

# Microsoft Copilot Studio guidance documentation

Microsoft Copilot Studio guidance documentation provides best practices, implementation, and architecture guidance information from the team that works with our enterprise customers.

## Building agents

### OVERVIEW

[Topics best practices](#)

[Trigger phrases best practices](#)

[Slot filling best practices](#)

[Reassign an agent to a new owner](#)

[Display data from arrays](#)

[Add feedback for every response](#)

[Pass context variables from your webpage to an agent](#)

[Pass web an analytics tracking ID from a webpage](#)

## Conversational user experience

### CONCEPT

[Introduction to conversational experiences](#)

[Design the ideal experience](#)

[Practical tips for designing a conversational experience](#)

## Measuring and improving agents

### CONCEPT

[Measuring engagement](#)

[Measuring outcomes](#)  
[Deflection optimization](#)  
[Custom analytics strategy](#)

## Planning a conversational AI project

### CONCEPT

[Projects best practices](#)  
[Getting started](#)  
[Building a team](#)  
[Project planning](#)

## Related content

### REFERENCE

[Power Platform guidance center](#)  
[Power Apps guidance center](#)  
[Power Pages guidance center](#)  
[Microsoft Copilot Studio docs](#)

# Topics in Copilot Studio

Article • 11/19/2024

[Topics](#) are the core building blocks of an agent. Topics can be seen as the agent competencies: they define how a conversation dialog plays out. Topics are discrete conversation paths that, when used together, allow for users to have a conversation that feels natural and flows appropriately.

To author topics, makers can:

- Create [new topics](#) from scratch.
- [Create and edit topics with Copilot](#): Copilot lets you create and iterate on topics by describing what you want and having AI build it for you.
- Customize [templates](#) and [samples](#).

## ⓘ Important

Avoid using periods (.) in your topic names. It isn't possible to export a solution that contains an agent with periods in the name of any of its topics.

## Topic content

A topic can optionally have trigger phrases associated to it, and contains conversation nodes:

- **Trigger phrases** are phrases, keywords, and questions that a user is likely to use, related to the topic. When a user says something to the agent that is close to the configured trigger phrases, the matching topic gets triggered.
- **Conversation nodes** can be seen as action steps and define what the topic should do once it's triggered (for example, ask questions, send a message, trigger a cloud flow, set variable values, or use conditions for branching logic).

## Default topics

Any new agent comes with a set of predefined topics that make it readily operational with core capabilities.

## Custom topics

The default custom topics offer fundamental conversational abilities, such as Greetings, Goodbye, Thank you, and Start Over, while the [lesson topics](#) serve as example topics for topic authors.

**ⓘ Important**

- Copilot authors can choose to disable or delete custom topics.
- Any new topic is created as a custom topic.

## System topics

Many [system topics](#) are also natively available and help manage specific conversation events.

**ⓘ Important**

System topics can be disabled (with caution, as they can impact the agent's behavior) but cannot be deleted.

[\[+\] Expand table](#)

System topic	Description
Conversation Start	Depending on the agent client, this topic gets proactively started to initiate the conversation with the user. The agent can greet users with messages, even before the users start entering any input.
End of Conversation	This topic is meant to be positioned at the end of the conversation with an agent, so that the user can confirm if their query has been addressed or not, and fill out a satisfaction survey. This topic is important to measure an agent's performance and act on it. When this topic is reached, the session outcome is assumed to be <i>resolved</i> , unless the user explicitly doesn't confirm resolution.
Escalate	The Escalate topic is used to hand off the conversation to an external system, typically to a live agent (when configured – for example to <a href="#">Dynamics 365 Omnichannel for Customer Service</a> ). When this topic is reached, the session outcome is <i>escalated</i> .
Fallback	This topic is triggered when the agent can't understand the user query and the query can't be associated with confidence with any existing topic. It's useful to have a strategy to catch these exceptions and handle them in a graceful way (with more data sources or through an escalation path).

System topic	Description
Multiple Topics Matched (also known as "did you mean")	This topic is triggered when multiple topics might address the user input and the agent doesn't have sufficient confidence to trigger one over the others. When this type of topic is triggered, the user is presented with a list of potential matching topics and can choose the most appropriate one.
On Error	The On Error topic informs the user that an error has occurred. The message includes an <a href="#">error code</a> , the conversation ID, and the error timestamp, which can be used later for debugging. You can customize this topic to change the way it presents errors to users and what should happen when an error occurs.
Reset conversation	This topic resets the conversation by clearing all variable values and forcing the agent to use the latest published content. It only triggers when redirected to, which is the default behavior with the Start Over topic.
Sign in	This topic prompts your users to sign in when user authentication is enabled. It triggers at the beginning of the conversation when users are required to sign in, or when the conversation reaches a node that uses authentication variables.

[Learn about triggering topics](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Trigger topics

Article • 12/03/2024

There are two main ways to trigger a topic:

- **Based on a user query** (also known as *utterance*).

Topic triggering based on what a user writes or says is also referred to as *intent recognition*.

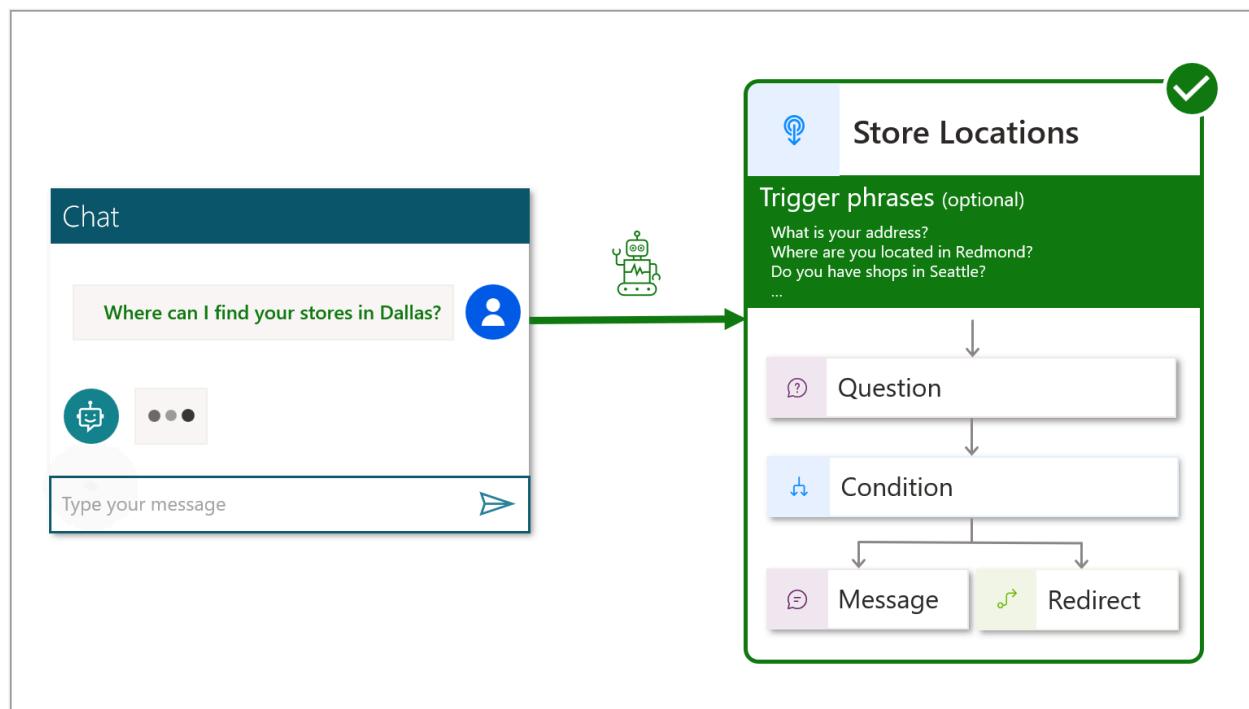
The AI uses its natural language understanding (NLU) model to parse the user utterance and finds the most relevant topic based on the configured trigger phrases that match the intent.

If multiple topics are found, the **Multiple Topics Matched** topic is triggered, so the user can select the appropriate topic from a list.

- **By redirecting to it from another topic**, with a redirect node.

## Trigger topics based on a user query

- ✓ Example: The "Store Locations" topic gets triggered based on the user utterance and the matching **trigger phrases**.

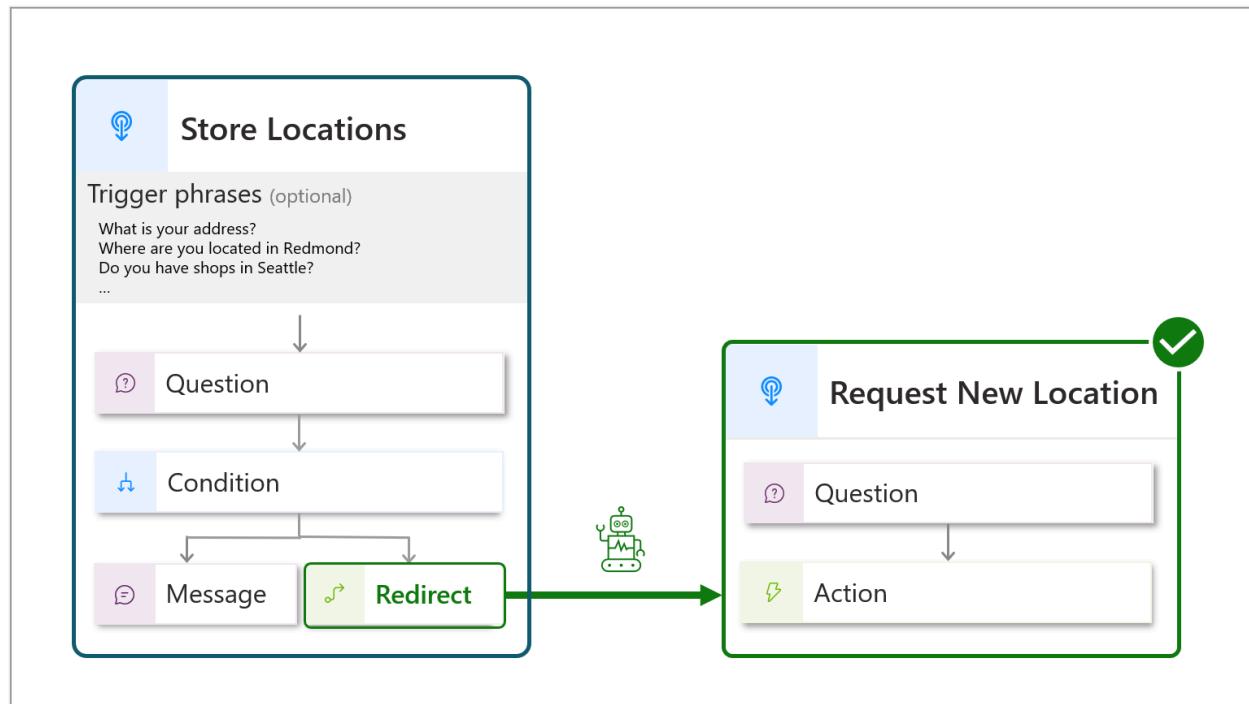


### Tip

For more information, see [Trigger phrases best practices](#).

# Redirect to a topic

- ✓ Example: the "Request New Location" topic gets triggered when it's explicitly called from an existing topic, with a **redirect** node.



Define agent topics

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Defining agent topics

Article • 11/19/2024

Defining the best topics for your agent requires an understanding of the questions users might ask or the tasks they try to accomplish, and the kind of information and automation you need to provide.

For example, a retail agent might start by asking the user to choose from four things they want to do: find a store, place an order, check the status of an order, or return a purchased product. Their answer could lead them into one of four topics, each with its own topic dialog.

An initial set of sources for this information include:

- Any existing frequently asked questions (FAQs) or knowledge bases (KB).
- Common subjects raised by your employees or customers in internal or customer service scenarios. If you're building a customer service agent, talk with existing service representatives to learn what the most common questions are and the order in which they're typically asked.

## Different types of topics

Typically, agent users have a specific question or problem they want to address or an issue they want resolved (their "task"). The types of tasks undertaken by agent users (and, therefore, the types of agent topics you need to create) fit into three categories:

1. **Informational:** for example, "*What is...?*", "*When will...?*", "*Why...?*"
2. **Task completion:** "*I want to...?*", "*How do I...?*"
3. **Troubleshooting:** "*Something isn't working...?*", "*I got an error message...?*"

You might also need to create agent topics to handle ambiguous user questions, such as "*I need help*" or "*shopping*." These topics would ask the agent user for clarification, so they can be routed to the correct topic.

## Topic design process

### 1. Identify the topic

- Choose topics agent users ask about.
- Start with high-impact topics.

- Think about topics from your user's point of view. Keep in mind that they might be less familiar or knowledgeable about an area than you are.

## 2. List all scenarios

- List all possible scenarios.
- Categorize scenarios: **informational**, **task completion**, and **troubleshooting**.
- Hierarchize topics: what are the initial questions a user asks?

## 3. Design a high-level conversation tree

- Draw the conversation tree.
- Define the hierarchy of questions asked inside each topic.
- Define the fewest number of questions to understand the situation and provide the right solution.

## 4. Validate and iterate on the design

- Read the dialog out loud before publishing to help identify if there are problems with the tone or words.
- [Get analytics and read session transcripts](#) for further optimization.
- Continuously iterate on and refine your agent topics by observing your agent users' interactions with the agent after each authoring iteration.

### ⓘ Important

- Don't just replicate what your website or app can already do—your customers likely are familiar with your website or app and can accomplish common tasks themselves without needing to interact with an agent.
- Focus on creating topics for issues or scenarios that generate a high volume of chats or calls first. Work on the longer tail of other, less-critical issues over a period of time.
- Be as thorough as possible in your design and consider all the potential scenarios that your users might ask or need help with.

# Plan for both single-turn and multi-turn interactions

For simple interactions, you create single-turn conversations with just one question and one answer. But more substantive topics require a multi-turn conversation with multiple

back-and-forth interactions between the user and your agent.

For instance, if a user asks a retail agent whether there are any stores nearby, the agent might respond with a question to narrow down the possible responses. These responses might be "*What city do you live in?*" or "*What is your postal code?*" The user's response determines the agent next response in the conversation.

## Use native generative AI capabilities

When designing an agent, you might not be able to anticipate all the types of questions your agent users ask. To help mitigate this, Copilot Studio incorporates a powerful AI-powered capability that uses the latest advancements in natural language understanding (NLU) models.

When you enable the [Boost conversations](#) option for your agent and link your agent to a publicly available, Bing-indexed website, your agent can provide automatically generated, conversationally friendly, plain language responses, without the need for you to create topics for every eventuality or "edge-case."

## Offload large volumes of single-turn question and answer pairs

Copilot Studio limits an agent to 1,000 topics.

If you have large numbers of FAQs or knowledgebases, you can avoid creating one topic for each question and answer pair by offloading them outside of Copilot Studio topics, while offering a single user experience in Copilot Studio by [using the Fallback topic](#).

### Tip

- [Question answering in Azure Cognitive Service for Language](#) provides cloud-based NLP that allows you to create a natural conversational layer over your data. It's used to find the most appropriate answer for any input from your custom KB of information.
- [Learn how to integrate question answering in Copilot Studio.](#)

[Topic authoring best practices](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Topic authoring best practices in Copilot Studio

Article • 07/03/2024

## ⓘ Important

Power Virtual Agents capabilities and features are [now part of Microsoft Copilot Studio](#) following significant investments in generative AI and enhanced integrations across Microsoft Copilot.

Some articles and screenshots may refer to Power Virtual Agents while we update documentation and training content.

## Create and organize topics in a way that's manageable and maintainable for your context

Copilot Studio offers much flexibility when it comes to topic management. While there's no *one size fits all*, it's a good practice to keep in mind the different ways that topics can be triggered:

- ✓ **Topics can trigger based on the user utterance and the natural language understanding (NLU) model.**  
These topics can almost be seen as your **entry points topics**.  
If you have trigger phrases that overlap multiple topics, it's good to have a catch-all topic (or *disambiguation topic*) and then redirect to other topics through slot filling and clarifying questions.  
Thanks to entity extraction, clarifying questions can be skipped and the conversation directly flows to the appropriate topic, without further input from the user.
- ✓ **Topics can trigger when called from a redirect action.**  
These topics can contain conversation and logic nodes, can be called by multiple topics, and can have input and output variables. They're ideally **Reusable, bite-size, topics**.
- ✓ **A topic can also be both.**  
For example, topics triggered through intent recognition or by an explicit redirect.

✓ Topics can be triggered by other events.

For example, custom events, inactivity, and so on.

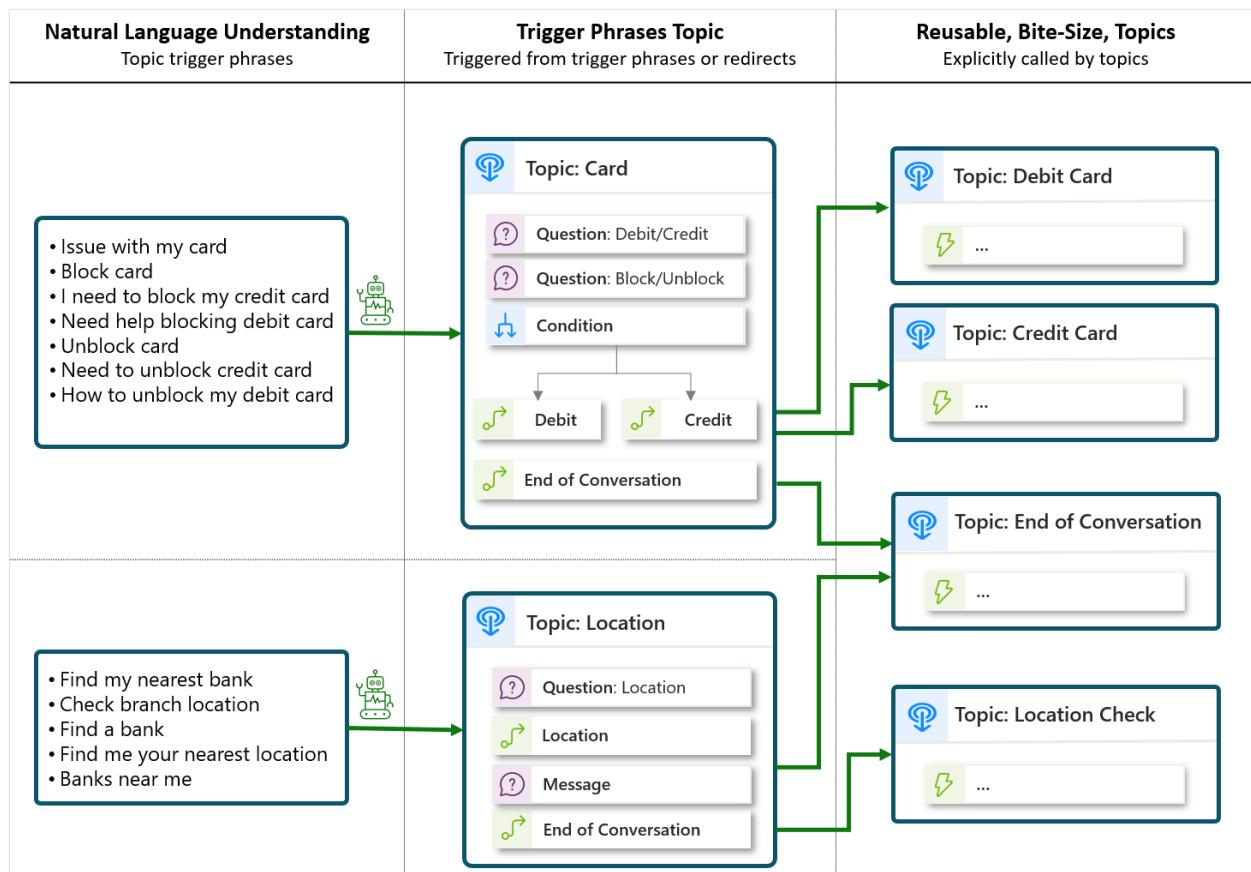
For more information, see [the full list of topic triggers](#).

💡 Tip

In the following example, trigger phrases are associated to two main topics that then break down their logic into multiple topics that are called with redirect actions.

Some topics can be called by multiple other topics: that's typically the case with the End Of Conversation topic.

Thanks to slot filling and entity extraction, if a user says "*I need to unblock my credit card*", the Card topic is triggered, and both Debit/Credit and Block/Unblock questions are skipped, since CardType and OperationType are deduced from the trigger phrase. Then, the appropriate Credit Card child topic is automatically called, without the user providing any additional input.



## Create bite-size topics

Large topics can be challenging to maintain and update. It's a good idea to break down your copilot logic whenever possible, especially if parts of your copilot conversation logic are shared by multiple topics.

Topics don't need to all have trigger phrases, as topics can redirect to other topics and pass variable information back and forth.

### 💡 Tip

It's often more manageable to create many bite-size topics rather than a few large topics. Taking this approach also helps making triggering more effective, by clearly mapping trigger phrases to the specific topics that address those areas.

The number of topics is a conversational design preference and decision. It either results in having a few bigger topics (if there's joining) or having smaller topics (if there's splitting).

### 💡 Tip

For intents that are more complex to recognize (for example, for generic words, such as "*request*" or "*issue*", as these could be about different things), then splitting the topics might be a better option, with adequate trigger phrases.

## Create reusable topics

Whenever conversation messages or logic nodes are shared by multiple topics, it's a good idea to group them in a new topic instead of duplicating their content. That way, a single update to that topic reflects on all the topics redirecting to it.

### 💡 Tip

If you call the same Power Automate cloud flows from multiple Copilot Studio topics with the same or similar before and after nodes, it's a good idea to group them in a dedicated topic.

## Avoid topic overlap

## Monitor and remove ambiguity between topics

When topics are triggered with NLU, it's important to avoid overlap issues. Overlap issues typically reflect through a large number of [multiple topics matched](#) topics (also known as "did you mean").

You can avoid these issues by reducing the overlap in intent meaning between the trigger phrases of different topics.

### 💡 Tip

For topics triggered with trigger phrases, you should:

- Compare the trigger phrases across your topics and remove ambiguous pairs.
- Try to not use the same words in different topics trigger phrases.

There are multiple ways to monitor topic overlap:

- Analyze the user utterances that trigger a "did you mean" topic, as it's a key indicator that you have overlap
- Copilot Studio offers a [topic overlap detection](#) feature, that lets copilot authors discover overlapping topics to resolve accordingly (deleting/moving trigger phrases between topics).

## Create a disambiguation topic

When multiple topics have trigger phrases that are too close, you can avoid topic overlap by creating a disambiguation topic.

### 💡 Tip

To continue on the previous example, to address with certainty scenarios such as "*unlock credit card*" or "*unlock debit card*", where users might just say "*unlock card*" and the NLU model doesn't know what specific topic and process to trigger, you can call a generic `Card` disambiguation topic, use an entity slot filling question for `cardType`, and launch the appropriate topic accordingly.

## Use entities to reduce the number of topics

You should avoid duplicating logic when the variation could be stored in a variable. For example, instead of creating similar topics, such as:

- Order Pizzas
- Order Burgers
- Order Drinks

You could create:

- One topic for `Order`
- One entity for `FoodType`, with `Pizzas`, `Burgers`, and `Drinks` as values.

[Leverage the Fallback topic](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Using the Fallback topic

Article • 11/19/2024

The Fallback topic gets triggered when Copilot Studio doesn't understand a user utterance and doesn't have sufficient confidence to trigger any of the existing topics.

There are multiple ways to use the Fallback topic.

## Offload single question and answer pairs

If you have a large number of question and answer pairs for single-turn conversations, you can also consider offloading them to an external system and access them through the Fallback topic.

Copilot Studio limits an agent to 1,000 topics. You can decide to manage exceptions to the Copilot Studio topics that should generally do things for the user through the Fallback topic. With the Fallback topic, you can get answers for users that you retrieve from external systems using Power Automate, by [calling a cloud flow from an action node](#).

### 💡 Tip

- [Question answering in Azure Cognitive Service for Language](#) provides cloud-based natural language processing (NLP) that allows you to create a natural conversational layer over your data. It's used to find the most appropriate answer for any input from your custom knowledge base of information.
- [Learn how to integrate question answering in Copilot Studio](#).

## Give your agent a personality

[Question answering in Azure Cognitive Service for Language](#) provides a readily available "chitchat" data source, of various tones such as "professional", "friendly", "witty".

Chitchat lets your agent answer trivial questions, such as "*how are you feeling today, agent?*".

## Integrate with generative AI models

You can also use the **Fallback** topic to call a generative AI model, such as those used by the [Azure OpenAI Service](#), to handle more questions from your customers.

Through the system prompt and [prompt engineering](#), you can also feed the model with instructions, rules and input data before it generates an answer.

## Learn from the Fallback topic

You should track what your customers are saying that doesn't trigger a standard Copilot Studio topic. The phrases can enrich an existing topic's trigger phrases, typically when the user utterance should have triggered a topic. The phrases can also lead to the creation of new topics to better serve your users as you understand better what they're trying to accomplish.

[Optimize trigger phrases and Natural Language Understanding](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Use the code editor to write and edit YAML in topics

Article • 07/03/2024

## ⓘ Important

Power Virtual Agents capabilities and features are [now part of Microsoft Copilot Studio](#) following significant investments in generative AI and enhanced integrations across Microsoft Copilot.

Some articles and screenshots may refer to Power Virtual Agents while we update documentation and training content.

[Topics](#) can be written and edited in the code editor, which supports the use of YAML, which stands for "YAML Ain't Markup Language™." YAML is used to write configuration files, and tends to be easier to read than XML or JSON. YAML files use either the .yaml or .yml file extension. YAML files are formatted with spacing and indentation to represent data structures, such as strings, numbers, Boolean, lists, and dictionaries.

To access the code editor from a topic, select the **More** icon (...) and then select **Open code editor**.

Watch this video to get an overview of how to use YAML in the code editor:

<https://learn-video.azurefd.net/vod/player?id=f7869585-6c3f-4878-9557-b04c1824dca0&locale=en-us&embedUrl=%2Fmicrosoft-copilot-studio%2Fguidance%2Ftopics-code-editor>

## ⓘ Important

All makers can leverage the code editor to visualize a single page snapshot of the full configuration of the current topic. Developers and advanced users might choose to design and edit within the code editor. However, it's recommended that caution is used when making revisions in the code editor, because errors in punctuation and other syntax patterns might result in complex error messages, and might break the current conversation. Please be advised that technical support teams can't assist in remediating code editor errors. Therefore, if you're a novice in revising YAML, [make a copy of the topic](#) before making any changes.

# When to use the code editor

Copilot Studio never requires you to write YAML from scratch. Instead, YAML is generated automatically when you create objects on the topic canvas. You can then either review the configuration in the code editor to better understand the settings, or you can choose to make changes to the YAML. The following are some examples for using the code editor:

- The code editor displays the complete configuration of a topic without the need to interact or navigate the individual dialogs in the topic. It's a one-stop-shop for displaying all the settings in a dialog.
- Makers might choose to copy specific nodes from a topic and paste them into other topics, rather than using the copy/paste icons. It might be easier than selecting multiple nodes of a configuration.
- Makers might choose to update a Power Automate flow ID in YAML, rather than wait on the synchronization to the Call and Action menu.
- Makers might choose to clone and revise questions or messages when they're similar. However, in this case, it's critical to ensure that all the IDs and variables remain unique. In this scenario, remember to revise cloned IDs into unique IDs.
- Makers might choose to do simple revisions, such as appending a term to Topic Phrases, within the code editor since you can type, or copy and paste the text into the YAML.

## Understanding YAML in the code editor

The following code is an example of a typical block of YAML. For more information about how that code applies to the dialog in the canvas, see [descriptions](#).

### YAML

```

1 kind: AdaptiveDialog
2 beginDialog:
3   kind: OnUnknownIntent
4   id: main
5   priority: -1
6   actions:
7     - kind: SearchAndSummarizeContent
8       id: search-content
9       userInput: <System.Activity.Text
10      variable: Topic.Answer
11      moderationLevel: Medium
12      tone: Friendly
13      additionalInstructions: Include emojis to make responses more fun.
14      publicDataSource:
15        sites:
16          - "https://www.chessusa.com/"
17          - chess.com
18          - lichess.org
19          - "https://www.weather.gov/alerts"
20
21      sharePointSearchDataSource: {}
22
23    - kind: ConditionGroup
24      id: has-answer-conditions
25      conditions:
26        - id: has-answer
27          condition: !=IsBlank(Topic.Answer)
28          actions:
29            - kind: EndDialog
30              id: end-topic
31              clearTopicQueue: true

```

## Descriptions

- Line 1: The type of dialog
- Line 2-3: Indicates the start of a conversational boosting system topic
- Line 7: Indicates a generative answers node
- Line 10: The relevant variable
- Line 11-13: Details about the prompt and special instructions
- Line 14-21: knowledge source links
- Line 23-27: Condition group-each condition has a unique ID, along with the condition's formula details and redirects

## Test YAML in the code editor

If you'd like to test out this code, perform the following steps in Copilot Studio:

1. Create a sample copilot.
2. In the copilot's settings, select the **Generative AI** tab and select **Generative (preview)**.
3. Go to the **Topics** page, select **System** and select the **Conversational boosting** topic.
4. Open the code editor.
5. Copy the sample YAML code in this article and paste it in place of the existing **Conversational boosting** topic's YAML code.

```
yml
```

```
kind: AdaptiveDialog
beginDialog:
  kind: OnUnknownIntent
  id: main
  priority: -1
  actions:
    - kind: SearchAndSummarizeContent
      id: search-content
      userInput: =System.Activity.Text
      variable: Topic.Answer
      moderationLevel: Medium
      tone: Friendly
      additionalInstructions: Include emojis to make responses more
fun.
publicDataSource:
  sites:
    - "www.chessusa.com/"
    - "www.chess.com/"
    - "www.lichess.org/"

sharePointSearchDataSource: {}

- kind: ConditionGroup
  id: has-answer-conditions
  conditions:
    - id: has-answer
      condition: !=IsBlank(Topic.Answer)
      actions:
        - kind: EndDialog
          id: end-topic
          clearTopicQueue: true
```

6. Select **Save** and close the code editor.

7. Test the copilot in the **Test** pane by asking a chess-related question, such as, "How does a knight move?"

## Related content

- [YAML Ain't Markup Language \(YAML™\) revision 1.2.2 ↗](#)
- [YAML Checker - The YAML Syntax Validator ↗](#)
- [YAML to JSON Converter ↗](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback 

# Optimizing trigger phrases and natural language understanding

Article • 11/19/2024

## What are trigger phrases in Copilot Studio

- Trigger phrases train your agent's natural language understanding (NLU) model.
- Trigger phrases are configured at the topic level and indicate to the agent for what typical user utterances a specific topic should be triggered.
- Trigger phrases typically capture the way a user would ask about a problem or issue. For example, "*problem with weeds in lawn*"

### Tip

- When creating a new topic, a maker only needs to provide a few sample phrases (ideally between five and ten). When the agent is used, the AI will parse what the user says and trigger the topic closest in meaning to the user utterance.
- For more information about effective trigger phrases, see [Choose effective trigger phrases](#).

## The importance of the triggering context

Copilot Studio NLU behaves differently based on the conversation state, which can sometimes lead to different behaviors for the same user utterance.

The following are the different conversation states:

- **Start of the conversation:** the agent has no context, so a user utterance is expected to either: trigger a topic directly (intent recognition), trigger a "did you mean" (Multiple Topics Matched) disambiguation question among intent candidates if there are multiple matching topics, or go to a fallback topic if the intent isn't recognized.
- **After a "did you mean" (Multiple Topics Matched) is triggered:** NLU optimizes to match one of the suggested topics, with higher thresholds to move out of the presented options.

- **Switching out from a current topic:** If the NLU is trying to fill a slot in a topic, and the user is giving a user query that could trigger another topic (topic switching).

## On punctuation

The NLU model is agnostic to punctuation, including question marks.

## Creating new trigger phrases

If possible, start with real production data over making up your own trigger phrases. The best trigger phrases are the ones similar to actual data coming from users. These phrases are the ones that users ask a deployed agent.

There's no need to leave specific words out: the model is designed to give less weight to unnecessary words, such as stop words (words that are filtered out before the processing of natural language data because they're insignificant).

## Optimizing trigger phrases

[\[+\] Expand table](#)

#	Tip	Examples
1	<b>Have at least 5-10 trigger phrases per topic</b> Iterate and add more as you learn from users.	<i>Find my nearest store</i> <i>Check store location</i> <i>Find a store</i> <i>Find me your nearest location</i> <i>Store near me</i>
2	<b>Vary sentence structure and key terms</b> The model automatically considers variations of those phrases.	<i>When are you closed</i> <i>Daily open hours</i>
3	<b>Use short trigger phrases</b> Fewer than 10 words.	<i>When are you open</i>
4	<b>Avoid single-word trigger phrases</b> This increases weight for specific words in topic triggering. It can introduce confusion between similar topics.	<i>Store</i>
5	<b>Use complete phrases</b>	<i>Can I talk to a human assistant</i>
6	<b>Have unique verbs and nouns or combinations of those</b>	<i>I need customer service</i> <i>I want to speak with a consultant</i>

# Tip	Examples
7 Avoid using the same entity variation You don't need to use all of the examples from the entity value. The NLU automatically considers all the variations.	<i>I want to order a burger</i> <i>I would like a pizza</i> <i>I want chicken nuggets</i>

## Balance the number of trigger phrases per topic

Try to balance the number of trigger phrases between topics.

### 💡 Tip

That way, the NLU capabilities don't overweight a topic versus another based on the configured trigger phrases.

## Assessing the impact of your changes

When updating trigger phrases, or when merging or splitting topics, there are multiple ways to assess the changes:

- ✓ An immediate change in agent behavior, which can be observed through the "test agent" canvas (for example, a topic that's now triggering or not based on trigger phrase updates).
- ✓ A change after agent deployment and facing traffic, which translates to higher or lower deflection (non-escalation) rates. This can be observed from the analytics tab in Copilot Studio.

### 💡 Tip

You can test topic triggering and how your NLU model performs against test data in bulk, by leveraging the [Copilot Test Framework](#).

Although the underlying features and components used to build the Copilot Test Framework (such as interacting with the Direct Line API) are fully supported, the Copilot Test Framework itself represents sample implementations of these features.

Our customers and community can use and adjust the Copilot Test Framework to implement bulk testing. If you face issues with the Copilot Test Framework, report the issue here: <https://aka.ms/PVAsamples>. (Microsoft Support won't help you

with issues related to these samples, but they will help with related, underlying platform and feature issues.)

### Entities and slot filling best practices

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#)

# Entities and slot filling best practices in Copilot Studio

Article • 11/19/2024

## What is slot filling?

Slot filling is related to the use of [entities](#) that let an agent acquire and use information more easily by identifying and extracting them from the user query.

An agent typically needs several pieces of information from a user to do its job. To acquire this information, you would typically ask separate questions, one for each fact the agent needs. When you use entities in your questions, Copilot Studio detects the required information and skips the questions where the information could be identified in the triggering user query.

For example, when a user asks: **I'd like to order 3 large blue t-shirts**

Copilot Studio Natural Language Understanding (NLU) can immediately understand:

- Topic is **Order**
- Quantity is **3**
- Color is **Blue**
- Item Type is **T-Shirt**

The agent can then skip unnecessary questions. If some pieces of information are missing, for example *Size*, it asks the unanswered questions before moving forward. Slot filling lets your agent acquire and use information more easily and reduces the number of questions you need to ask.

You first define which entities you want to use, and their types.

