, it is best to stay away from hard-coding the ID. Use dynamic content whenever possible. If you get the ID wrong and it cannot find a match, the flow will fail. In the following example, the ID of the record from the trigger *When a record is selected* is used by inserting the value from the dynamic content versus attempting to enter the actual ID.



Retrieving Records

When retrieving records, all filtering should be done as part of the initial *List records* action.

The following example brings back only the records in the database where the account's country field is equal to "USA." Here is what it would look like.



Filtering after the records have been retrieved from the source is inefficient and would require them to be filtered inside of the flow. Additionally, if you do not filter, you could easily surpass the 512-record limit for the number of records that a flow can retrieve without pagination.

TIP: A properly constructed filter will reduce the execution time by only processing the minimum number of records needed to support the business logic.

Controls

A control is a special type of action that works within the flow itself. They allow you to check for a value, wait until something reaches a certain state, or do something to all the records that meet a filter criterion. They are essentially the way to execute loops and conditions.

Switch Case vs. Condition Check

Microsoft Flow supports nesting conditional checks, which make it possible to have very advanced branching logic. Whereas switch case makes it very easy to have parallel branches of conditional logic based on a single value. If you go down the condition check path, be advised that flow has an upper limit of five nested levels.

Conditions

A condition check specifies which path flow will take when a check of the record resolves as true or false. Very often, the downstream logic is built completely for the true branch of the "condition" leaving no actions in the false branch. While in many scenarios this is correct, there are clearly instances where not having an action in the false branch is not recommended. At the minimum you might add a notification action to alert you that this path was traversed, notifying you that the expected logic path failed. Such an alert may influence you to revise the logic. Be careful when you use the terminate action to completely end the flow in a branch as there may still be successful parallel operations that should continue to process.

Switch

The switch case is a conditional check when you want to take a unique action for one or more of the data values. For example, with a call preference option value for a contact you might have different logic for a value of email versus a value of phone call.

It is possible to check up to 25 values from a source field. Although you can do that many, it would present a very busy look and challenging flow logic to manage. It is also likely that you end up duplicating a lot of logic when your switch case has that many conditions. If you do end up with more than you are comfortable configuring (and that is determined by you and not a preset number), you may want to review the underlying source data structure and the accompanying business logic. One of the options is to put common sub logic in another flow and call that flow as a child flow.

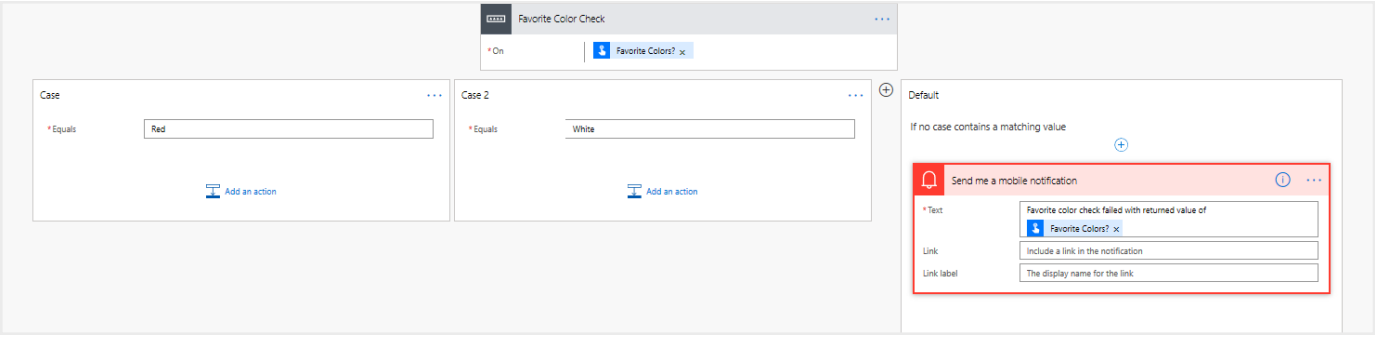
There are a few methods that can be used to accommodate exceptions to the expected values in a switch case decision tree. Using these will allow the flow to fail gracefully and provide you with the insight on the failure cause.

We will examine four concepts:

1. Using the *Default* case condition
2. *Configure run after* for failure trapping
3. Shared actions
4. Terminate

Use the *Default* Case Condition

When you add the switch control, it will automatically create one value check and a default branch for when there are no matches on the specific value checks. In the example that follows, you can observe that an additional condition check was added.



In this example we take a separate path for a customer and a vendor. But what happens if the returned value is not one of those two values? The flow will continue down the path of the default condition because it accepts any value that doesn't match the specific case branches. In order to plan for unanticipated changes to the source field, it is essential that you update the default condition check with an appropriate alert. This way, the condition will be trapped, and the maker informed that the flow didn't execute as expected.

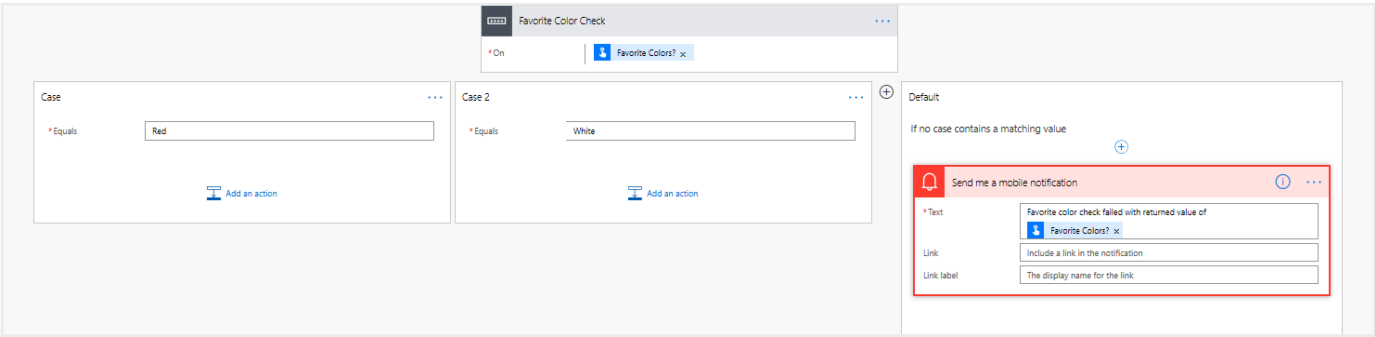
Configure run after for failure trapping

If the source data field contains a null value, the switch will fail, causing the flow to fail because a condition check will not handle a null value. Consequently, it is recommended that you use the *Configure* run after to trap the error and take an appropriate action. That action could be a notification action. This would let you know that there was a condition that could not be evaluated. In a larger environment this, however, might generate an overwhelming volume of notifications. See the section Null Values in this document for design patterns for handling null values.

There are situations where not utilizing the default condition is appropriate. Consider a scenario where you are only interested in taking an action if one of three conditions from an option set are updated on a record.

Shared Actions

Use the + symbol below the switch to create a common action for all the switch case conditions.



You can chain switches to get more case options if you need more than 25, but there should be a compelling reason and no better alternative to process the same logic.

Terminate

Terminate allows you to stop the further processing of the flow. When the terminate control is executed, it will cancel any actions in progress, skip any remaining actions, and return the specified status.

Be cautious when using it in conditions or when adding it to a path where there is a parallel path that may still be executing: the terminate action will end the entire flow.

> Note: This action doesn't affect already-completed actions and cannot appear inside a *For each* and *Until* loops, including a sequential loop.

Apply to Each

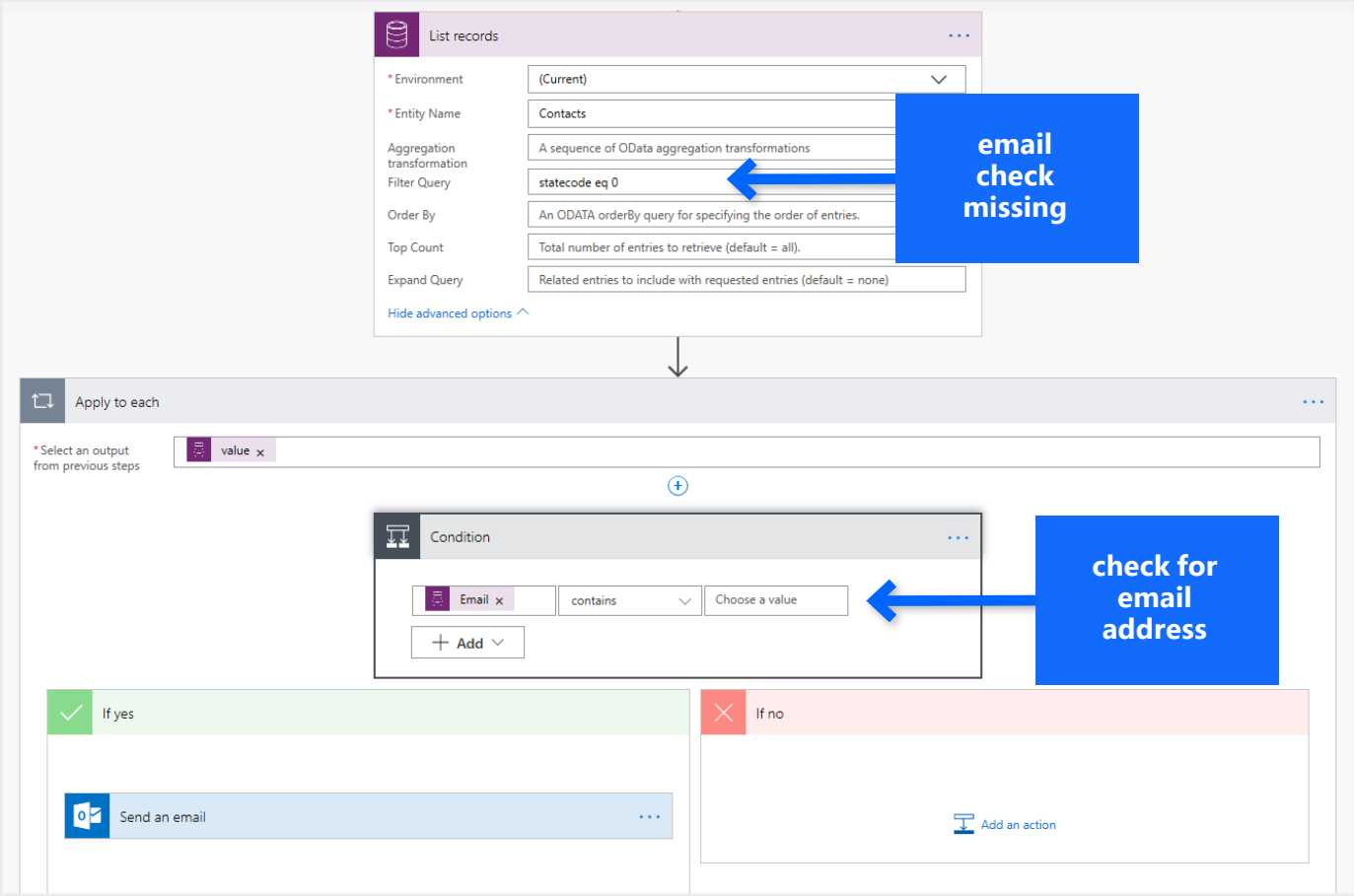
The *Apply to each* control applies a specific (set of) action(s) to all the records processed in the loop. It does not have any ability to filter on an array inside the loop. Therefore, you should filter the records at the data source query so that it only returns the items you need. Alternatively, you can use the *Filter* action to create that subset first and then use the apply for each only on that subset by referencing the *Filter* action output.

When using the CDS *List records* action, the filtering is done server side before the records are returned to flow.

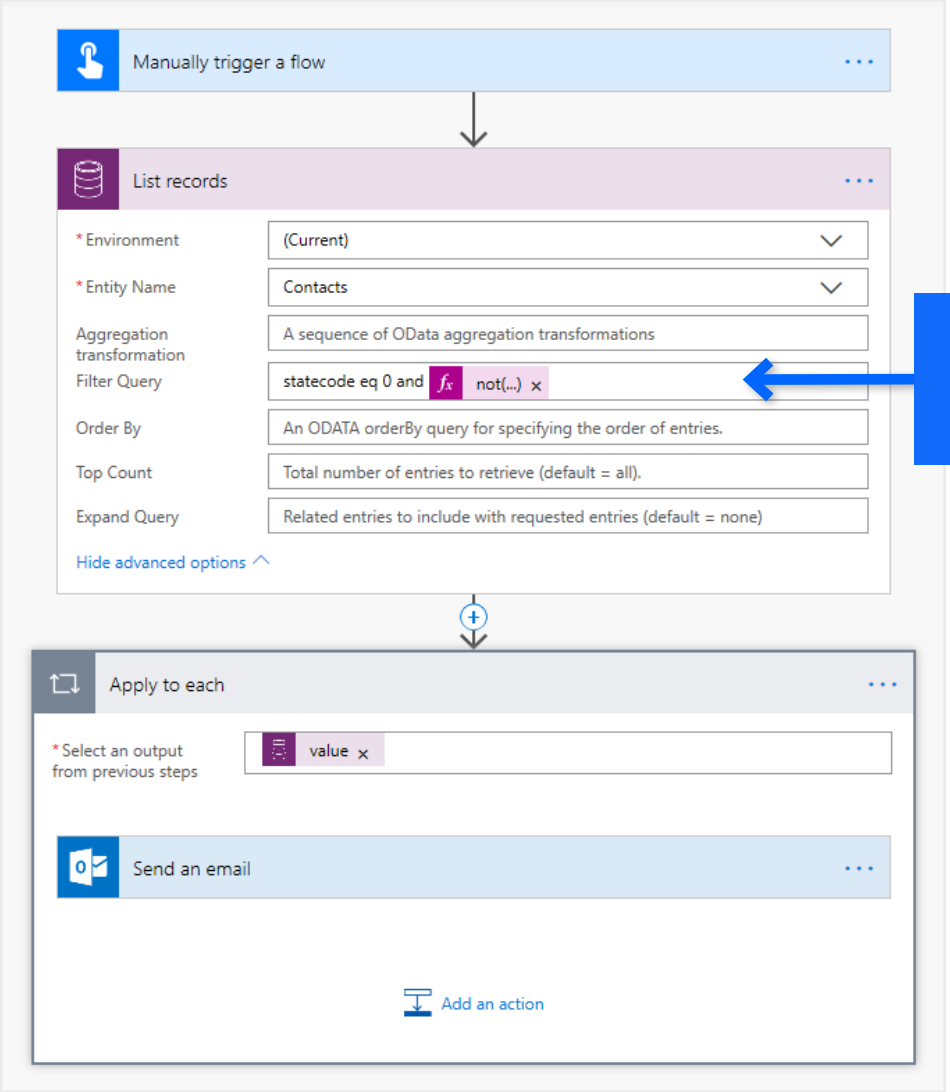
You may find yourself with an *Apply for each* being automatically added to your flow. This occurs if you try to use a dynamic content item that belongs to an array. If this happens and you didn't mean to select an array item, remove the reference to the array item just added, drag your action outside of the *Apply for each*, and then remove the auto-generated *Apply for each* action.

Nested Conditions Inside Apply to each

When using loops, do not include condition checks inside the *Apply to each* for the sole purpose of filtering records. Filtering of records should be always done on the *List records* action when possible. This does not mean that if you need to take different actions based on the value of a record that you should not have a condition check inside of the *Apply to each* loop. But only bring into the loop those records that you really need. The following is not the way it should be done.



Below is the recommended way to do it: include all the filtering in the *List records* action to bring into flow only the records that you need.

