🟥 Copilot Glossary

- Topics and Triggers (Agent's Senses)
  - Topics: Your Assistant Skills
  - Triggers: The Gateway to Your Topics
    - 1. By Agent (The Default and Most Important)
    - 2. On Redirect
    - 3. Advanced Orchestration Triggers
- Variables
  - 1. Topic Variables: For Single, Isolated Tasks (Default Scope)
  - 2. Global Variables
  - 3. System Variables
  - 4. Environment Variables
  - Supported Variable Data Types
  - Managing the Variable Lifecycle: Best Practices
- Understanding Topic Management (Agent's Nervous System)
  - 1. Go to another topic (Redirect)
  - 2. Recognize intent
  - 3. End current topic
  - 4. End all topics
  - 5. End conversation
- Activity
  - The Activity Protocol
  - Activity System Variable
- Create generative answers
- Breaking Down Complex Problems with Topics
  - Examples and Use-Cases for Topic Design
    - Use-Case 1: IT Help Desk
    - Use-Case 2: Retail Customer Service
- Triggers
  - User Message Triggers
  - Event and Activity Triggers
  - Orchestration Triggers (Generative Orchestration)
- Entities
  - How Topics Use Entities
  - Sharing and Reusability
  - Examples and Use-Cases for Designing with Entities
    - Use-Case 1: Ordering a Pizza
    - Use-Case 2: A Banking Agent
- Knowledge
- Actions
- Analytics
- Channels
- Generative Orchestration: The AI Planner in Action

Think of your agent as an intelligent assistant (the **Generative Orchestrator** or "planner") who is eager to help. Your job is to equip this employee with a set of skills (**Topics**) and teach them

when each skill is appropriate. When a user makes a request, the planner looks at all the skills it has been taught and chooses the best one for the job.

Topics and Triggers (Agent's Senses)

Topics: Your Assistant Skills

A **topic** is a self-contained "skill" or "capability" that you teach your agent. It represents a single, specific task the agent can perform from start to finish.

A *topic* defines how a segment of conversation with the user progresses. Topics are like conversation flows or dialog trees that handle a specific intent or scenario. Each topic contains nodes (steps) such as messages, questions, conditions, or actions that guide the interaction. Traditionally, a topic is triggered by certain user phrases (e.g. a "Store hours" topic might trigger when the user asks about opening hours). In Copilot Studio, you can also have the AI help create topics by describing what you want, rather than writing every prompt yourself.

There are two main types of topics you'll work with :

1. **Custom Topics**: These are the skills you build yourself. In your workshop, the "Guideline Q&A" and "Content Review and Revision" are perfect examples of custom topics. Each one is designed to handle a specific user need.
2. **System Topics**: These are pre-built, essential skills that come with every agent. They handle common conversational situations like greetings ("Conversation Start"), starting over ("Reset Conversation"), and escalating to a human. You can customize them, but you can't delete them.

Topics are the individual conversational paths that your agent can follow to complete a specific task. In our workshop, we have defined three custom topics:

- **Guideline Q&A**: For answering questions.
- **Content Review and Revision**: For performing a compliance check.
- **Measure Time Saved**: For collecting feedback.

Think of these as the primary "skills" your agent possesses. However, a skill is useless if the agent doesn't know *when* to use it. That's where triggers come in.

Triggers: The Gateway to Your Topics

A trigger is the event that causes a topic to start. The most common and important type of trigger is based on what the user says. Here are the key trigger types you must know:

1. By Agent (The Default and Most Important)

- **What it is**: This is the default trigger for new topics in generative mode. It tells the orchestrator, "You, the AI planner, have permission to use this topic whenever you think it's the best tool for the user's request."
- **How it works**: The AI planner makes its decision based primarily on the topic's **name** and, most importantly, its **description**. It reads the user's query and compares its semantic meaning to your description.
- **When to use it**: This should be used for **all** of your primary, task-oriented custom topics. Your "Guideline Q&A" and "Content Review and Revision" topics MUST use this trigger type.
- **Best Practice**: Your primary job as the agent builder is to write incredibly clear, descriptive, and accurate descriptions for these topics. A good description is the difference between an agent that works and one that doesn't. Include action verbs, key entities (inputs/outputs), and negative constraints (what the topic *doesn't* do).

2. On Redirect

- **What it is**: This makes a topic "private" or "hidden" from the AI planner. The planner will never choose a topic with this trigger on its own. It can only be activated when another topic explicitly "calls" it using a "Go to another topic" node.
- **How it works**: It's a direct, manual override of the AI planner. You are forcing the conversation down a specific path.
- **When to use it**: Use this for reusable "utility" topics. Your "Measure Time Saved" topic is the perfect example. You don't want a user to be able to just type "measure time saved" and trigger it randomly. You only want it to run immediately after a productive task has been completed.
- **Best Practice**: Identify any logic that is repeated in multiple topics (like asking for feedback, a complex sign-off message, or a multi-step authentication process) and build it as a separate topic with an "On Redirect" trigger. This keeps your agent organized and easy to maintain.