## Defining entities

Copilot Studio provides several built-in entities for the most common use cases, such as *Email*, *Date and time*, *Person name*, *Phone number*, *Color*, *Country*, *City*, *Number*, *Money*, and more.

Built-in entities are powerful as they can handle many variations of the form an information can take. For example, when using *Money* in a conversation, the user might

enter a value as "\$100" or "a hundred dollars" or "100 dollars". The NLU model in Copilot Studio figures out that the value is a monetary value of 100 dollars.

### 💡 Tip

However values are entered, as long as they're requested with questions in your conversation flow, they're stored in variables that you can re-use.

You can also define your own custom entities, such as *Item Type* from the previous example. Custom entities can be of two types:

- **Closed list:** for a predefined list of values.
- **Regular expressions (RegEx):** for information matching a specific pattern. RegEx is ideal when you need to capture data that always follows the same format (for example, *INC000001* for a ticket number)

## Improving the user experience

Using entities makes your life simpler by letting Copilot Studio more intelligently gather information from user queries. More importantly, it makes life better for users, since entities can find and store information from user queries, then make that information available later in the conversation. Entities improve the user experience—they make your agent seem smarter—and using them whenever possible is definitely a best practice.

## Entity best practices

### Use synonyms

You can add synonyms to closed list entities values to manually expand the matching logic for each item in the entity's list. For example, in the "*hiking*" item, you can add "*trekking*" and "*mountaineering*" as synonyms.

### 💡 Tip

- Using synonyms can also help topic triggering, as they increase the weight of a trigger phrase by adding related words as synonyms of an entity it contains. For example, for a "Complaint" option, add similar negative words or phrases as synonyms.

- The NLU model also generalizes on all entity variations (that is, all of their values and their synonyms) in case a topic trigger phrase includes at least one variation of this entity. In other words, agent authors should include one example of a trigger phrase with one usage of this entity for the NLU to generalize on other entity variations.

## Enable Smart Matching

For each closed list entity, you can also enable Smart Matching.

Smart Matching is part of the intelligence supported by the agent's NLU model. With this option enabled, the agent interprets the user's input using fuzzy logic, based on the listed items in the entity.

In particular, the agent automatically corrects misspellings and expands its matching logic semantically. For example, the agent can automatically match "softball" to "baseball".

## Be creative with regular expressions

Sometimes, entity extraction from a user query can lead to confusion for the NLU model, especially when several entities of the same type are present in the user query.

For example, if a user says: "can you bring 2 towels and 1 pillow to room 101"

Using the built-in *Number* entity would be confused between 2, 1, and 101. To work around this confusion, you could define the following regular expression entities:

- Towel Quantity: [1-9] towel
- Pillow Quantity: [1-9] pillow
- Room Number: [0-9]{3}

## Alternatives to entities to store referential data

For large or evolving datasets (for example a list of products or customers), instead of using Copilot Studio closed list entities, you might want to check external sources. You need to pass the user utterance to the external service using a Power Automate cloud flow.

Your topic logic can then validate the accuracy of the result (or ask the user to validate it) before moving forward in the conversation.

Dataverse is a good candidate to store such data, as it has a built-in [Dataverse Search](#) capability that supports fuzzy matching to return the best results along with a confidence score. Even if searching with a full sentence, it's able to retrieve potential matches.

 **Tip**

To review a sample implementation, see [Return a list of results](#).

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

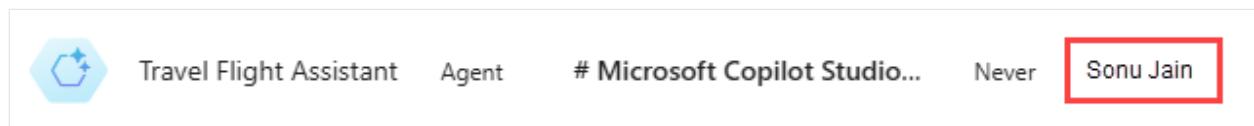
# Reassign ownership of an agent

Article • 11/19/2024

When the original owner of an agent is no longer available, due to resignation or role changes, it can be challenging to edit and manage the agent effectively. This guide walks you through the process of reassigning ownership of an agent to a new user.

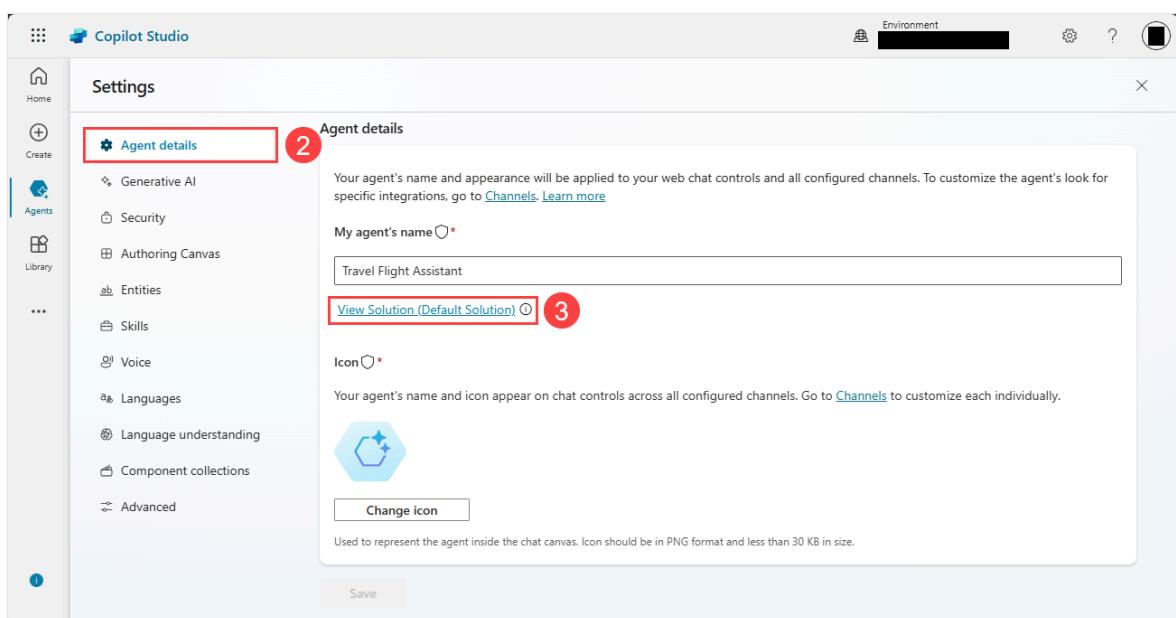
## Current agent owner

In this example, the current owner is *Sonu Jain*.



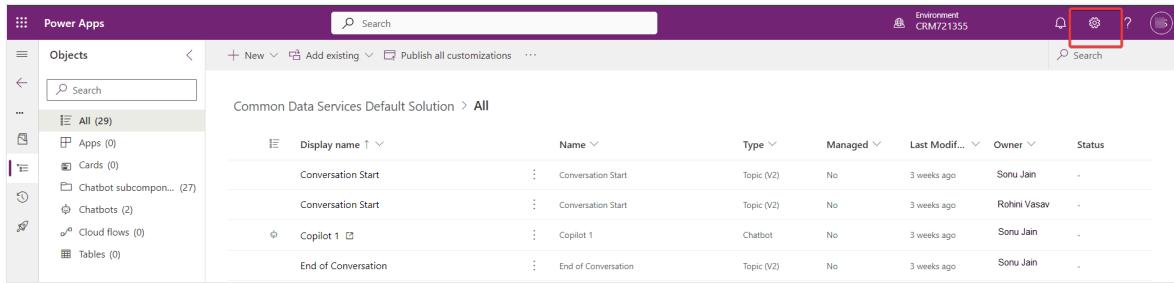
## Set up reassignment

1. Obtain the Dataverse URL. The DataVerse URL typically follows a format similar to:  
`<https://<organization_name>.crm.dynamics.com.>`
2. Open the agent that you intend to reassign ownership, then select **Settings > Agent details**.
3. Select **View Solution** to launch Power Apps, which is the platform used to manage agent.



## Access Advanced settings

## 1. In Power Apps, select the **Settings** icon.



The screenshot shows the Microsoft Power Apps interface. On the left, there's a navigation pane with 'Objects' selected, showing categories like 'All (29)', 'Apps (0)', 'Cards (0)', 'Chatbot subcomponent (27)', 'Chatbots (2)', 'Cloud flows (0)', and 'Tables (0)'. The main area displays a table titled 'Common Data Services Default Solution > All' with columns for 'Display name', 'Name', 'Type', 'Managed', 'Last Modif...', 'Owner', and 'Status'. Four rows are listed: 'Conversation Start' (Topic (V2), No, 3 weeks ago, Sonu Jain, -), 'Conversation Start' (Topic (V2), No, 3 weeks ago, Rohini Vasav, -), 'Copilot 1' (Chatbot, No, 3 weeks ago, Sonu Jain, -), and 'End of Conversation' (Topic (V2), No, 3 weeks ago, Sonu Jain, -). The top right corner of the interface has a gear icon, which is highlighted with a red box in the screenshot.

## 2. From the dropdown menu, select **Advanced settings**.

### ⓘ Note

The URL should follow this structure: <DATAVERSE\_URL>/main.aspx?forceUCI=1&pagetype=entitylist&etn=bot

The <DATAVERSE\_URL> should look like this:

<https://org21dabeee.crm.dynamics.com/>



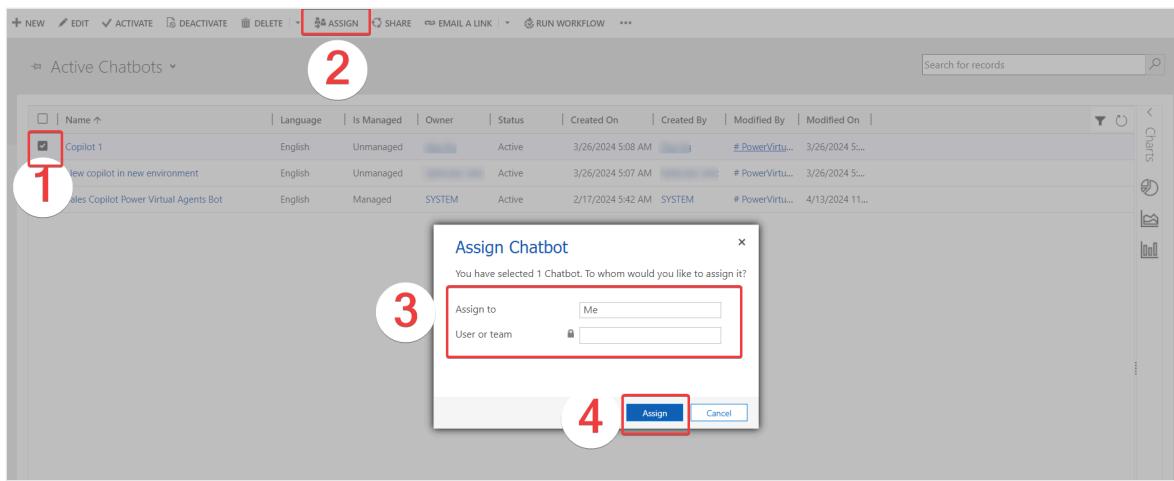
A screenshot of a web browser window. The address bar shows the URL: `https://org21dabeee.crm.dynamics.com/main.aspx?settingsonly=true#929962416`. The portion of the URL after the question mark, `?settingsonly=true`, is highlighted with a red box. Below the address bar, the page content displays the URL `forceUCI=1&pagetype=entitylist&etn=bot`.

## Assign agent to new user

### ⓘ Note

Before transferring ownership, ensure that the new user has the necessary permissions to manage the agent effectively.

1. Select the active agent.
2. Select **Assign** to launch the **Assign Chatbot** dialog.
3. Choose the new owner.
4. Select **Assign** to complete the ownership transfer.



## New agent owner

Now the new owner is *Mihaly Fodor*.

A screenshot of the Microsoft Copilot Studio Agent details page. At the top, there is a navigation bar with a hexagonal icon, the text 'Travel Flight Assistant', 'Agent', '# Microsoft Copilot Studio...', 'Never', and 'Mihaly Fodor'. The 'Mihaly Fodor' text is highlighted with a red box. Below this is a purple note box with the following content:

### Note

In agent Studio for Teams, any owner of the team should also be an admin of the agent Studio agent. For more information, see [Share agents with other users](#).

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Using Adaptive Cards in Copilot Studio

Article • 06/04/2024

Adaptive Cards are platform-agnostic UI snippets written in JSON, which apps and services can openly exchange. When delivered to a specific app (often called the "host"), the JSON is transformed into native UI that automatically adapts to its surroundings. This helps to design and integrate lightweight UI for all major platforms and frameworks.

Adaptive Cards are especially appreciated in products like Microsoft Teams and Outlook, because the designer of the UI can't predict whether the user is in dark or light mode. In either case, no worries! The Adaptive Card just adapts!

## ⓘ Note

For more information about adaptive cards schema, see [Adaptive Cards ↗](#).

Adaptive Cards are often used in Copilot Studio to validate information from the customer, to ask questions, or simply to enhance the conversation experience. There are several integration points where Adaptive Cards are valid choices, as outlined in the following articles.

**Demo Request**

Which product demo are you interested in?

Office 365  
 Dynamics 365  
 Azure Services  
 Power Platform

**Submit**

## Related content

- [Adaptive Cards documentation ↗](#)
- [Copilot Studio documentation ↗](#)

# Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Ask questions with Adaptive Cards

Article • 12/03/2024

In Copilot Studio, you can use Adaptive Cards to pose questions to the agent user. In this case, you're essentially creating a dialog that users can interact with, resulting in variables you can reuse later. The Adaptive Card node allows you to send an Adaptive Card containing one or more submit buttons, and optionally one or more form input fields. Copilot Studio then stores the user responses in variables for use later in the conversation.

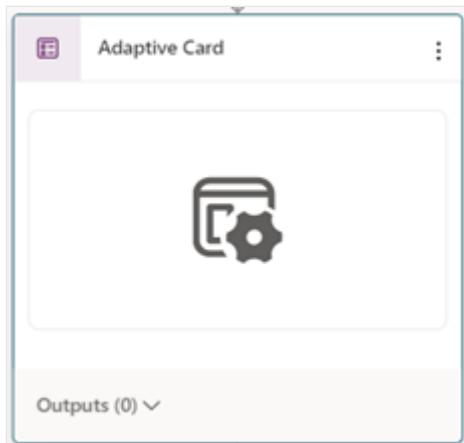
The screenshot shows a rectangular adaptive card with a light gray background and a thin black border. At the top, the title "Demo Request" is displayed in bold black font. Below the title, the question "Which product demo are you interested in?" is shown in a smaller, regular black font. Underneath the question, there is a list of four items, each preceded by an empty square checkbox: "Office 365", "Dynamics 365", "Azure Services", and "Power Platform". At the bottom of the card is a large, rectangular "Submit" button with a thin black border and blue text.

This functionality is best used in scenarios where you'd like to control the look and feel of a set of menu options for selection, offer a form to fill out, or when a combination of links and images around those things could help the customer complete the task at hand.

For instructions on asking questions with an Adaptive Card, see [Ask with Adaptive Cards](#).

## Update the JSON of an Adaptive Card Node properties

When adding a node, the card starts blank, with basic JSON.



Click the **Open Adaptive Card designer** link to create a JSON payload you can then copy and paste back into the card.

**Adaptive Card Node properties** X

[Open Adaptive Card designer](#)

`</> Edit JSON` ↻

```
{  
  "type": "AdaptiveCard",  
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",  
  "version": "1.0",  
  "body": []  
}
```

Below is a JSON example of a card with a demo request.

JSON

```
{  
  "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",  
  "type": "AdaptiveCard",  
  "version": "1.0",  
  "body": [  
    {  
      "type": "ColumnSet",  
      "columns": [  
        {  
          "type": "Column",  
          "width": 2,  
          "items": [  
            {  
              "type": "TextBlock",  
              "text": "Demo Request",  
              "weight": "Bolder",  
              "id": "Title",  
              "size": "ExtraLarge"  
            },  
            {  
              "type": "TextBlock",  
              "text": "This is a demo request.  
              It is intended to demonstrate how  
              adaptive cards can be used in  
              various scenarios.  
              Please ignore this message."  
            }  
          ]  
        }  
      ]  
    }  
  ]  
}
```

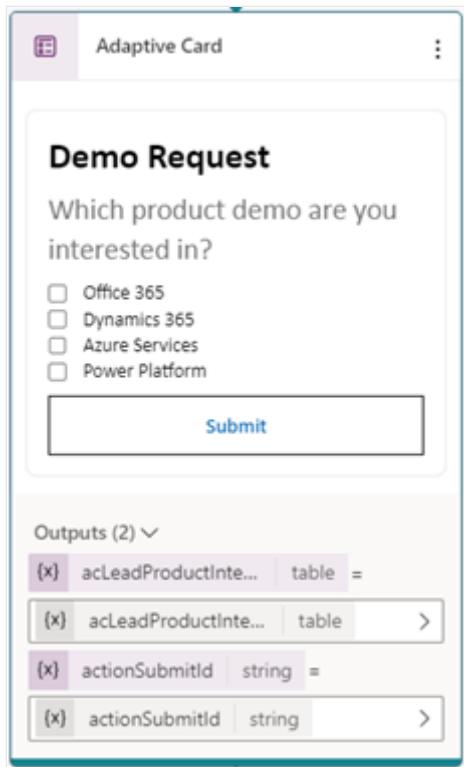
```

        "text": "Which product demo are you
interested in?",

            "isSubtle": true,
            "wrap": true,
            "id": "acInstructions",
            "size": "Large"
        }
    ]
}
],
{
    "type": "Input.ChoiceSet",
    "placeholder": "Placeholder text",
    "choices": [
        {
            "title": "Office 365",
            "value": "Office 365"
        },
        {
            "title": "Dynamics 365",
            "value": "Dynamics 365"
        },
        {
            "title": "Azure Services",
            "value": "Azure Services"
        },
        {
            "title": "Power Platform",
            "value": "Power Platform"
        }
    ],
    "style": "expanded",
    "id": "acLeadProductInterest",
    "isMultiSelect": true
},
],
"actions": [
    {
        "type": "Action.Submit",
        "title": "Submit"
    }
]
}

```

The JSON example produces a card like the following:

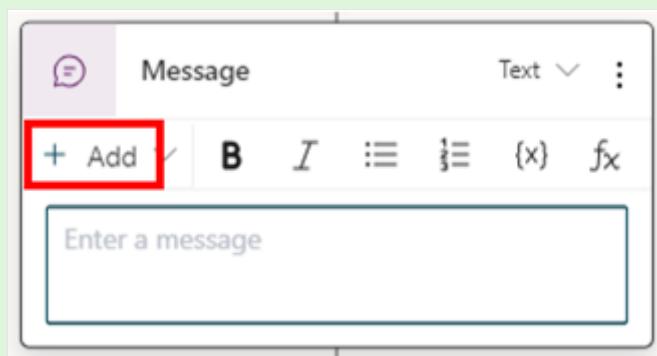


## 💡 Tip

The following resources are often used to create new Adaptive Cards:

- Adaptive Card IO schema documentation and samples.
- Previous agent topics (clone and revise previous cards to maintain consistency).
- ChatGPT or Copilot in Bing to generate the JSON.
- Power Apps or Power Automate (to design the card).

Using **Ask with an Adaptive Card** is *not* recommended if the customer will not directly interact with the card (such as *submit* information). If your Adaptive Card does not include a **Submit** button, and is only for display purposes, then it is recommended to use a Message Attachment, as shown in the following image:



---

# Feedback

Was this page helpful?

 Yes

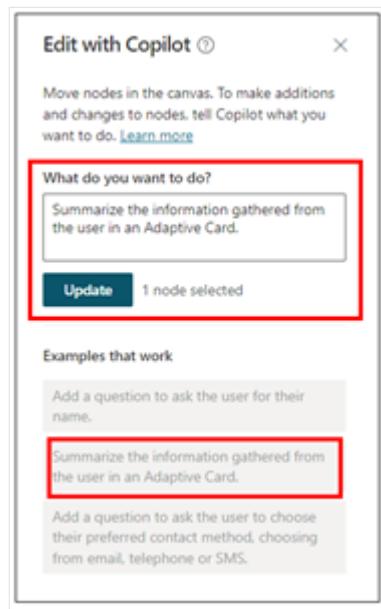
 No

Provide product feedback ↗

# Summarize responses with Adaptive Cards

Article • 11/19/2024

In Copilot Studio, you can use Adaptive Cards to summarize user responses using the embedded [AI for Makers](#). This embedded Copilot can help you to build the card, and incorporate the variables collected from previously answered questions.



This functionality is especially helpful to summarize answers provided by the user within a [topic](#). In these scenarios, the Adaptive Card node immediately follows the questions asked. This allows the agent to validate the information collected from the user, and to ensure it agrees with the information that's been collected from them. For example, it can be a best practice to validate order information before submitting it to a database or a Power Automate flow, to reduce the probability of errors in the processing of the request.

## Add a summary response

1. Open the topic where the questions have been asked, and select the new node immediately following them.
2. Open the embedded Copilot by clicking on the glimmer symbol at the top right of the topic canvas.
3. Scroll down to review the options for what the embedded Copilot can do.  
Remember, this Copilot is designed for the agent maker and is never seen by users.

4. Select **Summarize the information gathered from the user in an Adaptive Card** and then select the **Update** button in the prompt area.
5. If you see red errors generated after adding the Adaptive Card in the right panel, follow the instructions in this [video ↗](#). Otherwise, go to the next step.
6. Expand the formula panel using the double arrow in the corner.
7. Revise any values that are variables, converting them to formulas for dynamic responses.

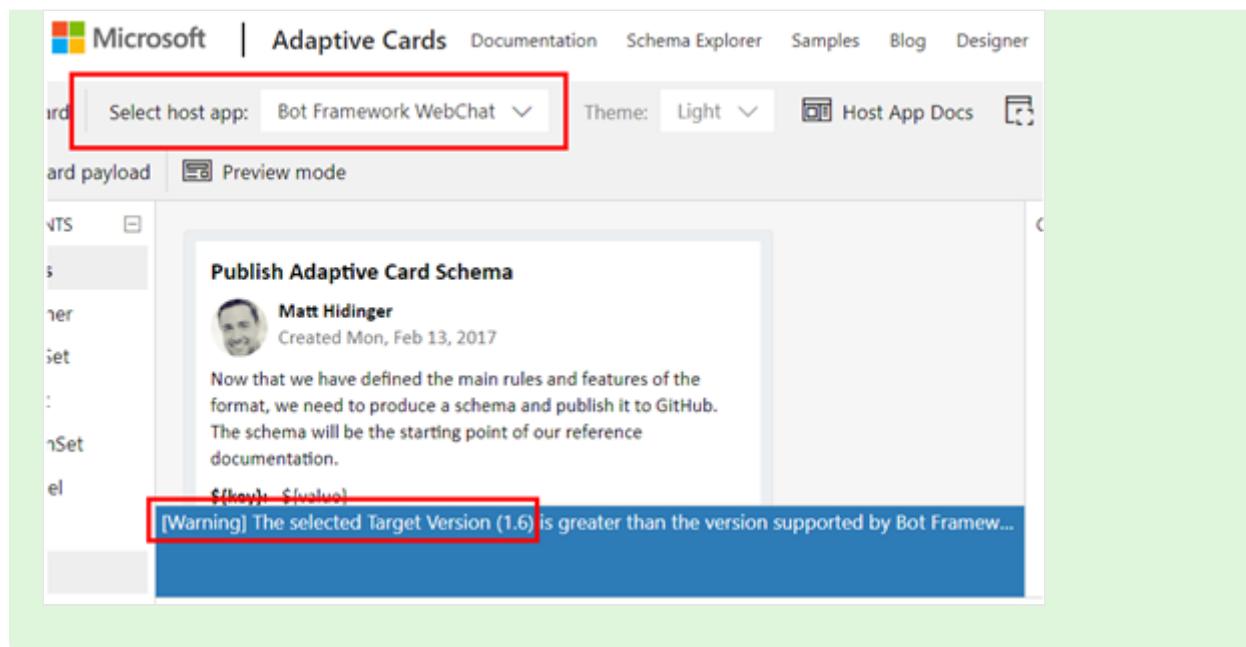
 **Tip**

The embedded agent provides the sample formulas, which you can reuse. However, remove the quotation marks around them. You can test if they're working as dynamic formulas by removing and adding back the dot after "Topic." If you see Intellisense showing all the possible variables, then you've succeeded in converting the strings to formulas. Example: Removing the quotes surrounding the variable `"Text(Topic.FullName)"` converts to a formula like `Text(Topic.FullName)` or `Topic.FullName`.

8. Test your agent.

 **Tip**

- Once you begin editing in the formula panel, you won't be able to reuse the original JSON code. To allow iterative design and changes, it's recommended to save the original JSON in your own notes, or as a comment in the node. This allows you to revert changes, if needed.
- You can further enhance the Adaptive Card with images and schema changes appropriate for Bot Framework WebChat hosts. For more information on supported formats and schema, see [Adaptive Cards ↗](#).
- If you enhance the Adaptive Card with other components, please ensure that the version is appropriate for the Bot Framework. You might see a **Warning** message under the card preview (as shown in the screenshot) when working and previewing in [Adaptive Cards ↗](#). If this appears, decrease the version number by .1. For example, if the setting is 1.6, change it to 1.5.



## Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#)

# Use Adaptive Cards to display carousels

Article • 08/26/2024

Adaptive Cards can be formatted to act like a carousel, where users can interactively cycle through messages. Users interact with a carousel by clicking on arrows, allowing them to scroll through many choices, as seen in the following image.



Carousels are helpful in scenarios where Adaptive Card menu options are lengthy, requiring a full sentence or short paragraph for the user to fully understand their selection options.

In the following example, a carousel is built to present our products to customers on the Contoso website. The carousel helps customers select the product they find most interesting, as part of the conversation flow.

## ⓘ Note

The icons and descriptions are for demonstration purposes, only. However, you can clone and revise the code samples for your own scenario.

## Build a carousel with Adaptive Cards

## ⓘ Note

The steps in this section of the article contain the following caveats:

- This example uses an input JSON data array as a string variable to simplify the conversation flow.
- It also uses example URLs for the icons to display the product icons and left/right arrows. However, this isn't required. You can also reference SVG files for media. For best results when referencing media, store the media in a static, unauthenticated location.

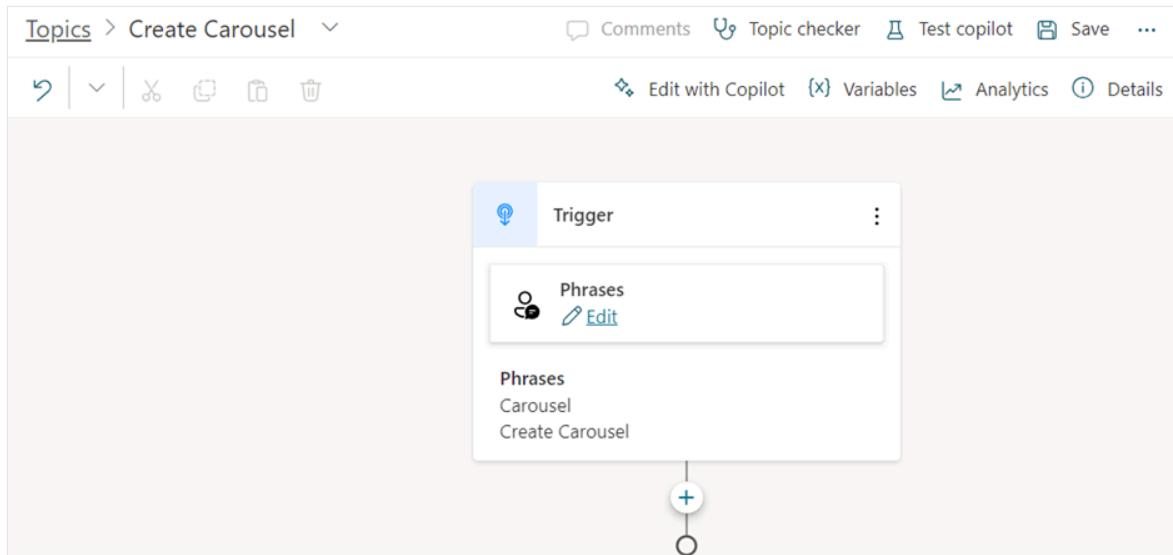
- For additional information regarding the functionality in this article, see the following:
  - [Ask with Adaptive Cards](#)
  - [Create expressions using Power Fx \(preview\)](#)
  - [Use variables to carry content across topics](#)
  - [AdaptiveCards sample YAML file for topic examples ↗](#)

1. In Copilot Studio, create a new topic called "Create Carousel."

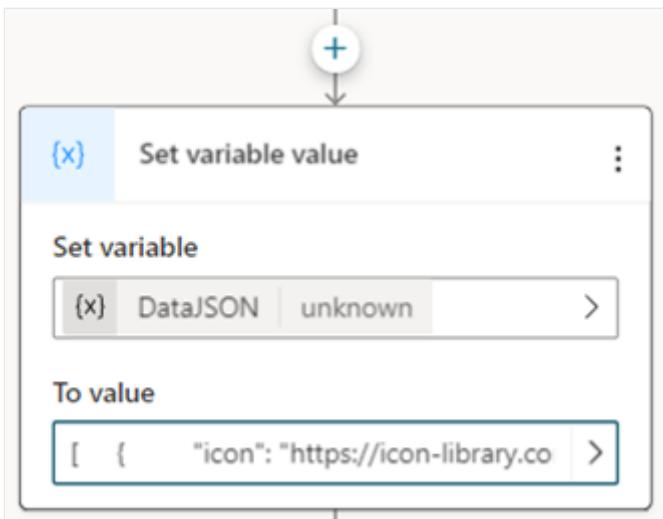
**!** Note

For more information on creating a topic, see [Create and edit topics](#).

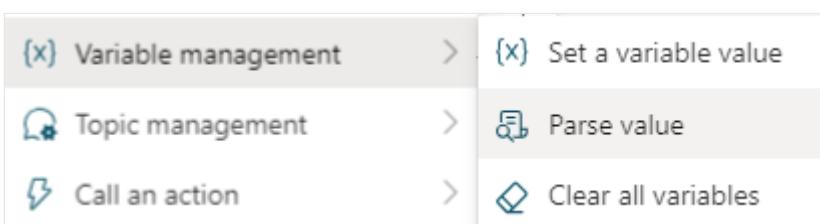
2. Use the words "Carousel" and "Create Carousel" for the trigger phrases.



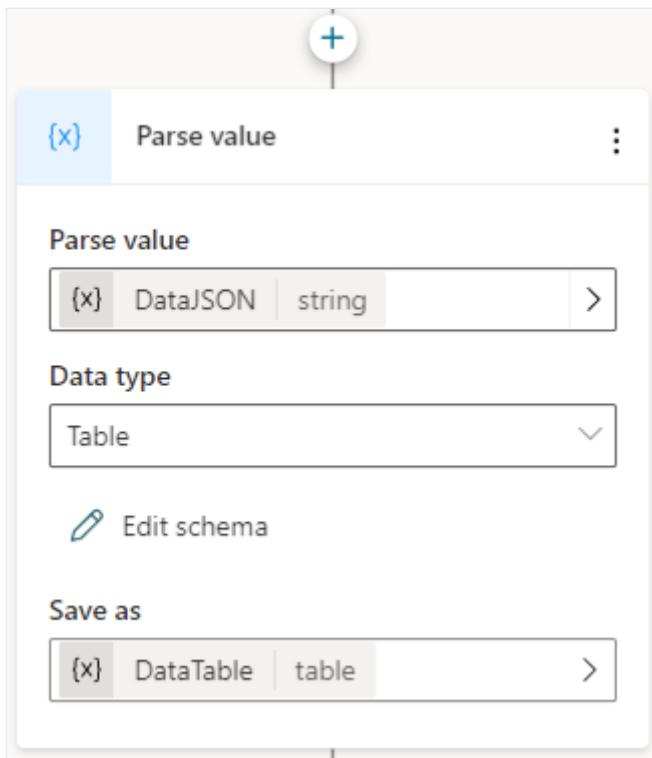
3. Press the + sign, select **Variable management** from the drop-down menu, and then select **Set a variable value**.
4. Create a new variable named "DataJSON." This variable is a string, and holds the JSON that describes the components of the Adaptive Card.
5. Copy and then paste the [DataJSON code block](#) into the **To value** field of the new node.



6. To leverage the response from follow-up nodes, the JSON array must be parsed. The parsing transforms the JSON array into an adaptive card for the customer.
7. Press the + sign under the recently created variable, then select **Variable management**, followed by **Parse value**. This is used to add another node to parse the previous variable.



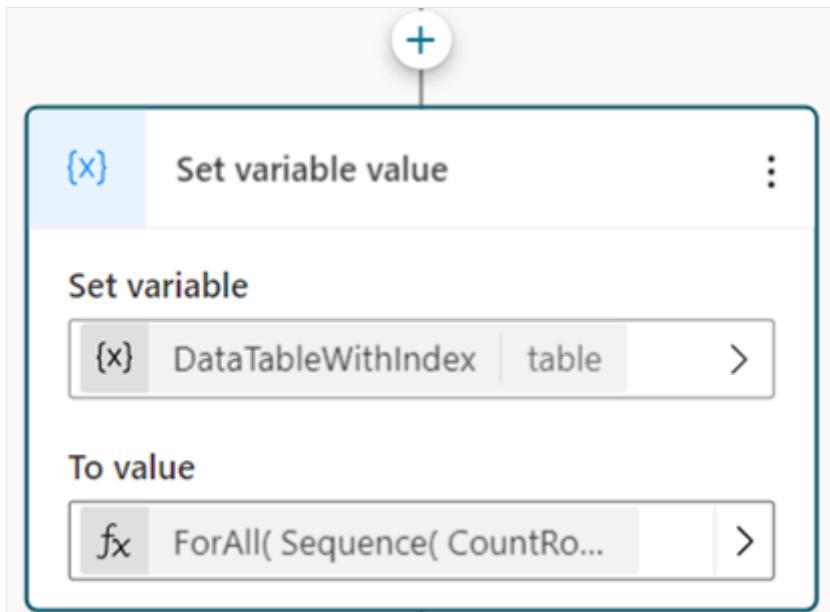
8. Choose the variable from the previous node as the **Parse value**.
9. Create another custom variable. This variable is a table named "DataTable." This variable holds the parsed results in the "Save as" field as a record.



### 💡 Tip

It's recommended that you determine the data-type based on the sample payload. Otherwise, it returns an empty schema.

10. Create an indexed table to be used by the Adaptive Card. Press the + sign and add another **Set a variable value** node. This is another custom variable, named "DataTableWithIndex." An index is added with the Sequence function to provide predictable, individual IDs for each record so the previous and next arrows of the carousel display or hide the correct record.