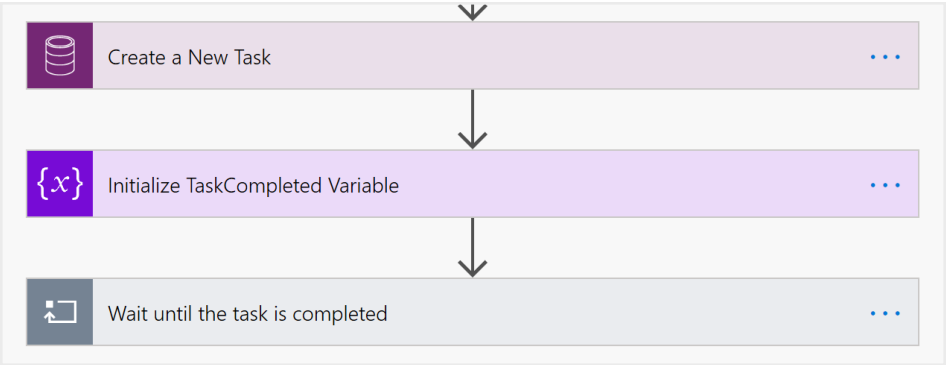


Do Until

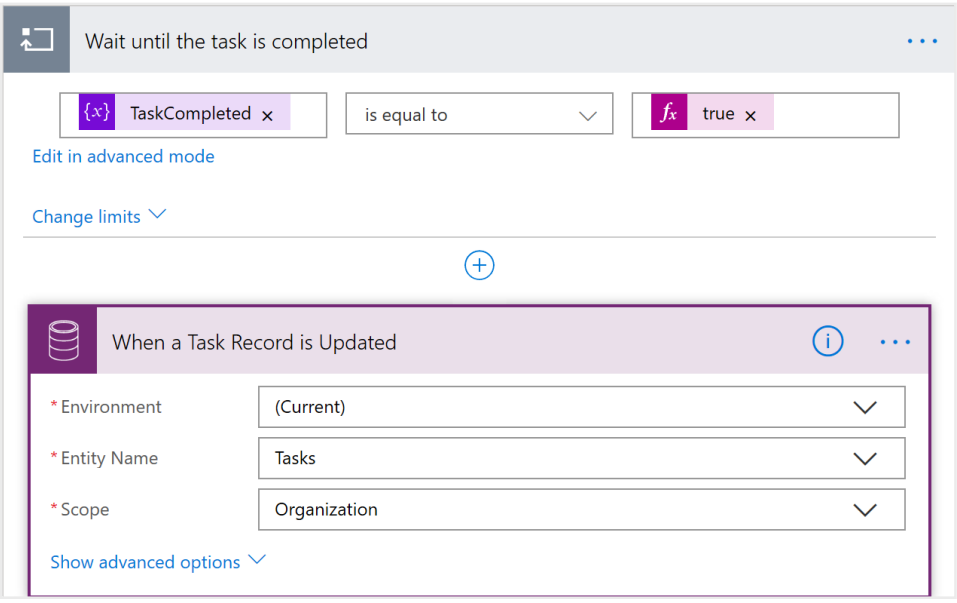
Kicking off a flow when a record is created and then waiting for a change to occur in that record to take an action is a powerful feature. It comes with some overhead. If you use it broadly, you will have a lot of flows running, all waiting for something to happen. The big risk is that these flows may be waiting for something that will never occur.

Here is how to set up the basic *Wait until* "something" flow.

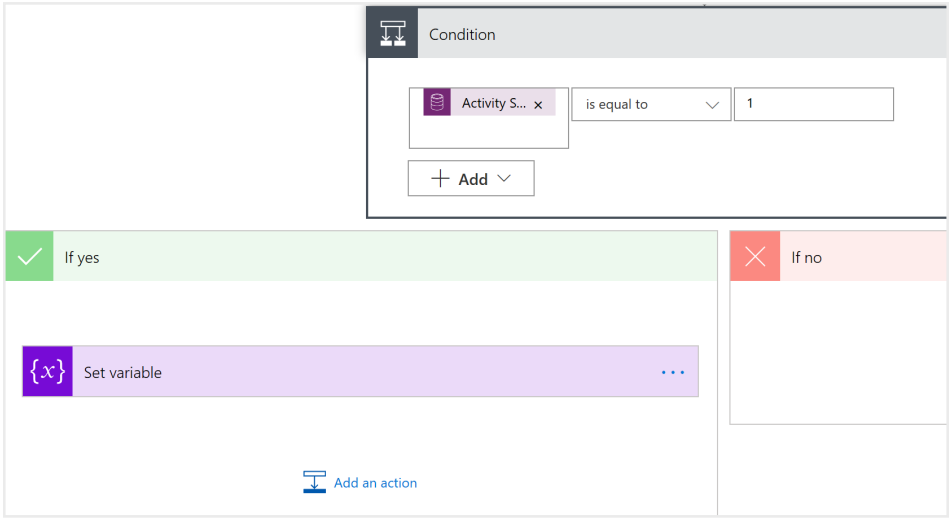
In this design pattern example, there are three components. The first component creates the task, then we set the variable to hold the status of task, and then we do the *Wait until*.



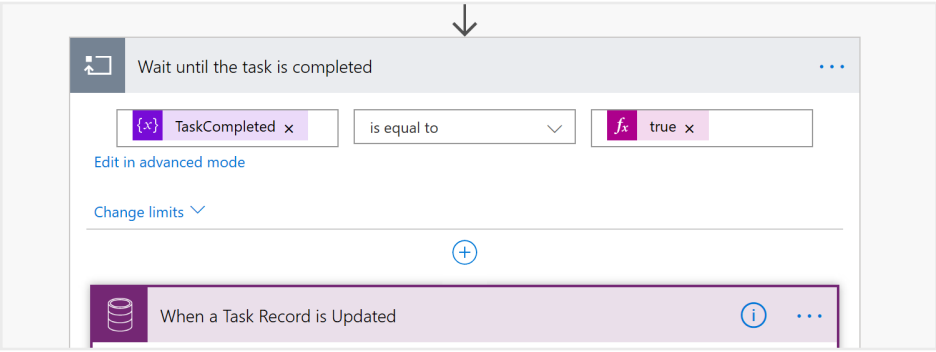
Let's take a look at the waiting process component. We have specified an attribute filter on the field of interest so that the wait only initiates when the filter is met and not for any record update. Otherwise, the condition stays in the waiting mode until the condition is met. When met, the flow gets the original task record that was created by the flow to check if it was completed.



Then there is a condition check. If the statecode is equal to 1, meaning the task has been marked complete, the *Yes* path is traversed updating the *Set variable*, and completing any other actions as specified. If not, the flow keeps running.



The maximum duration of a flow is 30 days. Also note the maximum iteration of a *Do Until* is 5,000. When you put these two constraints together, the highest rate of checking over a 30-day period would be once every 8.6 minutes. That is in most cases a much higher consistent rate of checking for a change over an extended period than most business applications would require. The default setting for the count is 60, and the timeout is set at one hour (PT1H) until you change it in the *Do Until* card.



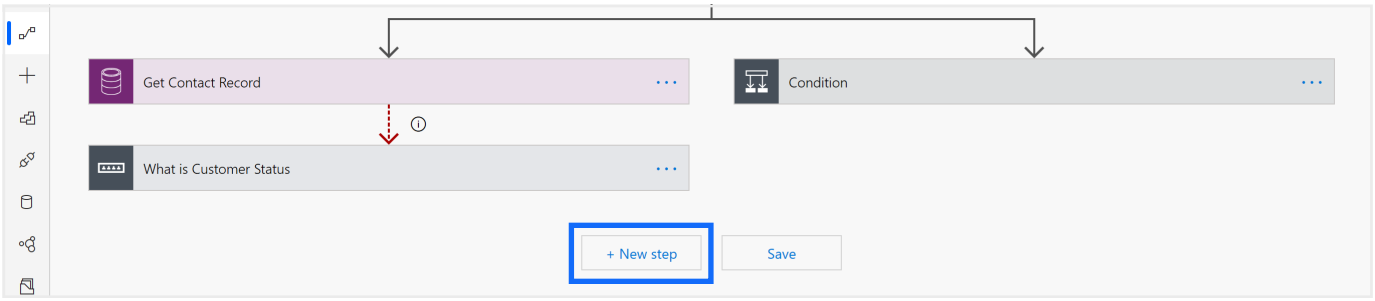
Scope

Scope allows you to group a set of actions that all roll up into one container. This structure is useful when you want to organize actions as a logical group, evaluate that group's status, and perform actions that are based on the scope's status. After all the actions in a scope finish running, the scope also gets its own status. For example, you can use scope when you want to implement exception and error handling.

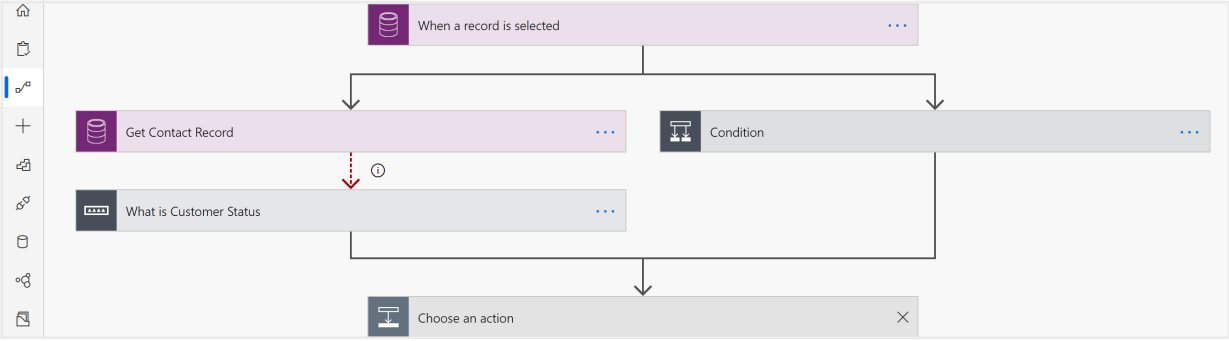
Parallel Branch

After a trigger, you can add an action or a parallel branch. While we won't go into all the uses of a parallel branch in this document, we will illustrate an important principle. The parallel branch doesn't have to stay independent; it can rejoin with one or more of the other branches. This can avoid duplicating other downstream actions. This is a much more efficient way to construct the flow rather than repeating the same logic across several parallel branches.

By selecting the + *New step* (highlighted in red below), which is not directly under each leg, you can bring the branches back to a common point.



After clicking on *New step*, the flow will now look like as follows where *Chose an action* will be processed after the two parallel branches:



Expressions

Flow has a rich expression language that can be used to manipulate data in actions, switches, conditions, or other steps. You can use expressions to do operations such as getting the current time, adding numbers, or replacing a part of string text. There is one key item that you need to prevent from happening when using expressions. Whenever you create an expression, you need to ensure that it doesn't evaluate as null. See the section Null Values in this document for design patterns for handling null values.

Data Operations

Data Operations are actions that allow you to manipulate the data inside of your flow.

Compose

Use the *Compose* action to save yourself from entering identical data multiple times when you are designing your flow. For example, if you need to enter the same array of digits several times while designing your flow, use the *Compose* action to save it in an array and reuse it across your flow. It is also a good method for combining multiple data source fields from one or more previous actions into one single field for future use.

Create CSV Table

This data operation allows you to change a JSON array input into a comma-separated value (CSV) table. The input field accepts either dynamic content from a previous step or a hard-coded value. It is the easiest way to output a list, but that output is in a very plain visual format.

Create HTML Table

To use this action, you will need some HTML markup language experience to construct the HTML snippet for the table in the body of the email to present the data. One easy way to accomplish this, if you are not HTML savvy, is to find a flow template that uses a formatted table and copy that HTML markup to your flow.

> Note: If you plan to send the HTML table via email, remember to select "IsHtml" in an email action that supports this switch. Not all email actions allow HTML input.

Filter Array

An array is a great method to reduce the number of objects in an array to a subset that matches the criteria you provide. For example:

- Perform an action on a single item.
- Make subsets of the list work with a different action.

Don't use *Filter array* as a substitute for doing a proper filter when retrieving a data set from its source.

A practical example is when you want to take different types of actions on different records in the same table. After you put records into an array, you can then use the *Filter array* to filter them by a certain column value. For each filtered result, you then invoke an action that is specific for that subset. This is the preferred approach over executing multiple *List records* actions to retrieve each of the different subsets from the data source. However, this is not an excuse for going too general with the *List records* action and retrieving all the records. Don't bring more records into your flow than you need for your flow to run.

Join

Use the *Join* action to delimit an array with a separator of your choice. This is an easy way to get the data received in previous actions formatted to meet the input requirements for the field in the next action.

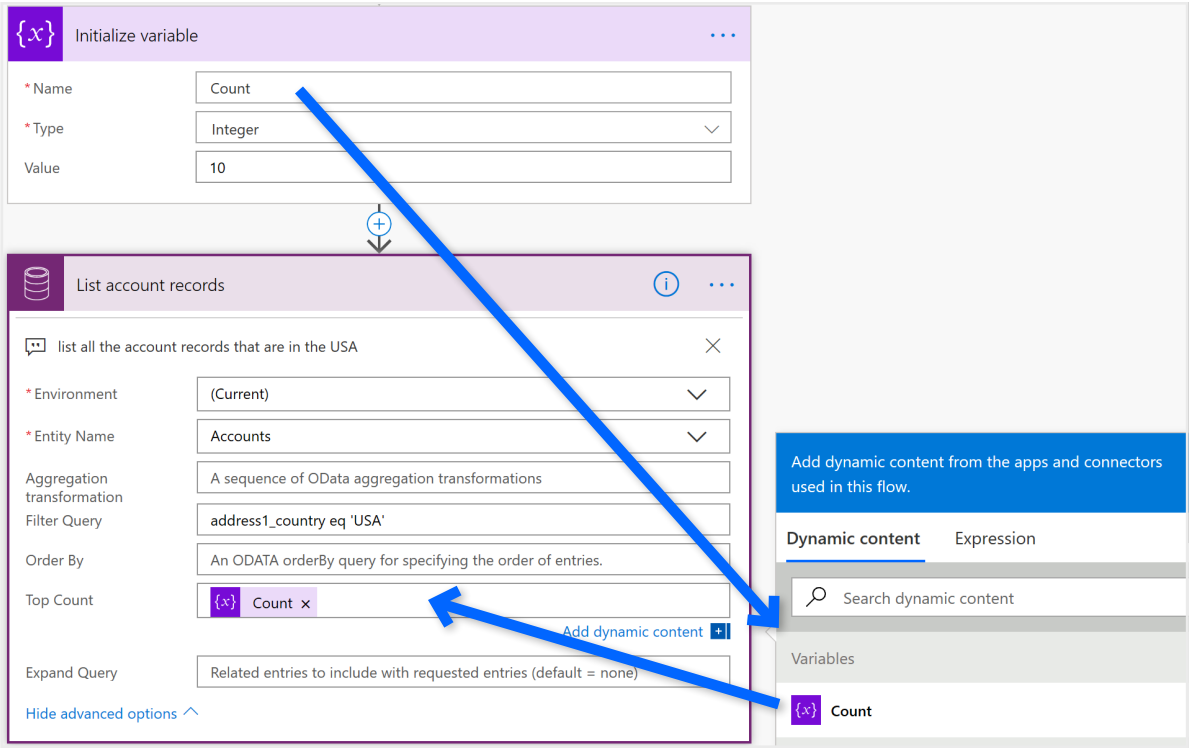
Parse JSON

The *Parse JSON* action is useful when working with external services that return JSON as part of their connector's actions. Using the *Parse JSON* action, you can consume the JSON and make it so that the properties of the data are available in the *Dynamic content* pane for downstream steps in the flow. When you configure the *Parse JSON* action, you provide a sample of the JSON data and the designer will infer the JSON schema from that sample.

Variables

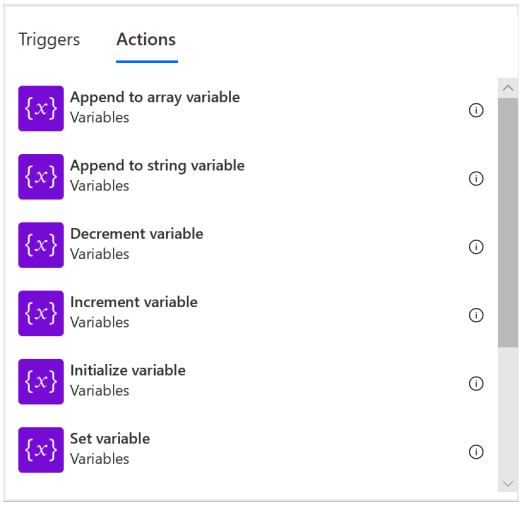
Variables are like a scratch pad. They can be set, incremented, or decremented depending on their data type.

Variables are found in the *Dynamic content* panel after you click in a field to set its value.



Variables can be helpful to keep track of information as you progress through your flow. Look for opportunities to use them to simplify your flow and to streamline condition checking and value setting.

Avoid using variables when they aren't needed. They should be used only when you have multiple steps updating the variable or when you are trying to accumulate some sort of value in a *Do until* or a *For each* loop. Considering using a constant with the Compose action instead, which has less overhead on the platform and correspondingly less complexity.



A variable must be initialized at the top level in your flow, and it can't be initialized inside of a loop, condition, or a scope. Variables support the following types: Boolean, integer, float, string, object or array.