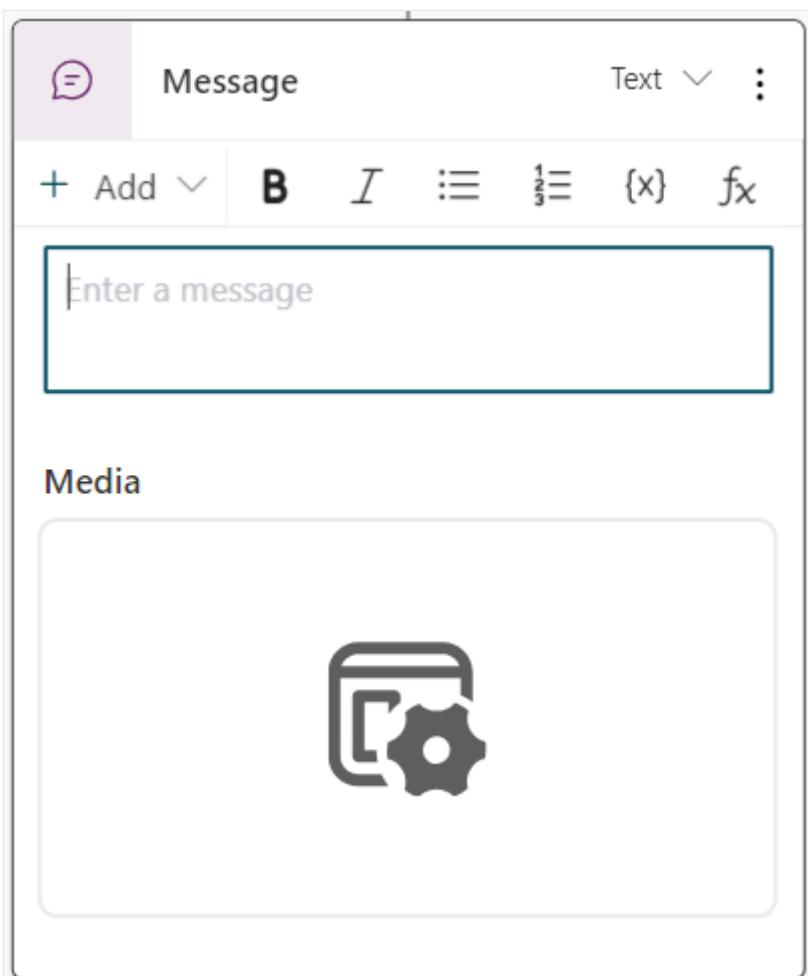


11. Paste the following formula code snippet into the **To value** field. The code snippet creates a JSON array for the display of each carousel window.

```
JSON

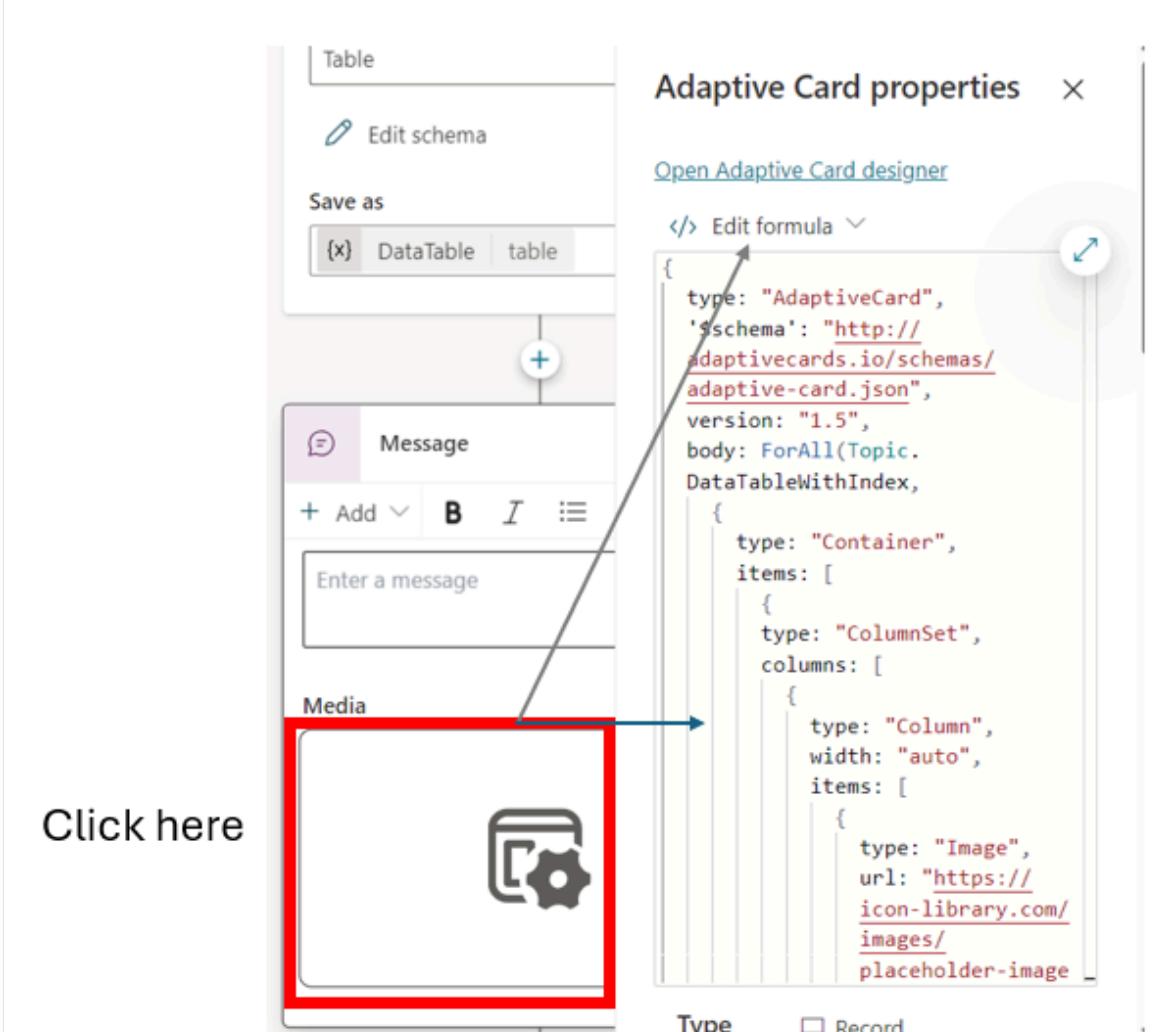
ForAll(
    Sequence(
        CountRows(Topic.DataTable)
    ),
    {
        ItemNumber: Value,
        Name: Index(Topic.DataTable, Value).name,
        Description: Index(Topic.DataTable, Value).description,
        ImageURL: Index(Topic.DataTable, Value).icon,
        URL: Index(Topic.DataTable, Value).linkurl,
        Label: Index(Topic.DataTable, Value).linklabel
    }
)
```

12. Add a message with the Adaptive Card attachment to the bottom of the topic nodes. Press the + sign under the last node and select **Send a message**.



13. Press the **Add** chevron at the top left and select **Adaptive Card**.

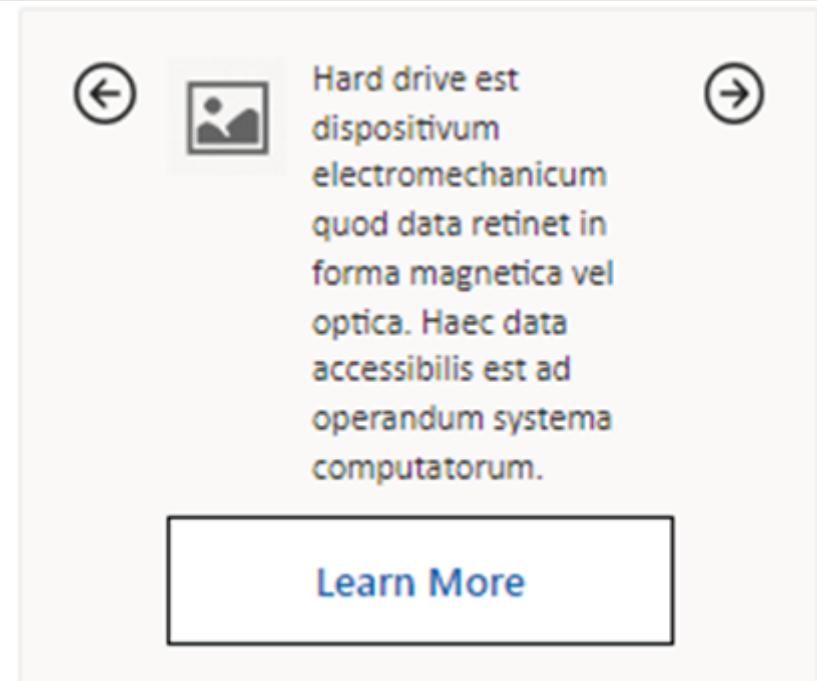
14. Select the **Media** section of the new card and change the properties area to formula. For more information about the formula, see [Adaptive Card formula](#).



The screenshot shows the Microsoft Bot Framework's Adaptive Card designer. On the left, there's a preview window titled "Table" showing a card with a "Message" section containing a text input field and a "Media" section with a placeholder image icon. A red box highlights the "Media" section. On the right, the "Adaptive Card properties" panel is open, displaying the JSON formula for the card. An arrow points from the "Media" section in the preview to the corresponding "Image" element in the JSON code.

```
type: "Image",  
url: "https://icon-library.com/images/placeholder-image"
```

15. Trigger the "Create Carousel" intent to test the experience.



The screenshot shows a carousel card with one item. The item contains a back arrow icon, an image icon, and a block of Latin text describing a hard drive as an electromechanical device that retains data in magnetic form. At the bottom is a "Learn More" button.

Hard drive est  
dispositivum  
electromechanicum  
quod data retinet in  
forma magnetica vel  
optica. Haec data  
accessibilis est ad  
operandum sistema  
computatorum.

Learn More

# DataJSON code block

Within the following JSON code block, there are repeated keys for "icon," "name," "description," "linkurl," and "linklabel." These keys represent each navigation point in the card. In the example, there are five products to browse between. If there are too many keys for your scenario, delete the unnecessary keys. If there aren't enough, copy and paste the keys as needed.

## Details on the keys and values

- icon: The key for the link to the displayed icon to the left of the carousel window.
  - name: The key for the label of the product name, not displayed in the Adaptive Card.
  - description: The key for the verbiage that describes the product.
  - linkurl: The key for the hyperlink associated with the product or selection. This hyperlink is intended to be a direct citation to where the customer should be directed.
  - linklabel: The key for the text displayed above the hyperlink.

## JSON code block

```

    "icon": "https://icon-library.com/images/placeholder-image-icon-12.jpg",
    "name": "Productum 3",
    "description": "Pluteus librorum est structura lignea vel metallica quae libros et alia volumina continet. Altitudine non amplius quam decem pedes, pluteus potest habere plures gradus vel tabulas ad libros exponendos. Hoc loco maximam aesthetiam et ordinem litterarum praestat.",
    "linkurl": "http://www.microsoft.com/",
    "linklabel": "Learn More"
},
{
    "icon": "https://icon-library.com/images/placeholder-image-icon-11.jpg",
    "name": "Productum 4",
    "description": "Servitium conficiendi et transmittendi tortarum ad nuptias, quae formas et delicias unicas offert, sub vices quam viginti quinque verbis descriptum.",
    "linkurl": "http://www.microsoft.com/",
    "linklabel": "Learn More"
},
{
    "icon": "https://icon-library.com/images/placeholder-image-icon-10.jpg",
    "name": "Productum 5",
    "description": "Bot GPT in correctione textus praecellit, grammaticam scrutans, errores corrigens, et textum meliorem reddens, sub vices quam viginti quinque verbis.",
    "linkurl": "http://www.microsoft.com/",
    "linklabel": "Learn More"
}
]

```

## Adaptive Card formula

For reference purposes, the following JSON defines the Adaptive Card formula:

JSON

```
{
  type: "AdaptiveCard",
  '$schema': "http://adaptivecards.io/schemas/adaptive-card.json",
  version: "1.5",
  body: ForAll(Topic.DataTableWithIndex,
  {
    type: "Container",
    items: [
      {
        type: "ColumnSet",
        columns: [
          {
            type: "Column",
            width: "auto",
            items: [

```

```
{
    type: "Image",
    url: "https://icon-library.com/images/placeholder-image-
icon/placeholder-image-icon-14.jpg"
}
],
selectAction: {
    type: "Action.ToggleVisibility",
    title: "Preview",
    targetElements: [
        "card" & If(ItemNumber = 1,
Last(Topic.DataTableWithIndex).ItemNumber, ItemNumber - 1),
        "card" & ItemNumber
    ]
},
{
    type: "Column",
    width: "stretch",
    items: [
        {
            type: "Container",
            items: [
                {
                    type: "ColumnSet",
                    columns: [
                        {
                            type: "Column",
                            width: "auto",
                            items: [
                                {
                                    type: "Image",
                                    url: ImageURL,
                                    size: "Small"
                                }
                            ]
                        },
                        {
                            type: "Column",
                            width: "stretch",
                            items: [
                                {
                                    type: "TextBlock",
                                    text: Description,
                                    wrap: true,
                                    size: "Small"
                                }
                            ]
                        }
                    ]
                }
            ]
        },
        {
            type: "ActionSet",
            items: [
                {
                    type: "Action.ToggleVisibility",
                    title: "Preview",
                    targetElements: [
                        "card" & If(ItemNumber = 1,
Last(Topic.DataTableWithIndex).ItemNumber, ItemNumber - 1),
                        "card" & ItemNumber
                    ]
                }
            ]
        }
    ]
}
```

```

        actions: [
            {
                type: "Action.OpenUrl",
                title: Label,
                url: URL
            }
        ]
    ],
{
    type: "Column",
    width: "auto",
    items: [
        {
            type: "Image",
            url: "https://icon-library.com/images/placeholder-image-icon/placeholder-image-icon-14.jpg"
        }
    ],
    selectAction: {
        type: "Action.ToggleVisibility",
        title: "Next",
        targetElements: [
            "card" & ItemNumber,
            "card" & If(ItemNumber =
Last(Topic.DataTableWithIndex).ItemNumber, 1, ItemNumber + 1)
        ]
    }
},
],
minHeight: "1px",
spacing: "None",
id: "card" & ItemNumber,
bleed: true,
isVisible: If(ItemNumber = 1, true, false)
}
)
}

```

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# Add feedback for every response

Article • 11/19/2024

As an agent owner, obtaining feedback from users is a vital step in improving the effectiveness and reliability of your conversational agent. You gain valuable insights into how well your agent is performing, identify areas for enhancement, and deliver a more satisfying user experience.

In this article, you learn how to use Adaptive Cards to collect user feedback after every interaction with your agent without annoying the user. Implementing this feedback mechanism allows you to assess and enhance the quality of agent responses while ensuring user satisfaction. The feedback mechanism works for a customer support agent, an informational assistant, or any other type of conversational agent. For this tutorial, we use a "Generative Answer," but this approach can be used for any response type as well.

The final output of this tutorial shows a message after the agent's response that says, "Generated answer, please rate it."

## Add a generative answer node

1. Open the authoring canvas for your topic and select the plus icon to open the new node menu.
2. Under **Advanced**, select **Generative answers**.
3. A new **Create generative answers** node is added to your topic, and the Properties pane is automatically opened.
4. Select **Add knowledge** and add a public website, such as  
`https://learn.microsoft.com/`.
5. Under **Advanced**, create a **global variable**. Enter a meaningful name for the variable, such as `VarStoreAnswer`. This variable stores the generative answers generated in response to user queries.
6. Clear the **Send a message** box.

The screenshot shows the Microsoft Bot Framework designer interface. On the left, a node titled "Create generative answers" is selected. It has an "Input" section with a text input field "Enter or select a value". Below it is a "Data sources" section with a "Edit" button. A blue plus sign icon is positioned above the node, and a white circle icon is below it. On the right, under the heading "Latency Message", there is a note: "In text conversations this message will be sent once. In voice conversations, this message will loop until the operation is complete." Below this are two options: "Send a message" (unchecked) and "Advanced" (expanded). Under "Advanced", there is another "Send a message" option (unchecked), followed by "Save LLM response" (set to "Text only") and "Save bot response as" (set to "{x} VarStoreAnswer string"). The "Save bot response as" field is highlighted with a red border.

## Add a question or message node

1. Use either a question or message node to add the adaptive card.

**Adaptive Card properties**

[Open Adaptive Card designer](#)

**Edit formula**

```
{
  type: "AdaptiveCard",
  '$schema': "http://adaptivecards.io/schemas/adaptive-card.json",
  version: "1.5",
  body: [
    {
      type: "Container",
      items: [
        {
          type: "TextBlock",
          text: Global.VarStoreAnswer,
          wrap: true
        }
      ]
    },
    {
      type: "Container",
      items: [
        {
          type: "ColumnSet",
          columns: [
            {
              type: "Column",
              width: "100px"
            }
          ],
          children: [
            {
              type: "TextBlock",
              text: "Enter a message"
            }
          ]
        }
      ]
    }
  ]
}
```

Type  Record

Output  {

```
type: "AdaptiveCard",
'$schema': "http://adaptivecards.io/schemas/adaptive-card.json"
```

2. Paste the following JSON to the Adaptive Edit Formula.

JSON

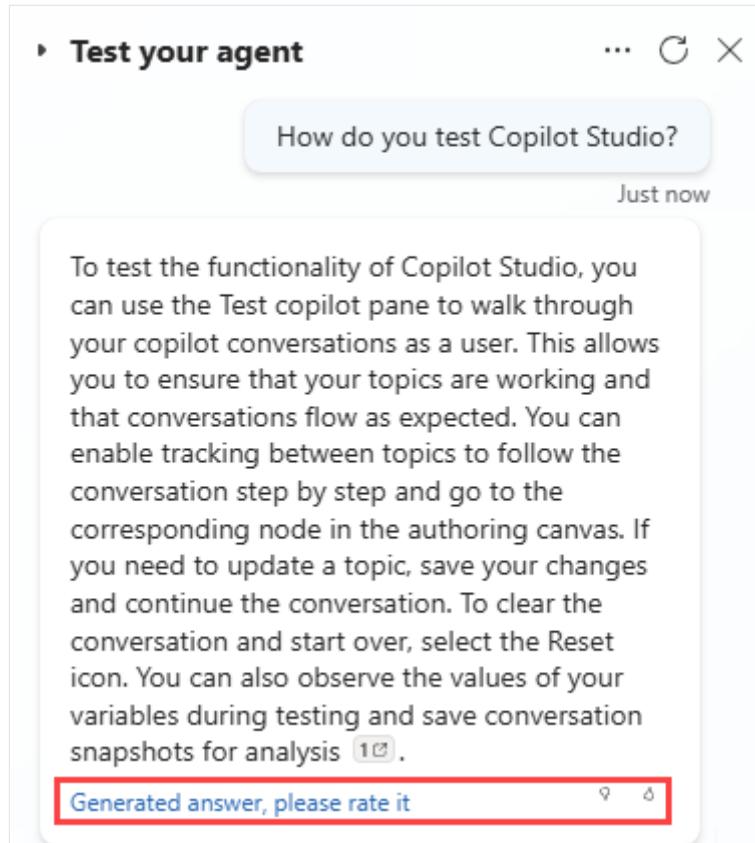
```
{
  type: "AdaptiveCard",
  '$schema': "http://adaptivecards.io/schemas/adaptive-card.json",
  version: "1.5",
  body: [
    {
      type: "Container",
      items: [
        {
          type: "TextBlock",
          text: Global.VarStoreAnswer,
          wrap: true
        }
      ]
    }
  ]
}
```



```
        type: "Action.Submit",
        data: "This generated answer wasn't useful"
    }
}
],
{
    type: "Column",
    width: "5",
    items: [
    {
        type: "Image",
        url:
"data:image/svg+xml;utf8,%3Csvg%20width%3D%2224%22%20height%3D%2224%22%20viewBox%3D%220%200%2024%2024%22%20fill%3D%22none%22%20xmlns%3D%22http%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%22%3E%0A%20%20%3Cpath%20d%3D%22M16.4996%205.20259C16.4996%202.76065%2015.3595%201.00391%2013.4932%201.00391C12.467%201.00391%2012.1149%201.60527%2011.747%203.00348C11.6719%203.2923%2011.635%203.43297%2011.596%203.57157C11.495%203.93031%2011.3192%204.54106%2011.069%205.40258C11.0623%205.42566%2011.0524%205.44741%2011.0396%205.46749L8.17281%209.95315C7.49476%2011.0141%206.49429%2011.8296%205.31841%2012.2798L4.84513%2012.461C3.5984%2012.9384%202.87457%2014.2421%203.1287%2015.5527L3.53319%2017.6388C3.77462%2018.8839%204.71828%2019.8748%205.9501%2020.1767L13.5778%2022.0462C16.109%2022.6666%2018.6674%2021.1317%2019.3113%2018.6064L20.7262%2013.0572C21.1697%2011.3179%2020.1192%209.54845%2018.3799%209.10498C18.1175%209.03807%2017.8478%209.00422%2017.5769%209.00422H15.7536C16.2497%207.37133%2016.4996%206.11155%2016.4996%205.20259ZM4.60127%2015.2672C4.48576%2014.6715%204.81477%2014.0788%205.38147%2013.8619L5.85475%2013.6806C7.33036%2013.1157%208.58585%2012.0923%209.43674%2010.7609L12.3035%206.27526C12.3935%206.13437%2012.4629%205.98131%2012.5095%205.82074C12.7608%204.95574%2012.9375%204.34175%2013.0399%203.97786C13.083%203.82461%2013.1239%203.66916%2013.1976%203.38519C13.3875%202.66348%2013.4809%202.50391%2013.4932%202.50391C14.3609%202.50391%2014.9996%203.48797%2014.9996%205.20259C14.9996%206.08708%2014.6738%207.53803%2014.0158%209.51766C13.8544%2010.0032%2014.2158%2010.5042%2014.7275%2010.5042H17.5769C17.7228%2010.5042%2017.868%2010.5224%2018.0093%2010.5585C18.9459%2010.7973%2019.5115%2011.7501%2019.2727%2012.6866L17.8578%2018.2357C17.4172%2019.9636%2015.6668%2021.0138%2013.9349%2020.5893L6.30718%2018.7198C5.64389%2018.5572%205.13577%2018.0237%205.00577%2017.3532L4.60127%2015.2672Z%22%20fill%3D%22%23212121%22%20%2F%3E%0A%3C%2Fsvg%3E",
        width: "15px",
        height: "15px",
        selectAction: {
            type: "Action.Submit",
            data: "This generated answer was useful"
        }
    }
]
}
]
```

```
]  
}
```

3. At the end of the response, you can give an option to the user to rate every response.



To capture the thumbs up and down, trigger the specific topic to get more feedback. Based on up or down selections, the topic catches phrases.

Here's the YAML file to redirect to this topic to capture those responses.

YAML

```
kind: AdaptiveDialog
beginDialog:
  kind: OnRecognizedIntent
  id: main
  condition: !=IsBlank(Global.VarStoreAnswer)
  intent:
    displayName: Untitled

  actions:
    - kind: SendActivity
      id: sendActivity_XauhTN
      activity:
        attachments:
          - kind: AdaptiveCardTemplate
            cardContent: |-
              ={
```

```
        type: "AdaptiveCard",
        '$schema': "http://adaptivecards.io/schemas/adaptive-
card.json",
        version: "1.5",
        body: [
            {
                type: "Container",
                items: [
                    {
                        type: "TextBlock",
                        text: Global.VarStoreAnswer,
                        wrap: true
                    }
                ]
            },
            {
                type: "Container",
                items: [
                    {
                        type: "ColumnSet",
                        columns: [
                            {
                                type: "Column",
                                width: "90",
                                items: [
                                    {
                                        type: "TextBlock",
                                        text: "Generated answer, please rate
it",
                                        wrap: true,
                                        size: "Small",
                                        color: "Accent"
                                    }
                                ]
                            },
                            {
                                type: "Column",
                                width: "5",
                                items: [
                                    {
                                        type: "Image",
                                        url:
"data:image/svg+xml;utf8,%3Csvg%20width%3D%224%22%20height%3D%224%22%
20viewBox%3D%220%200%204%204%22%20fill%3D%22none%22%20xmlns%3D%22http
%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%22%3E%0A%20%20%3Cpath%20d%3D%22M16.499
6%2017.9852C16.4996%2020.4271%2015.3595%2022.1838%2013.4932%2022.1838C1
2.5183%2022.1838%2012.1518%2021.6411%2011.8021%2020.3881L11.596%2019.61
62C11.495%2019.2574%2011.3192%2018.6467%2011.069%2017.7852C11.0623%2017
.7621%2011.0524%2017.7403%2011.0396%2017.7203L8.17281%2013.2346C7.49476
%2012.1736%206.49429%2011.3581%205.31841%2010.9079L4.84513%2010.7267C3.
5984%2010.2494%202.87457%208.94562%203.1287%207.63505L3.53319%205.54897
C3.77462%204.30388%204.71828%203.31298%205.9501%203.01106L13.5778%201.1
4153C16.109%200.521138%2018.6674%202.05607%2019.3113%204.5814L20.7262%2
010.1306C21.1697%2011.8698%2020.1192%2013.6393%2018.3799%2014.0828C18.1
175%2014.1497%2017.8478%2014.1835%2017.5769%2014.1835H15.7536C16.2497%2
```

```
015.8164%2016.4996%2017.0762%2016.4996%2017.9852ZM4.60127%207.92059C4.4
8576%208.5163%204.81477%209.10893%205.38147%209.3259L5.85475%209.5071C7
.33036%2010.0721%208.58585%2011.0954%209.43674%2012.4268L12.3035%2016.9
125C12.3935%2017.0534%2012.4629%2017.2064%2012.5095%2017.367L13.0614%20
19.2873L13.2731%2020.0786C13.4125%2020.5666%2013.4827%2020.6838%2013.49
32%2020.6838C14.3609%2020.6838%2014.9996%2019.6998%2014.9996%2017.9852C
14.9996%2017.1007%2014.6738%2015.6497%2014.0158%2013.6701C13.8544%2013.
1846%2014.2158%2012.6835%2014.7275%2012.6835H17.5769C17.7228%2012.6835%
2017.868%2012.6653%2018.0093%2012.6293C18.9459%2012.3905%2019.5115%2011
.4377%2019.2727%2010.5012L17.8578%204.952C17.4172%203.22415%2015.6668%2
02.17393%2013.9349%202.59841L6.30718%204.46794C5.64389%204.63051%205.13
577%205.16407%205.00577%205.83451L4.60127%207.92059Z%22%20fill%3D%22%23
212121%22%20%2F%3E%0A%3C%2Fsvg%3E",
width: "15px",
height: "15px",
selectAction: {
  type: "Action.Submit",
  data: "This generated answer wasn't
useful"
}
]
},
{
  type: "Column",
  width: "5",
  items: [
    {
      type: "Image",
      url:
"data:image/svg+xml;utf8,%3Csvg%20width%3D%2224%22%20height%3D%2224%22%20viewBox%3D%220%200%2024%2024%22%20fill%3D%22none%22%20xmlns%3D%22http
%3A%2F%2Fwww.w3.org%2F2000%2Fsvg%22%3E%0A%20%20%3Cpath%20d%3D%22M16.499
6%205.20259C16.4996%202.76065%2015.3595%201.00391%2013.4932%201.00391C1
2.467%201.00391%2012.1149%201.60527%2011.747%203.00348C11.6719%203.2923
3%2011.635%203.43297%2011.596%203.57157C11.495%203.93031%2011.3192%204.
54106%2011.069%205.40258C11.0623%205.42566%2011.0524%205.44741%2011.039
6%205.46749L8.17281%209.95315C7.49476%2011.0141%206.49429%2011.8296%205
.31841%2012.2798L4.84513%2012.461C3.5984%2012.9384%202.87457%2014.2421%
203.1287%2015.5527L3.53319%2017.6388C3.77462%2018.8839%204.71828%2019.8
748%205.9501%2020.1767L13.5778%2022.0462C16.109%2022.6666%2018.6674%202
1.1317%2019.3113%2018.6064L20.7262%2013.0572C21.1697%2011.3179%2020.119
2%209.54845%2018.3799%209.10498C18.1175%209.03807%2017.8478%209.00422%2
017.5769%209.00422H15.7536C16.2497%207.37133%2016.4996%206.11155%2016.4
996%205.20259ZM4.60127%2015.2672C4.48576%2014.6715%204.81477%2014.0788%
205.38147%2013.8619L5.85475%2013.6806C7.33036%2013.1157%208.58585%2012.
0923%209.43674%2010.7609L12.3035%206.27526C12.3935%206.13437%2012.4629%
205.98131%2012.5095%205.82074C12.7608%204.95574%2012.9375%204.34175%201
3.0399%203.97786C13.083%203.82461%2013.1239%203.66916%2013.1976%203.385
19C13.3875%202.66348%2013.4809%202.50391%2013.4932%202.50391C14.3609%20
2.50391%2014.9996%203.48797%2014.9996%205.20259C14.9996%206.08708%2014.
6738%207.53803%2014.0158%209.51766C13.8544%2010.0032%2014.2158%2010.504
%2014.7275%2010.5042H17.5769C17.7228%2010.5042%2017.868%2010.5224%2018
.0093%2010.5585C18.9459%2010.7973%2019.5115%2011.7501%2019.2727%2012.68
66L17.8578%2018.2357C17.4172%2019.9636%2015.6668%2021.0138%2013.9349%20
```

```
20.5893L6.30718%2018.7198C5.64389%2018.5572%205.13577%2018.0237%205.005  
77%2017.3532L4.60127%2015.2672Z%22%20fill%3D%22%23212121%22%20%2F%3E%0A  
%3C%2Fsvg%3E",  
        width: "15px",  
        height: "15px",  
        selectAction: {  
            type: "Action.Submit",  
            data: "This generated answer was  
useful"  
        }  
    }  
}  
]  
}  
]  
}  
]  
}  
}
```

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# Display data from arrays in Adaptive Cards

Article • 09/06/2024

Adaptive Cards are a versatile tool used to create interactive and engaging conversations in Copilot Studio, and can be used to display an array of items. In this article, for simplicity, we use a hard-coded example. However, you would likely get the data from a more dynamic source, like a SharePoint list, by using Power Automate.

The screenshot shows an Adaptive Card with the following content:

**Employee Information**

Employee Name: Alice

Employee ID: E12345

Department: HR

**Tasks**

- Task ID: T001, Description: Review employee benefits, Due Date: 2023-10-15
- Task ID: T002, Description: Conduct new hire orientation, Due Date: 2023-09-30
- Task ID: T003, Description: Update HR policies, Due Date: 2023-11-05

## Initialize a variable with the task list

Let's say that you have a list of tasks in an array, and you want to show the list of tasks in the copilot.

1. Select **Add node (+)** to add a node, and then select **Variable management > Set a variable value**.
2. Select the box under **Set variable**, and then select **Create new**.
3. Select the new variable (for example, `Var1`) to display the **Variable properties** panel.
4. Name your variable something meaningful, such as `EmployeeTaskList`.

5. Paste the following JSON in the **To value** field:

```
JSON

{
    "employeeName": "Alice",
    "employeeID": "E12345",
    "employeeDepartment": "HR",
    "employeeTasks": [
        {
            "taskID": "T001",
            "taskDescription": "Review employee benefits",
            "dueDate": "2023-10-15"
        },
        {
            "taskID": "T002",
            "taskDescription": "Conduct new hire orientation",
            "dueDate": "2023-09-30"
        },
        {
            "taskID": "T003",
            "taskDescription": "Update HR policies",
            "dueDate": "2023-11-05"
        }
    ]
}
```

## Parse the JSON data into a table

This variable management node is used to convert the JSON string into a table that can be used later in the Adaptive Card.

1. Select **Add node (+)** and **Variable management > Parse value**.
2. Under **Parse value**, select the **EmployeeTaskList** variable you created in the previous section.
3. Under **Data type**, select **From sample data**.
4. Select **</> Get schema from sample JSON**, then copy and paste the same JSON into the **From sample data** section of the **Data type** settings. The sample data automatically generates the schema and datatype. Select **Confirm**.
5. Under **Save as**, select **Create a new variable**.
6. Select the new variable, and change the **Variable name** to **TaskTable**.

The screenshot shows the configuration of a 'Parse value' node in Power Automate. The top bar includes an 'X' button, the title 'Parse value', and a '...' button. The main area is titled 'Parse value' and contains a row with '{x}' EmployeeTaskList string >. Below this is a 'Data type' section with a dropdown set to 'From sample data'. Further down are options to 'Get schema from sample JSON' (with a pencil icon and 'Edit schema' link) and 'Save as' (with '{x}' TaskTable record >).

## Display the data in an Adaptive Card

To display the data in an Adaptive Card, a [Send a message](#) is used.

1. Add a **Send a message** node.
2. Select + Add, and select **Adaptive Card** from the dropdown.
3. Select the **Media** section to show the **Adaptive Card properties** panel.
4. Inside the **Adaptive Card properties** panel on the right, select the </> **Edit JSON** dropdown and change it to **Formula**.
5. Paste the following code.

```
JSON
```

```
{
  type: "AdaptiveCard",
  version: "1.5",
  body: [
    {
      type: "TextBlock",
      text: "Employee Information",
      weight: "bolder",
      size: "large"
    },
    {
      type: "TextBlock",
      text: "Employee Name: " & Topic.TaskTable.employeeName,
      separator: true
    },
    {
      type: "TextBlock",
      text: "Employee ID: " & Topic.TaskTable.employeeID,
      separator: true
    },
    {
      type: "TextBlock",
      text: "Department: " & Topic.TaskTable.employeeDepartment,
      separator: true
    },
    {
      type: "TextBlock",
      text: "Tasks",
      weight: "bolder",
      size: "medium",
      separator: true
    },
    {
      type: "Container",
      items:
        ForAll(Topic.TaskTable.employeeTasks,
        {
          type: "TextBlock",
          text: "- Task ID: " & taskID & ", Description: " &
taskDescription & ", Due Date: " & dueDate ,
          wrap: true
        }
      )
    }
  ]
}
```

6. Now we can refer the JSON record properties using expressions like

`Topic.TaskTable.employeeName`.

7. To display array items in an Adaptive Card, use the **Container** element with the **items** property.

The items property accepts an array of elements as its value. Each element in the array is displayed in the Adaptive Card, using the '['ForAll'](#)' function. Reference the `Topic.TaskTable.employeeTasks` array, as it allows access to each of its properties.

If you want to create the topic without following these instructions, you can select Open code editor from the top right command bar, and paste the following YAML code in the code editor view.

YAML

```
kind: AdaptiveDialog
beginDialog:
  kind: OnRecognizedIntent
  id: main
  intent:
    displayName: Untitled
    triggerQueries:
      - array

  actions:
    - kind: SetVariable
      id: setVariable_uFs69M
      variable: Topic.EmployeeTaskList
      value: "{   \"employeeName\": \"Alice\",   \"employeeID\": \"E12345\",   \"employeeDepartment\": \"HR\",   \"employeeTasks\": [ {     \"taskID\": \"T001\",     \"taskDescription\": \"Review employee benefits\",     \"dueDate\": \"2023-10-15\" }, {     \"taskID\": \"T002\",     \"taskDescription\": \"Conduct new hire orientation\",     \"dueDate\": \"2023-09-30\" }, {     \"taskID\": \"T003\",     \"taskDescription\": \"Update HR policies\",     \"dueDate\": \"2023-11-05\" } ] }"
        - kind: ParseValue
          id: 58zKdp
          variable: Topic.TaskTable
          valueType:
            kind: Record
            properties:
              employeeDepartment: String
              employeeID: String
              employeeName: String
              employeeTasks:
                type:
                  kind: Table
                  properties:
                    dueDate: String
                    taskDescription: String
                    taskID: String

      value: =Topic.EmployeeTaskList
```

```

- kind: SendActivity
  id: sendActivity_oNXY1r
  activity:
    attachments:
      - kind: AdaptiveCardTemplate
        cardContent: |-
          ={
            type: "AdaptiveCard",
            version: "1.5",
            body: [
              {
                type: "TextBlock",
                text: "Employee Information",
                weight: "bolder",
                size: "large"
              },
              {
                type: "TextBlock",
                text: "Employee Name: " & Topic.TaskTable.employeeName,
                separator: true
              },
              {
                type: "TextBlock",
                text: "Employee ID: " & Topic.TaskTable.employeeID,
                separator: true
              },
              {
                type: "TextBlock",
                text: "Department: " &
Topic.TaskTable.employeeDepartment,
                separator: true
              },
              {
                type: "TextBlock",
                text: "Tasks",
                weight: "bolder",
                size: "medium",
                separator: true
              },
              {
                type: "Container",
                items:
                  ForAll(Topic.TaskTable.employeeTasks,
                    {
                      type: "TextBlock",
                      text: "- Task ID: " & taskID & ", Description: "
& taskDescription & ", Due Date: " & dueDate ,
                      wrap: true
                    }
                  )
                }
              ]
            }

```

Add feedback for every response

---

# Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# Pass context variables from webpage to an agent

Article • 11/19/2024

When your agent is deployed on your custom website, you might want to pass context variables from your webpage to the agent. You can effectively pass context variables from a webpage to your agent, and use these variables to enhance the user experience. Context variables allow your agent to provide tailored and more meaningful interactions with your website visitors.

## When to pass context variables

- Pass a web analytics tracking ID from a webpage to an agent to analyze web and agent analytics together.
- Pass an order ID from a webpage to the agent so the user doesn't have to type the order ID for the agent again. The agent can just take the order ID information from the webpage.
- Pass the current page of the website so the agent can customize its experience according to the webpage the user is visiting.
- Pass the language of the webpage to the agent so the agent can customize its experience according to the webpage's language.

Passing context variables is a two-step procedure: first pass the context from your webpage to the agent, then save the context in the agent as a variable.

## Pass the context from your webpage to the agent

In your JavaScript code, where you call the agent, include the values to be sent as context. You can pass any information, such as an OrderId, from your website to your agent. For example, in the following code snippet, three context variables are sent:

- Language
- currentURL
- OrderId

```
{  
    Language: siteLanguage,  
    currentURL: window.location.href,  
    OrderId: '12344'  
}  
})
```

ⓘ Note

For more information, see [Resuse variables across topics](#). In addition, for a complete custom canvas Webchat.js HTML/JavaScript code block, see [Customize the look and feel of an agent](#).

## Save context as a variable in the agent

In the [Conversation Start](#) system topic, add an "Asking a Question" node. To properly configure the topic, remember to do the following:

- Ensure that the question is shown to the user.
- Select **Save user response as Variable**. The variable should be set to **Global**.
- Select **External source can set values**. For more information, see [Set a global variable's value from external sources](#).

The screenshot shows the Microsoft Bot Framework designer interface. On the left, there's a trigger named "Conversation update received" and a question block. On the right, the "Variable properties" modal is open for a variable named "OrderId". The "Usage" section is highlighted with a red box, containing two radio buttons: "Topic (limited scope)" and "Global (any topic can access)", with the second one selected. There are also two checkboxes: "Allow to carry between sessions" (unchecked) and "External sources can set values" (checked).

**Trigger**

Conversation update received [Edit](#)

**Question**

Text

+ Add [B](#) [I](#) [≡](#) [≡](#) {x}

\* Provide a message or add other content.

Enter a message

**Identify**

User's entire response

**Save user response as**

{x} Global.OrderId string

**Variable properties**

Variable name: OrderId

Type: string

Reference:

Question

Type (string) derived from here

[View all references](#)

Usage

Topic (limited scope)

Global (any topic can access)

Allow to carry between sessions

External sources can set values

Timeout

The conversation will pause for the variable value to be set. Choose how

### Note

While this example uses the Conversation Start system topic, you can save the context received from a webpage in any topic.

With this variable in your agent, you can use it for any purpose you want, such as getting order details for the order ID, customizing the behavior of the chat for the language, and so on. For an example of how to use this concept to combine your web and agent analytics, see [Combine web and agent analytics](#).

## Feedback

Was this page helpful? [Yes](#) [No](#)

Provide product feedback ↗

# Link web analytics with Copilot Studio analytics

Article • 11/19/2024

Integrating web analytics with your agent analytics is essential for gaining insights into user interactions across various channels. There are two approaches to achieve seamless integration: pass tracking IDs from a webpage to your agent, and capture conversation IDs in your web analytics. By implementing these methods, you can enhance your analytics capabilities and gain a deeper understanding of user behavior.

## Methods for integrating analytics

There are two ways to integrate web analytics with agent analytics:

- Pass a tracking ID and other data to the agent as context variables
- Capture the agent's conversation ID in your web analytics

You can use either method or both.

For more information on how to build your own reports with your own custom dimensions on top of the conversation transcripts, see [Use custom analytics reports](#).

### Note

By default, conversation transcripts have a data retention of 30 days in Dataverse, but you can [change the retention period](#).

## Pass tracking ID from website to agent

You can pass any information, like a tracking ID, from your website to your agent. To pass information, you need to have a global variable in your agent that takes input from your custom canvas (website code).

To send this information from your custom canvas, add the piece of code outlined in [Pass context variables from your webpage to an agent](#).

```

const store = window.WebChat.createStore(
  {},
  ({ dispatch }) => next => action => {
    if (action.type === "DIRECT_LINE/CONNECT_FULFILLED") {
      dispatch({
        meta: {
          method: "keyboard",
        },
        payload: {
          activity: {
            channelData: {
              postBack: true,
            },
            name: 'startConversation',
            type: "event",
            value: {
              TrackingId: "Tracking Number"
            }
          },
          type: "DIRECT_LINE/POST_ACTIVITY",
        });
    }
    return next(action);
  }
);

```

With this setup, you see the tracking ID you sent from your custom canvas code in your agent analytics in the Dataverse Table ConversationTranscript.

```

{
  "id": "1GTgAUSUa9CCGL1l4exuHw-us|0000000",
  "type": "event",
  "timestamp": 1700449981,
  "from": {
    "id": "d18d8ce5-6ed5-f417-27af-1ba8da8045fa",
    "role": 1
  },
  "name": "startConversation",
  "channelId": "directline",
  "attachments": [],
  "value": {
    "TrackingID": "Tracking Number"
  },
  "channelData": {
    "postBack": true,
    "clientActivityID": "196iee8oxf"
  }
}

```

## Capture conversation ID in web analytics

This method uses a conversation ID to connect your web and agent analytics. To get this conversation ID, you must integrate the agent using the webchat.js in your HTML/JavaScript code, as outlined in [Customize the default canvas \(simple\)](#).

### Note

This method doesn't work if you embed the agent using the iFrame approach. For more information, see [Add an agent to a live or demo website](#).

As shown in the following sample JavaScript code, you can access the `conversationId` and save it to your own analytics:

```
<script>
  const styleOptions = {
    // Add styleOptions to customize Web Chat canvas
    hideUploadButton: true
  };

  // Add your BOT token endpoint below
  var theURL = "https://d2c97b9f13faeb97bd35c6b262b1f8.17.environment.api.powerplatform.com/powervirtualagents/botschemas/cr3a9_nsTestFrame/directline/token?api-version=2022-03-01-preview";

  var environmentEndPoint = theURL.slice(0,theURL.indexOf('/powervirtualagents'));
  var apiVersion = theURL.slice(theURL.indexOf('api-version')).split('-')[1];
  var regionalChannelSettingsURL = `${environmentEndPoint}/powervirtualagents/regionalchannelsettings?api-version=${apiVersion}`;

  var directline;
  fetch(regionalChannelSettingsURL)
    .then((response) => {
      return response.json();
    })
    .then((data) => {
      directline = data.channelUrlsById.directline;
    })
    .catch(err => console.error("An error occurred: " + err));

  fetch(theURL)
    .then(response => response.json())
    .then(conversationInfo => {
      window.WebChat.renderWebChat(
        {
          directline: window.WebChat.createDirectLine({
            domain: `${directline}v3/directline`,
            token: conversationInfo.token,
          }),
          styleOptions
        },
        document.getElementById('webchat')
      );
      console.log(`conversationInfo.conversationId`);
    })
    .catch(err => console.error("An error occurred: " + err));
</script>
```

Replace this code to log this `conversationId` in your own web analytics system

To test that this method is working correctly, press the F12 key. In the web browser's developer tools, search for `console.log`, as shown in the following image:

The screenshot shows the Chrome DevTools interface with the 'Sources' tab selected. The left sidebar shows the file structure: top, file://, C:/User, nsTestFrame.html (which is the active tab), (no domain), and cdn.botfr. The main pane displays the code of nsTestFrame.html, with line numbers from 63 to 107. Line 98 contains a console.log statement: `console.log( conversationInfo.conversationId);`. The output of this log statement, "I4XzJ8U3B3P9jQCUMNQ4rD-us", is shown in the 'Console' tab at the bottom, also highlighted with a yellow box.

```
63 // Add styleOptions to customize Web Chat canvas
64     hideUploadButton: true
65 };
66
67 // Add your BOT token endpoint below
68 var theURL = "https://d2c97b9f13faeb97bd35c6b262b1f8.17.environment.svc.powershell.com/api/v3/conversations"
69
70 var environmentEndPoint = theURL.slice(0,theURL.indexOf('/powershell'))
71 var apiVersion = theURL.slice(theURL.indexOf('api-version')).split('.')[0]
72 var regionalChannelSettingsURL = `${environmentEndPoint}/powershell/v${apiVersion}/regions`
73
74 var directline;
75     fetch(regionalChannelSettingsURL)
76         .then((response) => {
77             return response.json();
78         })
79         .then((data) => {
80             directline = data.channelUrlsById.directline;
81         })
82         .catch(err => console.error("An error occurred: " + err));
83
84 fetch(theURL)
85     .then(response => response.json())
86     .then(conversationInfo => {
87         window.WebChat.renderWebChat(
88             {
89                 directLine: window.WebChat.createDirectLine({
90                     domain: `${directline}v3/directline`,
91                     token: conversationInfo.token,
92                 }),
93                 styleOptions
94             },
95             document.getElementById('webchat')
96         );
97         console.log( conversationInfo.conversationId);
98     })
99     .catch(err => console.error("An error occurred: " + err));
100
101
102 </script>
103 </body>
104 </html>
105
106
107
```

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Generative AI strategies for math and data queries

Article • 11/20/2024

Currently, there are varying opinions on how well tools such as ChatGPT or other language models handle math and data queries. In this article, we're going to identify strategies and set expectations when building agents in Copilot Studio that handle math and data queries.

## Definitions of math and data queries in this article

The goal of this article isn't to evaluate whether generative AI can assist with calculating the perimeter of a rectangle or the diameter of a circle. Math, in this context, refers to typical natural language questions that someone would ask an agent. These questions assume the AI can aggregate and interpret sums, averages, and trends across the [knowledge sources](#) or data tables used to ground the models.

The desired outcome, in this case, isn't to answer a math equation. Instead, it's to help the user evaluate or understand data more efficiently. When users are looking for deep data analytics, such as looking for advanced predictive or prescriptive analytics, a custom agent is typically not the tool of choice. However, there are several agents in the Microsoft stack that are more directly focused on analytics. For example, the following agents supplement the language model with Microsoft application code for this purpose:

- Add Copilot for app users in model-driven apps
- Visualize data in a view with Power BI service
- Microsoft 365 Copilot for Sales
- Embed a Power BI report with a Copilot narrative visual
- Identify insights with Copilot in Excel ↗

## Data aggregates in Natural Language Understanding

When we ground an agent in our own knowledge sources, we're simplifying the discovery of information asked by a user in natural language. Keep in mind that language models are designed to predict the next word in a sequence rather than

perform rigorous math. However, they can still provide useful insights and explanations. These insights are faster for information discovery than browsing *keyword search results* or manually scrolling through all the records in a table.

Copilot Studio agents can scan knowledge sources on our behalf. These agents summarize answers across topics, actions, and knowledge sources, whether they involve aggregates of numeric data or not. However, as we ground the models with our data, we must contextualize the data required for the AI to respond. Based on this understanding, we know when we should provide more context or topic nodes. This extra understanding is relevant when niche terms or highly technical verbiage is found in the data sources. The following are examples of data queries that involve mathematical expressions:

 Expand table

Example questions	Things to consider
How many of our customers in North America purchased product X?	This prompt involves multiple structured tables in a relational database, and commonly looking over hundreds or even thousands of records.
What was the total cost impact for repair work items after the hurricane?	This prompt involves a table of items repaired, with a column for cost impact for each work item. If the table has more repairs than work items related to the hurricane, then a category or reason column would be needed for the AI to know which are related to the hurricane.
Which of our customers submitted the most requests for change?	This prompt involves a table with change requests, and related table with customer names. However, this first counts requests by customer, then returns the customer with the highest number of requests (and not the highest cost impact dollar value).

## Prompt clarity and structure

Language models rely heavily on how well the question is phrased. A well-structured prompt that clearly explains the math problem, defines variables, and breaks the task into steps leads to more accurate responses. For instance, asking for a direct answer to a simple arithmetic problem likely works well, but vague or multi-layered questions without clear context might confuse the model.

Here are some sample prompts based on a [structured knowledge source](#) such as a Dataverse table. This sample illustrates the addition of a Power Apps Dataverse table, as shown in the following image.

PCO columns and data

PCOID	Change Type	Change Amount	0.0 Schedule	Change Reason	Account	Status	+21 more	+
PCO-1000	Cost	5,000.00	0.00	Repairs needed due to the storm in July.	Fourth Coffee	Approved		
PCO-1001	Cost	12,000.00	5.00	Replacement of bridge bearings that could not sustain certain truck weights.	Fourth Coffee	Approved		
PCO-1002	Cost	2,000.00	0.00	Payment for aerial photography in Phase 2	Blue Yonder Airlines	Approved		
PCO-1003	Cost	8,000.00	10.00	Electrical work after the storm in July.	Adventure Works	Approved		
PCO-1004	Cost	300.00	0.00	Sandbags for expected storm protection.	Adventure Works	Approved		
Select option	Enter number	Enter decimal	Enter text		Select lookup	Select option		

The Dataaverse table was added as a knowledge source, and given an accurate knowledge description, along with synonyms and glossary definitions to aid the AI in interpreting the data.

Details Preview Synonyms Glossary

**Knowledge details**

Knowledge name \* 12/100  
PCO, Account

Knowledge description \* 214/2500  
This knowledge source is managing PCO requests (PCO stands for Potential Change Request). The priority for this knowledge source is cost impacts. Cost impacts are highlighted in the Change Type column as 991820000.

Selected tables Edit tables

- Account
- PCO cr507\_PCO

Details Preview **Synonyms** Glossary

Improve data discovery with synonyms  
Increase your copilot's data retrieval accuracy by adding synonyms for complex or unclear column names in your tables.

Select table  
PCO

Column name	Synonyms	Description
Account	Account Name,Customer Name,Originator,Source	Customer Account Name (requester)
Change Amount	Amount,Amount Requested,Change Amount,Cost Impact,E...	The cost amount requested in USD

## Specific prompts

These prompts are specific and scoped to the information being requested.

- "Can you provide full details on change order reference PCO-1003, including account name, amount requested, and reason for request?"

- "How many accounts submitted change requests in August of 2024?"
- "What is the total number of change orders requested to date?"
- "Which customer submitted the highest cost impact in 2024?"

## Generalized prompts

These prompts are generalized and are unlikely to consistently aggregate all of the results, likely only returning the top three results.

- "Please list our accounts in the order of their respective revenue."
- "Please list the change requests which were submitted this year in August, and include change amounts and status."
- "Can you list all the change order requests submitted to date?"

### ① Note

Enabling or disabling the AI's ability to use its own general knowledge can affect the accuracy or appropriateness of the returned answers.

## Tips and tricks

Here are a few suggestions when working with Copilot Studio which help you to set expectations around generative answers that rely on mathematical expressions.

- Plan for scenarios that highlight top trends, rather than expecting calculations over thousands of records. Raise awareness to this conversational approach that summarizes rather than itemizes, instead.
- Favor structured knowledge sources (tabular over nontabular) to optimize mathematical expressions.
- Support specific scenarios, and understand the dependencies for the differences. For example, note the difference between these two questions:
  - Which of our customers submitted the most requests for change? Counts request IDs, and returns the customer with the most requests – ignoring other columns
  - Which of our customers have the highest cost impact across requests for change? Sums the cost impact column by customer, and returns the customer who submitted the highest total dollar amount. It only returns this information if it finds an appropriate column that is numeric or currency-based.

- Be sure to identify and define any numeric columns for calculations. Ensure that they're formatted with the appropriate data type; both at the knowledge source level, and when used in any Copilot Studio variables. When possible, include a clear description, and include common synonyms for the relevant columns in the tables, columns, or action descriptions.

 **Tip**

With natural language understanding, if the table headers are too technical in their naming protocol, the AI might not be able to answer the human-centric questions asked during the conversation flow. Add descriptors with typical verbiage used by your users.

- Recognize that people only get answers over the data that they're permissioned to see. For example, a Sales table in Dataverse might only expose some records to specific business groups but not all. So, be sure that your agent doesn't set the wrong expectations on how the data is summarized. For example, a request for total sales in 2024 only sums the owned or shared records.
- Always set consumer expectations for AI driven answers. Use the agent [Conversation start](#) or the first message following [topic](#) triggers, to gently highlight the purpose and constraints for one or more relevant knowledge sources.

## Use AI builder prompt actions

[Prompts actions](#) enable you to add generative AI capabilities from Power Apps to your agents and solutions in Copilot Studio. This feature allows you to perform tasks such as classification, summarization, draft content generation, data transformation and much more. With prompt actions, you can also tailor generative AI responses to use specific filters and aggregations from tables.

In the following screenshot, you see how a Copilot Studio maker used AI Builder prompt actions in Copilot Studio to summarize change order requests from both the Account table and the related PCO table.

The screenshot shows the Microsoft AI Builder interface. On the left, under 'Prompt', there is a text input field containing a template for summarizing an account record. It includes dynamic prompt variables like 'AccountNumber' and 'Account.Account Name'. Below the input field, a 'Test prompt' button is visible. On the right, the 'Prompt settings' pane is open, showing 'Input 1' with a sample value 'ABSS4G45' for 'AccountNumber'. There are sections for 'Data used (preview)', 'Output', and 'Settings'. At the bottom right of the settings pane are 'Save custom prompt' and 'Cancel' buttons.

In the preceding example, the agent's knowledge sources weren't used. Instead, the prompt includes the dynamic prompt variable for the Account Number, and a table from Dataverse as Data.

### 💡 Tip

Related tables are assumed by the AI and don't need to be added in this case. (The PCO table has a many to one relationship with Accounts.)

## Feedback

Was this page helpful?

Yes

No

[Provide product feedback](#)

# Measuring agent engagement

Article • 11/19/2024

Tracking conversation engagement rates and outcomes are crucial to measure the agent performance metrics, and spot areas for improvements.

## Understanding conversations and analytics sessions

A **single conversation** between a user and an agent can generate one or multiple analytics sessions, typically happening when the user has new questions after an initial topic conversation path completed.

### ⓘ Note

Analytics sessions and billed sessions are distinct concepts.

## Engaged and unengaged analytics sessions

Analytics sessions are either **unengaged** (default) or **engaged**.

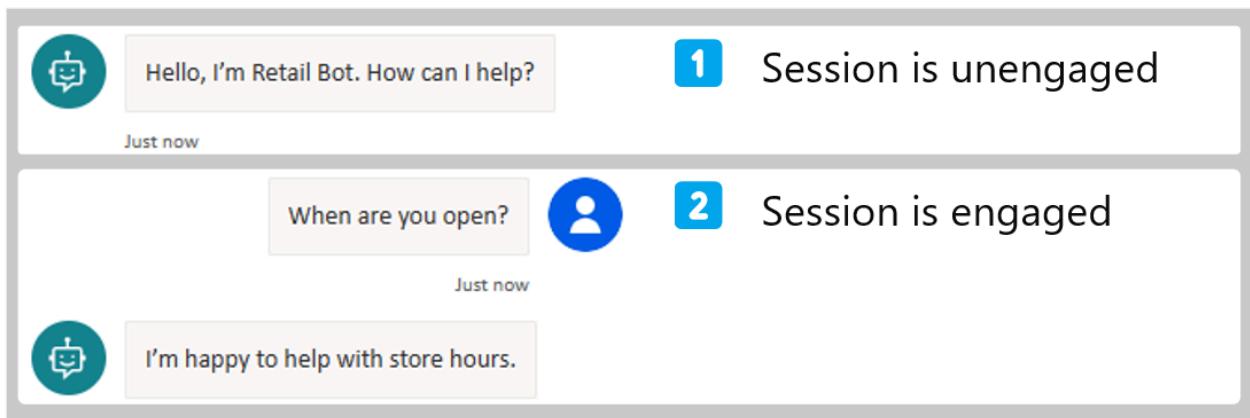
This notion is used in the Copilot Studio [analytics dashboard](#).

### 💡 Tip

Analytics session engagement is also tracked in [conversation transcript records in Dataverse](#), if you have a [custom analytics strategy](#).

In a conversation transcript record `content`, the engagement status of a session (either `unengaged` or `engaged`) is available as part of the `SessionInfo` activity, in the `value` type.

- ✓ A session starts when a user interacts with the agent or when the agent sends a proactive message to the user.
- ✓ The analytics session begins in an **unengaged** state.
- ✓ An analytics session becomes **engaged** when either entering a **custom topic** or the **Escalate** topic.
- ✓ The last custom topic triggered—or the first system topic triggered, if no custom topics were triggered—is associated with the analytics session.



ⓘ Note

Having an agent that proactively starts a conversation or that's placed on a website can increase the session total and the number of unengaged sessions.

Measuring agent outcomes

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Measuring agent outcomes

Article • 12/03/2024

Tracking conversation engagements and outcomes is crucial to measure the agent performance metrics and spot areas for improvements in the analytics dashboard.

## ⓘ Important

A Copilot Studio conversation between a user and an agent generates one or multiple analytics sessions. All analytics sessions are either [engaged or unengaged](#).

## Understanding analytics sessions outcomes

Analytics sessions each have an outcome:

- ✓ **Unengaged** sessions always have an outcome of **None**.
- ✓ **Engaged** sessions have one of the following outcomes:
  - **Resolved**
  - **Escalated**
  - **Abandoned**

## Engagement and outcomes

Conversation analytics sessions, their engagement status, and their outcomes are available as part of the [analytics dashboard](#) or in the [ConversationTranscript table](#) in Dataverse.

[+] Expand table

Engagement	Outcome	Definition
Unengaged	None (None)	All unengaged sessions have an outcome of <i>None</i> .
Engaged	Escalated (Handoff)	A session is considered <i>escalated</i> when the <a href="#">Escalate</a> topic is triggered, or the <a href="#">Transfer Conversation</a> node is reached.
Engaged	Resolved (Resolved)	A session is considered <i>resolved</i> when the <a href="#">End of Conversation</a> topic is triggered and the user confirms that the interaction was a success ( <code>impliedSuccess = FALSE</code> ) or doesn't answer and lets the session time out ( <code>impliedSuccess = TRUE</code> ).

Engagement	Outcome	Definition
Engaged	Abandoned (Abandoned)	A session is considered <i>abandoned</i> when an engaged session ends and didn't reach a <i>resolved</i> or <i>escalated</i> state.

## Properly ending conversations to measure outcomes

It's important to end conversation with the **End of Conversation** topic, so that the user can confirm their issue was resolved or not (and potentially escalate).

When the agent confirms success, a CSAT survey is displayed to the user [to capture a 0-5 score](#).

If you're certain of the outcome of a conversation based on the topic logic, you can directly flag it as a confirmed success or redirect to the Escalate topic.

### 💡 Tip

In the unified authoring canvas, in the code editor, you can set the `conversationOutcome` within an action. For example `conversationOutcome: ResolvedConfirmed` for confirmed success or `conversationOutcome: ResolvedImplied` for implied success.

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Deflection overview

Article • 11/19/2024

Return on investment (ROI) and improved customer satisfaction (CSAT) are top priorities for the organizations that implement Copilot Studio agents.

Optimizing the agent deflection rate is one of the top focus areas for organizations to achieve their business goals around ROI and CSAT, and to improve the agent's overall performance. There are major indicators in Copilot Studio that help improve agent performance, such as resolution rate, escalation rate, and CSAT.

While the metrics continue to evolve, there are several things you can do as an agent builder to improve the deflection rate of your agent. In these articles, we cover the importance of deflection in conversational AI and general techniques/considerations that are universal for optimizing deflection for agents.

## 💡 Tip

In the context of conversational AI, *deflection* is an indicator representing the percentage of requests that are completed in a self-service fashion that would otherwise be handled by live representatives. In other words, it refers to the number of items a team avoids having to deal with as a result of automation.

## Why deflection optimization?

- ✓ **Better customer experience:** more customers or employees can get their issues resolved by the agent instead of waiting for a human representative in chat or phone. This leads to a better customer experience and higher CSAT scores. While this helps reduce wait time, live representatives can also focus on more complex, higher-value tasks.
- ✓ **Cost savings:** one of the key ways the ROI of the agent is determined is by using deflection rates. Human representative call support typically costs around \$5 to \$10 in the contact center industry. However, an agent session that resolves a customer request costs around 50 cents. This means that higher deflection rates lead to higher cost savings.

## Understanding the key components of Copilot Studio analytics to improve deflection

A basic understanding of available analytics is required to be able to determine what deflection means for your organization. The following table describes the key metrics from Copilot Studio:

[\[+\] Expand table](#)

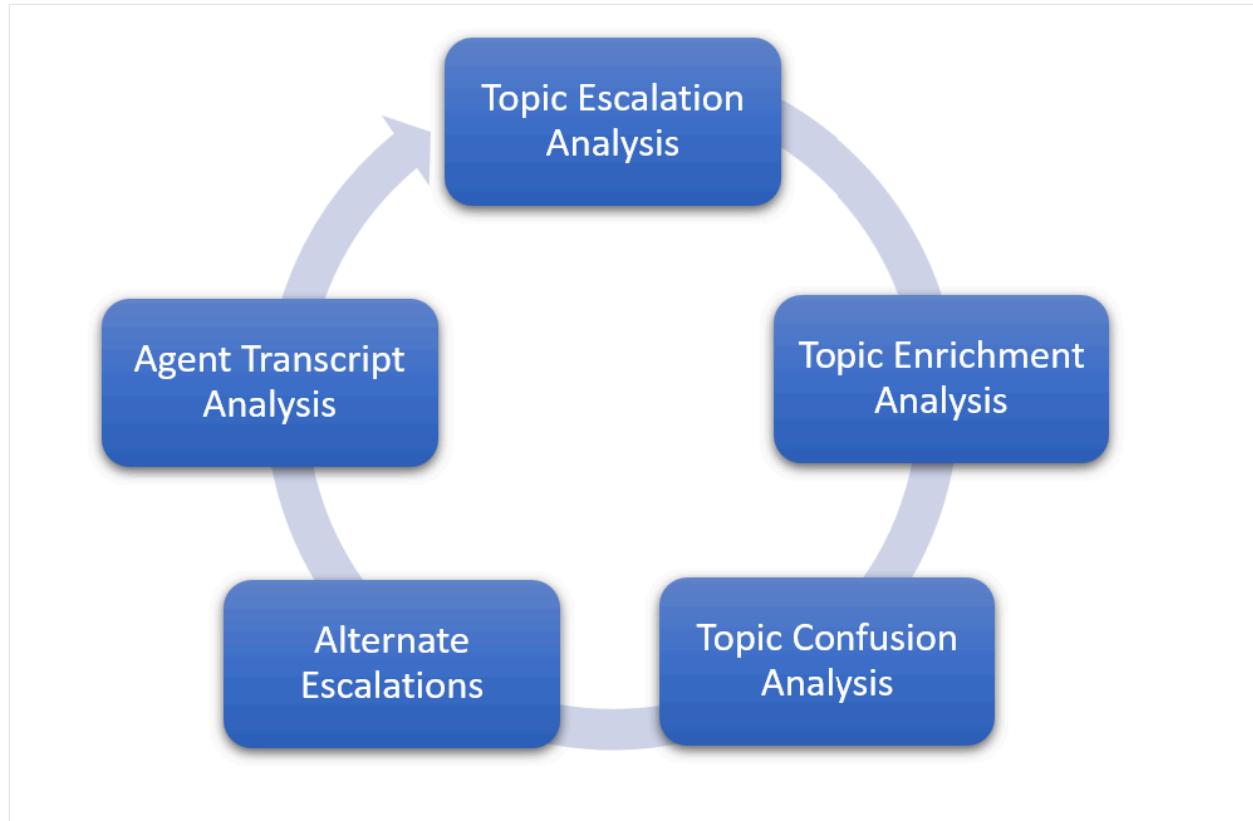
Description	Details
Total Sessions	The total number of <i>analytics sessions</i> within the specified time period. A conversation with an agent can generate one or multiple analytics sessions, each with their own engagement status and outcome. An analytics session begins when a user has new questions after an initial conversation completed (for example, reached the End Of Conversation topic).
Engagement Rate	The percentage of total sessions that are engaged sessions. An engaged session is an analytics session where a custom topic is triggered, or the session ends in escalation. Engaged sessions can have one of three outcomes —they're either resolved, escalated, or abandoned.
Resolution Rate	The percentage of engaged sessions that are resolved. A resolved session is an engaged session where the user receives an End Of Conversation question that asks "Did that answer your question?" and the user either doesn't respond or responds "Yes."
Escalation Rate	The percentage of engaged sessions that are escalated. An escalated session is an engaged session escalated to a human representative.
Abandon Rate	The percentage of engaged sessions that are abandoned. An abandoned session is an engaged session that isn't resolved or escalated after one hour from the beginning of the session.
CSAT	The graphical view of the average of CSAT scores for sessions where customers respond to an End of Conversation request to take the survey.

These metrics need to be continuously improved to optimize the agent ROI. However, each organization might have their own definition of what *deflection rate* means to them. For example, an organization might consider the *Abandon* rate along with the *Escalation* rate as part of their deflection calculation, while another organization might look purely at the *Escalation* rate.

Despite having different definitions for a deflection rate, these metrics still provide the foundation to calculate deflection. Based on our experience with various customers, in the context of deflection, *Resolution* rate, and *Escalation* rate play a major role.

Increasing the *Resolution* rate and reducing the *Escalation* rate typically has a direct result on overall agent deflection metrics.

# Key techniques



Topic escalation analysis

## Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#)

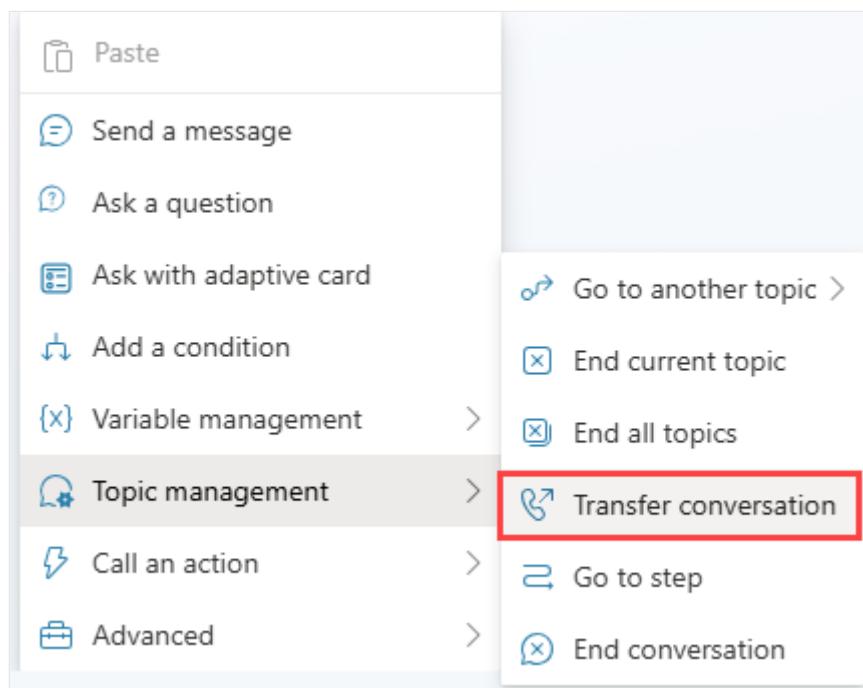
# Topic escalation analysis

Article • 11/19/2024

Escalation is the conversation flow during which the agent couldn't handle the conversation and escalated to a human representative. When an agent is able to answer the user query without having to escalate to a human representative, that's a deflection. The ideal goal is to increase the deflection rate of an agent by reducing the number of escalations.

Copilot Studio has multiple ways to handle escalation:

- The direct way of initiating an escalation to human representative is through the Escalate system topic. This system topic is triggered when the agent is no longer able to address the customer request and needs to escalate to a human representative. Through the Escalate topic, one can enable the agent to transfer the conversation to a representative service desk tool like Dynamics 365 Omnichannel for Customer Service for the live representative transfer, or an asynchronous support experience like creating a ticket, scheduling a callback, and so on.
- Another way of triggering this escalation is through the Transfer Conversation node in the authoring canvas.



## Escalation types

Copilot Studio has two types of escalations:

1. **Direct escalation:** in this case, the user comes to the agent and directly wants to talk to a human representative. This kind of escalation can't be avoided as the intent of the customer is to trigger the Escalate topic directly.

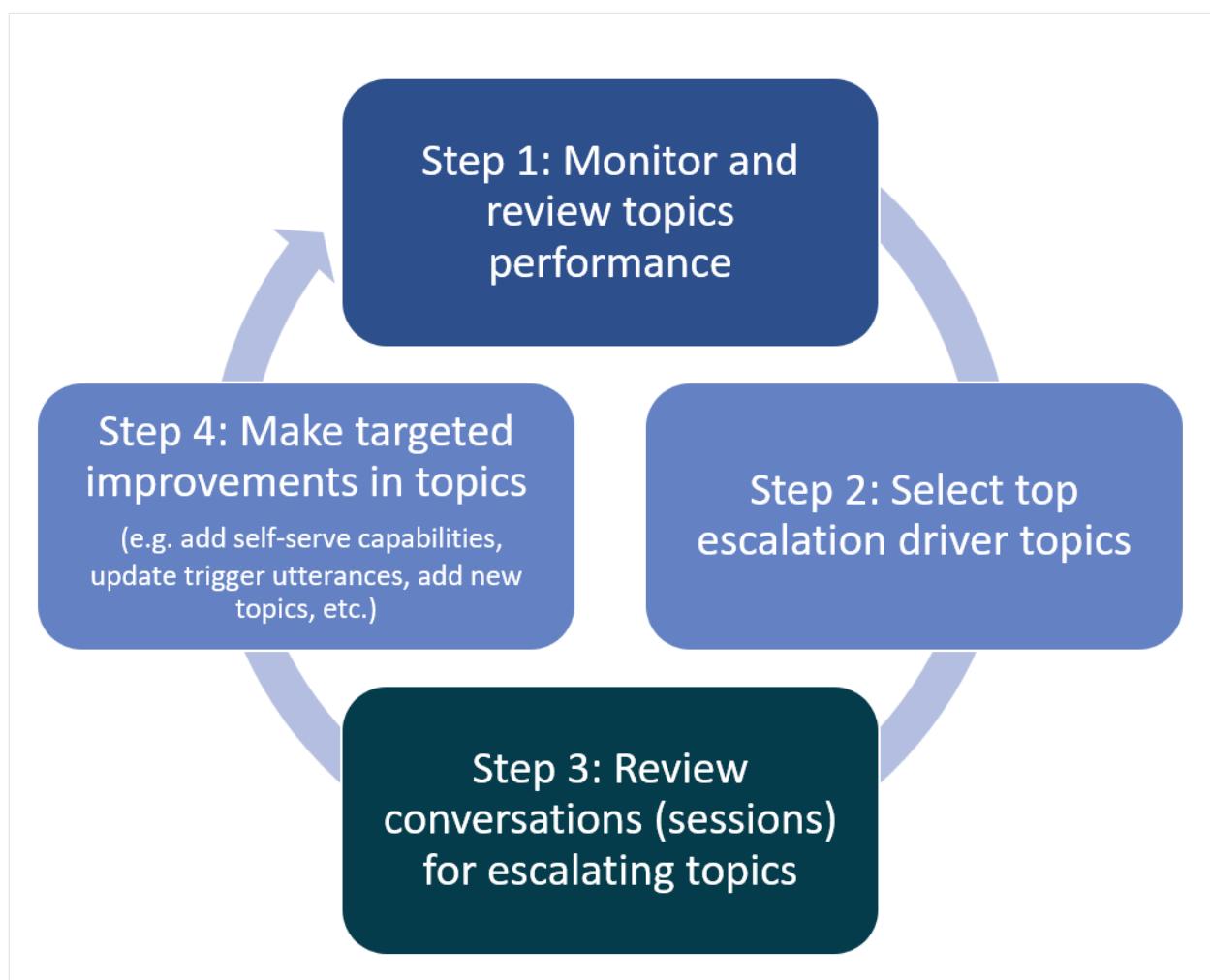
Examples of customer queries:

- "can I talk to someone"
- "talk to a live representative"
- "speak with representative"
- "talk to a representative"

2. **Indirect escalation:** in this case, the user gets escalated to a representative during the conversation.