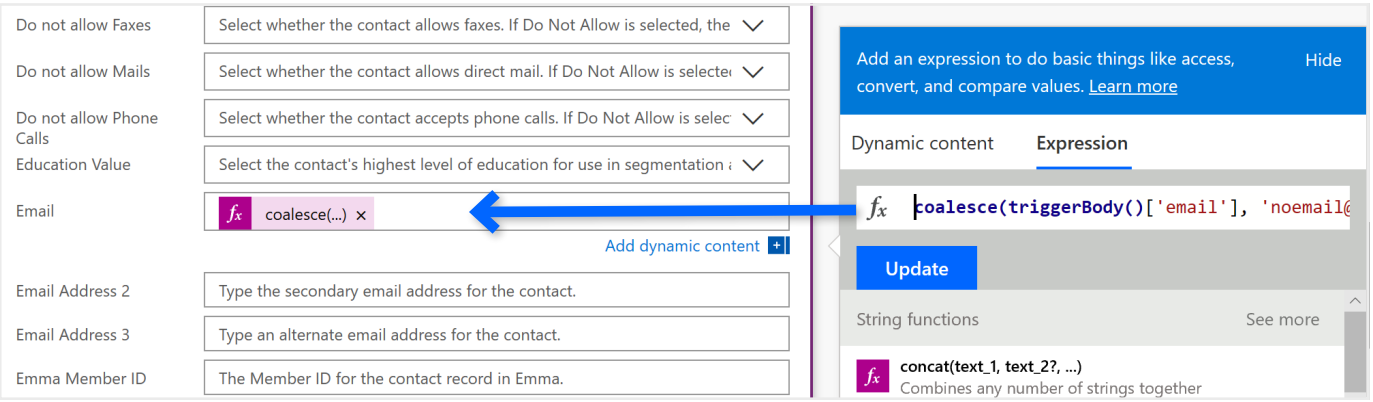
What are some of the great ways to use variables in your flows?

- Counters: you can increment or decrement variables.
- Append: use them to concatenate in arrays or with strings.
- Set State: set the status of a record, proceed downstream in your flow, and then check the variable again to see if the state has changed.
- Simplify expressions: no need to keep checking object?['p1']?['p2']?[p3'], just store the result in a variable.

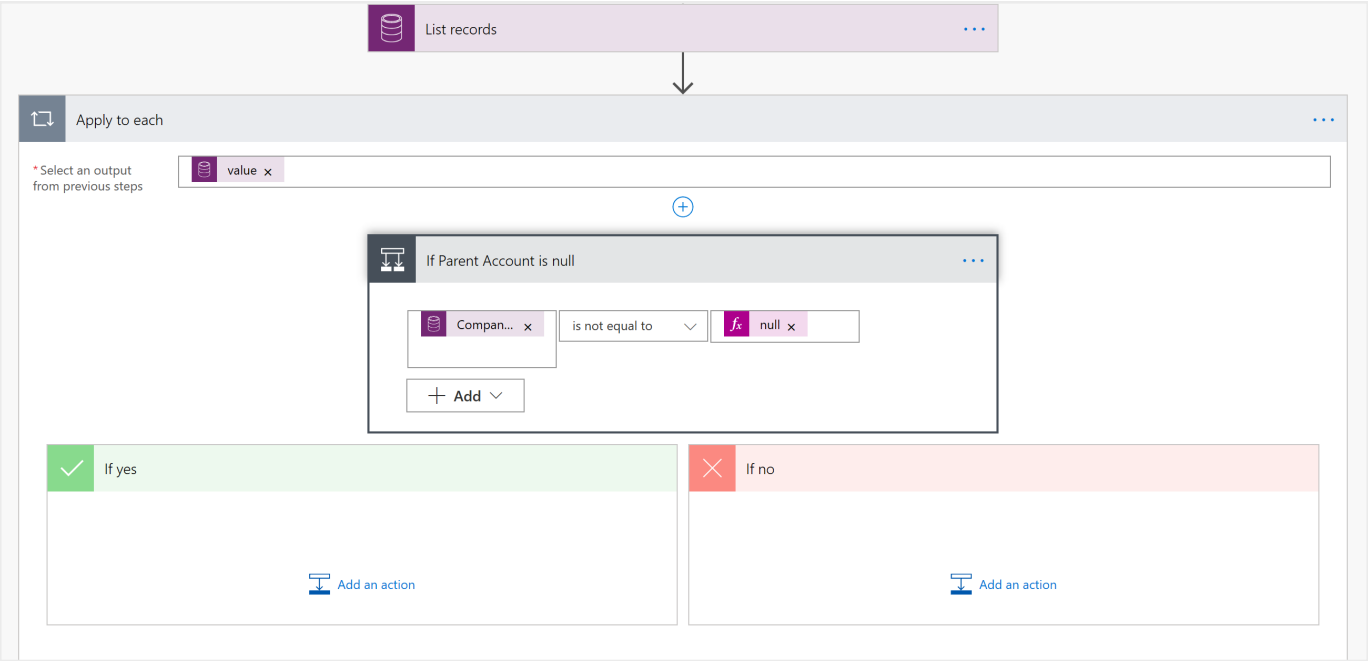
Null Values

The handling of null values in flow requires some special attention. If you reference a field or a table value that is null, the flow will fail. There are two methods for preventing this from happening.

1. Use the *Coalesce* function, which evaluates its arguments in order and returns the first value that isn't null. For example, to handle a possible null value from a trigger, you can use this expression: @coalesce(trigger().outputs?.body?.<someProperty>, '<property-default-value>'). A practical example of this pattern might be where you insert a placeholder email address when one is expected but none is provided: coalesce(triggerBody()?['email'], 'noemail@noemail.com'). The following is an example of how to write and insert the coalesce expression into the flow.



2. Alternatively, you could also use a condition check for the field being null; however, this approach could quickly increase the complexity of the flow. However, this would be superior to using the coalesce approach if your business practice doesn't support inserting a placeholder value. In the example below, each record is being checked in the condition embedded in the *Apply to each* loop and if the *Company lookup* is not null, then traverse the *Yes* path, but it is null, then go the *No* path.



Note: A *Null* is when the data value is unspecified or nonexistent for any reason. It should not be confused with an empty string, which is a defined value that contains no characters.

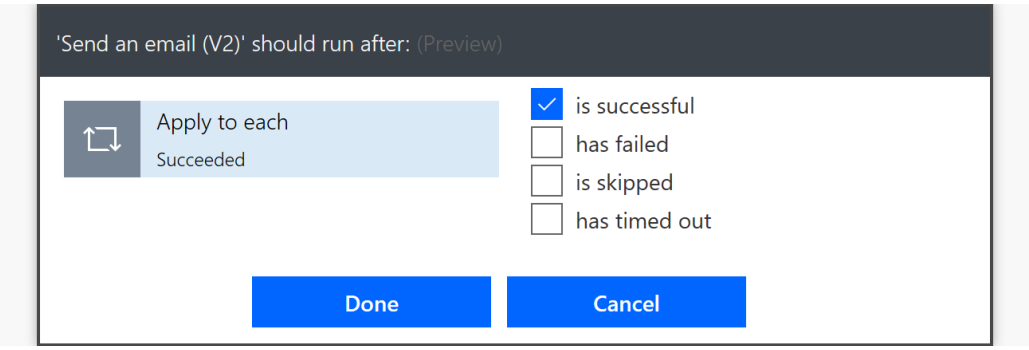
Error Handling

The reality is that sometimes things don't run exactly as you expected and that is why you need to build in error handling.

Microsoft Flow provides a *Configure run after* option that can be used specifically for processing various outcomes of the action that you configure it for. When you use it for error handling, you can define any number of actions to run after the failure of an action.

The options are:

- Is successful: Use this in a scenario when you want to explicitly call out another parallel action when the step is successful.
- Has failed: Use for any type of failure except timeout. It is best used for hard failures such as where the user didn't have permission to do the action on the record or the field value required by flow was invalid.
- Is skipped: Use when the prior action was not executed due to its own *Configure run after* setting.
- Has timed out: You can use this for approvals that do not complete in the time allotted; then use this option to branch to an escalation process. It can also be used when the action to a service times out.



Special Types of Flows

There are some types of flows that have a component within them that requires a human response before it can proceed.

Approval flows are such an example. They require a person (or persons) to approve a document or a process before moving on to the next action. For example, you can create document approval flows that approve invoices, work orders, or sales quotations. You can also create process approval flows that approve vacation requests, overtime work, or travel plans.

Pending approval requests that have not been responded to are automatically cleaned up when the flow times out, fails, or is canceled. Currently there is no method for bulk-deleting approvals that have been sent. They will expire on their own at the end of 30 days. If you are using V2 approvals, the admin has to clean them up in CDS; they no longer expire on their own.

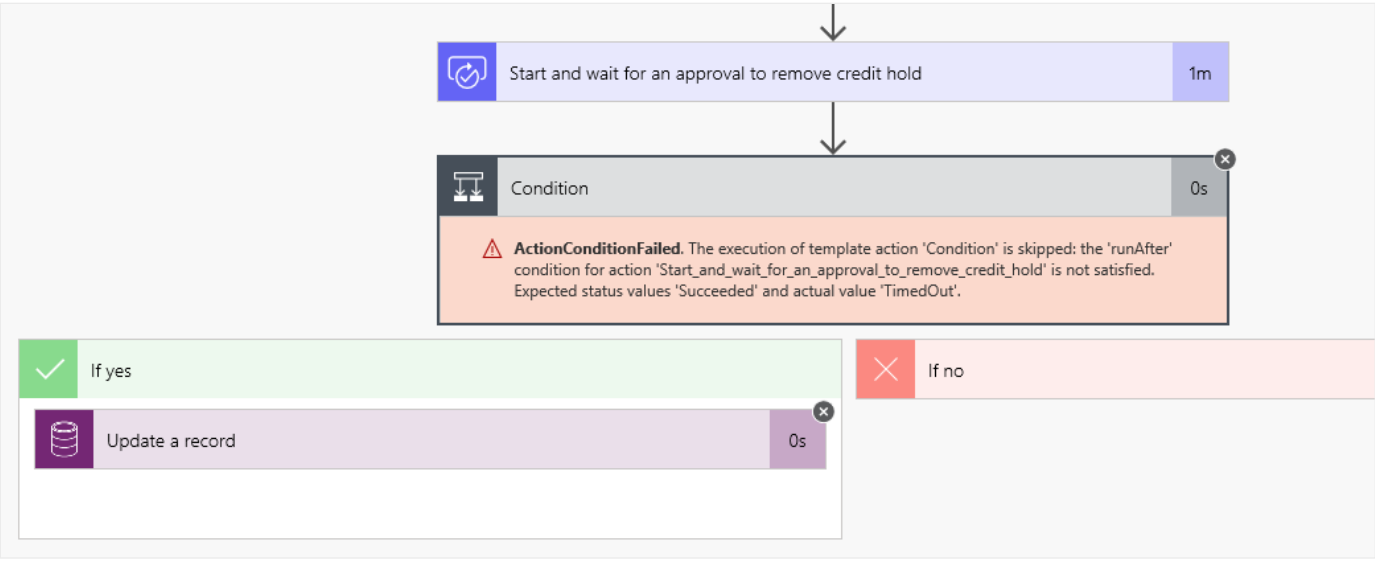
Approval flows can be routed for different types of approvals: single, sequential, parallel, or everyone must approve. How different businesses handle approvals, or the lack of responses thereof, varies. Consequently, we will illustrate a baseline design pattern that you can extend and customize to fit your organization's specific business requirements for these scenarios:

- What if the approver doesn't respond?
- What if the item to be approved no longer exists?
- What if the flow is canceled?

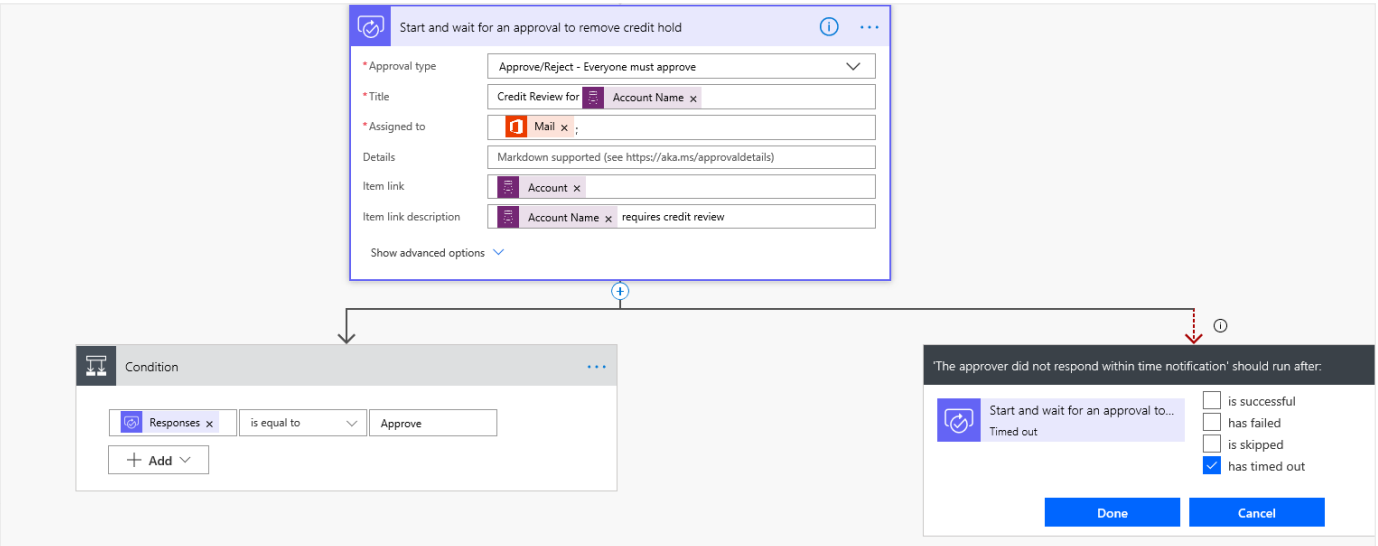
What if the approver doesn't respond?

When flow sends out an approval request and the approver doesn't respond, it will either timeout when it hits the maximum time allowed for a flow of 30 days or when it hits the *Timed out* in the settings for the approval action.

This is what a flow failure looks like when the condition can't be satisfied due to a timeout.



The corrective action is to use *Configure run after* with a parallel action.



What if the item to be approved no longer exists?

You may need to consider the possibility that the original requirement for the request is no longer necessary. For example, the request for time off has been canceled by the submitter. A way to approach this is to add a parallel branch with a *Do until* loop that keeps checking the state of the record that triggered the approval. If the approval is no longer needed, cancel the flow with a *Terminate* action in that parallel branch.

What if the flow is canceled?

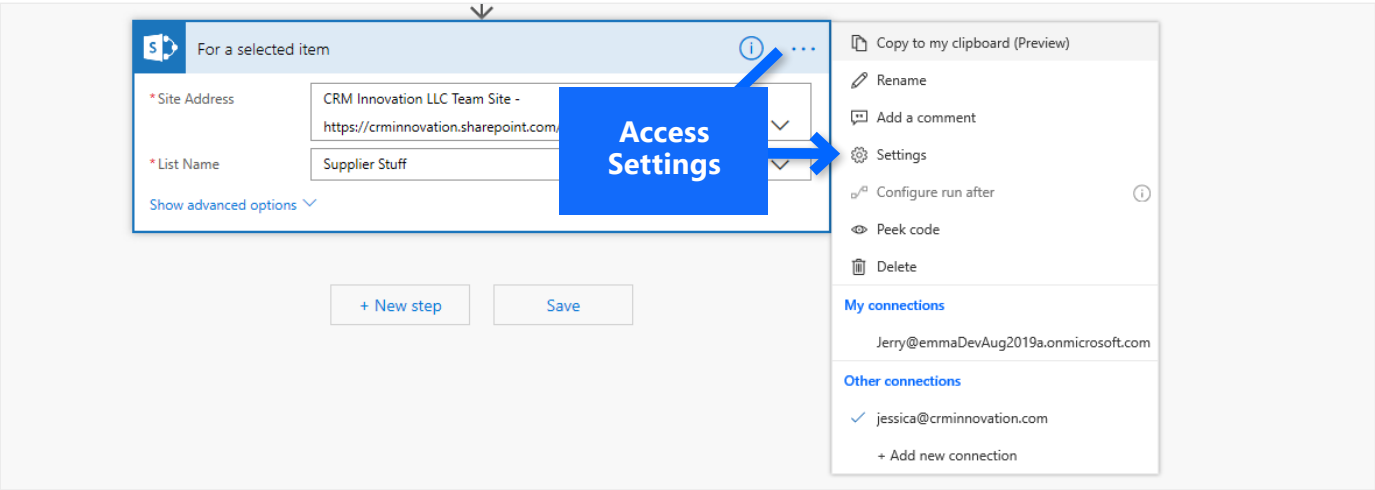
If the flow is canceled, the sent approval email will not be deleted from the recipient's inbox. The approval requests are only removed from the mobile app and the flow portal. If you are using V2 approvals, they have to be removed manually in CDS.

> Note: Pending approval requests are automatically cleaned up when the flow times out, fails, or is canceled. Currently there is no method for bulk-deleting approvals that have been sent. They will expire on their own at the end of 30 days.

Advanced Settings

Microsoft Flow also provides more advanced control over how triggers and actions run through the *Settings* property. The settings vary depending on the type of trigger or action. Going with the default is a reasonable approach and generally will be satisfactory. However, being aware of the settings options may allow you to make the flow more efficient and resilient. It doesn't hurt to be familiar with these settings and leverage them when appropriate.