These can be grouped in *expected* vs *unexpected* escalations.

*Expected* escalations happen when the topic is designed to escalate at some point during the conversation, or the user chooses to escalate since the agent didn't answer their query, while an *unexpected* escalation can happen when the agent errors out due to other issues.



# Topic escalation analysis

## STEP 1: Monitor and review topics performance

Identifying and optimizing escalation rate drivers can be done through the [built-in analytics](#) or through [custom analytics](#).

### Built-in analytics

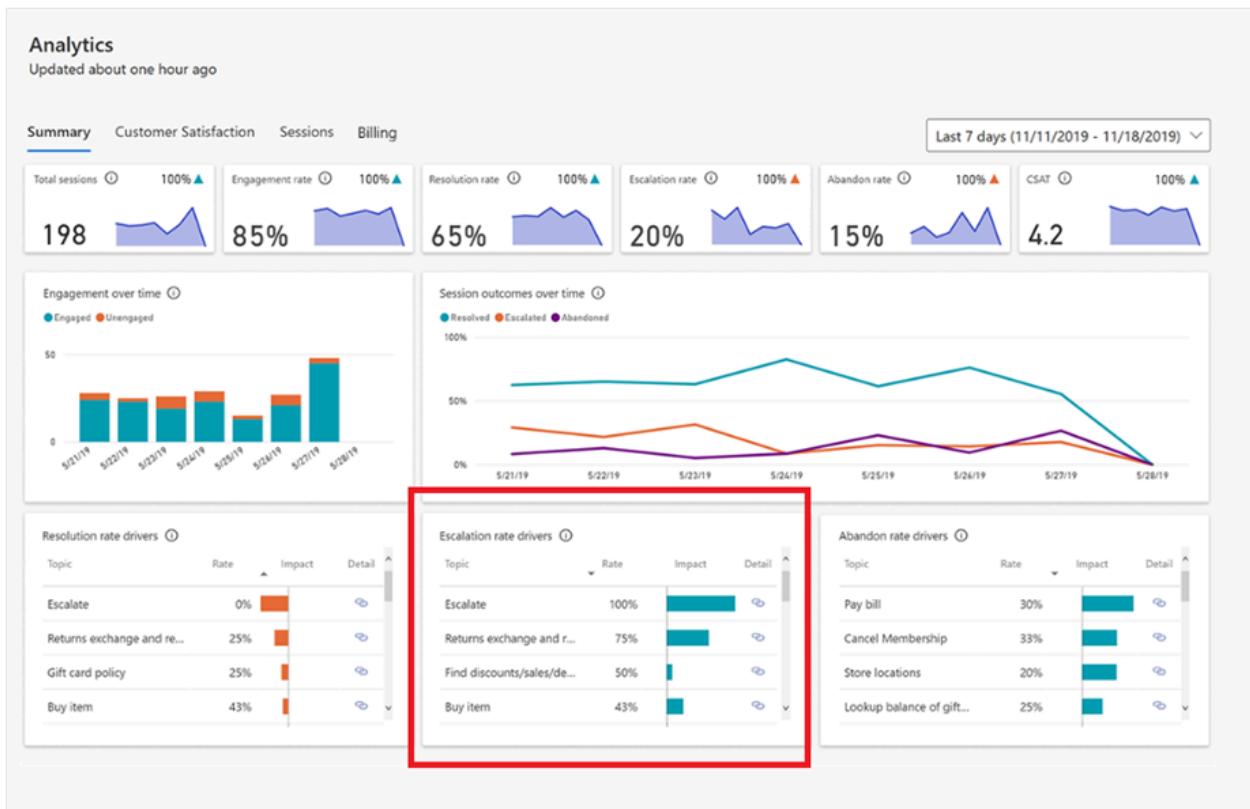
All the agent sessions that led to an escalation or transfer to an representative are captured from beginning to end at the topic level. The escalation drivers in this scenario are the agent topics.

The analytics dashboard has a section for "Escalation Rate Drivers", which gives the details on which agent topics were escalated to human representatives most of the time and why. This information is available in the numerical point of view and derived from the chat transcripts.

For example, in the following screenshot, under the **Escalation Rate Drivers** section, the **Returns, Exchanges...** topic has a **Rate** value of 75%. This means that 75% of all the sessions that triggered the **Returns, Exchanges...** topic were escalated to a human representative. Since the agent couldn't solve the issue for the user, 75% of the times when user asked about returns, the agent had to escalate to a human representative. Now the agent author can improve the **Returns, Exchanges...** topic to reduce the number of escalations happening through this topic.

The chart also shows the **impact** as a red or blue bar. The escalation rate impact score is the overall escalation rate including the topic minus the overall escalation rate excluding the topic. In short, **impact** helps you understand what this topic is contributing to the overall escalation rate. If the impact is high, then that's the topic you have to focus on because if you improve that topic, the potential impact it has on escalations also improves.

A red bar indicates that the topic's escalation rate is greater than the average escalation rate, resulting in a negative impact on overall escalation rate. A blue bar indicates that the escalation rate is smaller, resulting in a positive impact on overall escalation rate performance. Lowering the escalation rate for the topics in red has the greatest impact on improving the overall escalation rate (the impact score isn't represented as a number but as a bar chart).

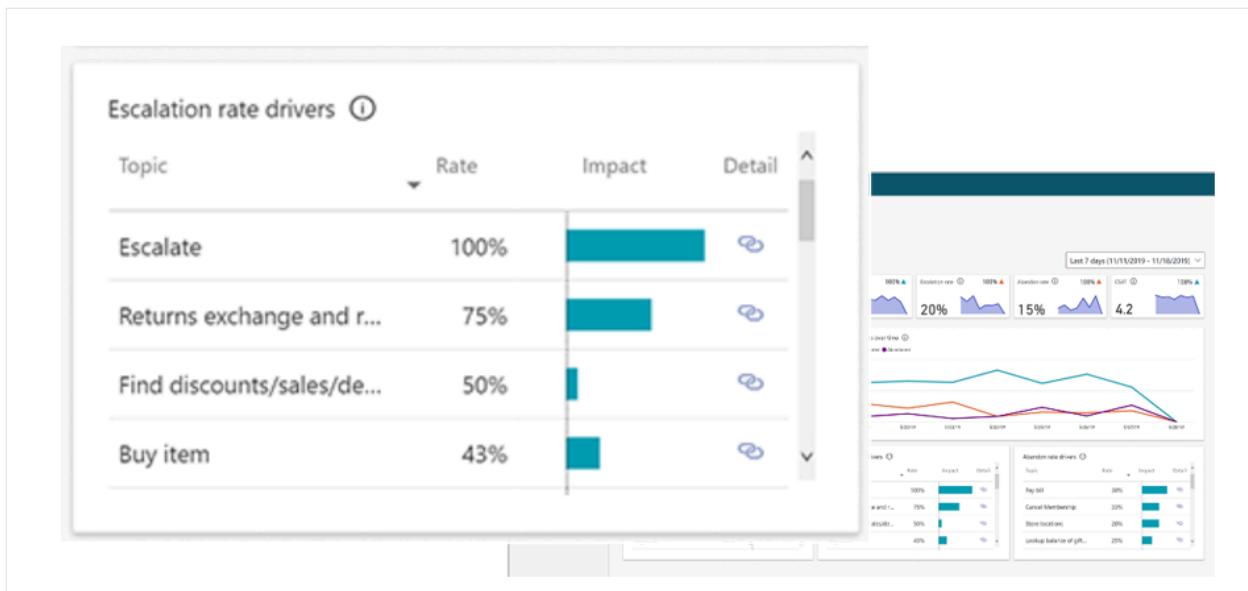


## Custom analytics

You can also build on your own custom analytics on top of the conversation transcripts data. Microsoft provides a [sample template report](#) that can be reused or extended to identify the top escalation driver topics and add custom details specific to your business and context. For example, if you need the number of escalated sessions per topic.

## STEP 2: Select the top Escalation topics

The general guidance is to target the top 5-10 topics under **Escalation rate drivers** to start with, for deflection rate optimization. On a ball park estimate, if you improve the rate of escalation by 10% for each of the top five topics, you can improve the overall deflection by 1% for the agent.



## STEP 3: Review conversations for selected topics

Analyzing the conversation transcripts for the top escalation topics can provide more insights into the reasons for escalation. Conversation transcripts capture the turn by turn, as "*user says*" and "*copilot says*". It also captures the topic name that was triggered and the session outcome (for example, Resolved, Escalated etc.).

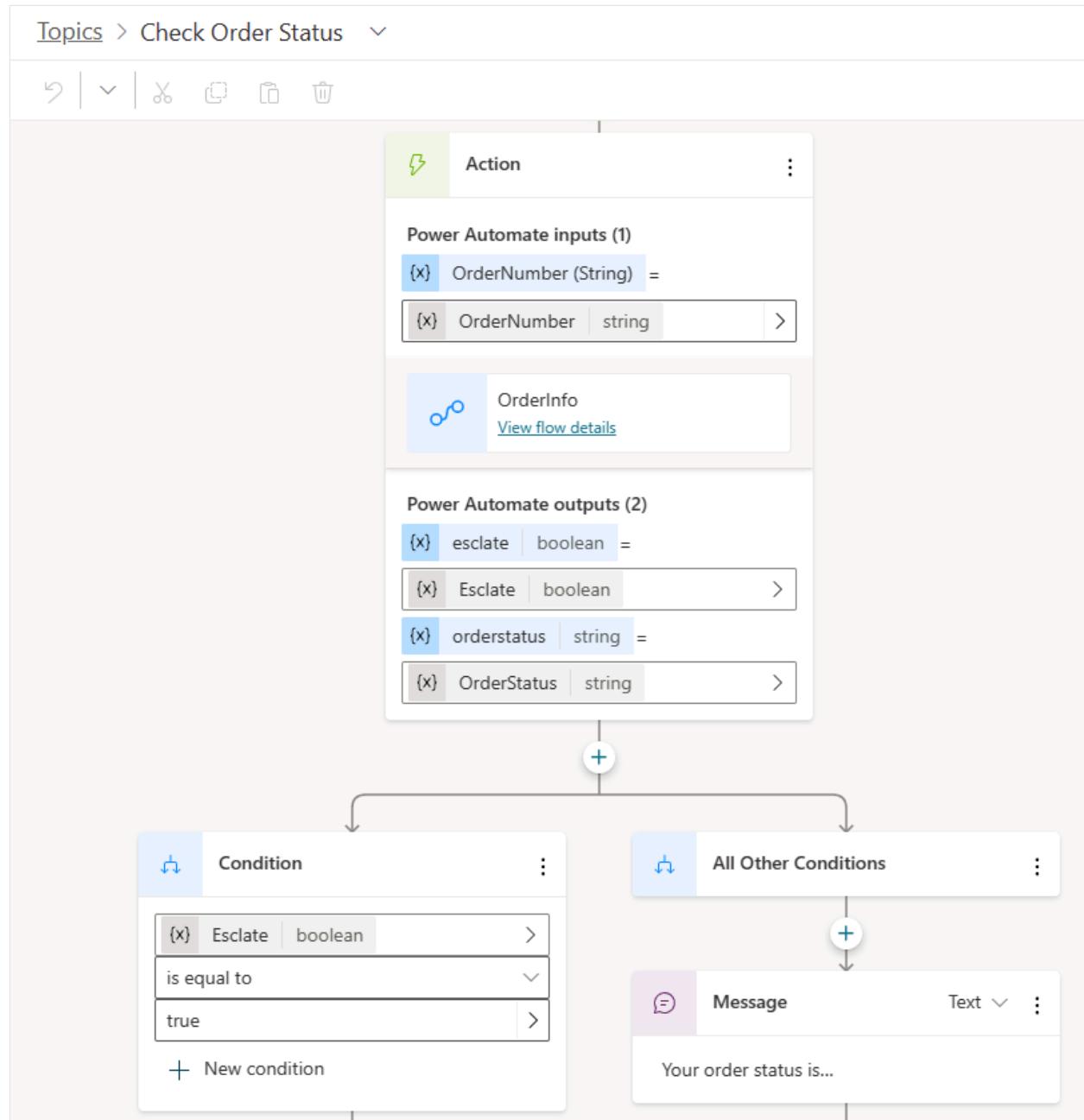
Now, you can filter these sessions based on outcome for **Top escalated topics** and review a few sample conversations to see what caused the escalation. This helps you identify the pattern that's causing the escalation. This exercise can be repeated in periodic cadence to keep improving the deflection rate and reducing the escalation rates.

Below is the step-by-step guidance you can use for dissecting the chat transcripts and coming up with the right recommendations for improving the topic performance:

1. Take one of the top five topics to make improvements to reduce escalation.
2. Filter the transcripts and sort by the session outcome for **Escalation**.
3. Select the most recent sample set of conversation transcripts (for example, 10 sessions). The size of the sample set depends on how much accuracy you're looking for. For a quick analysis you can start with 10 sessions.
4. Read through each of the sessions and identify the various repeated dialog paths that are emerging for that topic related conversation.
5. List out the dialog paths identified for each session and group them as per the dialog path.
6. For each dialog path group, come up with a recommendation for improvement.

7. Implement the recommendations in the agent topics and observe the change in the escalation rate and deflection.

Applying the above approach for the **Check Order Status** topic example described in the next section would look like this:



## Topic description

**Check Order Status** is supposed to provide order and shipping information for the user.

## Observation from transcripts

After reviewing multiple conversation transcripts for this topic that ended with escalation, there are multiple dialog paths emerging that are leading the user to

escalate to a representative, even if the agent provided the order information as designed.

There could be a dialog path #1 where the agent provides the order info when the user asks about missing shipment. There could also be another dialog path #2 where user is looking for the status of multiple orders while the agent is currently providing a status for only one order at a time. The recommendation for dialog path #1 could be adding a new topic exclusively addressing the **Missing Order** scenario, while the recommendation for dialog path #2 could be to update the self-service action to provide status for multiple orders instead of just one.

## Conversation transcript review summary

- Sample set size: analyze sample conversations for escalated sessions from the downloaded transcripts. All triggered the correct topic. All got escalated at the end.
- Expected dialog path: Go to **OrderInfo** action and provide the order status to the user.

## New dialog paths identified by reviewing transcripts

- Dialog path 1: **OrderInfo** replies with **order info adaptive card** but the user query is related to missing package, so the user decides to escalate (7 out of 10 sessions).
- Dialog path 2: **OrderInfo** action replies with: "*your order contains multiple shipments*", but doesn't show shipping information for all the orders, so the user decides to escalate (2 out of 10 sessions).
- Dialog path 3: Other (order number mismatch), user was unaware that they were entering an incorrect order number, so the user decides to escalate (1 out of 10 sessions).

## Recommendations for the dialog path groups

- Path 1: Add a new topic for handling missing order.
- Path 2: Enhance the **OrderInfo** action to support providing multiple order shipping information.
- Path 3: Enhance the **OrderInfo** action to validate the order ID format and provide error message for incorrect order IDs.

## STEP 4: Make targeted improvements in selected topics

Based on the outcome of the review of conversation transcripts, you can now make targeted improvements in those selected topics.

Some of the techniques you can apply for reducing topic level escalation rates include adding self-serve capabilities so the user doesn't have to rely on human representatives for an action (for example, checking shipping status), improve the triggering performance to make sure the right topics are presented to the user instead of having to escalate to a human representative (this includes adding missing trigger phrases and updating existing trigger phrases).

### Topic enrichment analysis

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Topic enrichment analysis

Article • 11/19/2024

## Analyzing unrecognized intents

Copilot Studio has built-in [conversational boosting](#) and [fallback](#) system topics. These topics are set to trigger when the Natural Language Understanding (NLU) model isn't able to find a matching topic or action for a given user query. In terms of priority, conversational boosting triggers before the fallback topic.

If most unrecognized utterances are escalated to a human representative—that is, not deflected—there's an opportunity to improve the deflection by addressing the usage patterns of the user that triggers fallback consistently.

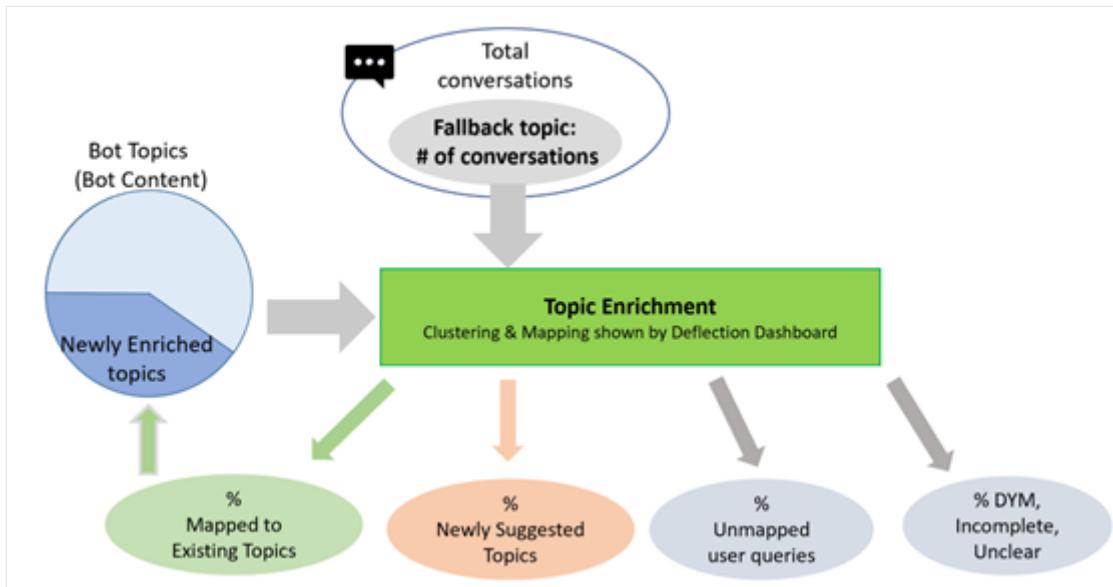
### Tip

Topic enrichment is an offline data analysis exercise, focused on repurposing user queries that triggered the Fallback topic into triggering relevant topics in Copilot Studio.

The analyzed user queries from the Fallback topic typically fall into these buckets:

1. User queries that are expected to trigger existing topics, but somehow are missed by the agent's NLU.
2. User queries that can be converted to newly suggested topics.
3. Unmapped user queries that aren't relevant to any existing or new topics.
4. Other categories, including user queries that triggered a [Multiple Topics Matched](#) (also known as "did you mean") topic followed by conversational boosting or fallback. Unclear user queries that hit conversational boosting or fallback. As well as user queries from incomplete conversations that led to conversational boosting or fallback.

Of the four categories, the first and the second are immediately actionable. Based on the findings from these categories, you can enrich the topics by adding more trigger phrases for existing topics or creating new topics.



## Topic enrichment through out-of-the-box analytics

Copilot Studio provides advanced AI capabilities out of the box. To identify a list of suggested new topics, turn on the [AI features](#) for [topic suggestion from chat transcripts](#), when the author doesn't want to enable the Conversational boosting or Fallback topic.

ⓘ **Important**

Only [classic chatbots or Teams copilots](#) have access to these legacy AI capabilities.

This information can also be used to create new topics to improve deflection rate.



**Topic suggestions from chat transcripts (preview)**  
Each topic suggestion represents related queries about the same issue. Your bot was unable to match these to an existing topic - consider adding a new topic to expand coverage.

Updated 17 hours ago

Topic suggestions (4)	Sessions ↓
memorial day discount	4
holiday deals	3
Surface laptop 4	3
Hololens	2

## Topic confusion analysis

# Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# Topic confusion analysis

Article • 08/01/2024

## Important

Power Virtual Agents capabilities and features are [now part of Microsoft Copilot Studio](#) following significant investments in generative AI and enhanced integrations across Microsoft Copilot.

Some articles and screenshots may refer to Power Virtual Agents while we update documentation and training content.

Using semantically similar trigger phrases for two different topics can lead to confusion; the copilot might not know which topic to open, and needs to ask follow-up questions to the end-user.

## Tip

Confusion is typically measured by the frequency of the [Multiple Topics Matched](#) topic (also known as "did you mean") being triggered.

This often results in the escalation to an agent, and spikes the deflection rate of the copilot.

A **topic confusion analysis exercise** helps you improve topic triggering accuracy by finding overlaps between topics. Resolving topic overlaps can help reduce the need for the copilot to ask clarifying questions before triggering a topic.

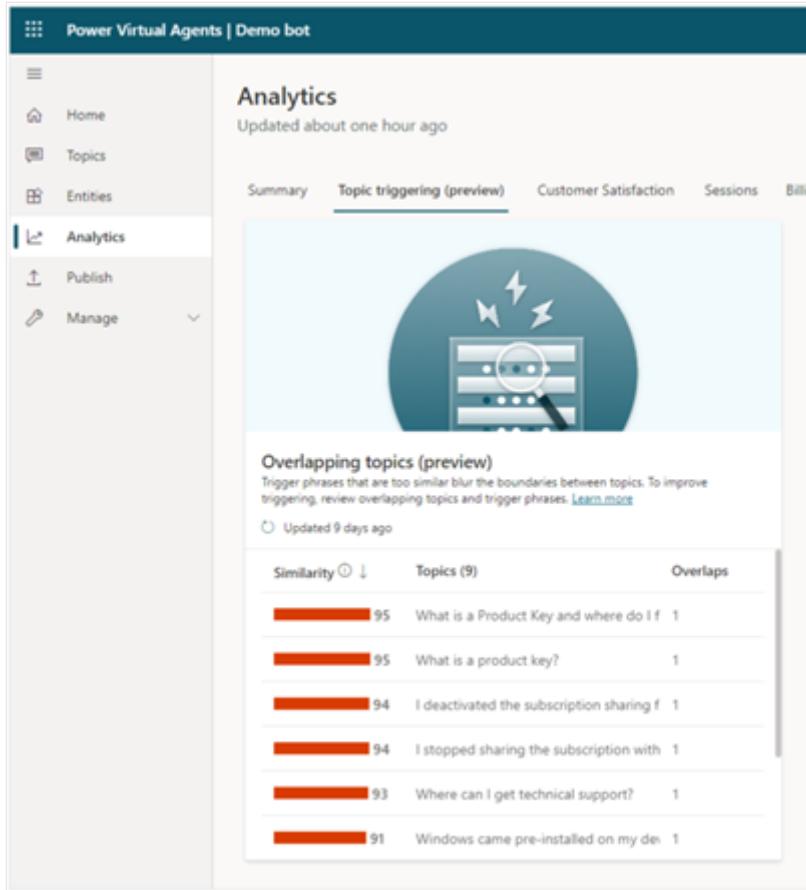
Identifying semantically similar trigger phrases can also help you determine if you have topics that themselves are similar and could be consolidated to simplify the copilot authoring process, or edited to make the topics more distinct with high triggering accuracy, which in turn improves the deflection rate.

You can identify the list of topics causing confusion during triggering, by enabling the [AI features for Teams and Classic copilots](#) in Copilot Studio for [topic overlap detection](#).

This standard capability helps you identify the trigger phrases causing confusion, remove duplicates, and consolidate similar topics.

## Important

Only [classic chatbots](#) or [Teams copilots](#) have access to these legacy AI capabilities.



The screenshot shows the 'Analytics' section of the Power Virtual Agents interface for a 'Demo bot'. The left sidebar includes options like Home, Topics, Entities, Analytics (which is selected), Publish, and Manage. The main content area is titled 'Analytics' and shows it was updated 'about one hour ago'. Below this are tabs for Summary, Topic triggering (preview) (which is selected), Customer Satisfaction, Sessions, and Billing. A large icon with a magnifying glass over a bar chart represents 'Overlapping topics (preview)'. A note states: 'Trigger phrases that are too similar blur the boundaries between topics. To improve triggering, review overlapping topics and trigger phrases.' A link to 'Learn more' is provided. A timestamp indicates the data is 'Updated 9 days ago'. A table lists 9 topics with their similarity scores and overlap counts:

Similarity	Topics	Overlaps
95	What is a Product Key and where do I f...	1
95	What is a product key?	1
94	I deactivated the subscription sharing f...	1
94	I stopped sharing the subscription with ...	1
93	Where can I get technical support?	1
91	Windows came pre-installed on my dev...	1

## Alternate escalation paths

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# Alternate escalation paths

Article • 08/29/2024

## Tip

There are a few strategies you can use to deflect the user from reaching the human agent when the copilot user decides to escalate, without deprecating the user experience.

## Check for operating hours and queue size during escalation

In this approach, you can have a self-service action inside Copilot Studio, for example, a Power Automate cloud flow.

The action that checks for the operating hours of the human agent and agent queue size from the engagement hub using, for example, Dynamics 365 Omnichannel for Customer Service. It then transfers the chat only if the agent is available in the current hour or if the queue isn't full.

If the escalation happens outside the operating hours of the agent or the agent queue is full, then the copilot can redirect the user to email support or schedule a callback.

This approach avoids unnecessary escalation and, in turn, improves the deflection rate.

## Provide an option to create a support ticket during escalation

When the user escalates to an agent, the copilot can redirect through a self-service action to create a support ticket by providing the required details.

Agent transcript analysis

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Agent transcript analysis

Article • 11/19/2024

## Tip

Analyzing representative transcripts can provide insights on what new topics or self-service actions can be added to an agent to reduce escalation and improve deflection.

The representative transcript review process is similar to the [copilot transcript review](#):

1. Identify the top escalation driver topics.
2. Download the representative conversation transcripts from your engagement hub (for example, [download transcripts from Dynamics 365 Omnichannel for Customer Service](#)).
3. Select the most recent sample set of human representative conversation transcripts for that topic (for example, 10 sessions). The size of the sample set depends on how much accuracy you're looking for. For a quick analysis you can start with 10 sessions.
4. Read through each of the human representative conversations and identify the resolution path provided by the human representatives and patterns that are emerging for that conversation.
5. List out the resolution paths identified for each session and group them as per the resolution paths.
6. For each resolution path group, come up with a recommendation to implement in Copilot Studio as a topic or a self-service action, and so on.
7. Implement the recommendations in the agent topics and observe the change in the escalation rate and deflection.

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Analytics considerations for Copilot Studio vs Dynamics 365 Omnichannel

Article • 11/19/2024

Analytics is a top priority for Dynamics 365 Omnichannel and Copilot Studio customers tracking the performance of their agents and calls center representatives. There are major efforts currently underway to improve the analytics across both these products. While the analytics continue to improve, there are several considerations you can adopt as an agent builder or customer service expert to improve your understanding of analytics across Copilot Studio and Dynamics 365 Omnichannel to enhance agent performance.

This article touches on concepts that help you understand the difference between Dynamics 365 Omnichannel analytics and Copilot Studio analytics, and the purpose behind the analytics tools in each of them. This helps you make informed decisions about the agent performance for your implementation.

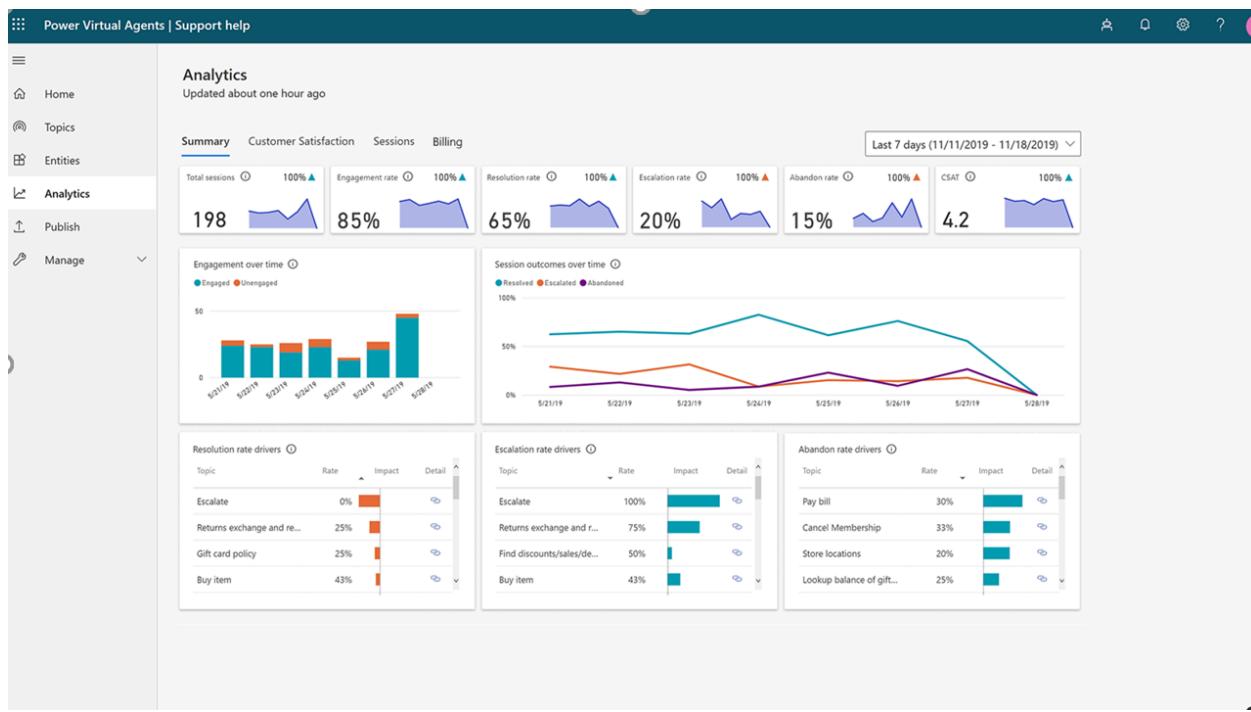
## Using Copilot Studio Analytics

The standard analytics tools in Copilot Studio agents are focused mainly on providing insights about agent topics performance through during chat sessions. They don't provide an end-to-end insight that includes any human-representative conversation. But they can be used to improve the topic performance inside the agent.

The insights you can get from Copilot Studio analytics include metrics for questions like:

- How engaged are users with the agents?
- Which topics are driving resolutions, escalations, and abandonment?

In general, Copilot Studio analytics tools are useful to enhance the agent performance in terms of the topics. This gives insights about what new topics need to be created, and what topics need to be updated to achieve better resolution rates for future user queries.



To learn more, see [Key concepts – Analytics](#)

## Understanding Copilot Studio analytics definitions

It's important to understand the way Copilot Studio calculates the various metrics made available to [Analyze agent performance and usage in Copilot Studio](#). These calculations are based on the agent sessions. A session can be an *unbilled* or a *billed* session. A billed session is a continuous interaction between an agent and a user to address all of user's question within a 60-minute time window.

A billed session can include multiple user authored topics and can't last longer than 60 minutes or 100 turns.

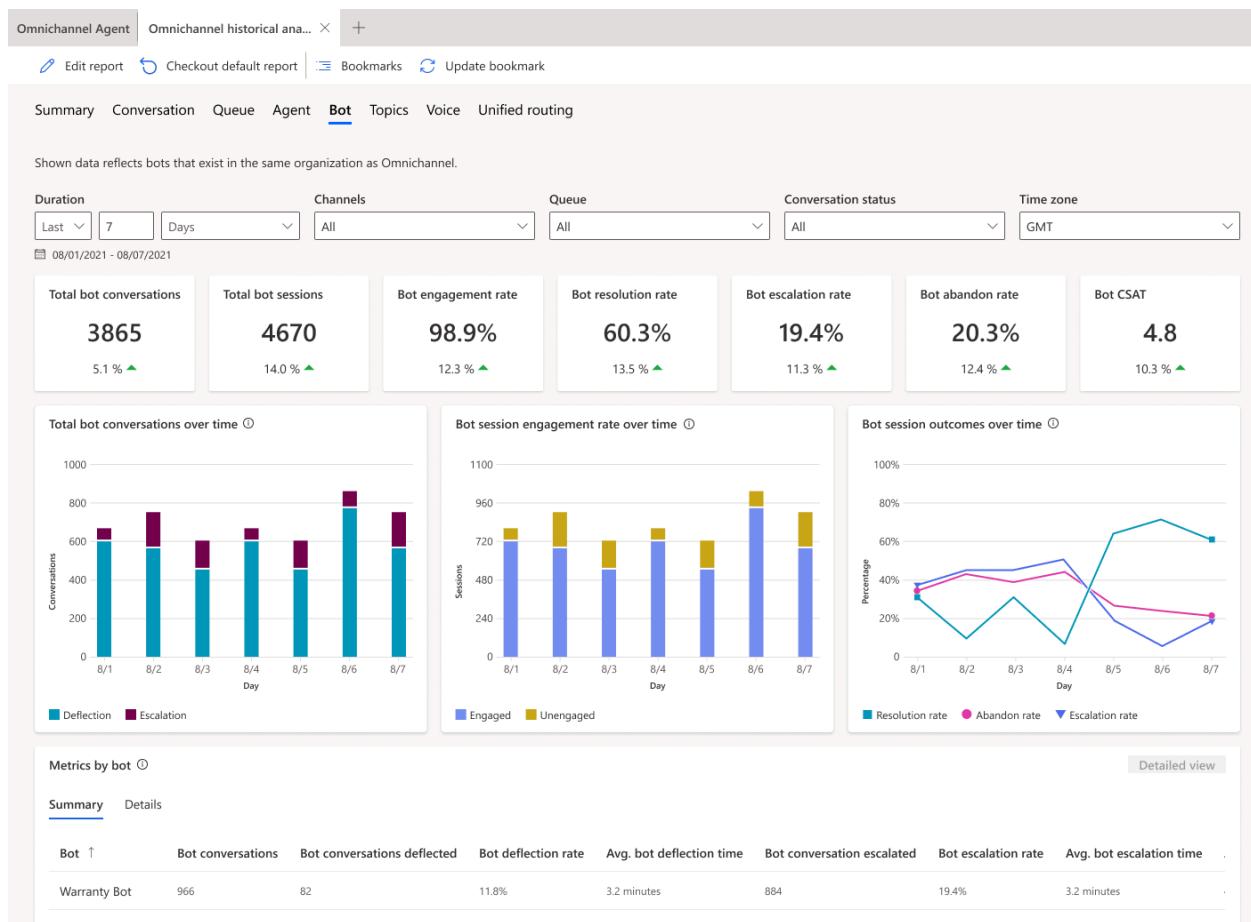
A session is interaction between an agent and a user to address or assist a user with one query or intent (one user authored topic). Most sessions are *engaged*, which means that a user-authored topic or an escalation was triggered. There are three possible outcomes for an engaged session:

- Resolved: Customers are offered an end-of-conversation survey and either responds yes or leaves without a response.
- Escalated: Customers indicate that they need a customer service representative to help resolve the issue.
- Abandoned: An engaged session that isn't resolved or escalated and is inactive after 60 minutes from the beginning of the session. This would happen, for example, if the user leaves.