You can access these settings by selecting the … menu at the top-right of a trigger or action and then choosing *Settings* from the menu:

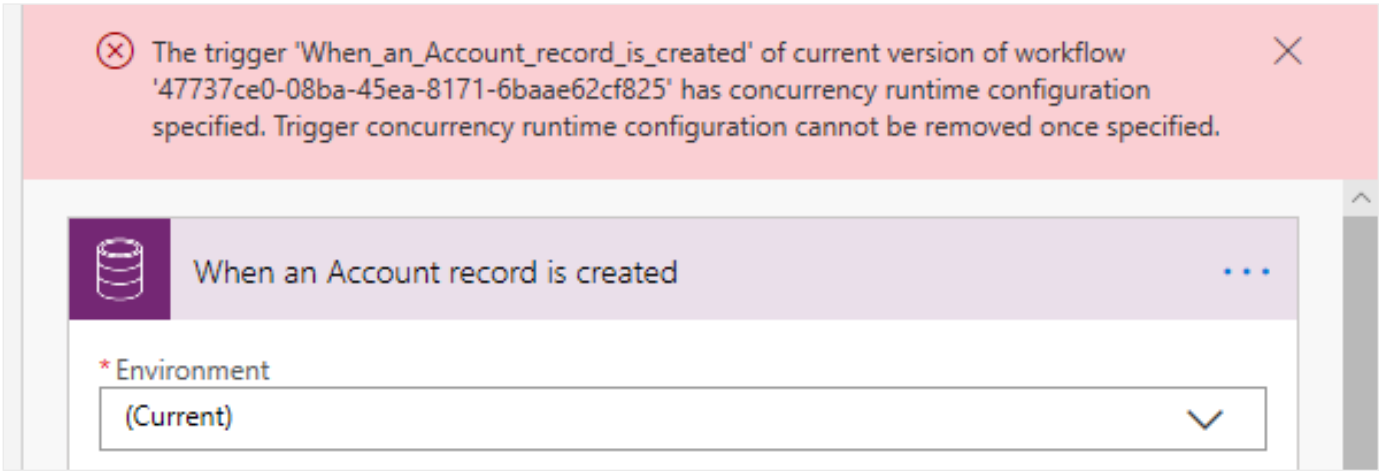


Here is quick review of the different settings and some examples on how they can be used to improve your flow.

- Split On: If your trigger returns an array for flow to process, a *For each* loop might sometimes take too long to process all array items. Instead, you can use the Split On property in your trigger to debatch the array. Debatching splits up the array items and starts a new logic app instance that runs for each of those array items.
- Retry Policy: For the most basic exception and error handling, you can use a retry policy in any action or trigger that supports it. A retry policy specifies whether and how the action or trigger retries a request when the original request times out or fails. Any web request that results in a 408, 429, or 5xx response code would be supported by the retry policy. The value you choose for this setting should reflect your business requirements.

- Pagination: This enables you to handle more records than are returned in a single call from a service. For example, by default CDS will only returns 512 records, and SharePoint returns 100. However, you may need more records to be processed in your flow. By enabling pagination, the flow engine will continue to call the service until it has all the items or hits the limit of the pagination. This limit can be set from zero up to 100,000 for CDS and SQL but may also have limitations imposed by the connector. The limit you configure will be rounded up to the next 512. As such, in the case of CDS, the maximum number of records that can be retrieved is 100,352 or, alternatively, the limit for the service that is being called if it is lower.
- Concurrency Control: Set the Degree of parallelism to 1 when you need to have a flow that processes its records in order. If that is not a requirement, and you would like to try to decrease the overall time that the flow runs, you can increase the value to 50. But be advised, while Microsoft Flow may support it, the services that you are connecting to may not. It is quite possible that you exceed the allowed API call rate and therefore generate a flow error, causing the process to fail.

Some settings like concurrency control, which are available on some triggers, only allow you to set them once and not revert to the default. Below is the error message you get when you try to return to the default after configuring concurrency. The only fix at this point is to rebuild the flow.



- Timeout: There are some operations that may run for a very long time. Use this setting to change how long the action runs before it "times out." In this case, it is also recommended to set the *Configure run after* setting to define the actions that should be executed when the timeout occurs.
- *Apply to each* Concurrency Control: By default the *For each* loop will execute sequentially. You can alternatively set its concurrency to a range from 1 to 50. Use a value of 1 when you have a business requirement that depends on records being updated one at a time. Use a higher value when there is no dependency of the order of execution and you have large volumes of records to process.

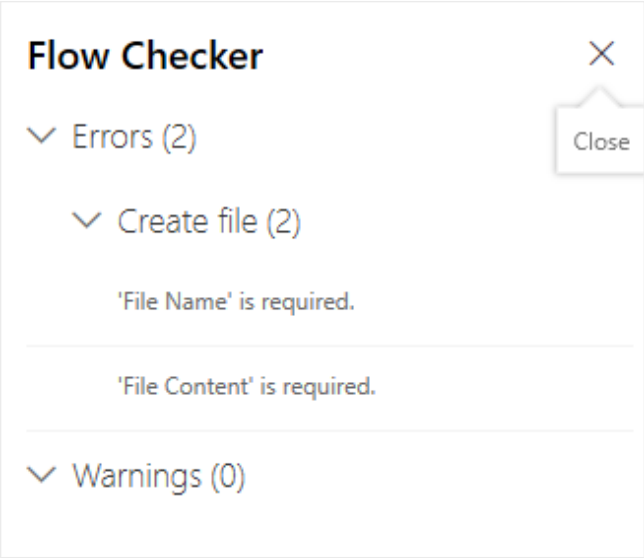
Flow Checker

The flow checker is a feature designed to promote higher-quality flows by helping you to follow best practices. The checker helps provide an answer to a question like: Which areas of my flow implementation pose a performance or reliability risk? While running the checker should be a regular step in your testing process, it is additionally well served when troubleshooting flows that aren't running as expected or flows that are periodically failing.

For each issue identified, the flow checker points to specific occurrences within the flow where improvements may be required. And you learn how to implement these improvements by following the detailed guidance.

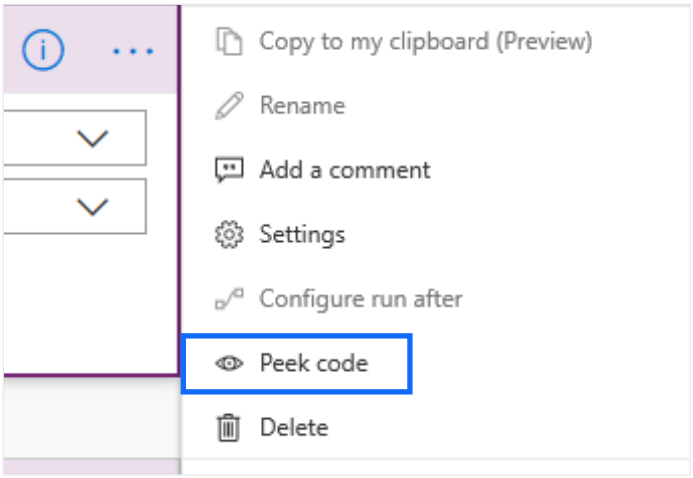
The flow checker will display two categories of information:

- **Error** is something that is guaranteed to fail. The most common example is omission of a field that's required.
- **Warning** is something that may or may not cause the flow to behave in an unexpected way. An example is infinite loops.



Peek Code

In some cases, you may want to take look behind the UI to see the actual code that is running. This is particularly helpful for troubleshooting a large action with many different inputs. To get a better understanding of what exactly is sent to the connector from the flow use the *Peek code* option in the ellipsis (...) menu of triggers and actions in the flow designer.



When you select this option, you will see the full JSON representation of the action. This includes all the inputs to the action, such as the text you entered directly and expressions used. For example, you can select expressions here and paste them into the dynamic content expression editor to verify the data you expect is present in the flow.

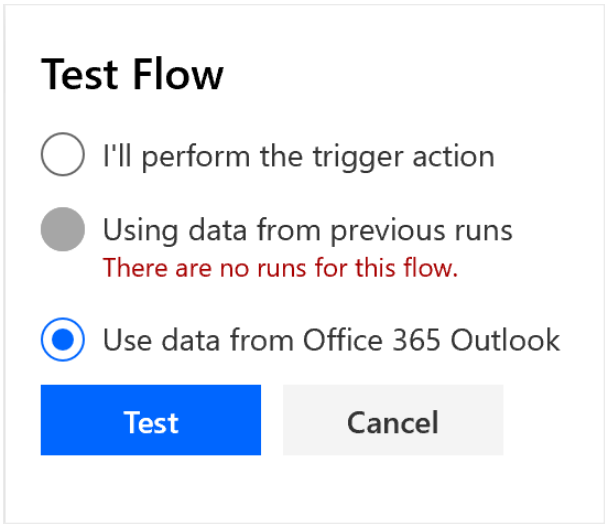
Testing

An important task in building a flow is testing and validating results. Microsoft Flow has built-in features that make it easy to interactively test flows. This means that a common pattern for flow modifications becomes: Modify -> Test - > Review Results -> Repeat. We will review the three types of built-in testing methods and why or why not to select that method:

- Real example data
- Previous run data
- On demand

Real Example Data

Sometimes you need real data for testing, and you don't want to bother with creating realistic dummy data. There is nothing better than hitting a data source that has real data rather than made-up information during the development process. When you are ready to test your flow, select the test button in the command bar at the top of the screen. In the test pane, there are three options for testing your flow. The one you want for this purpose is the third one that may appear after the standard two. For example, this is how the window appears when using Office 365 Outlook as the trigger.

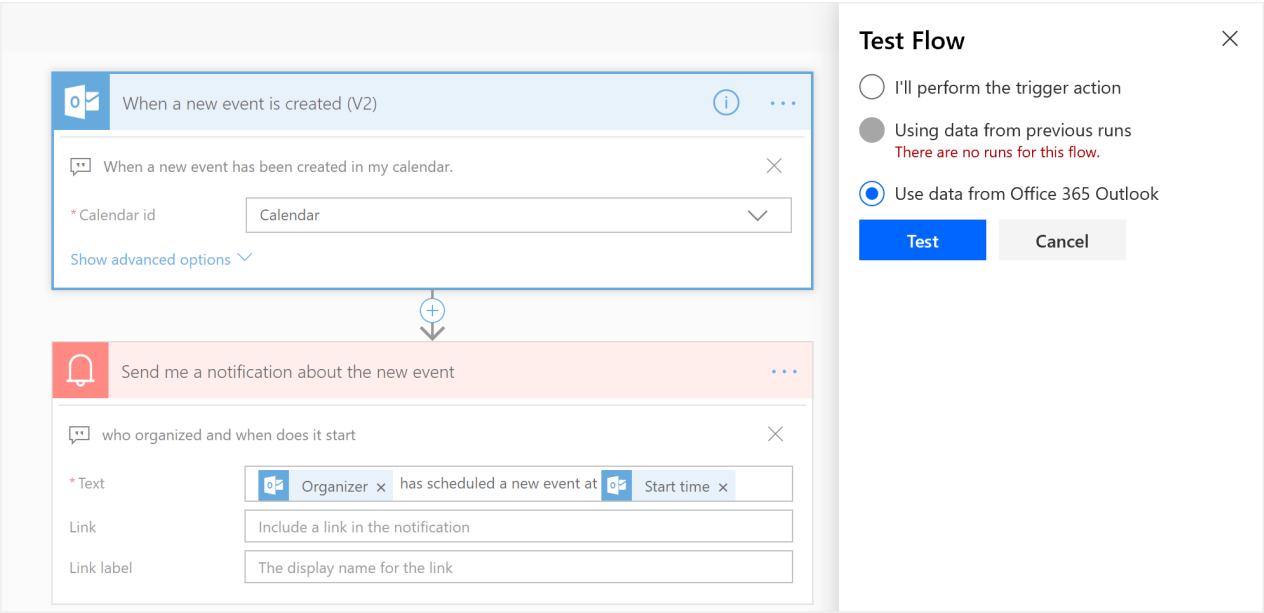


As of the publishing of this document only the following four connector triggers have this feature available:

- Office 365 Outlook
- SQL Server
- Gmail
- Outlook.com

For example, if you are you are developing a flow using the Office 365 *When a new event is created* trigger, you can use this testing process to pull data from your existing calendar. This bypasses the need to create a test event in your calendar. When using, for example a trigger like *When a new event is created* or *When a new email arrives*, Microsoft Flow will automatically get the most recent record of that type. In the case of the SQL Server trigger, it will find a row in SQL that matches the query you pass.

Here is what the flow looks like with the test flow dialog window open.

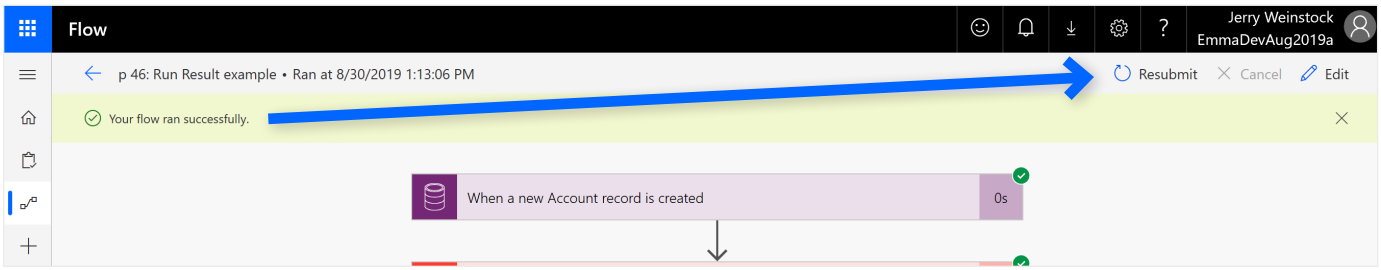


Previous Run Data

When you are editing a flow that you have run in the past, it can be useful to rerun the flow with the trigger data from that previous run. This gives you a way to verify that your flow behaves as expected with the same test data.

There are two ways to do a test with previous run data:

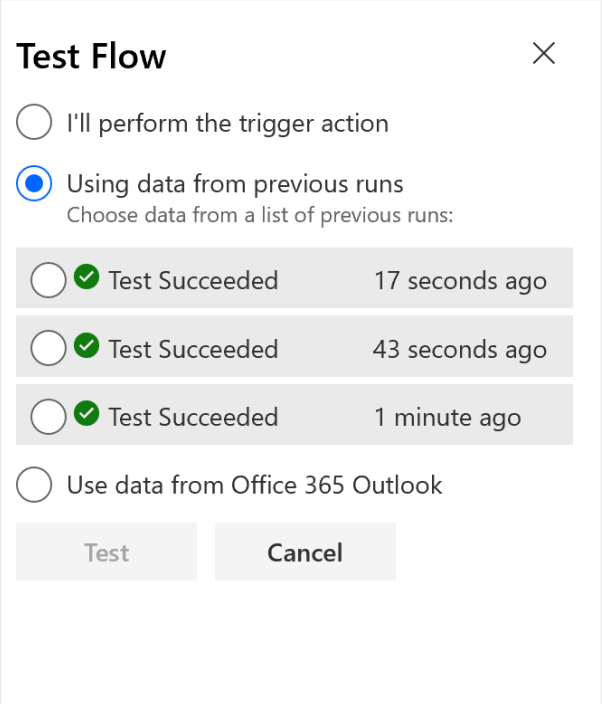
1. Go to the *Run history* view and select Resubmit. The upside of this method allows you to select from any of the previous runs from a longer list presented in the test flow approach.



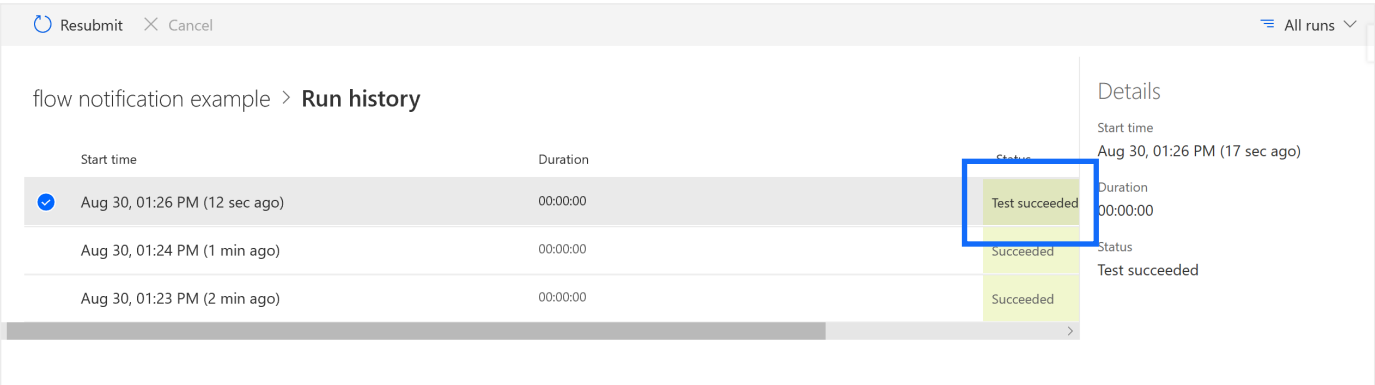
2. Use the *Test flow* button directly in the designer.

When you select *Test flow*, you will see a new pane with options:

- Perform the trigger action yourself. The action depends on what the trigger is. For example, for a button trigger you'll run the flow button, for an email trigger you'll send yourself an email, or for a file trigger you'll upload a file to SharePoint Online.
- Use data from previous runs. Here, if your flow has run before, you'll get a list of the most recent runs and be able to pick one to test your flow.



Once you run the flow, you will immediately see the details of the flow running and you will be able to watch each of the steps execute. Note that runs started via either one of these approaches are marked as test in the *Run history* view so you can tell them apart from flow runs that were triggered by their respective events.



It is important to keep in mind that when you use a prior run for testing, it is similar to the *Resubmit* option. It simply starts a new flow with the same initiation data as the prior run. This can affect your testing depending on how your flow is constructed. For example, if your flow creates a record if it doesn't exist and then takes path A for the rest of its processing, it won't run the same if resubmitted.

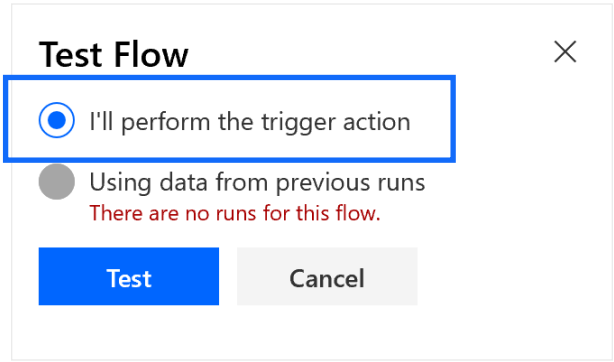
The second test would not be a repeat because the record already exists and then path B in the flow would be executed. In this case, performing the tests with new data would be a better option to ensure consistent results.

Where testing with data from prior runs is most helpful is when a failure occurs, you can keep rerunning the failed record until you have successfully fixed the flow logic.

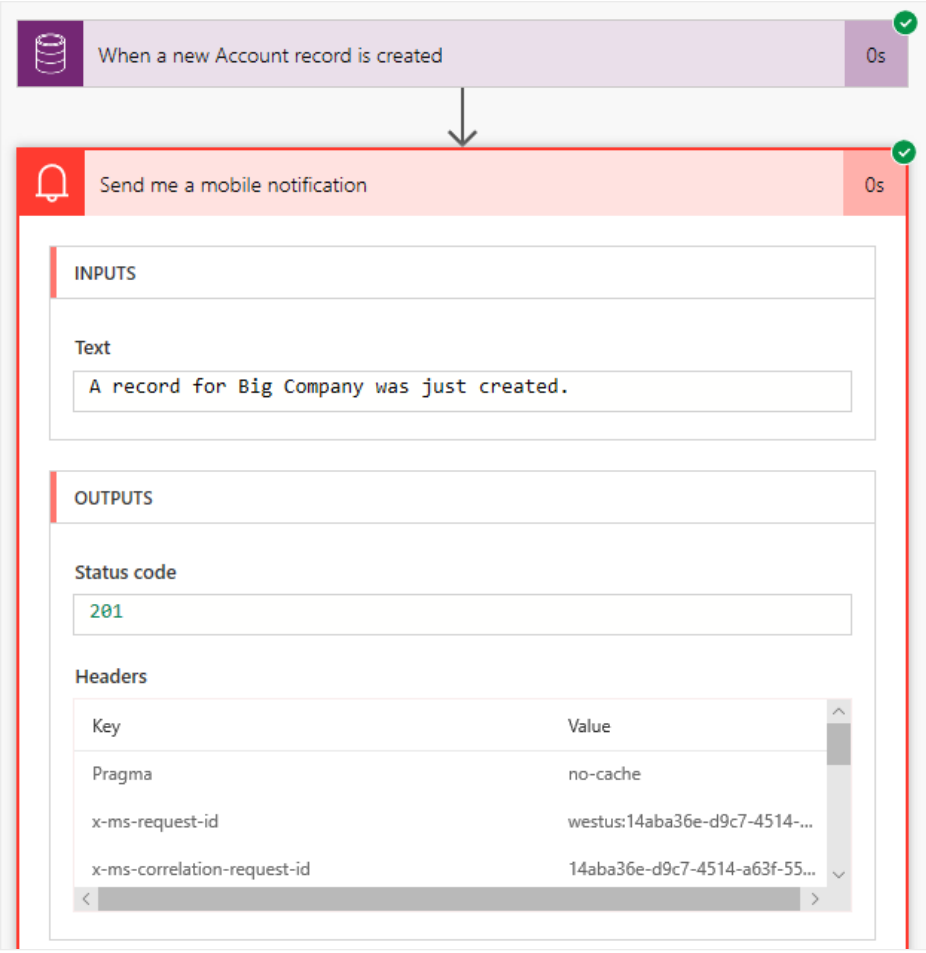
> Note: When you are testing your flows and use prior runs, be aware that the filtering of scope in the CDS trigger is not performed. That is only done on the actual check, not a rerun of a prior run, so you can end up testing with a record that wouldn't otherwise qualify. The record that is being acted on may be in a state that prevents the flow from running.

On Demand

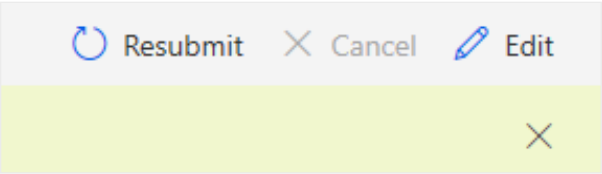
If neither of the two preceding approaches work for you, then using the *I'll perform the trigger* option is the way to go. When you invoke this method, it puts flow into test mode. To see the flow work, simply perform the starting action. The flow will respond almost immediately or at the minimum measurably sooner than it would if it was just running and doing its normal listening.



After the flow runs, you will see the results of the run directly in the web UI. You can then inspect the outputs of the triggers and actions and, if required, make changes to the flow.

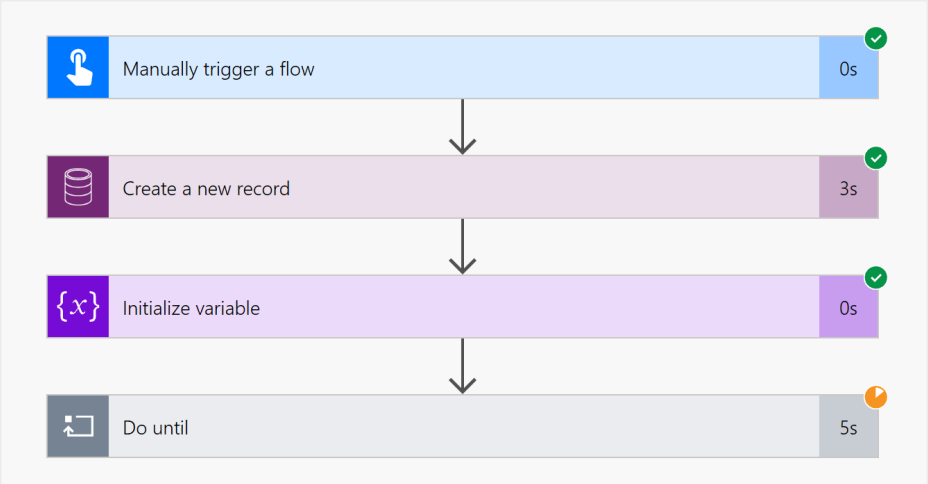


From this point, you can resubmit the flow to run against the same data or return to edit mode, make changes, and test again.



Testing Tip

When the flow runs through any of the testing options described previously, you can watch the execution and preview the results as they occur.



Testing from within the UI will help you build better flows,thanks to the real-time insights provided on the steps. The green check in the upper right corner of the step indicates the step successfully completed and the orange 7/8 circle indicates that the step is still executing. You can click on any of the completed steps to expand them to see the data that was used as input as well as what was produced as output from that step.

Deployment

An integral part of deploying a software application to production is the consideration of security. Within the context of Microsoft Flow, the security considerations relate to who owns the flow, what their exclusive rights are, who else can make changes to the flow, and whose credentials will be utilized when the flow runs. Within this section we address some of these considerations and provide you with a deeper understanding of each of the options. You can decide which of these choices best fits your organizational requirements and the specific flows in question.

Flows that are part of enterprise solutions are supported in the same way workflows are supported in CDS. Anything that is possible to model in the CDS permissions stack will now also work for Microsoft Flow. We cover this in more detail in the application life-cycle management section that follows.

However, when the flow is not part of CDS solution management, there are only two options:

1. Use the sharing option when you want to allow other users to make changes to the shared flow.
2. Use the *send a copy* function when you want to allow users to take a copy of the flow and create their own personalized version of it without affecting anyone else, including yourself.

Sharing Options

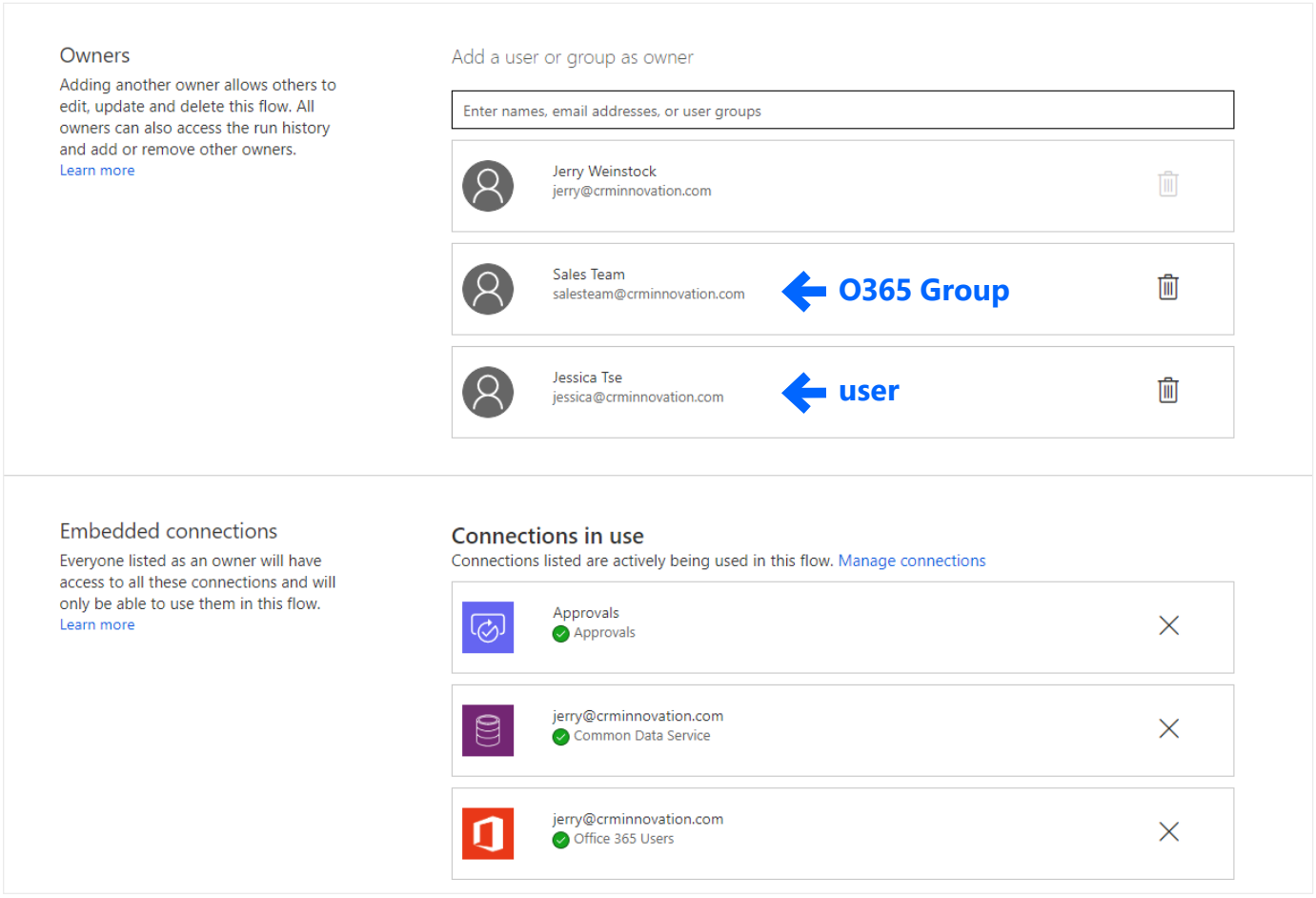
If you plan on sharing a personal productivity flow you created and personally use, you will need to consider how you will grant access to others. Let's take a closer look at the options.

When you share a flow, you add all the people you share it with as owners of that flow. The flow is now a team flow. All owners can view, update, and delete the flow. They also all have access to the embedded connections that will continue to run under your credentials. However, they will only be able to use those within the shared flow. They will not be able to create their own flow using the connections that use "your" connector credentials that came with the shared flow. Finally, all owners can also access the *Run history* and remove other owners, except you. If you want to prevent other users from viewing previous outputs of the flow, then make a copy of the flow using *Save as* and then share the copy.

This is an excellent choice if you want them to assist you in managing the flow but without allowing them to utilize your credentials to create brand-new flows.

> TIP: Use this approach with a significant degree of discretion and with a full understanding of all the broad rights you have granted to them.

Here is a view of the screen where you can add a user or a group as an additional owner. Sharing with an Office 365 group is more scalable than adding individual users for larger organizations.



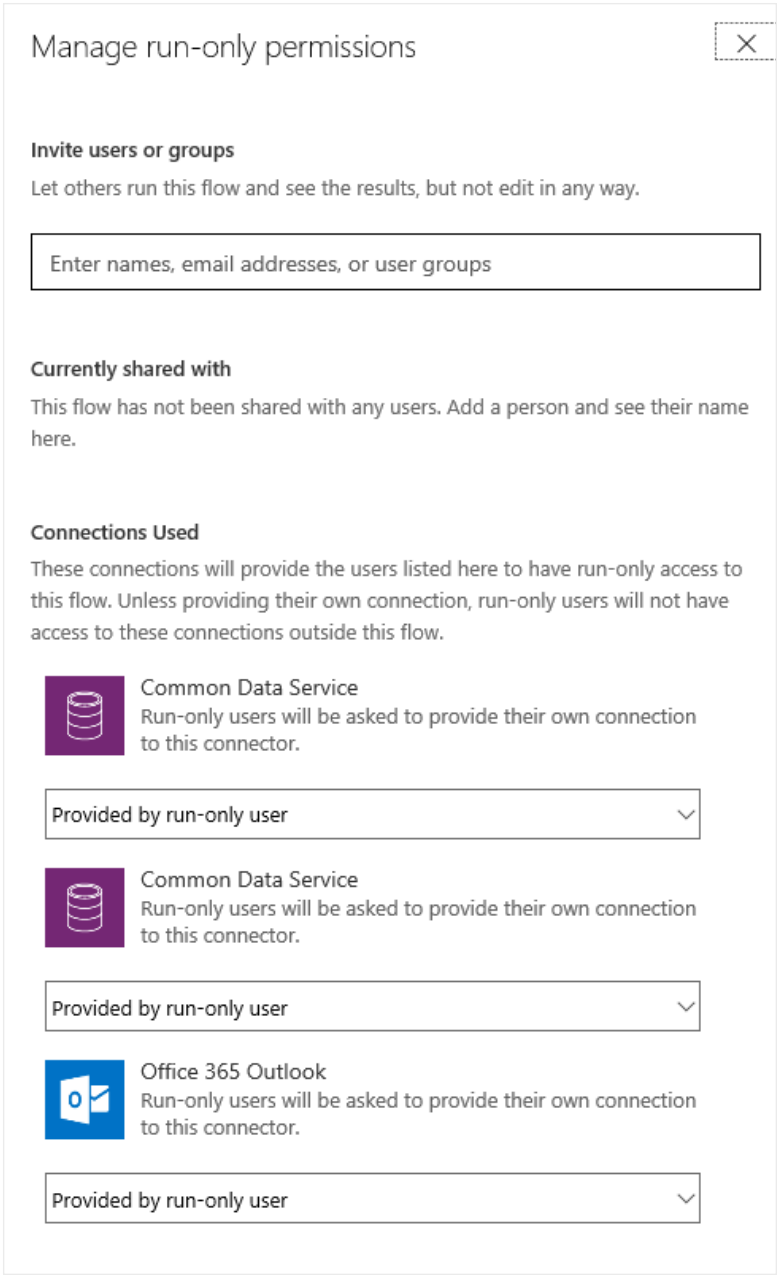
Note: When you remove an owner whose credentials are used to access Microsoft Flow services, you must update the credentials for those connections so that the flow continues to run properly.