A topic is tied to a dialog flow built in Copilot Studio. One topic contains one dialog flow.

# Using Dynamics 365 Omnichannel chatbot analytics

The [standard bot analytics tools in Dynamics 365 Omnichannel](#) are focused on providing insights about how the overall agent deflection is performing through the conversations. These analytics are optimized to give insight into the overall interaction/conversation across both agent and human representative.



The insights you can get from Dynamics 365 Omnichannel include:

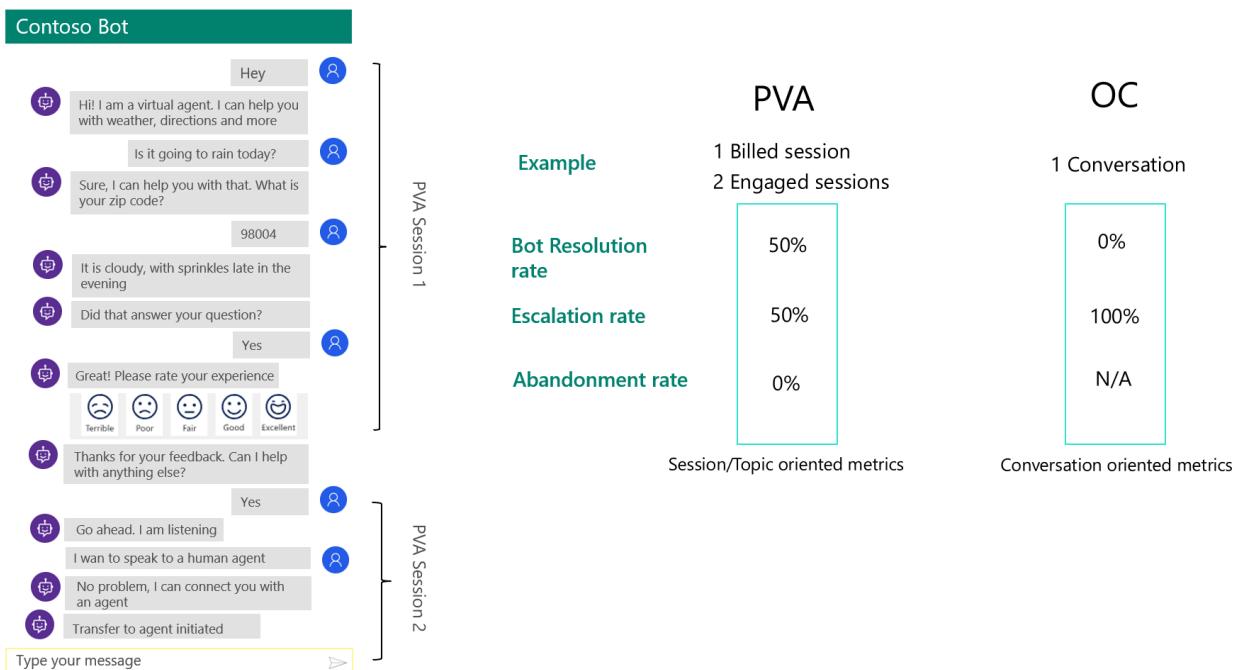
- Bot resolution time
- Bot escalation time
- Conversation escalation

## Copilot Studio vs Omnichannel analytics comparison

Analytics calculations differ between Copilot Studio and Dynamics 365 Omnichannel for metrics with similar names, such as resolution rate, escalation rate, and abandonment rate.

For example, Copilot Studio analytics might consider an ongoing interaction with a user that would be considered one conversation, and one agent session in Dynamics 365 Omnichannel analytics as multiple separate analytic sessions. As a result of this, the resolution rates, escalation rates, and abandonment rates can be different across Copilot Studio analytics and Dynamics 365 Omnichannel analytics.

Here's an example showing the differences for analytics between the two for a single conversation:



These key differences in the calculation of resolution rates and escalation rates need to be considered when consuming the Copilot Studio and Dynamics 365 Omnichannel analytics to determine key metrics like deflection rate, resolution rate, and so on. The general recommendation is that you rely on Dynamics 365 Omnichannel analytics for tracking the overall agent deflection rate for the organization and use the Copilot Studio analytics for enhancing the topic performance.

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Custom analytics strategy

Article • 11/19/2024

Copilot Studio provides comprehensive [out of the box analytics](#) that allow customers to understand an agent's usage and key performance indicators.

Customers can view reports related to the following analytics:

- Performance and usage
- Customer satisfaction
- Session information
- Topic usage
- Billed sessions

However, there are often scenarios where you need to create or use custom analytics.

For example, you might need to:

- Share analytics with stakeholders or users.
- Report on conversation transcripts data for a period longer than the default last 30 days.
- Design a report not covered by out-of-the-box analytics.

There are a few ways you can take the analytics data recorded by Copilot Studio and use it in customized reports.

## Copilot Studio analytics sample template report

To create custom analytics, our recommended approach is to start with the [Copilot Studio analytics sample template report](#). The sample template report is a set of open-source assets, distributed through GitHub, which accelerates the time it takes to create a report that renders in Power BI.

### Warning

- The solution isn't part of the core Copilot Studio offering and requires configuration.

- The sample report isn't supported by Microsoft, but you can raise issues in the GitHub repository to get help from the community.

## Dataverse

### Conversation transcripts data

The analytics shown in the Copilot Studio portal come from a data service residing within Copilot Studio. Usage data is also written to the associated Dataverse environment, in the [conversation transcripts](#) table.

### Retention periods

By default, both sources have a data retention of 30 days, but customers can [change the retention period for conversation transcripts in Dataverse](#).

### Copilot Studio Dataverse tables

Copilot Studio uses the following tables for custom analytics in Dataverse:

- [Chatbot](#) (`Bot`). This table includes details of each agent in an environment. The details are often a small amount of data.
- [Chatbot Subcomponent](#) (`BotComponent`). This table lists the topics, entities, and dialogs associated with the agent in your environment. The details are often a small amount of data.
- [Conversation Transcripts](#) (`ConversationTranscript`). This table contains detailed conversation data for all the agents in your environment. The size of the data in this table is related to the use of the agent and can be large.

### Azure Synapse Link for Dataverse (Azure Data Lake Storage Gen2)

When there's need to retain conversation transcripts longer than the default retention period, build custom metrics or dashboards. Our recommended approach is to export raw conversation transcripts data to a more cost-effective data store like Azure Data Lake Storage Gen2 using the [Azure Synapse Link for Dataverse feature](#).

The export creates an incremental sync of configured Dataverse tables in the Azure data lake, using the Common Data Model format.

There are more steps you need to take, including what's configured in the base template:

- [Create an Azure Data Lake Storage Gen2 and connect it to Dataverse](#).
- During configuration, select the *ConversationTranscript* table (*Chatbot* and *Chatbot Subcomponent* don't support incremental sync).
- [Follow the guidance](#) ↗ for setting up Power BI dataflows to process the incoming data.

### ⚠️ Warning

By default, Azure Synapse Link for Dataverse mirrors the configured table data from Dataverse to the Azure data lake. Therefore, any record that gets deleted in Dataverse (for example, by the recurring bulk delete job that deletes conversation transcripts older than 30 days by default) will also be removed from the Azure data lake. To work around this, you can create copies or snapshots of your data in the Azure data lake, or you can configure the synchronization to use the [append-only mode](#).

## Power BI

The custom analytics solution template includes a Power BI report that processes the raw transcript data (using Power Query) into a report that matches the Copilot Studio default analytics.

In addition, users of the report have access to the following information:

- Data for all agents in an environment.
- Data as far back as the feed provides (in Dataverse or in the Azure Data Lake Storage Gen2).
- Raw data tables extracted from the Conversation Transcript table data – which can be used to create custom reports.
- A transcript viewer, allowing users to see the actual conversations that occurred.

### ⚠️ Warning

This a complex report that performs significant transformations on the base data. Customers with Conversation Transcript tables greater than 80 MB should look to use the Azure Synapse Link for Dataverse and Power BI dataflows version of the report.

---

# Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗

# Optimize prompts and topic configuration

Article • 11/19/2024

When using Generative AI, it's important to understand the best practices related to prompting and topic configuration. This article shares a few key practices, and helps you integrate guardrails to optimize results, and reduce errors and hallucinations in generated responses.

## Generative AI settings

Normally, Generative AI settings are configured at the start of the agent build (see the Generative AI tab once the agent is created). These settings enable you to define a knowledge base for the agent and clarify the content leveraged for the generation of answers, including the following:

- Website URLs
- Uploaded documents
- Desired moderation level

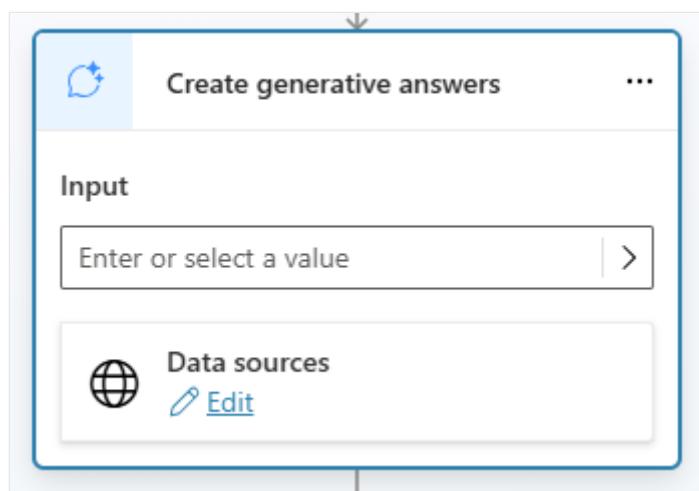
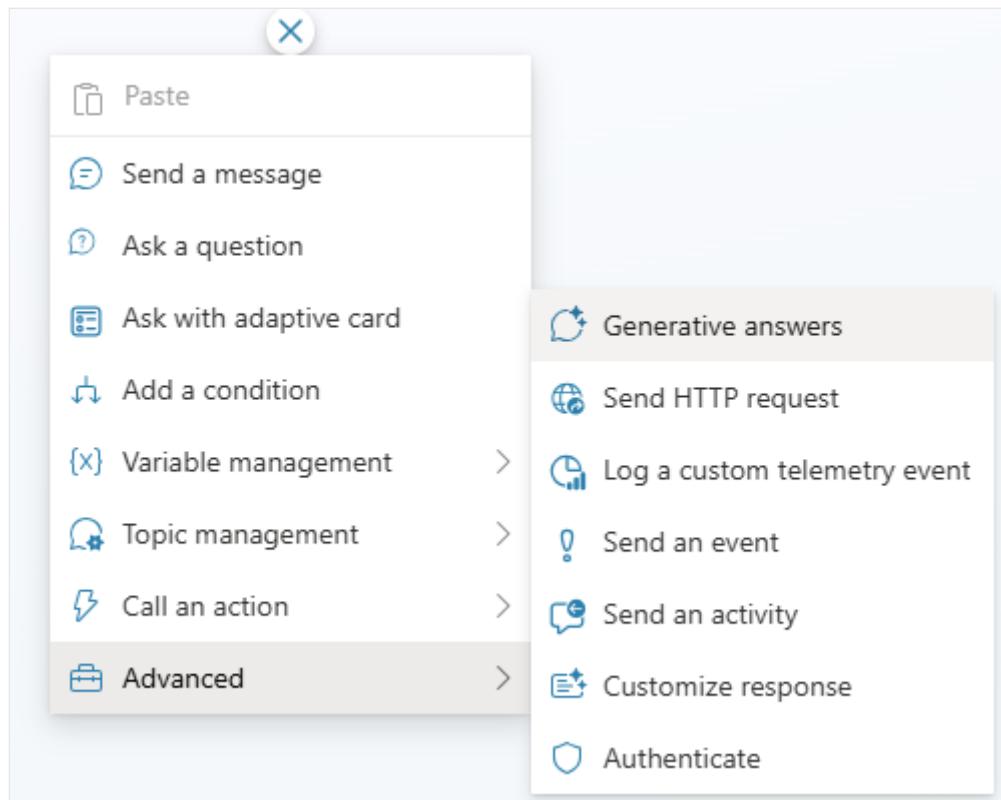
Consider these settings as the broadest knowledge base of information for the agent to be used, unless there are specific instructions configured in a triggered topic flow.

To learn more about the content types supported by Generative AI, and how to boost natural language capabilities, see the following articles:

- [Quickstart: Create and deploy an agent](#)
- [Add a generative answers node](#)
- [FAQ for generative answers](#)

## Generative answers with custom instructions

In addition to the Generative AI settings page, you might also add generative answers as a node to any topic flow.



Once a node is added, there are several ways to help optimize prompts. You might include fields for configuring the input prompt, data sources, and custom instructions. Custom instructions have a high return on investment, but are also often ignored.

To locate custom instructions, from the **Create generative answers** node dialog open the **Properties** pane, by selecting the node's **...** then **Properties**.

The screenshot shows the 'Create generative answers properties' dialog box. On the left, there's a vertical stack of nodes: a 'Trigger' node (blue icon), a 'Phrases' node (blue icon) with an 'Edit' button, a 'Phrases' node (grey icon) showing 'No phrases to show', a 'Create generative answers' node (blue icon), an 'Input' node (grey icon) with a text input field 'Enter or select a value', and a 'Data sources' node (blue icon) with an 'Edit' button. Below the 'Data sources' node is a blue plus sign button. On the right, the dialog box has sections for 'Knowledge sources' (with a 'Search only selected sources' toggle and an 'Add knowledge' button), 'Classic data' (with a 'Allow the AI to use its own general knowledge (preview)' toggle), 'Content moderation level' (with a 'Customize' checkbox), and a large red-bordered area for 'Customize your prompt with variables and plain language'. Inside this area is a placeholder text '{x} fx' and a character count '0/2000'.

Create generative answers properties

Search for content or provide your own context, use OpenAI to create a summarized response using the provided data.

[Learn more](#)

**Knowledge sources**

Get search results from the knowledge sources using selected mode.

Search only selected sources

[+ Add knowledge](#)

> **Classic data**

Allow the AI to use its own general knowledge (preview)

Customize

{x} fx

Customize your prompt with variables and plain language

0/2000

## Sample prompts

The following examples illustrate a typical agent experience involving a customer asking questions, which results in the flow of the topic arriving at the generative answers node. The first example shows a prompt without custom instructions, while the second shows a prompt using custom instructions.

### Sample prompt without custom instructions

*Prompt: What should I consider when buying an electric car (for a Lexus agent)?*

*Response: When considering the purchase of an electric car, there are several factors you should take into account to ensure that you make an informed decision that suits your needs and preferences. Here are some key considerations:*

1. Charging Infrastructure: Assess the availability of charging stations at home, work, and in public areas. Some companies offer charging stations for employees, so it's worth checking if this is available at your workplace.<sup>1</sup>
2. Range: Consider how far you can drive on a single charge and whether it meets your daily commuting needs.
3. Cost: Evaluate the total cost of ownership, including the purchase price, incentives, tax breaks, and the cost of electricity versus gasoline.
4. Maintenance: Electric cars typically require less maintenance than traditional gasoline cars, but it's important to understand the long-term maintenance requirements and costs.
5. Performance: Test drive different models to experience their acceleration, handling, and comfort.
6. Size and Capacity: Ensure the car fits your personal or family needs in terms of size and cargo space.
7. Technology: Electric cars often come with advanced technology features. Determine which technology features are important to you, such as regenerative braking, advanced driver-assistance systems, or connectivity options.
8. Environmental Impact: Consider the environmental benefits of driving an electric car, such as reduced emissions and a smaller carbon footprint.
9. Resale Value: Research the car's resale value and how well it retains its value over time.

 **Note**

The remaining numbered items, and a recommendation to talk to a friend, were truncated. The answer was already long, which doesn't help Lexus to inspire a car purchase.

## Sample prompt with custom instructions

*Prompt: What should I consider when buying an electric car (for a Lexus agent)?*

*Response: The top three factors to consider when buying an electric car in 2024 are:*

1. Range: Ensure the car's range aligns with your daily travel needs.

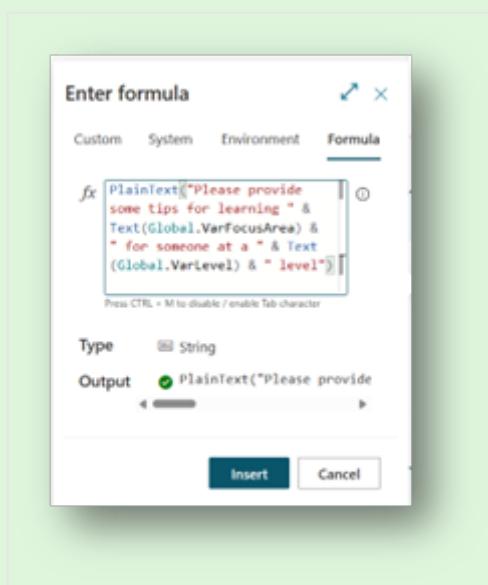
2. Charging Options: Consider the availability and convenience of charging facilities.

3. Price and Incentives: Factor in the cost, including any available tax benefits or incentives.

For Lexus, the RZ450e stands out in our 2024 electric lineup, offering a blend of luxury and performance with advanced technology features. For detailed information on our electric vehicles, please refer to the Lexus website or contact us directly for personalized assistance.

### 💡 Tip

In both the custom instructions, as well as the actual input prompt, you can also leverage a formula to make the prompts more dynamic, based on variables existing before the node. The following example shows where the prompt changes, based on the desired focus area and the student's level (previously asked in the Topic flow).



Note that any prompts and/or instructions configured by the agent maker aren't submitted to the same level of safety filtering as are customer prompts.

## Feedback

Was this page helpful?

Yes

No

[Provide product feedback ↗](#)

# Optimize prompts with custom instructions

Article • 11/19/2024

This article discusses the best practices for adding custom instructions to [generative answer nodes](#) within Copilot Studio topics. Key areas of consideration include assigning a role to the agent, specifying format and style, and clearly indicating what you *don't* want in the response.

Here are some other factors you want to keep in mind:

- **Clarity and context:** Ensure that instructions are clear and provide enough context for the agent to understand and respond accurately. This clarification helps to get more relevant and precise results.
- **Supplemental instructions:** Add extra instructions in the following situations:
  - The user likely answers with a single word or short sentence, for example when they're responding to Adaptive Cards, or previous questions from the agent. (Variables from previous answers might be involved in the prompt sent to the generative AI.)
  - The user is expected to appreciate the information in a more structured format, such as bullet points, numbered lists, or with specific metadata. For example: "Make a list of the most popular product options with the Product Name, SKU, and Price."
  - When you don't want something returned as part of the answer. For example: "Don't speak in the third person, instead play the role of a Contoso Hotel Suites concierge."
  - When you want a unique personality level. For example: "Your tone should be friendly, helpful, cheerful, and expressive." (Words like "friendly" and "expressive" often include emojis in the response.)
- **Add step-by-step explanations:** When seeking complex information or formats, supplement user prompts with step-by-step guidelines in the custom instructions. These instructions can lead to more dependable results, since it gives your agent enough information to generate the best response.
- **Audience:** Mention the intended audience. Is it for students, professionals, or the general public? Understanding the audience allows your agent to tailor its tone

and level of technicality appropriately.

- **Review analytics and measures for refinement:** Regularly review your agent results and refine custom instructions based on the desired outcomes. This practice helps enhance the performance of your agent, and tailor it to your specific needs.

## Practical custom instruction examples

  Expand table

Weak instruction	Improved instruction
<div style="border: 1px solid #ccc; padding: 5px;"><p>Custom instructions (preview) ⓘ</p><p>(x) fx</p><p>List the most popular coffee shops.</p><p>35/2000 maximum characters</p></div>	If a user asks about coffee shops, focus on promoting Contoso Coffee in US locations, and list those shops in alphabetical order.
<div style="border: 1px solid #ccc; padding: 5px;"><p>Custom instructions (preview) ⓘ</p><p>(x) fx</p><p>Provide a detailed explanation.</p><p>31/2000 maximum characters</p></div>	Format the response as a series of steps, starting each step with "Step 1:", "Step 2:" in bold. Don't use a numbered list.
<div style="border: 1px solid #ccc; padding: 5px;"><p>Custom instructions (preview) ⓘ</p><p>(x) fx</p><p>Customize your prompt with variables and plain language.</p><p>0/2000 maximum characters</p></div>	Return a list of the top five universities within 100 miles of {State Variable} with University Name, Location City, and Number of Students.

## Related content

- [Use prompt modification to provide custom instructions to your agent](#)
- [FAQ for generative answers](#)
- [Generative answers pointing to SharePoint sources don't return results](#)

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Optimize prompts using Bing Custom Search

Article • 11/19/2024

When preparing for Generative Answers using public data, it's sometimes challenging to target specific content in the context of expected question flows. In these cases, we've facilitated easy integrations with [Bing Custom Search](#) to optimize generative answers and prompting opportunities.

Say we have an agent for an automobile company website that's designed to help customers locate the right car for their next purchase. The website might mention a variety of vehicles, from many pages, and from different perspectives. So, when a customer asks questions about a particular model, the agent leverages any content related to that model found on the website, as configured in the settings or the Generative Answers node. A broad context approach like this might result in suboptimal answers and low levels of customer relevance.

Bing Custom Search supports Copilot Studio Generative Answers as it helps to target and/or exclude public facing content and can be leveraged by the agent maker to test and optimize queries. This also affords a platform for testing and optimizing common queries.

- Targeting Content: Use the **Active** tab of Bing Custom Search to specify which URLs are active and available for search queries.
- Excluding Content: Use the **Exclude** tab of Bing Custom Search to specify which URLs are to be excluded (and therefore won't be used in query results).
- Optimizing Queries: You can pin queries that you're sure your users will ask frequently. For more information about pinning queries, see [Pin slices to the top of search results](#).

Therefore, it's a best practice to leverage Bing Custom Search when it enables the agent to provide the most accurate answers in context of user intent. How might we do this in the scenario mentioned above for the Contoso Premium Vehicles Copilot?

## Responding to the example scenario

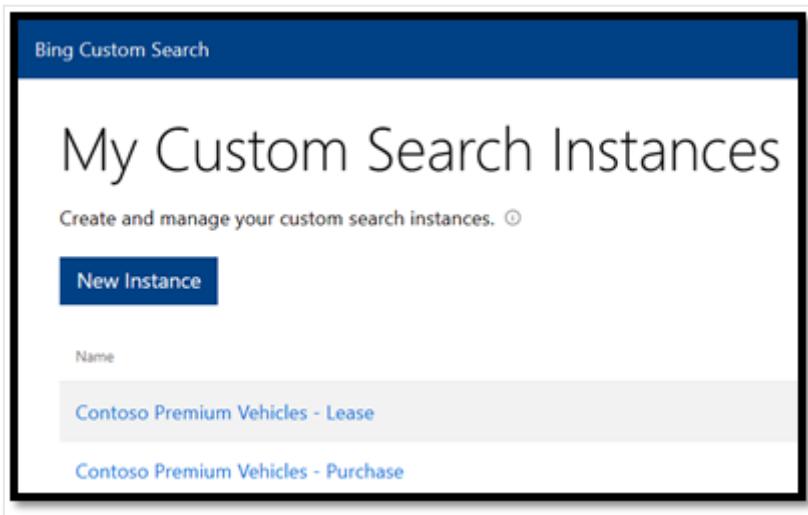
In the scenario mentioned earlier in this article, the Contoso Premium Vehicles company has an agent to help customers find the right car for their needs. Their goal is to increase conversion rates and expedite purchases. To begin the customer journey, we'd

first need to determine the preferred purchase option (i.e. purchase or lease), then we can use Bing Custom Search to send the agent to the appropriate page(s) for that type of purchase, while excluding any expired offers.

### ⓘ Note

Contoso Premium Vehicles is a fictitious website. Therefore, you won't be able to test these steps with its website links. Just note the logic behind these steps for your own purposes, replacing the links with your own website specifics.

1. Create an account for Bing Custom Search at [Bing Custom Search](#).
2. Create two new instances for the Contoso Premium Vehicle's Copilot, one for *Purchase* and one for *Lease* (as shown in the following screenshot). We're creating two instances because there are different content targets depending on the purchase option, and we plan to surface each instance in a different plugin/topic:



3. Let's start with Lease. Open the instance "Contoso Premium Vehicles – Lease" and add a page for <www.ContosoPremiumVehicles/LeaselInventory> on the **Active** tab.
4. We exclude the page with last year's expired lease inventory by clicking the **Blocked** tab and adding this URL: <www.ContosoPremiumVehicles/2024Specials>
5. Press **Publish** to save this change (the first publish also gives a configuration ID for use in Copilot Studio).
6. Next, open the instance "Contoso Premium Vehicles – Purchase" and add a page for <www.ContosoPremiumVehicles/PurchaselInventory> on the **Active** tab.
7. Press **Publish** to save that change.

Now you're ready to leverage the results of these configurations in topics with a **Create generative answers** node, and your agent will only generate answers based on the content you specified.

The screenshot shows the Microsoft Bot Framework designer interface. On the left, there's a vertical stack of nodes:

- Identify**: Contains a **User's entire response** input field.
- Save user response as**: Set to **{x} Global.OrderId string**.
- Create generative answers**: This node is highlighted with a red circle containing the number **1**.
- Data sources**: Set to **Bing Custom Search**. This section has three numbered callouts:
  - Search with Bing Custom Search** (red circle with **2**)
  - Configuration ID** (red circle with **3**)
  - SharePoint** and **Manual input**
- Input**: Contains a **Text("What offerings do yo...")** input field.
- Data sources**: Shows **Bing Custom Search** selected. A red circle with **1** is placed over the "Data sources" header.

On the right, there's a panel titled **Classic data** with sections for **Data sources**, **Search public data**, **Bing Custom Search**, **SharePoint**, and **Azure OpenAI Services on your data**.

## Feedback

Was this page helpful?

Yes

No

Provide product feedback ↗

# Conversational AI project best practices

Article • 11/19/2024

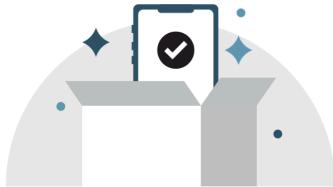
Adoption of low-code technologies like Copilot Studio can help customers envision and deliver solutions with less management and fewer maintenance costs.

## Engagement principles

Customers approaching a Copilot Studio project should align project delivery with their own company culture and objectives. However, there are some core principles that are adopted again and again among successful Copilot Studio projects.

## Principles

[Expand table](#)

		
Engage your flywheel	Keep it simple	Keep goals in focus
<p>When the Copilot Studio agent goes live, the "customer feedback flywheel" takes effect quickly. Actionable customer feedback comes in almost instantly. Delaying the publishing of your agent for more topics or processes means foregoing valuable user feedback. Encourage the team to see going live as the beginning of your project, not the finale.</p>	<p>One of the core value propositions of Copilot Studio is iteratively delivering a conversational AI experience quickly. After the project goes live, maintainability is key to staying agile. Tooling can allow users to create complex conversation flows. However, extensibility with the Bot Framework enables flexible interactions, therefore decide if complexity is always necessary.</p>	<p>It's easy when implementing new technology, like an agent, for developers to focus on the minor details of implementation. Define your key business goals, for example to deflect 30% of customer queries or score above 4.0 customer satisfaction. Goals can keep the project moving and ensure correct feature prioritization.</p>

## Best practices

## 1. Focus on delivering immediate value

Building agents is an iterative initiative. You learn from your users what they're looking for. Use these observations to drive further investment.

## 2. Establish role clarity

Make it clear who's the decision maker, so there's no ambiguity in delivery.

## 3. Identify risks early and plan contingency accordingly

Occasionally customers run into technical issues when implementing specific requirements or integrating with other systems. These issues can take time to diagnose.

## 4. Agree on issue priorities

Define a *blocker*, a *must have*, and a *nice to have*. Decide how aggressively you want to push for workarounds, while the underlying issues are addressed. Workarounds are a natural part of projects as they allow the project to make progress, while a longer-term solution is found.

## 5. Determine your Application Lifecycle Management (ALM) strategy

Move to the production-level process as early as possible. Don't linger in development phases, when there are timelines to be met.

### ⓘ Note

For more information about analyzing usage, see [Combine web and agent analytics](#).

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Getting started

Article • 11/19/2024

As you get started using Copilot Studio, it's important to understand how project goals, use cases, and pricing affect the success of your Copilot Studio project.

## Set goals

We recommend customers separate the project goals into three distinct areas: core delivery, business metrics, and governance. If agents are brand-new to your organization, then the examples in this article are a good starting point. Once you go live, you quickly learn what's important, how your customers behave, and where you would like to improve.

## Project goals

Here's an example of the goals a customer defines at project inception.

### 1. Delivery goals - Minimum Viable Project (MVP)

- a. Go-live with an agent on a target date fewer than three months.
- b. The agent can answer your target use cases, using a script of test phrases.

### 2. Business goals after you go live

- a. *Resolved* session percentage: the project team aims to increase this number over time. This figure is often referred to as the *deflection* amount.
- b. *Escalated* session percentage: the project team aims to reduce this number over time.
- c. *Abandoned* session percentage.
- d. *Deflection or Resolution*: the business value per session.
- e. Monthly Active Users (MAU).

### 3. Establish agent governance and management model

- a. The Application Lifecycle process and automation setup.
- b. The agent author onboarding process.

- c. The reporting and review cadence.

## Build use cases

Creating inspiring use cases is a key part of beginning your agent building journey.

The program includes:

1. Interested employees participate in a [Copilot Studio in a Day](#) training, where they learn the fundamentals of conversational agent building.
2. Once attendees have experience building agent, split the training group into small teams of two to three people.
3. Allow teams to identify moments in the use case they could solve using an agent.
4. Each team should establish milestones over a four-week period that includes activities for each team member. This practice is similar to a project plan.
5. After two weeks, each team of the training group presents their agent, demonstrates their progress, and identifies any challenges or gaps.
6. After four weeks, each team of the training group reveals their final product to the rest of the group and leaders from the respective business areas.
7. When possible, provide some level of reward to the participants of this initiative. Incentives can vary from recognition to monetary rewards.
8. Include a *path to production* period to ensure all agents are finalized into production.
9. Share success with the rest of the organization, and inspire others to do more.
10. Rinse and repeat. Now you have a program that can run on a regular cycle to promote agent development within your organization.

### 💡 Tip

Hackathons encourage organizations to learn and apply what they've learnt in real-world use cases to develop agents.

## Pricing

You can subscribe to plans for Copilot Studio and for Microsoft 365 Copilot. Which plan you should choose depends on you or your organization's needs. To find more information or to purchase a plan, go to [Pricing information for Copilot Studio](#) or contact your Microsoft account team.

[Building a team](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#)

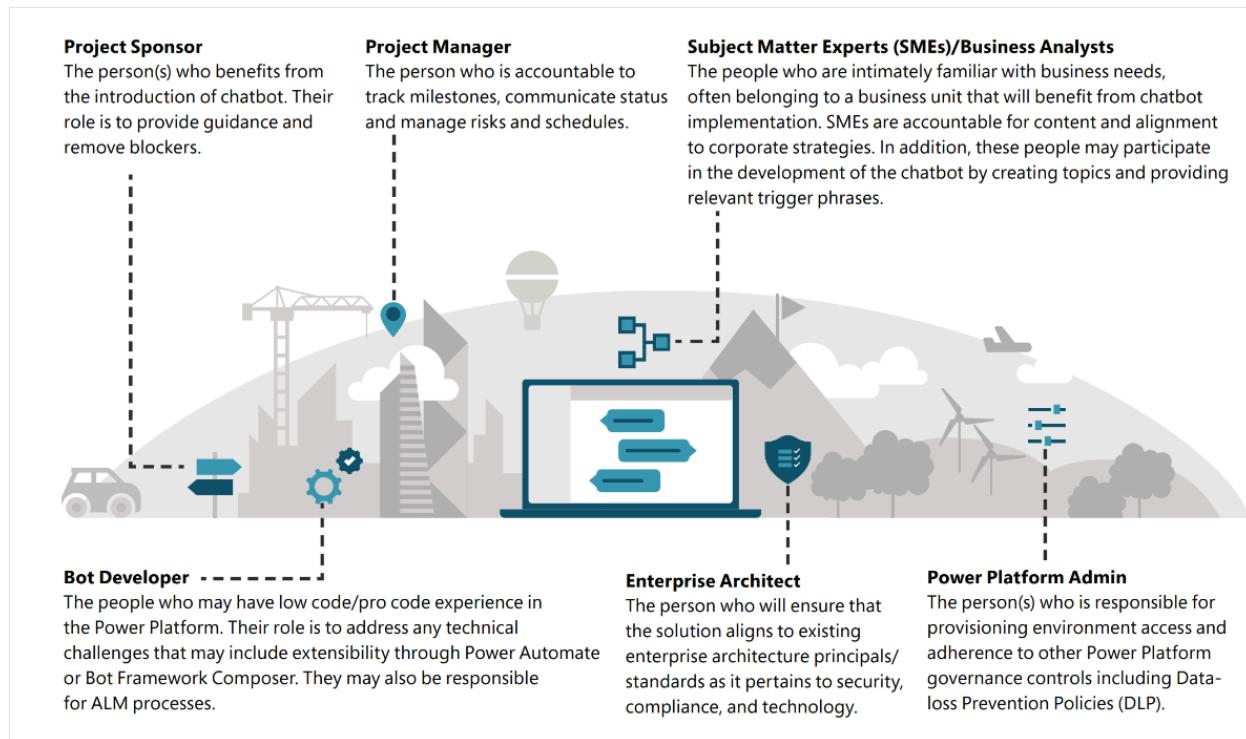
# Building your team

Article • 11/19/2024

Once you understand your customer use cases for your agent, you can form a team and get them trained. Delivering a large agent is a complex project with many different stakeholders within and beyond your organization. This section outlines the roles and responsibilities to align everyone in the project.

## Internal team

Most customers treat an agent implementation like any other project. It usually consists of the following roles:



## External team

The **Third party delivery partner** is typically a System Integrator that leads various activities.

1. **Project management**—Sets up workstreams and reviews with the customer.
2. **Copilot development**—Developers and analysts who have experience with Copilot Studio, agents, and related technologies.
3. **Training**

**4. Administration/governance setup**

**5. Hackathons and other nurture events**-Drives technology adoption and knowledge sharing.

## Microsoft

**Microsoft delivers the Copilot Studio service.**

**1. Account team** - Account Executive, Account Technical Strategist, Technical Specialist, and Customer Success Manager

- a. Alignment on success and goals with the customer.
- b. Ultimate advocate for the customer within Microsoft.
- c. Leads business discussions and escalations.
- d. Attends all business calls until goals are met.

**2. Product group**

- a. Develops product features based on customer needs.
- b. Customer Advisory Team (CAT)–Engages with select customers, providing guidance on how to adopt and deploy the technology. Includes architecture reviews and monthly executive syncs.

**3. Microsoft Support** - Works with customers to solve issues with Microsoft software.

Aims to solve issues directly but can also bring in engineering resources for challenging issues.

## Training your team

For organizations that are new to building Copilot Studio agents, training might be needed for the customer to become familiar with the technology. The following resources can assist customers with building Copilot Studio agents, [aka.ms/LearnPVA](https://aka.ms/LearnPVA).

Training can be delivered through various channels. These channels include Technical Sales Professionals (TSPs), Microsoft Technology Center (MTC) architects, and Microsoft certified partners. In exceptional circumstances, engineering resources like Power CAT can facilitate these training sessions.

## Copilot Studio in a Day

[Copilot Studio in a Day](#) is an instructor led, hands-on course that familiarizes agent makers with Copilot Studio.

The course includes:

- Build your first agent
- Topic suggestions
- Improve your agent with entities, variables, and topic redirections
- Integrate with prebuilt Power Automate flows
- Build Power Automate flows for your agent
- Authentication
- Export and import agents, using solutions
- Create an agent in Microsoft Teams
- Topic design
- Extend your agent using Bot Framework skills (optional)
- Add a telephone number to your agent using AudioCodes (optional)

## Copilot Studio Architecture Series

The [Copilot Studio Architecture Series](#) can be consumed on demand. This course focuses on considerations that customers should account for when embarking on an agent project.

The course includes:

- Planning your first agent
- Building your first agent
- Extending your Copilot Studio agent with Power Platform
- Extending your Copilot Studio agent with Bot Framework (to be announced)
- Advanced topics
- Security and governance

**Project Planning**

---

## Feedback

Was this page helpful?

 Yes

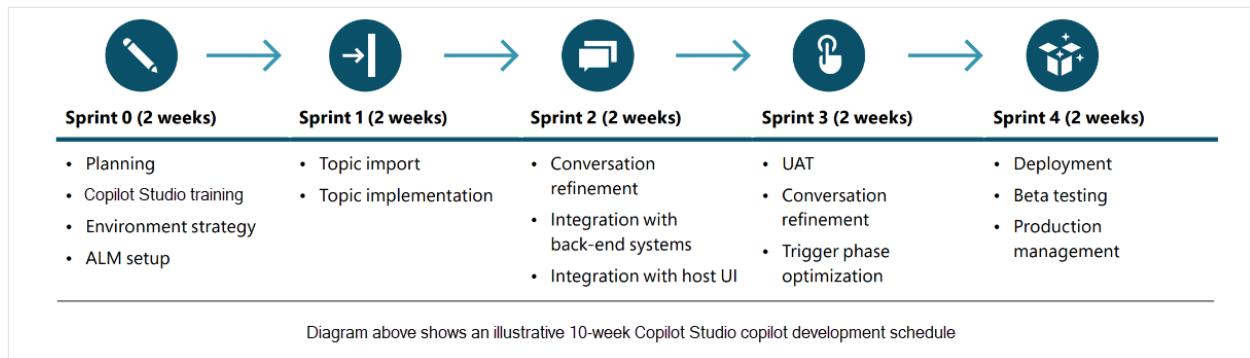
 No

[Provide product feedback](#)

# Project planning

Article • 11/19/2024

Copilot projects based on Copilot Studio can have different timelines and specifications depending on the use case and organization. However, one advantage that low code technologies bring is the *rapid time to market* as shown in the 10-week timeline.



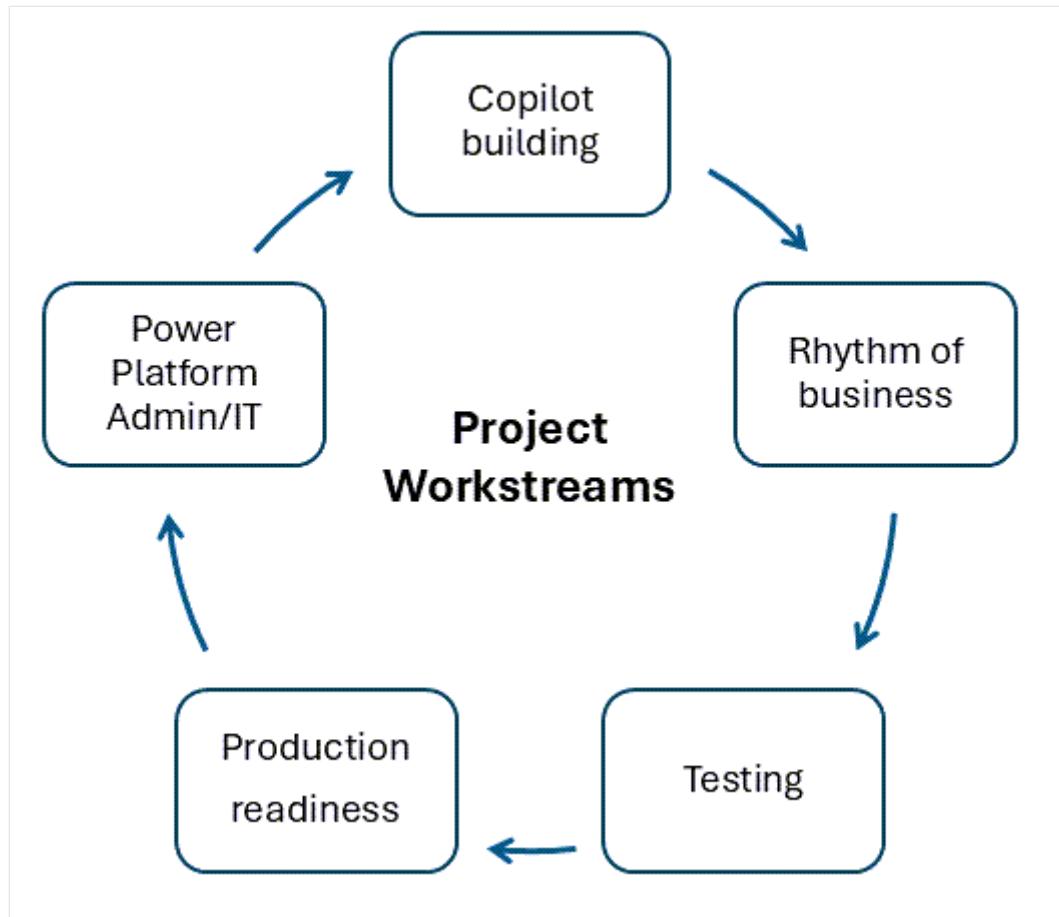
## Sample project overview and timelines

The advantages in choosing a rapid delivery timeline as your initial project phase include the following:

- Accelerate customer adoption by first developing a minimum viable product.
- Generate customer feedback (see [Engage your flywheel](#)) to provide insights on what can be improved.
- Review insights to return value to your product and organization, enabling new developments faster.

Whether this approach is right for your use case depends on organizational need, support, and the ability to align the necessary workstreams.

## Project workstreams



Setting up an adoption platform requires several workstreams that can execute in parallel, each with their own goals, timelines, and owners.

The following five workstreams summarize and describe responsibilities:

## Copilot building

1. Build out related content for agent use cases.
2. Content creation and development tasks.

## Rhythm of Business (ROB)

A Rhythm of Business (ROB) meeting allows all participants to provide status updates and blocking issues, which stakeholders aim to resolve. These meetings should be actionable and driven by dates and business goals.

### 1. Customer driven

Microsoft and other partners attend.

The customer is responsible for booking meetings and setting the agenda. Run this meeting as an engineering project with success metrics and scorecards.

Activities and features for the meeting include:

- Standup meetings for agent development, one or two times per week.
- Weekly demos to show progress.
- Weekly status update emails, highlighting blockers.
- Bi-weekly bug prioritization.
- Celebrate successes.
- Periodic retrospectives.
- Weekly office hours to support customers adoption.
- Teams channel for open communication.

## 2. Jointly owned

Microsoft and other partners attend.

- Architectural reviews guide core decisions for the platform.
- Monthly executive reviews.
- Scorecard review with progress, risks, blockers, and feedback.
- Forum for demos.

## Testing

User acceptance testing is a large part of delivering your Copilot Studio agent. Testing ensures the integration of the complete solution with the back-end, client API, and agent responses to customer input.

## Production deployment and readiness checklist

Being assured that you're ready to deploy is a critical step to a successful launch.

Here are some considerations:

1. Have all scenarios been tested to validate business requirements?
2. Do you have a cutover plan that identifies all the activities required to successfully go live?
3. Has this cutover plan been tested and validated?
4. Do you have access to all the production resources you need?

Resources such as:

- Environments

- DLP policies
- Gateways (optional)
- Azure resources (optional)
- External APIs (optional)

5. Do you have a communication plan to inform both internal and external stakeholders of the changes? This plan includes both pre- and post- deployment.

## Power Platform administration

Many customers have Power Platform, Microsoft 365 administration, or a Center of Excellence (CoE) team that's accountable to the management of a technology set.

This team can focus on activities such as:

- Environment setup.
- Security/Access.
- DLP Policies.
- Gateways.
- Custom analytics.

For more information, see [custom analytics strategy](#).

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Design your Copilot Studio production environment strategy

Article • 11/22/2024

For customers building and deploying agents at scale, they need guidance on best practices and key considerations to design their production environment strategy in alignment with their operational goals and technical needs. A crucial decision involves choosing between a shared or dedicated production environment for their agents. This article outlines the key factors that customers should consider when developing their agent production environment strategy.

## Choose between a shared or dedicated production environment

A shared production environment is where multiple agents coexist within the same production environment, while a dedicated production environment hosts a single agent. Generally, customers are advised to set up a separate production environment for each agent deployment. However, as customers expand their use of agents, they can find it burdensome to manage multiple production environments and prefer a single shared production setup that can handle several agents since agents are offered as software as a service (SaaS).

## Key considerations for production environments

Designing an effective production environment strategy for Copilot Studio involves careful consideration of your operational needs, performance requirements, and security concerns. By weighing the pros and cons of shared versus dedicated environments, and monitoring key indicators, you can make informed decisions that support the success and reliability of your agent applications.

The following key factors help you decide between a shared or dedicated production environment for your agent deployments.

- Business criticality and impact

It's highly advisable for business-critical agents with significant value and impact to run in dedicated production environments. It helps prevent disruptions from

possible changes to shared resources, such as data sources, connection references, data loss prevention (DLP) policies, and message allocation.

- Data security and compliance requirements

Customers need to consider their data loss prevention, data privacy, and geographic residency requirements when they evaluate deploying agents into shared or dedicated production environments. For example, depending on industry regulations, certain agents might handle highly sensitive data which might require dedicated environments to comply with data residency or security standards. Dedicated environments help maintain data privacy.

- Governance and user access control

Usually, the best practice is to restrict production access to service principals or managed accounts. Depending on the structure of support teams and business requirements, various IT and user groups might need to access specific applications or agents in production environments. So, consideration of dedicated versus shared environments for granular permission management becomes necessary.

- Solution architecture

The dependency of your agent on cross-platform solutions significantly affects your production environment strategy. For instance, if your solution includes Canvas apps utilizing a custom agent that invokes several Power Automate flows, you must ensure all solution components are hosted within the same environment. Adding several agents with numerous dependencies into shared production environments can make environment management a daunting job.

- Application life cycle management

Considering your organization's existing Dev Ops process is crucial. For instance, utilizing a fully automated deployment system with tools like Azure Pipelines can significantly streamline the management and support of multiple production environments, whereas relying on a manual process can present more challenges.

- Resource allocation and storage capacity

When managing agents in shared production environments, it's essential to consider storage requirements and usage patterns. For example, spikes causing increased message consumption, or large conversation transcript data that might affect Dataverse capacity and utilization. Assigning agents to dedicated environments facilitates the allocation of individual resources (such as agent

messages or Dataverse storage) for each agent separately. This separation allows more detailed management of agent message capacity assignment at the environment level. Therefore, your production environment strategy should take the expected traffic load, message consumption, and data storage needs of your agents into consideration.

- Reporting and monitoring

The following factors need to be considered during the planning of the production environment strategy:

- An organization's reporting and monitoring requirements
- The configuration of the system's monitoring and alerts

- Service protection limits

Critically, your organization must consider the service protection API limits for each service utilized by their agent (such as Dataverse calls, Power Automate, customer connectors, AI Builder, and shared connections). Consider whether these limits are assessed at per user or per environment levels. For more information about Power Platform API requests limits, see [Requests limits and allocations](#).

## Related content

- [Recommendations for performance planning](#)
- [Requests limits and allocations](#)
- [Quotas, limits, app registration, certificates, and configuration values](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Introduction to conversational experiences

Article • 11/19/2024

Conversational user experience (CUX) is a mode of interaction that's based on natural language. When humans interact with each other, they use conversation to communicate ideas, concepts, data, and other mutually relevant information. In a CUX, your customers interact with their devices, apps, and digital services the way they do with people, through voice, text, or chat that comes naturally.

Other types of interactions can force users to behave in ways that are meaningful to the system: to provide the expected syntax to a command line, for instance, or understand the information architecture of a graphical user interface. CUX turns the tables. Instead of users having to learn the system's "language," the system understands users' natural language—with its patterns of speech, colloquialisms, chit-chat, and even misspellings—so that it can respond appropriately.

## Conversation is personal

There's more to developing a conversational experience than crafting dialogue. The question of what constitutes an appropriate response—and the choice of who gets to decide what's appropriate—are ethical concerns and fundamental to the agent design process. More than other modes of interaction, conversation is especially personal, ingrained with human qualities of emotion, agency, and personality. Conversational experiences that don't address these human traits risk being unsatisfying at best.

Take personality, for example. When a device, app, or service responds to user input, the user begins to assign a personality to it, even when the interaction isn't conversational. This phenomenon of assigning a personality is more pronounced when the user is communicating in their natural language, whether in text or speech. Our latest research shows that users become more attached to devices when they interact using natural language. It's therefore in the best interest of the CUX designer to consider the social and emotional implications of establishing a personal relationship between the user and the device, app, or service. CUX designers have a responsibility to honor the emotional state of the human on the other end of the conversation.

## The power of being understood

When a CUX responds intelligently to natural language, it's not only signaling that it received a message or command, it's communicating that it comprehends the user's unique form of human expression. The user feels heard. Moreover, if the response includes emotional nuance that's relevant to the user's cues, the user not only feels heard, they feel understood. In demonstrating intelligence and empathy, CUX lays the groundwork for building a relationship of trust with every person who uses the system.

CUX can be multi-modal and employ text or voice, with or without visual, auditory, and touch-enabled components. Fundamentally, though, CUX is human language. It lets you harness the power of words to build trust and forge deep connections with your users.

## Related information

- [Principles of conversational experience design](#)
  - [Quickstart: Create and deploy an agent](#)
  - [Create and edit topics](#)
- 

## Feedback

Was this page helpful?



[Provide product feedback ↗](#)

# Principles of conversational experience design

Article • 09/11/2024

Conversational user experiences (CUX) let users interact with technology using natural language. CUX can be voice-based, text-based, or a combination of both. For some situations, CUX can be the most efficient, accessible, intuitive, and empathetic way to accomplish a task. It also offers a powerful way to help build trust with your users. In this article, you'll learn about the principles of great CUX and how they can make your product better.

## Efficiency

Often, the fastest way to get something done is with natural language. Take setting an alarm. With a traditional clock app, users need to find and open the app, select a time, choose AM or PM, and save the new alarm. With CUX, they can just say, "Set an alarm for 8 AM."

Other tasks that involve commands or questions, like adjusting volume, turning on Bluetooth, setting reminders, or looking up a weather or traffic report, can also be simple hands-free experiences with CUX. Users don't need to search through menus or guess the right words to use.

When you design CUX for digital assistants, think about your user's most common questions and offer simple answers that help them achieve their goals quickly and efficiently.

## Accessibility

CUX can make technology more accessible for everyone. A disability is a mismatch between a person and their environment. It can be situational, temporary, or permanent. Voice-driven CUX can match people's abilities with their environments when visual or tactile modalities aren't an option. For example, someone who is driving or cooking might use voice commands instead of touch screens. Someone who has limited mobility might use voice input instead of keyboards. Visual media in the form of images and videos can also enhance CUX by providing alternative ways to present information or feedback.

# Intuitiveness

CUX can help users discover how to do something without knowing the exact steps beforehand. A well-designed CUX uses natural language processing to capture the user's intent and dialog design to guide the user through the task. For example, someone who wants to connect their new wireless headset but isn't familiar with Bluetooth can just say "Connect my headset" and the CUX walks them through the process. Another example is troubleshooting a problem. "I can't print" can lead to a conversation that helps users solve their problem without having to understand the technical details.

# Empathy

CUX can create an emotional connection with users by acknowledging and responding to their feelings. Conversations are inherently emotional, even when users know they're interacting with something artificial, like a copilot. In scenarios that deal with emotional situations, CUX can be a powerful way to honor people's emotional reality. For example, when a user feels frustrated or happy, CUX can use kind and careful language or positive reinforcement to support them. A thoughtfully designed CUX also considers linguistic and cultural context and creates an opportunity for users to feel respected as well as supported.

# Trust

CUX can build trust with users by having a consistent and appropriate personality. Even if your CUX has no identity, name, or avatar, it still communicates through language. Language is inherently human, so it always conveys a persona. The quality of the persona's dialogue tells your user what to expect from your product or company.

An intentionally crafted persona communicates consistency, trustworthiness, and care. Even if your CUX is a sarcastic, belligerent crank with a cutting wit, as long as it's consistent with its personality, your users know how to communicate effectively and achieve their goals. Personality makes it easier to teach your user how to speak to your CUX and makes the experience more enjoyable.

Chit-chat is another way to show personality and build rapport with users. Chit-chat is casual conversation that doesn't necessarily relate to the task at hand, but rather allows users to get to know the personality of your CUX. A few well-chosen chit-chat scenarios can help establish loyalty and smooth over some of the inevitable errors in other conversational exchanges. Thoughtfully crafting the personality of your CUX gives you some control over how users respond to your experience and interact with your brand.

# Related information

- [Types of conversational user experiences](#)
- [Quickstart: Create and deploy a copilot](#)
- [Create and edit topics](#)

 **Note:** The author created this article with assistance from AI. [Learn more](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback](#) 

# Types of conversational user experiences

Article • 09/11/2024

Conversational user experiences (CUX) can be voice-based, text-based, or start in one type of conversational user interface and continue in another. There are different types of CUX that have different capabilities and functions. The terms for these kinds of experiences are often used interchangeably, but they differ in important respects. In this article, you'll learn about some common examples of CUX and how they differ from each other.

## Copilots

Copilots are typically text-based CUX that help users find information or complete tasks on a website. Users type a question or request, and the copilot responds. The copilot might answer with natural language, show something visual like a map or a product image, or offer an action for the user to take, like adding an item to a cart or booking an appointment.

Copilots make the search experience more intelligent by helping users refine their search or disambiguate their query. Copilot conversations are either single-turn or multi-turn. Single-turn conversations are question and answer pairs that don't keep context from previous exchanges. Multi-turn conversations are more like human conversations that build on or follow from previous exchanges.

## Virtual agents

Virtual agents are similar to copilots, but they have a more limited domain of specific functions or skills. They help users complete tasks like booking a flight, placing an order, or troubleshooting a technical problem. They might provide a conversational way to do things like fill out a form or look up information.

Virtual agents can be voice-based or text-based, depending on the platform and the task. They might have a name or an avatar to create a sense of personality.

## Virtual assistants

Virtual assistants, or personal digital assistants, are usually branded personas, like Siri, Alexa, and Cortana, that can do many different tasks on behalf of the user. They also

have an extensible platform that allows third-party companies to add new skills to them. For example, a user can ask a virtual assistant to order a ride, check the weather, or play music from different services.

Virtual assistants are usually voice-activated, but they can converse in text as well. Because of their voice and their branded name, they have a deliberately crafted personality to create an emotional connection with the user.

## Related information

- Design ethical and empathetic conversational experiences
- Quickstart: Create and deploy a copilot
- Create and edit topics

 **Note:** The author created this article with assistance from AI. [Learn more](#)

---

## Feedback

Was this page helpful?



Yes



No

[Provide product feedback ↗](#)

# Design ethical and empathetic conversational experiences

Article • 09/11/2024

Conversational design is a new and exciting field that uses AI and machine learning to create engaging, natural-language interactions between humans and machines. As conversational designers, we have the opportunity to shape the future of this technology and the way people use it. But we also have a responsibility to design with care and respect for our users, our values, and our society.

Ethics and empathy are essential for creating conversational user experiences (CUX) that people trust, enjoy, and benefit from. Ethics help us define our principles and goals, while empathy helps us understand our users' emotions and needs. Together, they guide us to make design choices that are responsible, inclusive, transparent, and accessible.

In this article, you'll learn how to design ethical and empathetic conversational experiences with Copilot Studio. You'll discover some best practices for building your team, crafting your principles, designing with empathy, revising your work, and handling voice-only interactions.

## Build a diverse and creative team

Designing a great conversational experience requires more than technical skills. It also requires insight into what your users want or need to accomplish with your CUX. Whether they want to complete a task, get unstuck, have a fun conversation, or learn something new, you need to design a conversation that meets their expectations and preferences.

To do that effectively, you need a team that can bring different perspectives, backgrounds, and expertise to the table. A diverse team can help you avoid biases, uncertainties, and assumptions that might limit or harm your design. A creative team can help you explore new possibilities, write engaging dialogues, and craft a unique voice for your CUX.

If you have the opportunity to build a team for your project, consider inviting people who have various lived experiences or humanities backgrounds. For example, you might include novelists, musicians, playwrights, filmmakers, poets, or other people who know how to tell stories and create characters. They can help you design a conversational experience that feels natural, authentic, and welcoming.

# Define your principles and values

What are the core values that you want to reflect in your conversational experience? What are the ethical standards that you want to uphold? How do you want your users to feel when they interact with your CUX? These are some of the questions that you should ask yourself and your team before you start designing.

Defining your principles and values can help you clarify your vision and goals for your project. It can also help you identify the potential risks and challenges that you might face along the way. For example, how will you handle sensitive or controversial topics? How will you protect your users' privacy and security? How will you deal with abusive or inappropriate inputs?

Having clear principles and values can help you make consistent and confident design decisions. It can also help you communicate your intentions and expectations to your users, stakeholders, and partners. You can use your principles and values as a reference point throughout your design process, and as a way to evaluate your work and measure your impact.

Here are some examples of principles and values that you might consider for your conversational experience:

- Transparency: Be honest and clear about who or what your CUX is, what it can do, and how it works. Make sure your users know how to talk to a person if they want to.
- Privacy and security: Respect your users' personal data and protect it from unauthorized access or misuse.
- Boundaries: Set clear limits on responding to abusive, violent, sexually explicit, or otherwise problematic requests.
- Inclusivity and sensitivity: Design for a diverse range of users, abilities, cultures, languages, and preferences. Avoid stereotypes, discrimination, and cultural appropriation.
- Accessibility: Make sure your CUX is easy to use for everyone, regardless of their physical or cognitive abilities.
- Empathy: Understand your users' emotions and needs, and respond with kindness, compassion, and support.
- Clarity: Use simple, concise, and accurate language that your users can understand and follow.

- Personality: Give your CUX a distinctive voice, tone, and style that matches its purpose and audience.

These aren't the only principles and values that you can choose for your project. You might have different or additional ones that suit your specific context and goals. The important thing is to be intentional and deliberate about what matters to you and why.

## Design with empathy

Empathy is the ability to put yourself in someone else's shoes and feel what they feel. It's a key skill for conversational designers because it helps us create experiences that resonate with our users on an emotional level. When we design with empathy, we don't just focus on the functionality or usability of our CUX. We also consider how it makes our users feel, what motivates them, what frustrates them, what delights them.

To design with empathy, we need to learn as much as we can about our users. We need to research their needs, goals, preferences, behaviors, pain points, and emotions. We need to observe how they interact with our CUX or similar products. We need to listen to their feedback and suggestions. We need to test our assumptions and validate our solutions.

But learning about our users isn't enough. We also need to act on what we learn. We need to use our insights to inform our design choices and improve our CUX. We need to create conversations that are relevant, helpful, respectful, and engaging. We need to anticipate our users' needs and expectations, and provide them with appropriate responses and guidance. We need to avoid or minimize errors, confusion, frustration, or disappointment.

Designing with empathy isn't only good for our users. It's also good for our brand and business. When we design with empathy, we build trust and loyalty with our users. We create positive and memorable experiences that make them want to come back and recommend us to others. We differentiate ourselves from the competition and stand out in the market.

## Revise your work

Designing a conversational experience isn't a one-time event. It's an ongoing process that requires constant revision and improvement. As you design, test, launch, and monitor your CUX, you'll discover new insights, challenges, and opportunities that will help you refine your work and make it better.

To revise your work effectively, you need to collect and analyze data from various sources. You need to use analytics tools to track how your CUX performs, how your users behave, and what outcomes you achieve. You need to use feedback mechanisms to gather opinions, ratings, reviews, and suggestions from your users. You need to use testing methods to evaluate the usability, functionality, and quality of your CUX.

But collecting data isn't enough. You also need to act on it. You need to use your data to identify the strengths and weaknesses of your CUX, the gaps and opportunities in your design, and the trends and patterns in your user behavior. You need to use your data to make informed decisions that enhance your CUX and increase your user satisfaction.

## Handle voice-only interactions

Voice-only interactions are conversational experiences that rely on voice as the primary input. They might not have a graphical user interface (GUI) or have a limited one. Examples include voice-activated navigation systems in cars, automated voice systems for customer service or support over the phone, and Bluetooth devices that offer voice-activated features without a virtual assistant.

Voice-only interactions pose some unique challenges for conversational design because they depend entirely on speech recognition and synthesis technologies. These technologies aren't perfect and can introduce errors or misunderstandings in the communication between the user and the CUX. To design effective voice-only interactions, you need to consider some specific factors:

- Variation: Users can say the same thing in many different ways. For example, "yes," "yup," "yeah," and "uh huh" all mean an affirmative confirmation. Your CUX needs to be able to recognize all these variations as equivalent inputs.
- Volume: Users can speak at different volumes depending on their environment, mood, or preference. Your CUX needs to be able to adjust its volume accordingly and ask the user to repeat or speak louder if necessary.
- Clarity: Users can speak with different levels of clarity depending on their accent, pronunciation, or speech impediments. Your CUX needs to be able to handle different accents and dialects, and ask the user to clarify or spell something if needed.
- Breaks: Users can pause, hesitate, or interrupt themselves while speaking. Your CUX needs to be able to detect these breaks and respond appropriately. For example, it can wait for the user to resume, ask a follow-up question, or offer some suggestions.

- Pronunciation: Users can pronounce words differently depending on their language, region, or culture. Your CUX needs to be able to understand different pronunciations and use the correct ones for your target audience.
- Inflection: Users can convey different meanings or emotions with their voice tone, pitch, or stress. Your CUX needs to be able to interpret these cues and use them in its own speech. For example, it can use a rising intonation to ask a question, a falling intonation to make a statement, or a higher pitch to express excitement.

In addition to these factors, you also need to consider the audio characteristics of your CUX's voice. The voice you choose for your CUX can have a significant impact on how users perceive and interact with it. You need to select a voice that matches the purpose, personality, and audience of your CUX. You also need to consider the implications of your voice choice in terms of gender, culture, identity, and bias.

## Related information

- [Research and prioritize scenarios](#)
- [Configure handoff to Dynamics 365 Customer Service](#)
- [Create and edit topics](#)

ⓘ **Note:** The author created this article with assistance from AI. [Learn more](#)

---

## Feedback

Was this page helpful?



[Provide product feedback ↗](#)

# Research and prioritize scenarios

Article • 09/11/2024

Before you create content for your conversational user experience (CUX), you need to understand what your customers want and need from your conversational experience. To do that, you can use data analysis and brainstorming techniques to identify and rank the most relevant and impactful scenarios for your CUX.

## Analyze existing data sources

Start by reviewing your data and knowledge about how your customers interact with your brand. This might include website visits, blog traffic, social media engagement, search data, support calls, or any other source that's relevant to your users. Look for areas of high volume, high user engagement, or any other information that might relate to the purpose of the conversational experience. You want to find out all the potential topics that your CUX could handle.

## Brainstorm possible scenarios

After analyzing the data, brainstorm scenarios that you could cover with your CUX. Think carefully about your current user experiences, what the data tells you about those experiences, and topics that could help answer your most common questions. You might start by answering these questions:

- Who is the target audience for the conversational experience?
- What are your business goals?
- What are your biggest challenges?
- What are your users' biggest pain points?
- What are the priorities for your business?

At this stage, don't worry about narrowing down or prioritizing your scenarios, or about current resources and timeline. Make a list of all the topics you might want to cover with your CUX at some point. It can help at this point to dig into these scenarios and think through how users would interact with the CUX in each case.

## Rank scenarios by impact

When you've finished brainstorming, the next step is to rank and choose a few scenarios to build in your conversational experience first. The key to this step is to start small in

terms of number of scenarios, but significant in terms of impact.

Don't discard your brainstorming once you're done! Those ideas are the building blocks for future content development, and they act as a basis for your innovation roadmap.

When you're ranking your initial scenarios, consider the data that you analyzed and your answers to the questions about business goals, challenges, and priorities.

## Test, monitor, and improve

The process of scenario development should never be stagnant. Even after you've started creating content for your CUX, continue to monitor usage and analyze data to make sure that it addresses the most important, relevant, or impactful user scenarios. Update your content plan over time as needs change.

## Examples of scenario development

Here are two examples of how these steps can be applied to develop scenarios for conversational experiences. In each example, available data is analyzed, possible scenarios are brainstormed, and then a list of ranked scenarios is created based on balancing the business goals with the scenarios that have the biggest impact on users.

### Example 1: Customer support copilot for a mobile phone retailer

**Available data:**

- Questions asked on social media channels
- Traffic data from the frequently asked questions (FAQ) page on the website
- Call volume to customer support

**Brainstormed scenarios:** Questions about mobile phone features, pricing questions, troubleshooting mobile data issues, troubleshooting automatic factory reset error, ordering replacement parts, repair services, wireless carrier compatibility, billing issues, new phone release information, shipping information, service outages, warranty information.

**Prioritized scenarios:**

1. Troubleshooting mobile data issues
2. Troubleshooting automatic factory reset error
3. Repair services

**Why these scenarios were prioritized:**

- The first and second scenarios generate the most customer support call volume.
- The third scenario has the highest traffic on the FAQ website.

## Example 2: Personal shopping assistant for clothing brand

**Available data:**

- Past sales data
- Current industry trends
- Traffic data from the FAQ page on the website

**Brainstormed scenarios:** Clothing recommendations, accessory suggestions, completing purchases, returns and exchanges, size and fit information, current discounts and sales shipping information, what's new, store locations, store hours, inventory information

**Prioritized scenarios:**

1. Clothing recommendations
2. Size and fit information
3. Returns and exchanges

**Why these scenarios were prioritized:**

- The first scenario is the main reason for creating the shopping assistant.
- The second and third scenarios have the highest traffic on the FAQ website.

## Next steps

- Identify customer intents
- Key concepts - Analytics in Copilot Studio
- Analyze your copilot's customer satisfaction metrics

ⓘ Note: The author created this article with assistance from AI. [Learn more](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Identify customer intents

Article • 09/11/2024

You can think of user intent as the reason a person is interacting with your conversational experience, or conversational user experience (CUX). Your CUX must identify what the user wants so that it can match them with the most accurate content to help them complete their goal.

## Intents vs requests

It can be easy to confuse *intents* with *requests*. Intents define the goal of the interaction, while requests are the context-setting words, phrases, or questions that users say or type to begin a conversation. It's important to note that you can't rely on a single request to accurately describe an intent. You might discover that requests that appear similar are different intents that require separate content, or you might find that users describe the same intent in different ways.

## Where to start with user intents

Review your key scenarios and any data that shows how users are interacting with your conversational experience or brand. Start with one scenario and think critically about what your user's purpose might be in that scenario. Keep in mind that a single scenario might have multiple intents.

Here's an example of developing user intents:

**Conversational experience:** Personal shopping assistant for clothing brand

**Scenario:** Returns and exchanges

**User intents:**

- Return an item
- Return a defective or damaged item
- Return an item after 90 days
- Exchange an item for a different size
- Exchange an item for a different product

Don't worry if you feel that you aren't capturing all customer intents. In fact, it's better to start with a smaller set of intents and make sure that they can be traced back to the goals and priorities that you defined at the beginning of your CUX design process. Just

as you did when you defined your scenarios, keep the scope small and focus your efforts on the most important and high-impact areas. Learn more in [Research and prioritize scenarios](#).

## Monitor and iterate your intents over time

Identifying user intents isn't a one-time activity. Like any content that you create for a conversational experience, you need to monitor the success of the content. Let the data that you collect guide how you refine intents, and make sure to map intents back to real user needs with matching requests.

## Related information

- [Disambiguate customer intents](#)
- [Topic overlap detection in AI features for Teams and Classic copilots](#)
- [Topic suggestion from chat transcripts in AI features for Teams and Classic copilots](#)

---

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Disambiguate customer intents

Article • 09/11/2024

Disambiguation is the process of narrowing down a user intent by asking clarifying questions in your conversational experiences. With a clearer understanding of a user's true intent, you can better direct them to the information they're looking for.

Disambiguation is helpful when you have multiple intents that are similar or when the user request is unclear. Whether disambiguation is necessary depends on the type of scenarios you support, how you identify and route different intents, and the content you have available for users.

For example, the request "How do I upgrade my computer?" might seem straightforward, but is it?

- Does the user want to upgrade their operating system or the computer hardware that it runs on?
- Does the user want to upgrade their desktop PC or tablet, if the process is different for each?
- Does the user want to upgrade their operating system to the newest version or update their current operating system with security patches?

## Disambiguation approaches

You can approach disambiguation in a few ways, such as by asking more questions, asking targeted questions, and providing options.

### Ask more questions

Ask questions that help you narrow down the user intent. Based on the preceding example, the conversation might look like this:

The screenshot shows a conversational interface with two participants: a user and a bot. The user's messages are in blue boxes, and the bot's messages are in white boxes with blue borders.

- User message: "How do I upgrade my computer?" (08:36)
- Bot message: "Do you want to upgrade your computer's hardware or the operating system?" (08:36)
- User message: "My computer hardware" (08:36)
- Bot message: "Is your computer a desktop PC or a laptop?" (08:36)
- Bot message: "Laptop" (08:36)

This method of disambiguation leads to more conversational turns before the user's intent is identified. While clarity is important, balance the number of questions you ask with how helpful they are. If disambiguation unnecessarily extends an interaction or makes the conversation feel more like an interrogation, it might lead to user frustration.

## Ask targeted questions

Another approach is to ask targeted questions that build on the initial request to get the user to the best possible content faster.

The screenshot shows a conversational interface with two participants: a user and a bot. The user's messages are in blue boxes, and the bot's messages are in white boxes with blue borders.

- User message: "How do I upgrade my computer?" (08:36)
- Bot message: "I understand you want to upgrade your computer. To help, I'll need a little more info." (08:36)
- Bot message: "Can you tell me what operating system your computer is running right now and which version you want to upgrade to?" (08:36)

Similar to the other approaches, the goal is to try to understand the user's intent as quickly as possible, without accidentally asking them irrelevant questions. However, open-ended questions can make the interaction more complex. If the user includes too much detail in their response, the conversational user experience (CUX) might become confused or trigger the wrong flow.

## Provide options

Provide suggestions at the start of the interaction to help the CUX disambiguate more quickly and reduce the need to ask multiple follow-up questions.

The screenshot shows a conversation between a user and a bot. The user asks "How do I upgrade my computer?". The bot responds with "Here are a few ways I know to help with your computer. What can I help you with?" Below this response are four blue rectangular buttons labeled "Hardware", "Operating System", "Latest Security Patches", and "Other Issue".

This method is up-front about what kind of information is available to the user. However, if none of the options address the user intent, the conversation must provide a way for the user to say so and then be redirected to the correct information. Examples include offering options like "None of these" or "I have a different question" and then following up with a request to provide more context or details.