You can add SharePoint lists as co-owners to a flow so that everyone who has edit access to the list automatically gets edit access to the flow. Once the flow is shared, you can simply distribute a link to it. See https://docs.microsoft.com/business-applications-release-notes/april18/microsoft-flow/sharepoint-lists-libraries.

If sharing a flow with users as described above gives them more rights than appropriate, you can instead share them as run-only using the flow mobile app or using the *Manage run-only permissions* in the Microsoft Flow portal.

In the view below, you see one of the additional options when sharing with users with run-only permissions: you can selectively determine whether they need to use their own credentials for the connection(s) or whether, will use yours as the originator.

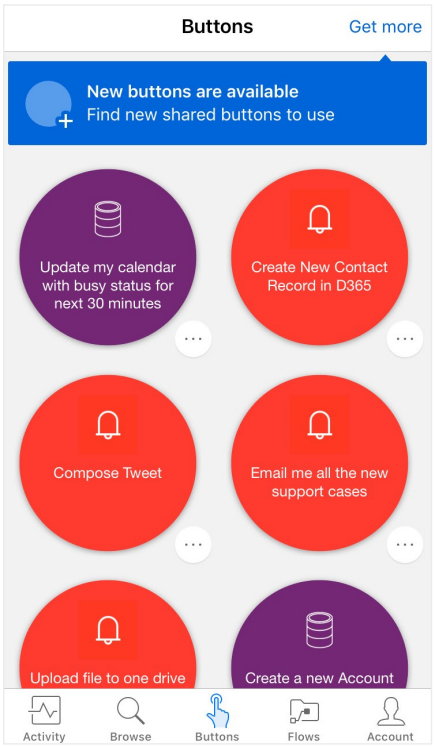
This is best used when you want to share the flow with other users but require them to utilize their own credentials for making the service connections, while not allowing them to edit the flow.

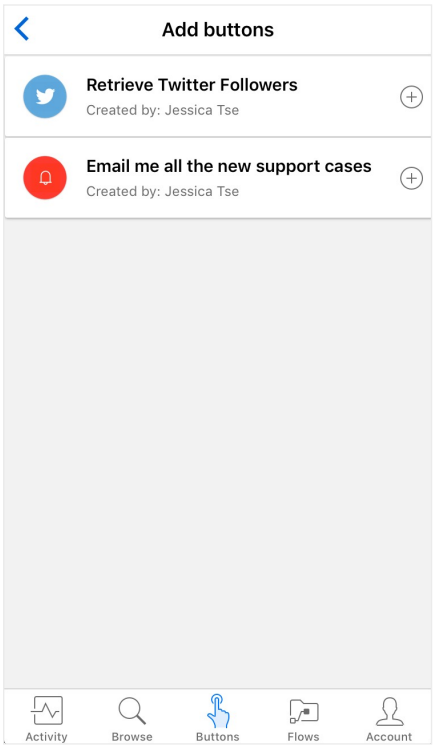


Note: If you allow others to use your connections in the shared flow, they can't access the credentials in your connection, or reuse them in any other flow.

When you share button flows, the behavior is a little different than when you share automated flows.

1. In the case of button flows, runs for all users will appear in the activity tab of the button creator's moble app.
2. Users to whom a button flow was shared will not get to see any of their own run activity.
3. When you share a button, the person or group with whom you share, can run the button the same way they run their own buttons.
4. You can stop sharing at anytime.



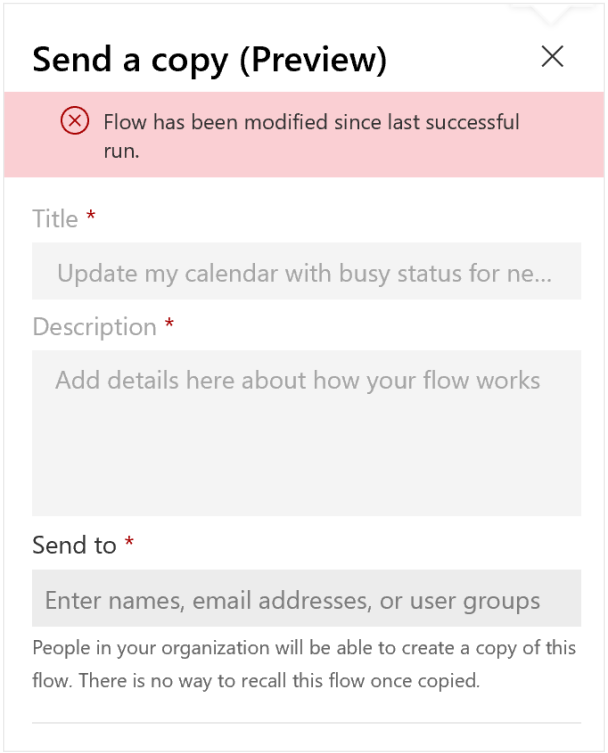


The display of the shared buttons isn't automatic, though. Users that the button was shared with must use the *Get More Buttons* command on their mobile app to manually add the button(s). On the upside, this allows users to manage their own mobile user interface. It is an additional step that may require user training. However, if you are in a larger organization that might have a lot of button sharing across the organization, it might be considered a benefit to allow them to be selective. The best approach, if buttons are going to be a significant part of your use of flow, is to create an enterprise plan for button-sharing guidelines.

Send a Flow as a Copy

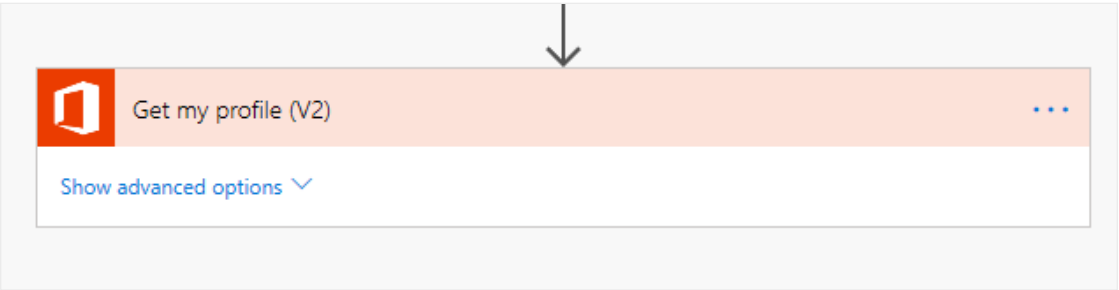
If you create a new flow and you want to share the flow logic with your peers, then sending a flow as a copy is the way to go. When you send as a copy, they will be able to use the copy they were sent to create their own version of the original flow. This approach has significant advantages over sharing as the copy will use the recipient's connections and be owned by the recipient. Each user will have their own independent flow. The downside is that improvements you make to the original flow will not update the copies you sent. You would need to send a copy of the new version.

Note: As shown in the screen that follows, you are not able to send a copy if the last run of the flow was not successful or if it hasn't had a successful run since it was last modified. This prevents you from distributing a design that will fail for the new user.



Portability

Always avoid hard-coding references in your flows. Instead use dynamic content or retrieve the information you need using logic within the flow. An excellent example of this is to use the Office 365 *Get my profile action*. You don't need to set any parameters and it will retrieve all the profile information of the current user from Office 365. You can read more about the powerful *Get my profile* action online at: https://docs.microsoft.com/graph/api/resources/user?view=graph-rest-1.0#properties.



Note: It is essential that you ensure your flow doesn't contain any hard-coded personal information. For example, a flow that sends a message to your email address or creates an activity for you would likely not be fitting when sent as a copy to others.

Regardless of the sharing methods you select, or even if you don't initially plan to share your flow, using these practices are important for future-proofing your flows.

Application Life-Cycle Management

Flows are part of the application life-cycle management (ALM) capabilities offered within the Power Platform. Using the solutions functionality that is available from the Microsoft Flow user interface, you can logically group and package related flows. By bundling flows into solutions, you will be leveraging the enterprise-class solution system that provides considerable benefits to the software development and versioning process used by many project methodologies. Using solutions, you will be saving yourself a significant amount of time. Additionally, when you are building flows that run on CDS, you can bundle related flows within a single deployable unit with other customizations and PowerApps components.

An important aspect of ALM is that you can validate that your flows work in a test or sandbox environment first before moving them, in a single action, to your production environment.

There are few preparatory design steps that you should take to minimize the changes needed after importing solutions into your production environment.

Preplanning

1. Use a Service Principal account for your connections when it is appropriate. When is it appropriate? For just about any service that supports it when the flow is running asynchronously (in the background) using CDS triggers or actions. This removes the dependency on a connection tied to a specific account used by a person. A service principal is created by registering an Azure AD application and then creating a corresponding application user in CDS. This application user in CDS can then be assigned the minimum set of permissions necessary to support the flow. Only certain services allow a service principal user. Services such as Office 365 or SharePoint don't. In those cases, you will need to use a real user account. For further information on how to create a service principal user, go to: https://docs.microsoft.com/azure/active-directory/develop/app-objects-and-service-principals.
2. Don't hard-code values if you can use dynamic content. This is especially true when referencing email addresses or record ownership in the flow, for example.
3. Always select the *Current environment* value when setting up a trigger or action that refers to the environment. Don't be tempted to choose the friendly name for the environment you are working in. As the flow traverses environments from dev to test, to staging, to production, it will then automatically connect to the environment that it is running in.

After Import

1. By default, flows will be in a disabled state after importing. This is because connections still need to be correctly configured. For example, you may need to change a SharePoint connection from the development SharePoint list to its location in the production SharePoint site.
2. Besides connection updates, changes to flows should not be made in the production instance. Fix the flow in the sandbox and then move the updated solution to production.

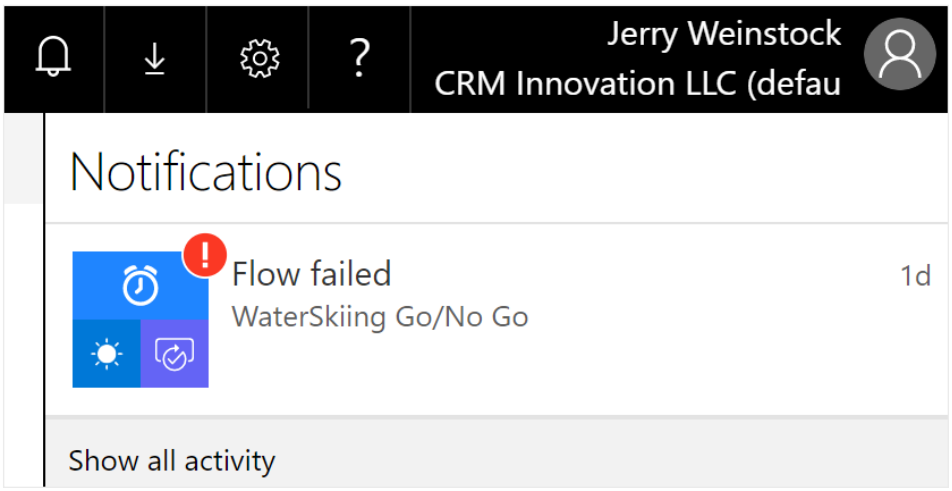
TIP: Don't circumvent the ALM solution process by exporting and importing flows from one environment to another environment. Including your flows in your organization's ALM process is critical to maintaining application consistency and functionality.

Troubleshooting

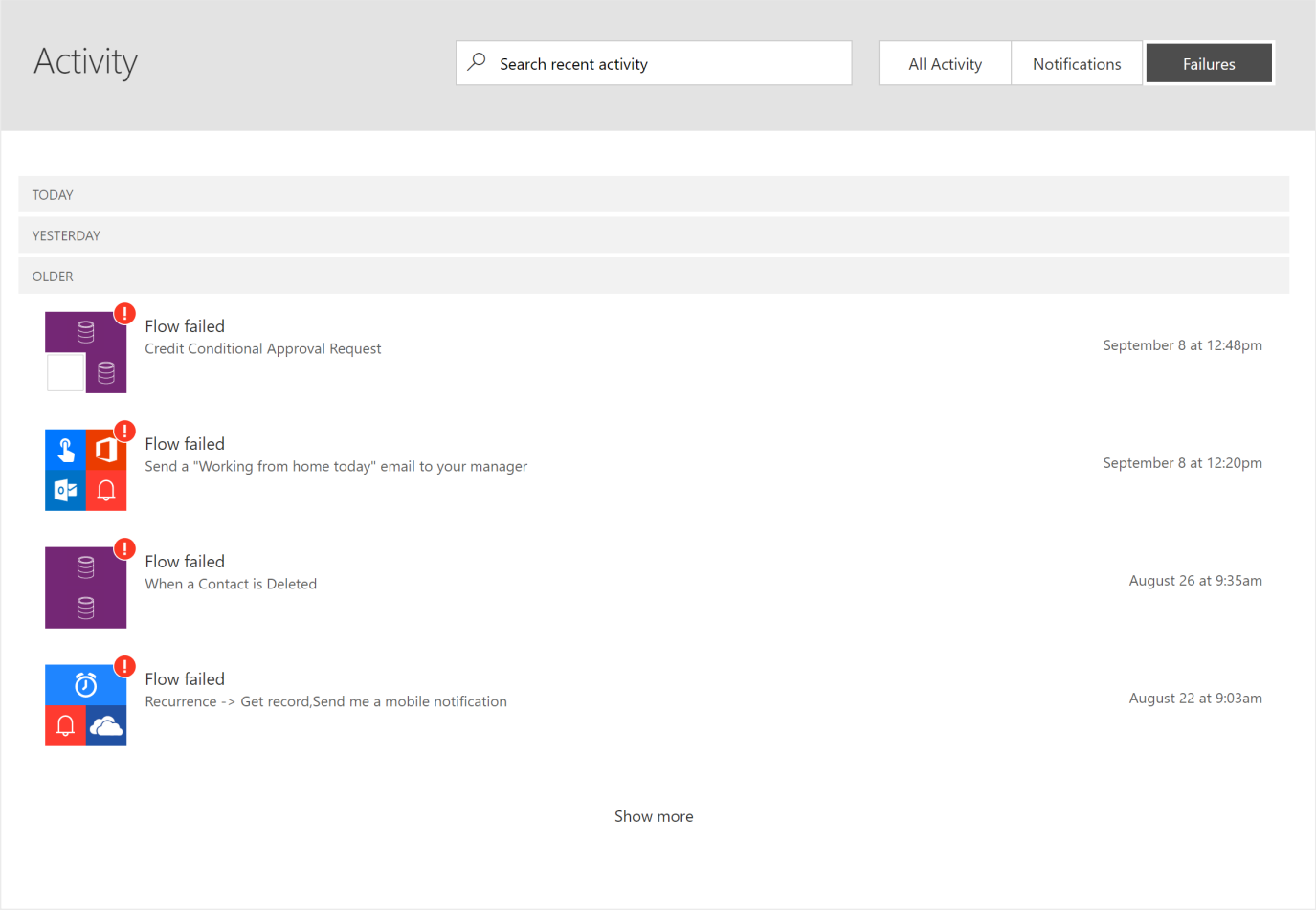
One of the important tasks in using Microsoft Flow is troubleshooting failures. Microsoft Flow has several built-in features that support doing this efficiently and effectively. Using a third-party tool to help with troubleshooting is unnecessary.

Failure Notification Web UI

You will see a notification of any recently failed flows every time you log into the Microsoft Flow web UI.

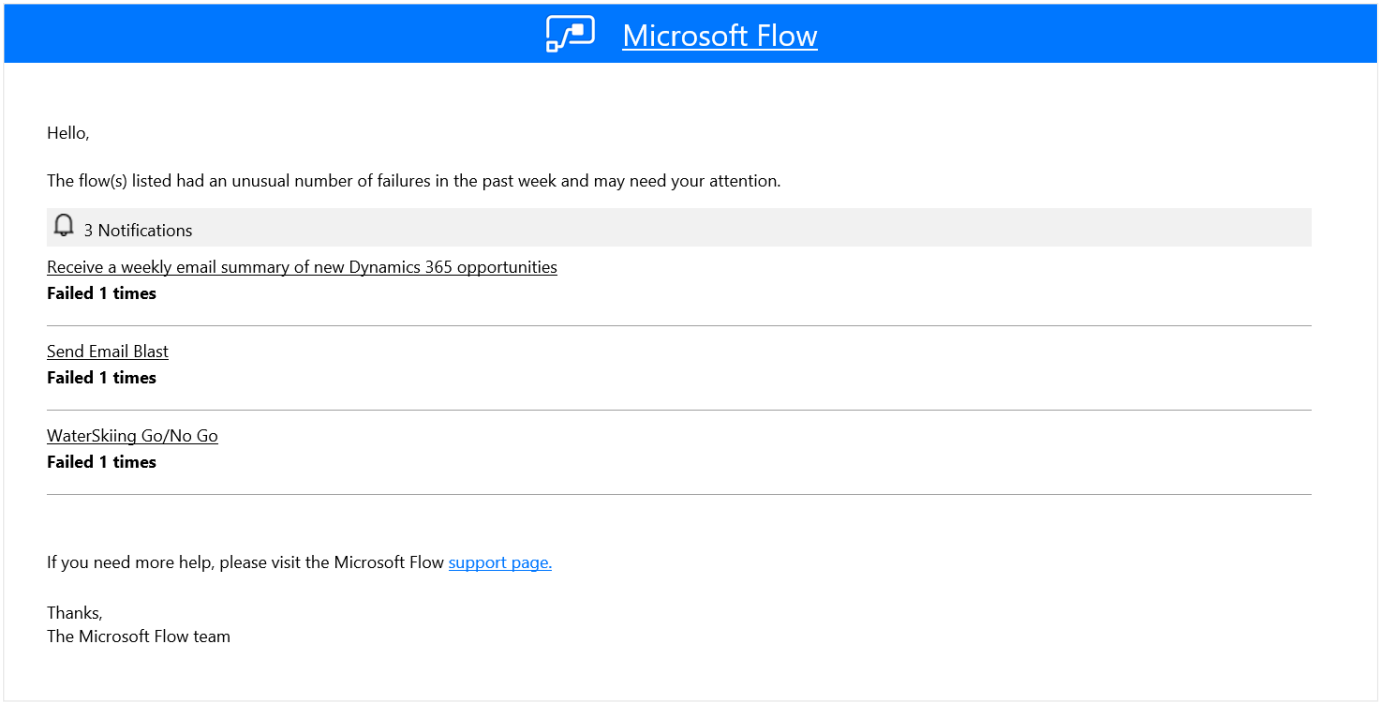


When you click on *Show all activity* in this notification, you will be taken to a screen where you can see all the recent notifications and failures. Clicking on any event in this screen will take you directly to a detailed view of that specific failure showing the flow run that failed and the error details.



Failure Notification Email

You will receive an email notification regarding flows that have had an unusual number of failures during the preceding week. You won't receive any details in the email, but clicking on any flow listed will take you directly to the run history for that flow.



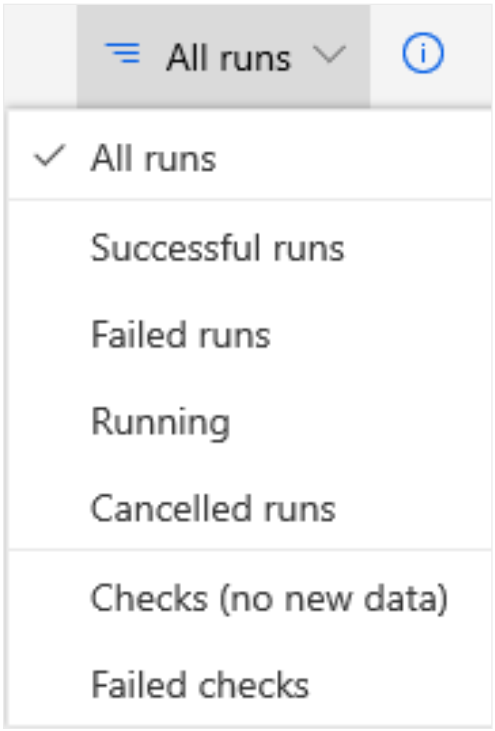
TIP: The notification emails are sent from maccount@microsoft.com. Be sure to add this email address to your email client white list to keep them out of your spam bucket.

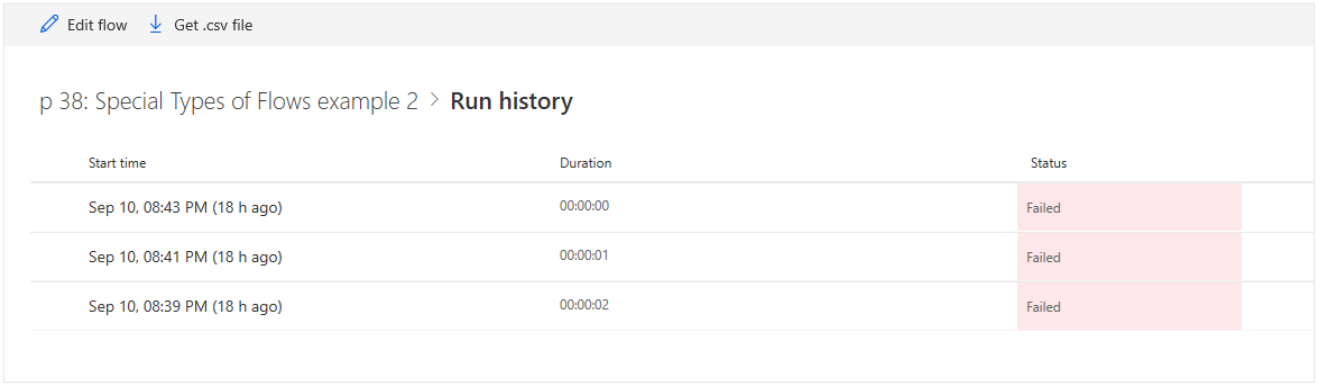
Flow Run History

There are two types of failures that appear in the run history:

- Trigger failures
- Action failures

You can filter on just these failures by selecting *All Runs* and then *Failed checks* from the view switcher:





Processing Failed Flow Runs

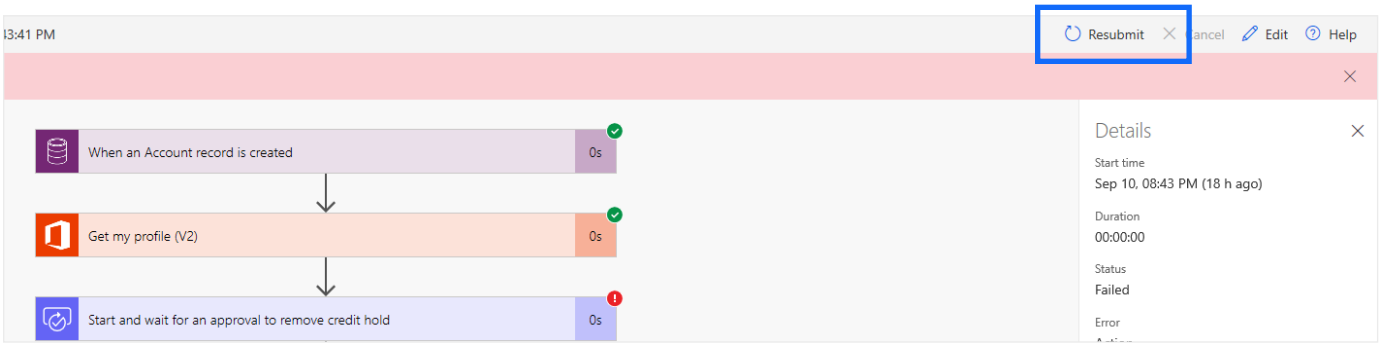
Occasionally, the flows you have authored may fail. These flows may have failed, for example, because the connector you use exceeded the API calls allowed to its service or because of a password issue.

You can rerun the failed flow with the exact same data using the *Resubmit* feature, or you can choose to cancel the flow run.

Access the flow run history from the *My flows* section for that specific flow.

On this screen you will be able to identify exactly why the flow failed and try to fix the underlying issue. If it is due to a connection failing, then you can sign in to fix the connection. If it is because of a misconfigured parameter, then update the action.

Once you feel you have corrected the issue, select the *Resubmit* button in the command bar on the failed flow run.



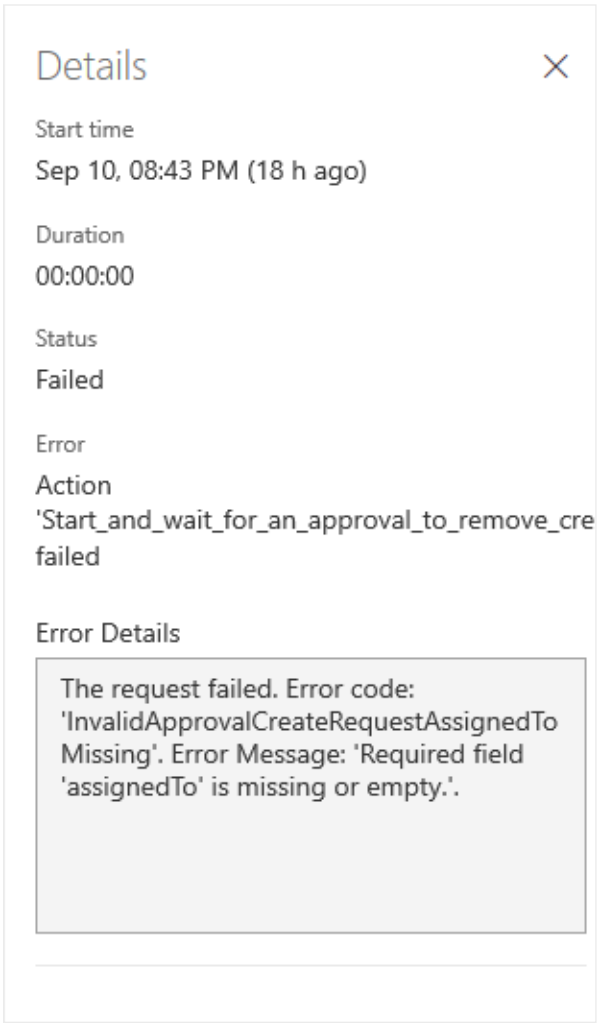
This will immediately rerun the flow with the exact same data that originally triggered it. This does mean that if the reason your flow failed was because the incoming data was incorrect, your flow will still fail. *Resubmit* is useful only for errors in the actions of the flow that you fix, such as by repairing a connection or by updating the flow definition (such as filling in a missing parameter).

Note: Trigger failures are automatically retried–so if the first step failed because of a broken connection, for example, it will automatically run again, and you will not see a *Resubmit* button for that flow.

Repair Tips

When a flow fails, there are two ways to get direct help on what to do to fix it.

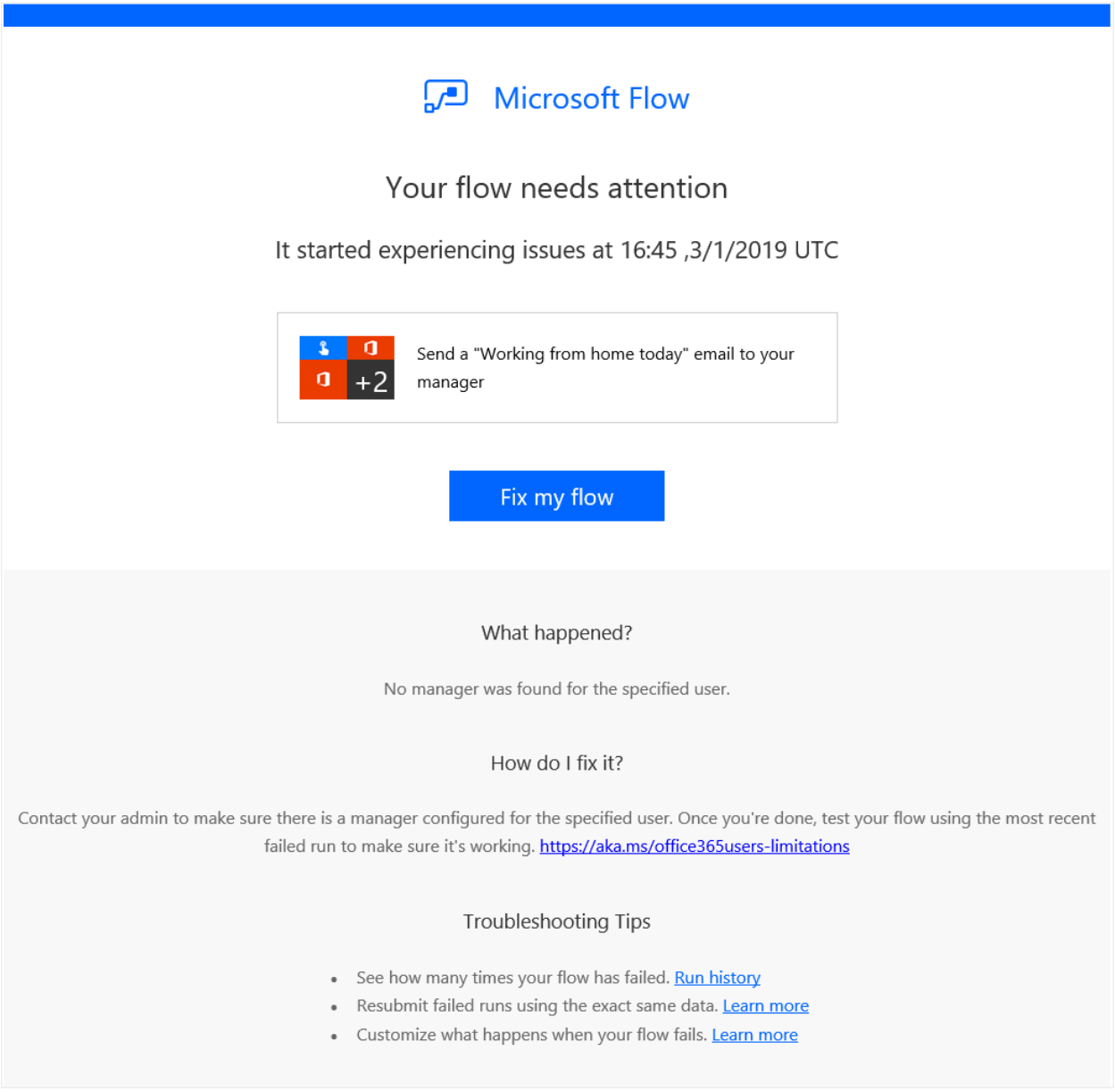
The first way is from the Microsoft Flow web UI. When you go to the activity view and inspect the failure, a run details window will open. For every failure you will at least get the error details describing why the run failed and, if available, you will also get a *How to fix* tip.



The other method is through the repair tip email that is sent to you. If you don't want to receive repair tips for a given flow, you can disable them from the *More…* menu on the flow's properties page.

The repair tips will always be available on the flow run screen even if the email feature is turned off.

> Note: Repair tip emails are only sent for flow failures that have *How to fix* tips.



Notification Intelligence

It also may be helpful to complement the *system-generated* flow notifications sent to you, as the maker of the flow, with additional insight on what is going on in the flow and any issues that have come up. To do this, use the *Send me an email notification* action. Configure it to send information about the flow itself in the notification.