The screenshot shows a conversation between a user and a bot. The user asks "How do I upgrade my computer?". The bot responds with "Here are a few ways I know to help with your computer. What can I help you with?". The user then selects the "Other Issue" option. The bot follows up with "OK, can you provide a little more detail about the issue you're experiencing?".

## Preparing for out-of-scope requests

You might find some cases where disambiguation is impossible because the user intent falls outside the scope of your conversational experience and you don't have the content to support the scenario. It's fine not to have an answer for every question a user might ask. However, try to identify the most likely out-of-scope requests and prepare answers that make it clear that you understand the intent and help users find the assistance they need, if possible. Planning ahead prevents users from reaching a dead

end, and can also provide valuable data if you see an increase in traffic to areas that you originally planned to be out of scope. Learn more in [Design graceful fallbacks and handoffs](#).

## Related information

- [Choose effective trigger phrases](#)
  - [Configure the system fallback topic](#)
  - [Use entities and slot filling in copilots](#)
- 

## Feedback

Was this page helpful?

 Yes

 No

[Provide product feedback ↗](#)

# Choose effective trigger phrases

Article • 09/11/2024

In Copilot Studio, the context-setting words that a user provides to begin a conversation are referred to as *trigger phrases*. You might be familiar with such alternative terms as *queries*, *questions*, *requests*, and *prompts*. They're often presented at the beginning of a conversational experience, but they might appear anytime a user's intent changes. Trigger phrases can be one word, a group of words, a sentence or question, or even a paragraph.

Your conversational user experience (CUX) must correctly identify and understand the user's trigger phrase to match it to an intent and present the correct response. The more effective your trigger phrases are, the more likely it is that your conversational experience will provide the right information or assistance to the user. In this article, you'll learn tips for creating effective trigger phrases that help users get the information they're looking for.

## Create unique trigger phrases

As you build a conversational experience, develop a library of trigger phrases that are unique to each of your identified user intents. Such a library allows you to capture the correct conversation when users present any form of the trigger phrases that pertain to a specific intent. In most cases, start with three to five unique trigger phrases per intent. Adjust the library over time as users interact with your CUX.

## Brainstorm trigger phrases

When creating trigger phrases, think about the different ways that users might express their intent. Here are some tips to help you brainstorm trigger phrases:

- Try saying it out loud. How might a user phrase their question or request if you were speaking with them in person or over the phone? Try roleplaying scenarios to identify natural-sounding trigger phrases.
- Try to think like a user. What words would you use if you were looking for the information or assistance that your CUX provides? What questions would you ask?
- Make sure to identify any common slang, contractions, acronyms, synonyms, or other variations for terms that a user might use.

## Keep track

Monitor your user data and make sure to refine and update your trigger phrases as necessary over time. Remove or rephrase any that are causing confusion.

## Identify user intents first, trigger phrases second

As a rule of thumb, you should identify the user intent before you start to develop trigger phrases. Learn more in [Identify customer intents](#).

Here are a few things to keep in mind as you begin compiling trigger phrases for a user intent:

- Think about the different ways that users might phrase the same intent—that is, build variations on a theme. For example, users who want to know the weather might ask, "What's the weather like today?" or "What's the forecast for today?" or "Is it going to rain today?" or "Do I need an umbrella today?"
- Anticipate similarly worded trigger phrases with different intents. For example, "How do I return my purchase?" could be a request for a refund, an exchange, or a store credit. If you identify multiple intents that are similar or related to one another, they need to be clearly differentiated. Make sure that your natural language model is well trained and can distinguish between different intents. You can also use disambiguation questions to help narrow down the meaning of the user's true intent. Learn more in [Disambiguate customer intents](#).
- Identify critical differentiation words. When you're identifying the key words or phrases for a trigger phrase, think about which words are important to help set context and differentiate between intents. Think of how a single word changes the meaning of these two trigger phrases:
  - I want to book a flight to Shanghai.
  - I want to book a flight from Shanghai.

## Clarify ambiguous trigger phrases

A trigger phrase like "I need help" might not be specific enough to match a predefined intent with certainty. You can gather extra context with disambiguation questions that help guide the user in refining their intent. Learn more in [Disambiguate customer intents](#).

# Examples

Here are two examples of developing potential trigger phrases for two separate user intents in the same scenario: troubleshooting laptop display issues.

## Example 1: Laptop display is only in black and white

**Scenario:** Troubleshooting laptop display issues

**User intent:** Display color troubleshooting

**Potential trigger phrases:**

- Screen color is missing
- Display is in black and white
- Display isn't showing color
- Change screen to show color
- Screen stuck in grayscale
- Color settings for screen

## Example 2: Laptop doesn't connect to external display

**Scenario:** Troubleshooting laptop display issues

**User intent:** Monitor connection troubleshooting

**Potential trigger phrases:**

- Can't extend laptop screen
- Troubleshoot external display issues
- Can't connect laptop to external display
- External display doesn't work with laptop

## Related information

- [Design graceful fallbacks and handoffs](#)
- [Quickstart: Create and deploy a copilot](#)
- [Create and edit topics](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback 

# Design graceful fallbacks and handoffs

Article • 09/11/2024

For situations where your conversational user experience (CUX) can't determine or fulfill your user's intent, you need to develop a series of fallback responses. "Series" is intentional here. Don't think that an isolated "Oops, sorry about that" will cover you. You don't want to break the user's trust—and possibly damage your brand loyalty—by creating an experience that leads to a dead-end. Good fallbacks and handoffs help the user complete the task they set out to accomplish with as little frustration as possible.

## Set clear expectations up front

Conversational experiences aren't appropriate for handling tasks that a human handles better. During your design process, you identified the things your CUX does and the things it doesn't. One way to reduce the need for fallbacks is to be clear right from the start what those things are.

For example, if you're designing a banking copilot, tell customers in the greeting that they can check their balance or make a transfer between accounts. If you're designing a travel copilot, tell customers that they can book a round-trip flight, reserve a hotel room, or make changes to their itinerary.

When you get to the point where a user has asked your copilot to do something that it can't, a fallback response is another place to provide this clarity. Fallbacks like "Sorry, I didn't get that. I can help you [X] or [Y]. Do you want to try one of those things?" help redirect the user toward the things your copilot *can* do.

## Think of fallbacks according to function

Regardless of whether your CUX can't understand or can't deliver on what the user wants, it's helpful to think of your fallbacks according to their function: seeking understanding, disambiguating, and establishing domain expertise.

- Seek understanding: When your CUX can't understand the user's intent, ask the user to rephrase or clarify their request. For example:
  - "I didn't get that. Can you say it another way?"
  - "I don't quite understand. Can you try rephrasing?"
  - "I'm a little unsure how to help. Try asking again using just a few key words."

- Find other ways to disambiguate. Sometimes, a fallback is an appropriate place to seek more clarity that helps you determine what the user wants. Offer a suggestion or two that closely matches the user's intent. For example:
  - "Did you mean [suggestion]?"
  - "Sounds like you want to [suggestion]. Is that right?"
  - "I found [suggestion 1] or [suggestion 2]. Is it one of those?"
- If your CUX understands the intent but can't fulfill it, be transparent with your users. Redirect them toward what the CUX can do or offer other resources that might help. For example:
  - "Sorry, I can't help with that. Did you want to try [suggestion1] or [suggestion 2]?"
  - "Sorry, I don't think I can help you with that. Say "main menu" to learn what I can do."
  - "I don't have any info about that, but I found this topic that might help: [topic]."

Be cautious about using phrases that suggest the CUX learns how to address the user intent, such as "I can't do that yet" or "I'm still learning how to do that," unless you have concrete plans to build that capability into your experience.

## Create fallback variations

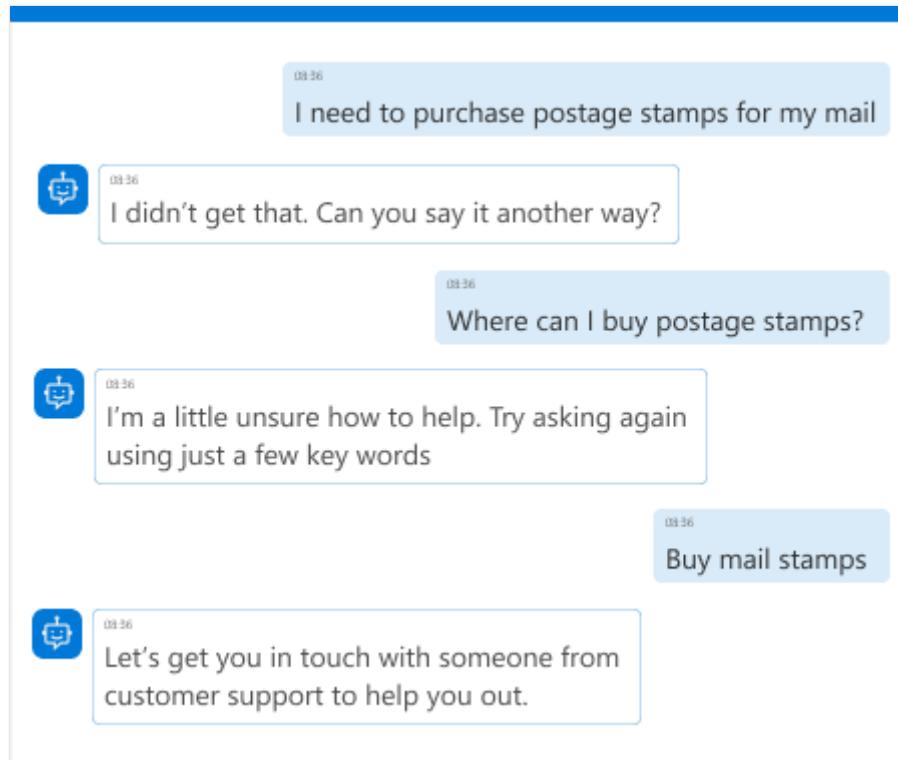
If you were having a conversation with someone and they made several errors in a row, it would be odd for them to offer the exact same apology over and over again. The same goes for your CUX. When you're writing fallbacks, it's a good idea to create a few variations for every situation. That way, when customers encounter a fallback more than once, the experience doesn't feel overly robotic. The number of fallbacks you need depends on how many paths customers can follow in your conversation, but generally, try to write at least three.

## Know when to hand off

It's important to build a handoff process for when your CUX can't understand the user or can't help them. You might direct the customer to a human support agent or to resources such as support websites or online documentation. One of the trickier questions you need to answer is *When should the CUX direct the user to a human or other resource?*

It can be helpful to think about how many times you'd be willing to repeat or rephrase a question during a conversation before you became frustrated. We recommend asking

the user *no more than two fallback questions* in one session before directing them elsewhere.



Make the handoff as smooth as possible. Make sure that the user knows what's happening, whether they're being connected to a human or another resource, and what they need to do next. Even if they got stuck, it doesn't necessarily mean that they need to start over—and they don't want to. A good handoff effectively remembers where the user left off and helps them continue the task they set out to accomplish. Asking the user to repeat the same process the CUX started isn't a great experience, and might result in the user abandoning the effort.

## Ask for feedback

When your CUX concludes the conversation, whether it helped the customer or not, is a great time to ask for feedback. Make the request simple and quick. Here are some easy ways to ask people about their experience:

- Thumbs up/thumbs down
- Smile/frown
- Numeric rating (five-point scales are typical)
- Positive/negative (either a binary scale or a broader five-point scale)

Ideally, include an open text field after the rating so that the customer can say whatever they want. You can add more questions, but the more questions you add, the less likely people are to engage with the feedback form.

As valuable as feedback can be, though, it's equally important to be thoughtful about how often you ask for it. Asking too often is annoying at best and alienating at worst. If possible, try to use frequency signals from your customers so that you're not asking them for a rating more than once a week. Even then, you might want to prioritize experiences where feedback would be most useful, such as new experiences or more complex ones. You might also want to avoid asking for feedback where the user might want to quickly move on to something else, such as after getting a phone number. Make sure that the task they set out to complete is done before you distract them with the task of completing your survey.

However, *don't ask for feedback if you have no way to manage it*. If customer feedback goes into a void, if there's no process for reviewing, labeling, tagging, storing, and reporting it, don't bother asking for it. If customers sense that their feedback isn't being reviewed, they lose trust and likely won't submit feedback in the future.

## Related information

- [Handle errors](#)
  - [Configure the system fallback topic](#)
  - [Configure handoff to Dynamics 365 Customer Service](#)
- 

## Feedback

Was this page helpful?



[Provide product feedback ↗](#)

# Handle errors

Article • 09/11/2024

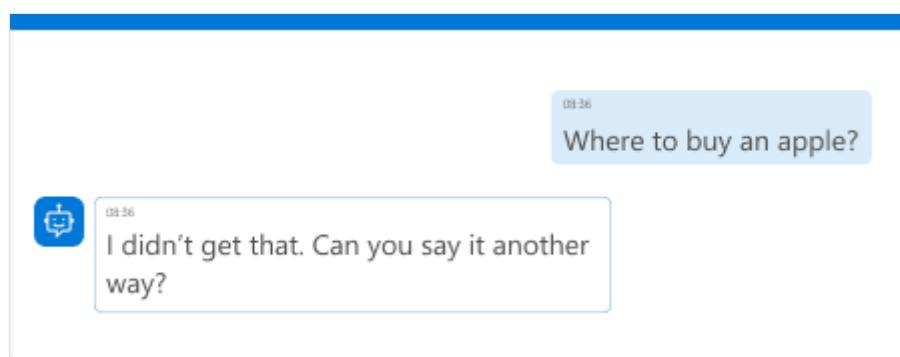
When something goes wrong during a conversational user experience (CUX), you should offer ways to get the conversation back on track. Error handling is a form of fallback. Errors can occur in a user's interactions with the CUX and in the functioning of the CUX itself. Learn more in [Design graceful fallbacks and handoffs](#).

Different types of errors require different messaging. Regardless of the cause, try to be as transparent as possible about what happened and how to fix it. The messaging might include details about the error. It must include information about what the user can do to get unstuck, or at least to understand why they're stuck.

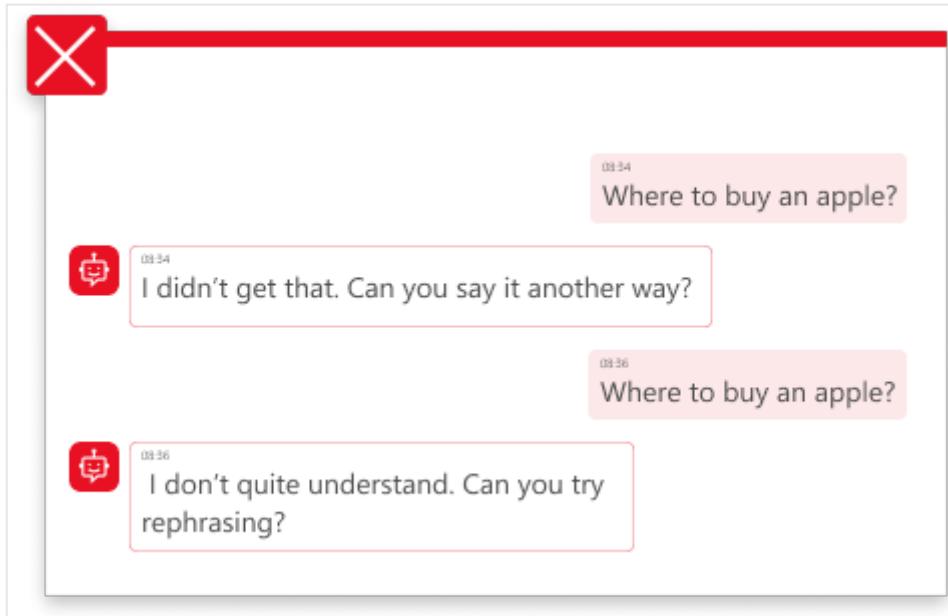
## Misrecognition errors

When your CUX doesn't understand the user's intent, ask for clarification. This step is especially important when the user's input is ambiguous or unclear. The user might not realize that their input was unclear, so it's up to your CUX to ask for clarification. Here are some examples of how to handle misrecognition errors.

Example 1: Ask the customer to rephrase their request.

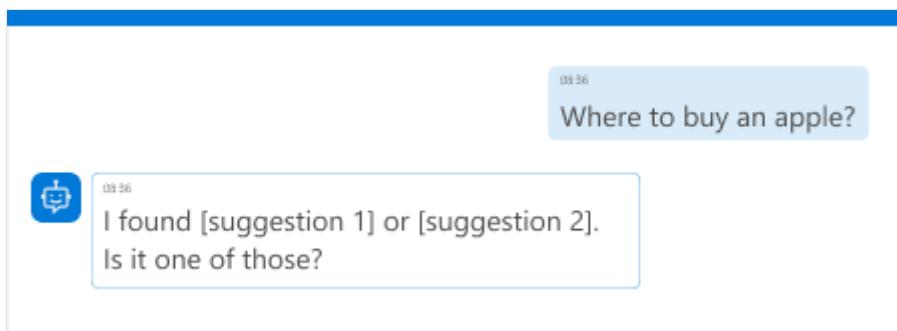


Example 2: Don't ask the user to repeat their request more than twice.



If the misunderstanding happens more than once, choose your responses carefully. If the first response asks the user to repeat or rephrase their request, offer specific options in the second response.

Example 3: Present the customer with specific options.



We don't recommend having the user repeat their request more than twice. If your CUX can't understand the user's intent in two turns, hand off to a human or other mitigation tactic, like email.

## Execution errors

Execution errors result from misrecognition errors. The user asked for something, and your CUX either misunderstood the user's input or matched it incorrectly.

Example 1: Misunderstood input (two items added to list instead of one)

08:36

Add ginger tea to my shopping list



08:36

OK, I added ginger and tea to your shopping list.

Example 2: Incorrect match for user's request (item added to wrong list)

08:36

Add ginger tea to my shopping list



08:36

OK, I added ginger tea to your to do list.

When an execution error happens, provide a way for the user to go back and fix it.

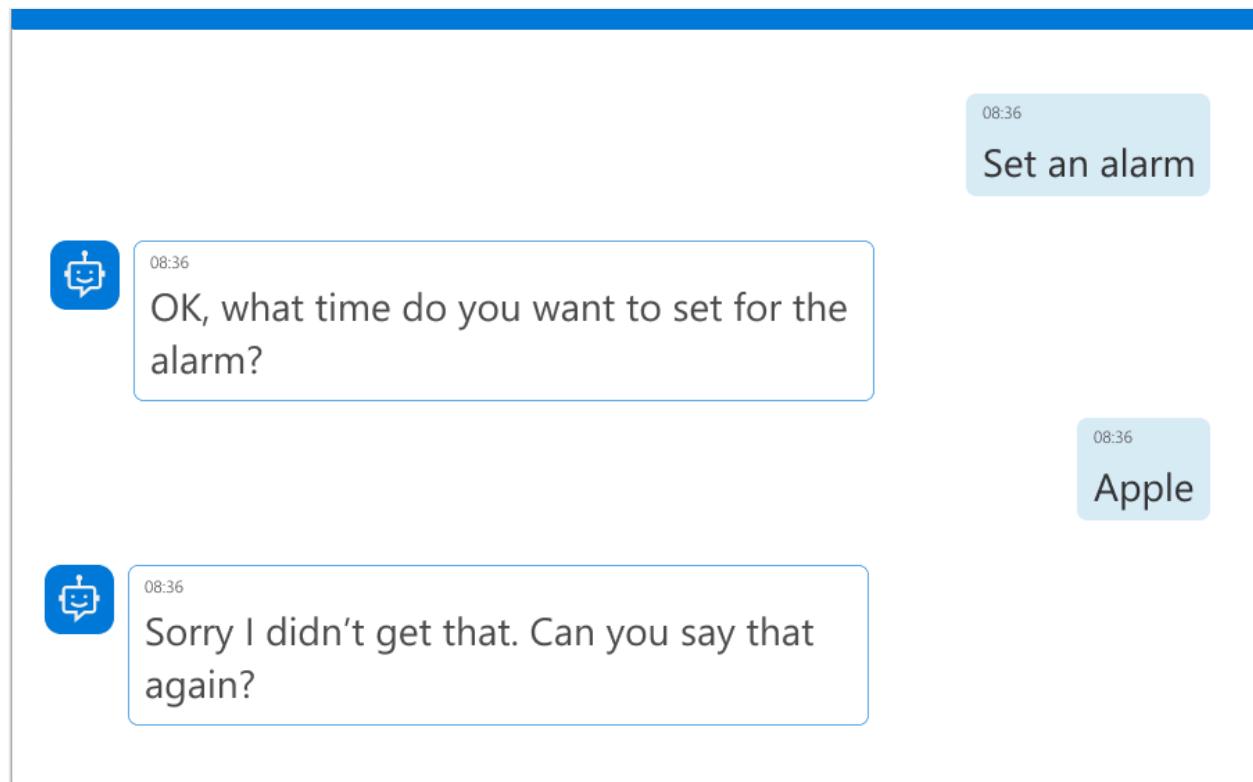
- **Recognize secondary inputs.** Users have an automatic impulse to immediately fix errors, and you can use that to your advantage. For example, a user might enter a phrase like "wrong list," "that's not right," or simply "no." From a technical standpoint, implementing this type of understanding can be challenging, especially if the interaction model is a question-and-answer design instead of a multi-turn conversation. However, taking the time to integrate these types of keyword phrases into your design for error handling can help make the user's experience much more natural.
- **Ask for confirmation before executing an action.** Including a confirmation, such as "OK, I'll add ginger and tea to your grocery list. Is that correct?" gives the user an opportunity to correct the error before it happens. However, having to confirm every action can quickly become annoying, so use this approach with caution.

## Input errors

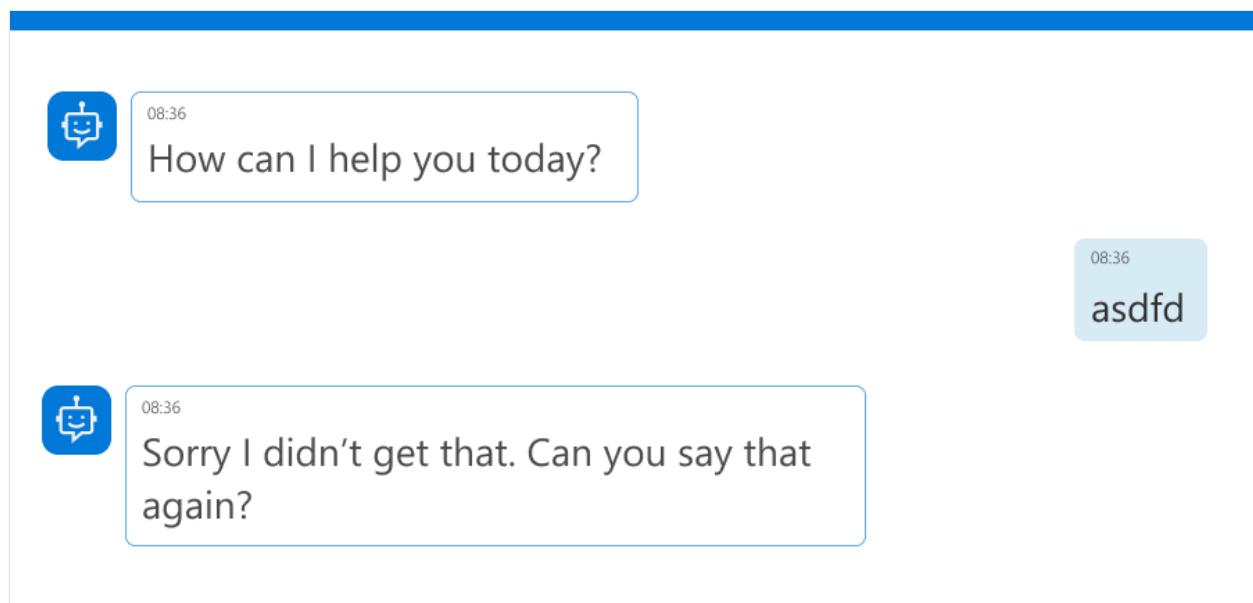
Sometimes your conversational experience isn't to blame for an error. The user might say something that isn't an answer to the question the CUX asked or that it doesn't

understand. In these cases, a simple "Sorry, I didn't get that. Can you say that again?" is acceptable.

#### Example 1: Unexpected user response



#### Example 2: Unintelligible input

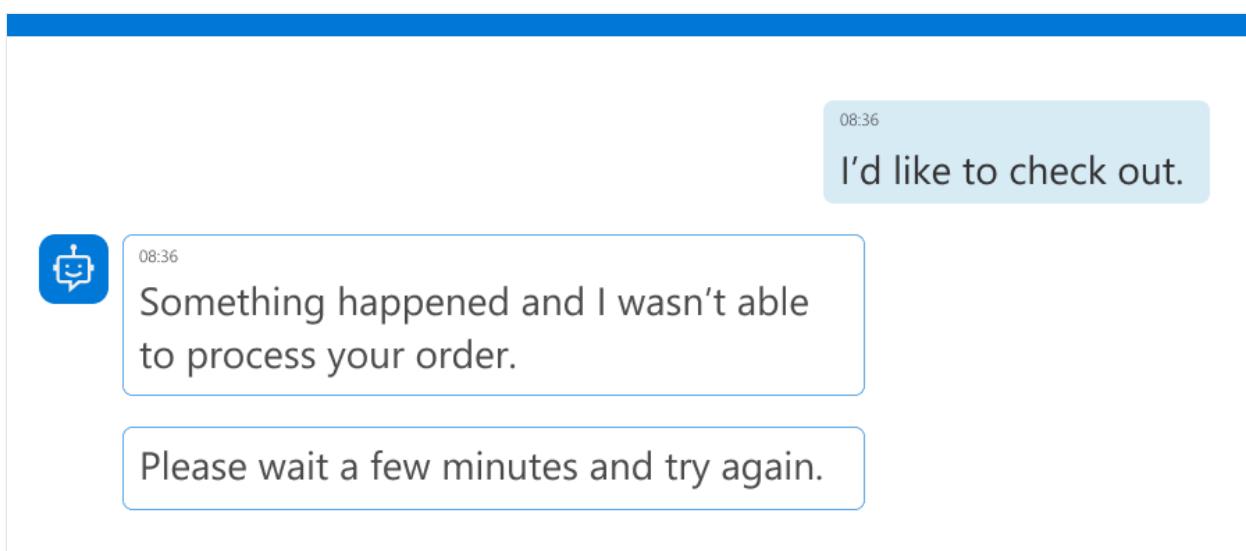
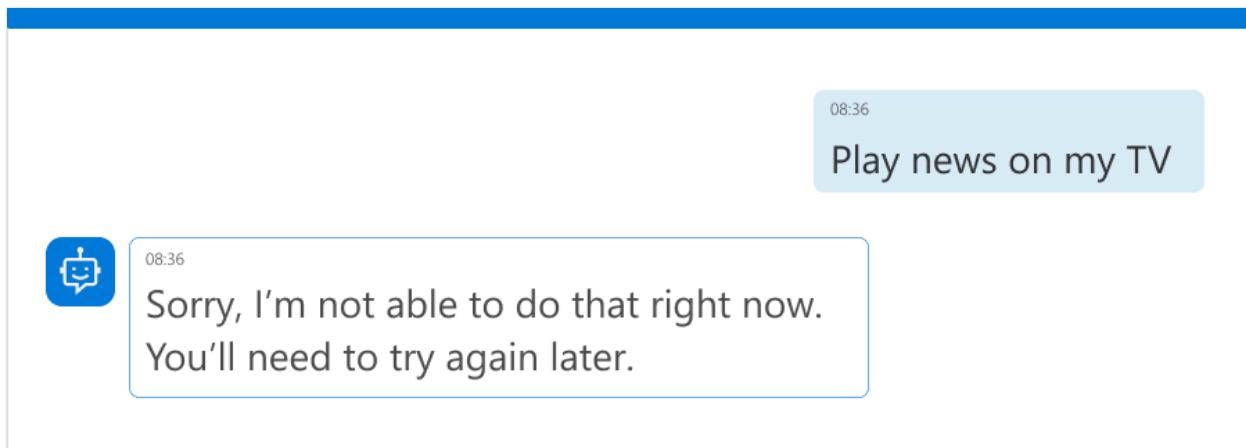


## Technical errors

Other types of errors might cause the copilot not to function correctly. These types of technical errors can include:

- **Errors with a solution.** You're aware that something is wrong with the CUX. You have a workaround or fix for the user to try or you're actively working to fix the error.
- **Errors without a solution.** You're aware that something is wrong with the CUX, but no workaround or fix is available.
- **Unknown errors.** The CUX isn't working correctly and you don't know why.

If you include responses that handle technical errors, make sure to let users know that there's a problem with the experience.



It's helpful to let the user know when they should try again, but only if trying the request again later is likely to succeed. If they try again and get the same response, it can lead to mistrust and frustration. If the error is either unknown or doesn't have a solution, be sure to direct the user somewhere else, like to a customer support agent, FAQ, or email or chat support.

## Related information

- End conversations
  - Understand error codes
  - Test your copilot in Copilot Studio
- 

## Feedback

Was this page helpful?



Provide product feedback ↗

# End conversations

Article • 11/19/2024

How you end a conversational user experience (CUX) is as important as how it begins and progresses. Real conversations have endings, and your CUX should, too. How do you know when a conversation ends?

Think about your own conversations with friends and coworkers. What signs indicate that a conversation is about to end? An awkward silence, an abrupt goodbye, or, if the conversation is in person, a subtle change in facial expression or body language can all signal that the other person is disengaging from the conversation.

Such subtleties of human interactions are lost on conversational experiences. A CUX is reactive. The most accurate way to know if the CUX has reached the end of a conversation is to ask the user if they're done.

## Ask a follow-up question

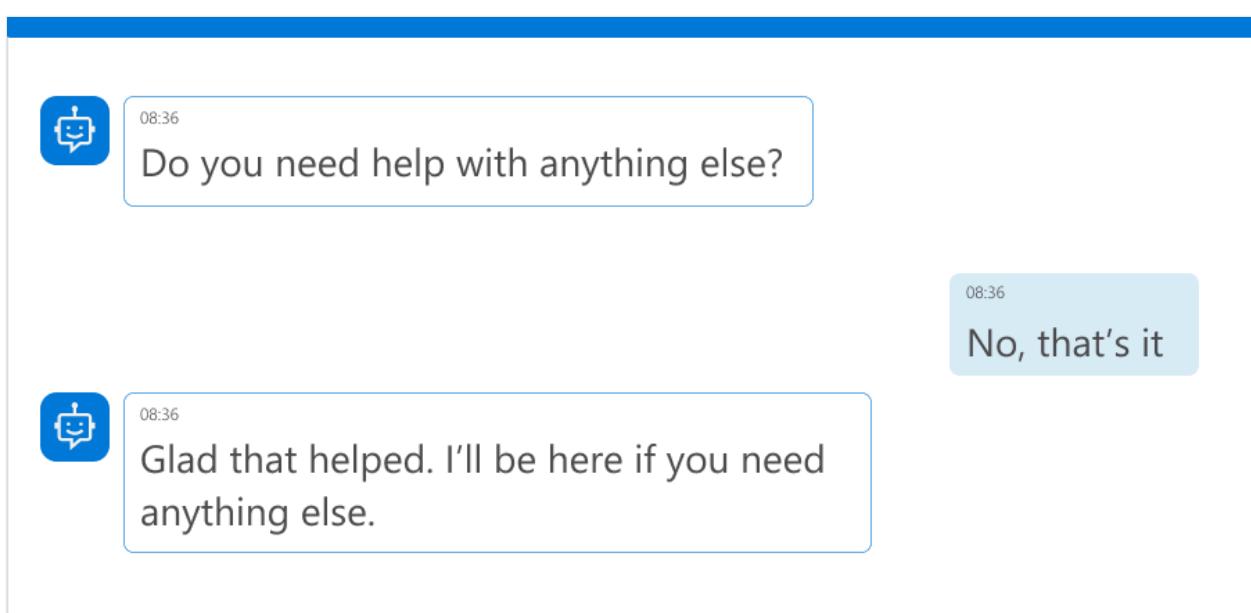
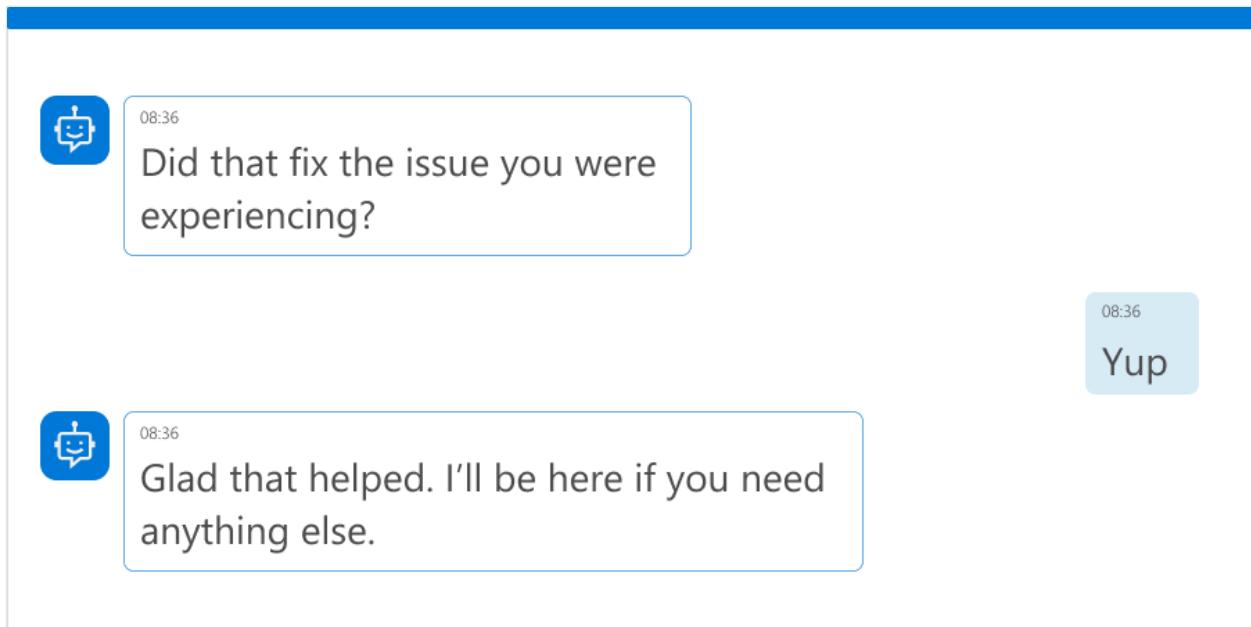
After the CUX completes a user's request, ask a follow-up question to check if the user is ready to end the conversation:

- "Did this answer your question?"
- "Did that fix the issue you were experiencing?"
- "Can I help with anything else?"
- "Do you need help with anything else?"

If the user responds that they didn't get the help they need or they have other requests, it signals the conversation should continue.

## Say more than goodbye

When the user confirms a conversation is ready to end, respond with more than "Goodbye" or "Bye!" Such an abrupt close can make it seem like the user doesn't have the opportunity to continue with a different conversation. A better approach is to let the user know that the current conversation has come to a close and leave open the possibility for further interactions.



In both of these examples, when the user responds that they're ready to end the conversation, the CUX confirms the user's response and lets them know that they can always ask another question later.

## Related information

- [Quickstart: Create and deploy an agent](#)
- [Create and edit topics](#)
- [Key concepts - Publish and deploy your agent](#)

---

## Feedback

Was this page helpful?

 Yes

 No

Provide product feedback ↗