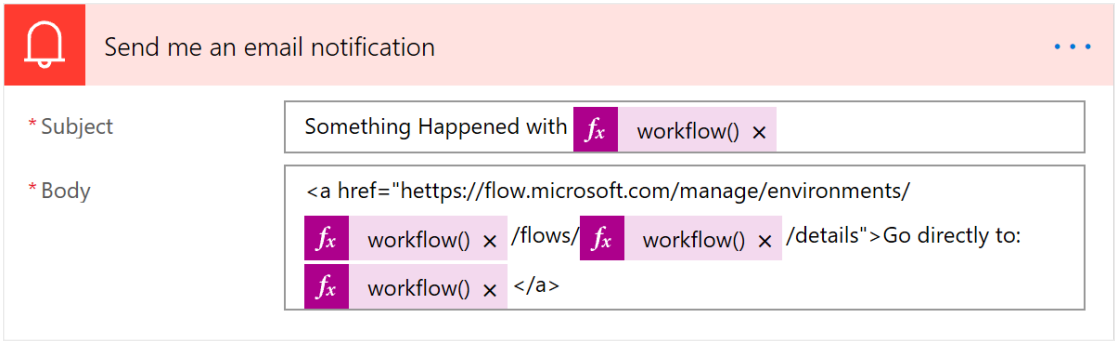
There is an output property called tags in the workflow() expression. The tags property contains properties like flowDisplayName and environmentName. This means that from the flow itself, you can send custom email notifications that link back to the flow. For example, the following will create an HTML link back to the flow with the display name of the flow in the title and body of the notification action:

Subject

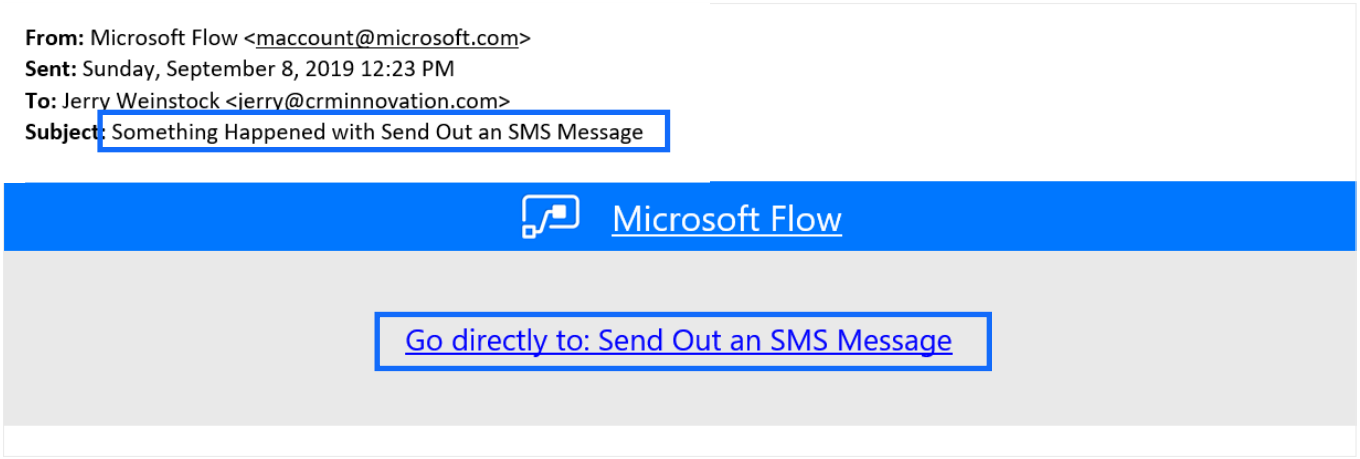
```
workflow()['tags']['flowDisplayName']
```

Body

```
<a
href="https://flow.microsoft.com/manage/environments/@{workflow()['tags']
['environmentName']}/flows/@{workflow()['name']}/details">Go directly to:
@{workflow()['tags']['flowDisplayName']}
</a>
```

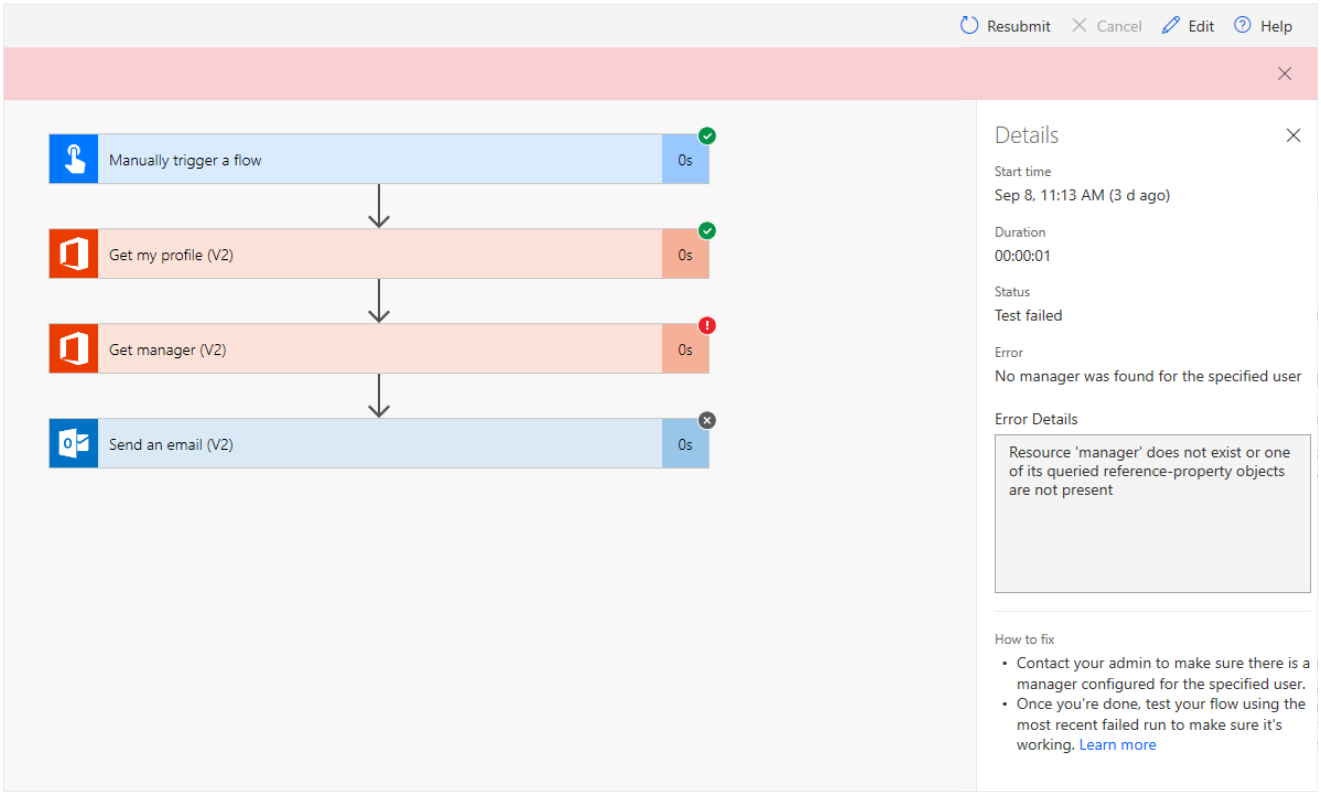


This will send the following email, alerting you to an issue and providing a direct link to the flow.

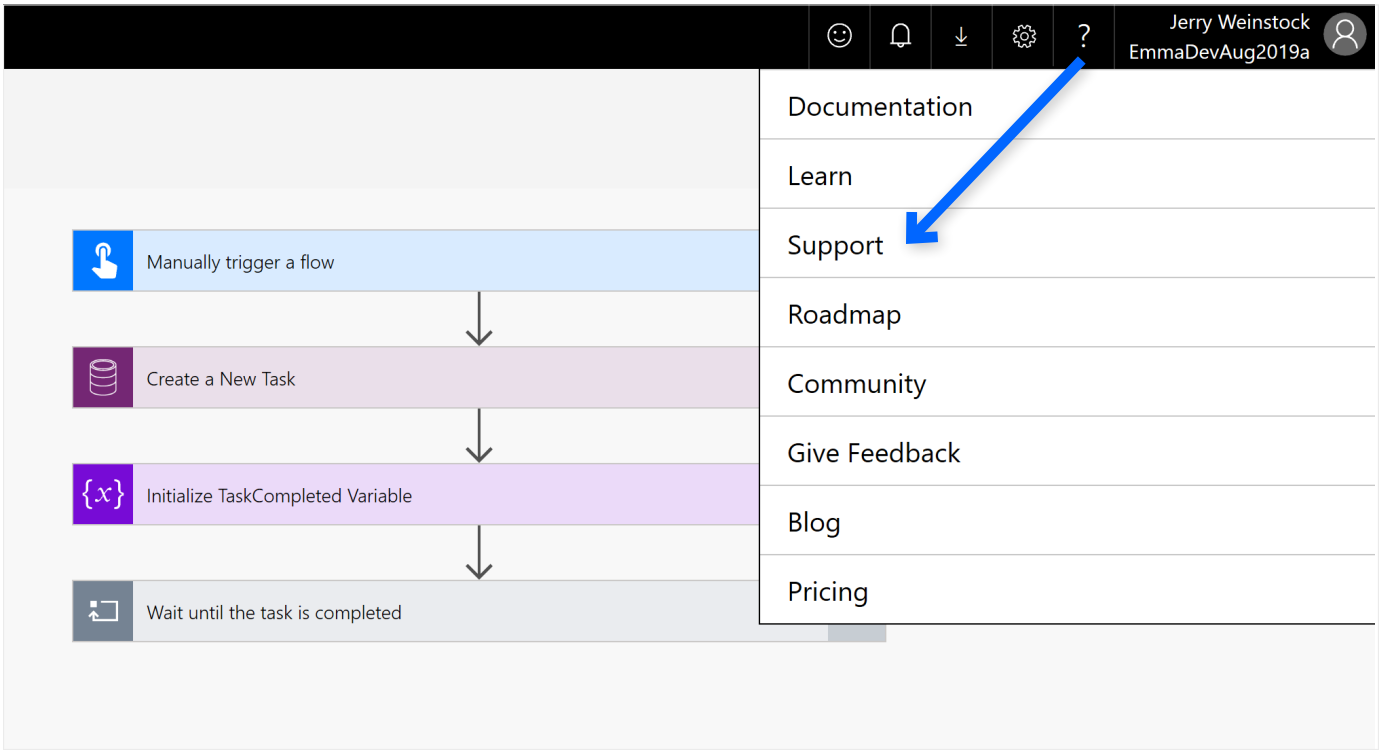


Getting Support

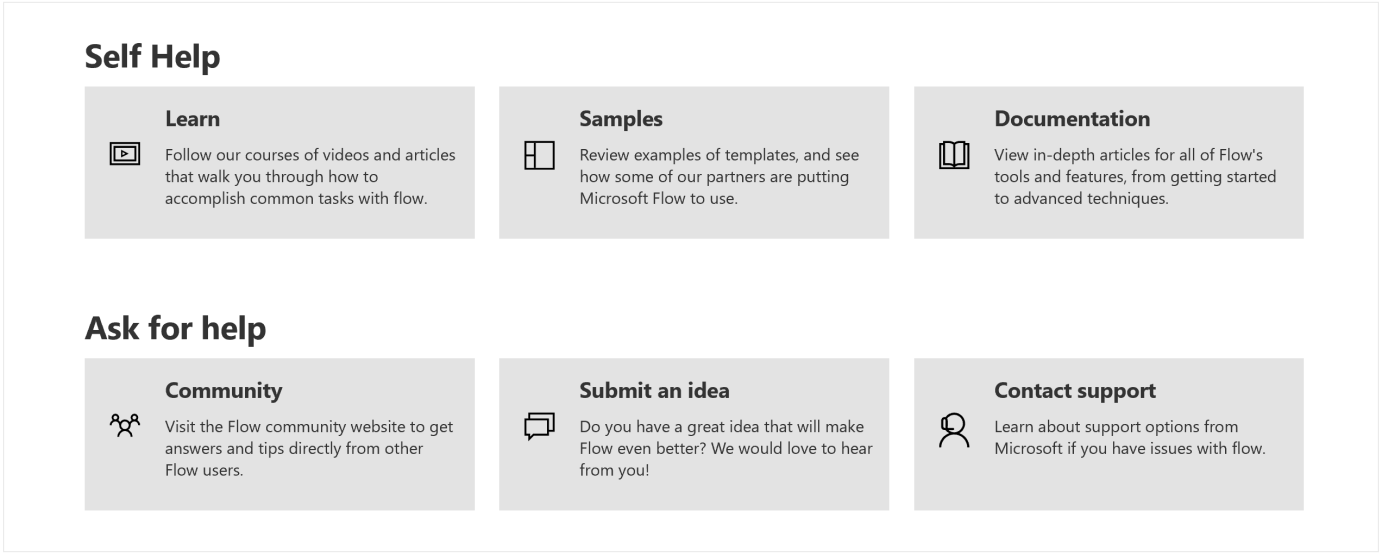
From the failed flow you can easily see the reason why the flow failed and make corrections.



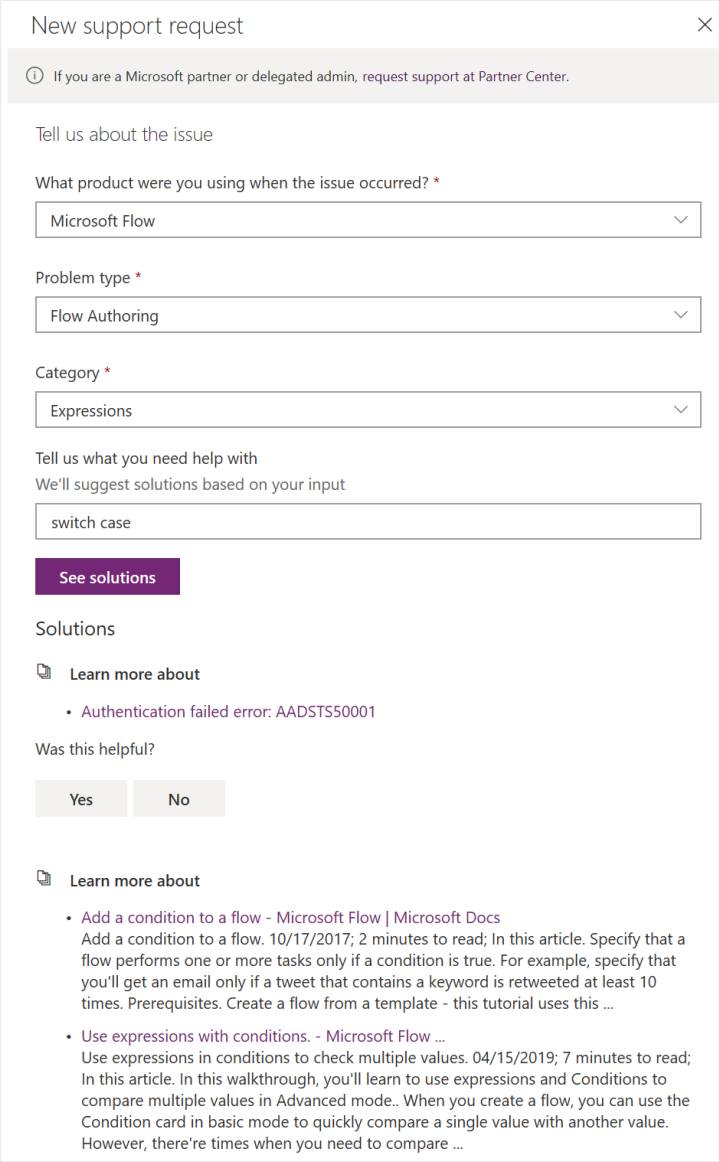
However, sometimes you just can't figure out how to fix a flow. You can seek help from online learning courses, samples, documentation, and the community.



Select the help resource that best fits your needs.



But sometimes you may need to reach out to Microsoft with a support request. When you use this process (https://admin.powerplatform.microsoft.com/?newTicket&product=Flow) you are presented with a dialog to assist in describing the issue through a series of questions. Next, you receive a curated list of possible solutions. These solutions are sourced from the Microsoft Flow official documentation, blog posts, community forums, and Knowledge Base articles.



These self-help solutions should enable you to fix issues quickly. If you still need further assistance, you can open a support ticket from that same page without losing your context.

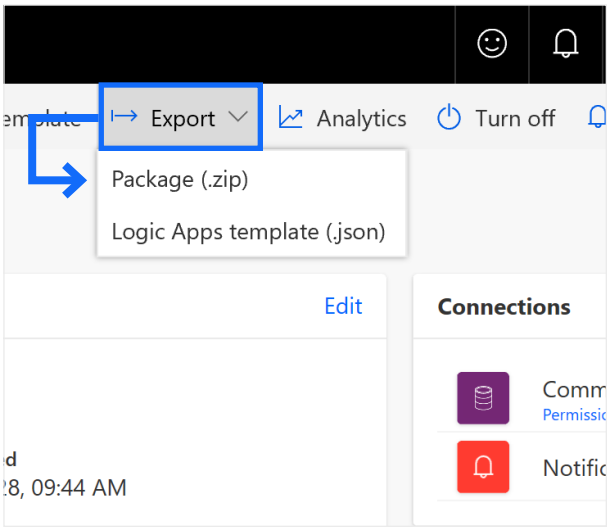
Maintaining

When you have completed the testing and are satisfied with your flow, you need to consider two more contingency areas: disaster recovery and unexpected issues with your flow logic.

Recovery

If you are building flows that are part of the CDS ALM processes, that methodology includes built-in disaster recovery and backup support. However, if you are working with flows that are used with platforms that don't have an ALM, such as SharePoint, or if you are building and sharing personal productivity flows with team members then you need to create your own system. Microsoft Flow does not have a versioning capability built into the platform. What can go wrong? Fellow team members make changes or delete flows. You make a change that accidently updates a flow used for production purposes and it stops working. There is currently no undo function in the maker interface.

The best way to prepare for these scenarios when the flow isn't part of the ALM process is to export the flow as a package to your local PC. From the web UI select from the *More* drop-down list, *Export* and then from the flyout menu select *Package (.zip)*



You will be given the opportunity to provide a name for the package and a description. This is your chance to provide the details and versioning information so that if you ever need to recover from an issue, you have a current backup of the flow to reimport into your environment.

Appendix

Microsoft Flow Limitations

As a flow maker, it is important to be aware of the product's limits and configuration settings.

Most of them are included inline in this document if they related directly to the practices and patterns described. However, not all of them are covered, and over time they may change. Therefore, you should seek out the support page where they are outlined: https://docs.microsoft.com/en-us/flow/limits-and-config.

Microsoft Flow Resources

Check out this list of places to learn more about flow, go through training courses, keep up with the latest videos and community activities, and get support from the community:

- Flow Documentation
- Flow Team Blog
- Community Blog
- Microsoft Learning
- Microsoft Flow Documentation
- Community Support