3. Advanced Orchestration Triggers

These are more advanced and allow you to "hook into" the AI planner's thinking process. You may not need them at first, but it's good to know they exist :

- **Plan Complete**: This triggers a topic *after* the AI planner has finished executing its chosen steps but *before* it shows the final answer to the user. You could use this to add a standard closing statement or a "Was this helpful?" prompt to every single response the agent generates, regardless of which topic was used.

- **AI Response Generated**: This triggers *after* the AI has generated its response but *before* it's sent. This gives you a final chance to inspect or even modify the AI's answer. For example, you could check the AI's response for certain keywords and, if found, add a legal disclaimer before sending it to the user.

For your workshop, focusing on mastering the **"By Agent"** and **"On Redirect"** triggers is the most important step. They are the fundamental building blocks for creating a well-structured and effective generative agent.

Variables

Agent's Memory

When you build your agent, variables are the "memory" that allows it to handle multi-step tasks. Choosing the right type of memory (variable scope) for each piece of information is the key to preventing errors and creating a smooth user experience. Let's explore this using your agent's topics.

Understanding Variable Scopes is crucial; otherwise, agent behavior may become inconsistent, leading to corrupted agent states and responses containing outdated or invalid data across different user conversations in the same session.

1. Topic Variables: For Single, Isolated Tasks (Default Scope)

A topic variable is temporary memory that only exists for the duration of one specific topic. **Once the topic ends, the memory is wiped clean.** This is your most-used tool and prevents data from one task from accidentally "leaking" into the next.

- **When to Use**: Use topic variables for temporary data needed for a single conversational flow.

Example: Use Case in the "Content Review and Revision" Topic

This topic has a complex job: get the content, figure out what *type* of content it is (email, blog post, etc.), and then analyze it. This involves multiple steps that are all part of *one single task*.

The RIGHT Way to Use Topic Variables:

- **Ask for Content Type**: The first step in your topic is a question: "Certainly! What type of content are you reviewing?" You save the user's answer (e.g., "Blog Post") into a **topic variable** named `Topic.ContentType`.
- **Ask for Input Method**: Next, you ask, "How would you like to provide the content?" You save the answer (e.g., "Paste Content") into another **topic variable**, `Topic.InputMethod`.

- **Capture the Content**: Based on the input method, you ask the user to paste their text. You save this large block of text into a third **topic variable**, `Topic.OriginalContent`.
- **Perform the Review**: All three of these topic variables (`ContentType`, `InputMethod`, `OriginalContent`) are passed as inputs to your prompt tool or Power Automate flow to perform the compliance check.
- **Topic Ends, Variables Disappear**: The topic displays the results and ends. At this moment, all three topic variables are automatically destroyed.

Why This is CRUCIAL (The WRONG Way):

Imagine if you had used **global variables** for this instead (`Global.ContentType`, `Global.OriginalContent`). A user successfully reviews a "Blog Post." A few minutes later, in the same conversation, they say, "Now check this email for me," and they paste the new email text.

- **The Problem**: When the "Content Review" topic runs the second time, the `Global.OriginalContent` variable gets updated with the new email text, but what if your logic doesn't re-ask the content type? The `Global.ContentType` variable might still hold the old value, "Blog Post." The agent would then incorrectly review the new email using the rules for a blog post, leading to a completely wrong compliance report. By using **topic variables**, you guarantee that every time the review topic runs, it starts fresh and must gather all the necessary information for *that specific review*, preventing this exact kind of data corruption.

2. Global Variables

A global variable is memory that is shared across **all topics** for the duration of a single user session. This is for information that defines the user or the overall context of their conversation. You create them by changing a topic variable's scope to "Global" in the variable's properties.

- **When to Use**: Use global variables for information that needs to be remembered and reused throughout the entire conversation, like user details or session-wide preferences.

Example: Use Case Across "Guideline Q&A," "Content Review," and "Measure Time Saved" Topics

Let's say this agent serves multiple teams (marketing, sales, product). Knowing who the user is can help tailor the experience and, more importantly, log feedback correctly.

The RIGHT Way to Use Global Variables:

1. **Welcome and Identify**: In your welcome topic (or the first time the user interacts), you can ask, "Which team are you on? (Marketing, Sales, Product, Other)." You save this answer into a **global variable** named `Global.UserTeam`. The agent could save user information like `Global.UserName`, `Global.Location`, `Global.LinkedInProfile`.
2. **Personalization**: Now imagine if a user wants the agent to review and generate a social media post as per Progress guidelines. The agent simply generates a revised social media post and automatically shares it your linkedIn profile using its global memory variables.
3. **Log Feedback**: When the user gives their feedback ("Yes, it did!"), you can log this response along with the `Global.UserTeam` value. You can track if the agent is saving more time for the sales team than the product team, identifying areas for improvement.
4. **Session Ends, Variable is Cleared**: The global variable persists until the user ends the session or the conversation is explicitly reset (e.g., by typing "start over"), ensuring the next user starts fresh.

Why This is CRUCIAL (The WRONG Way):

- If you used a **topic variable** (`Topic.UserTeam`) to capture the team name in the "Welcome" topic, that information would be lost the moment the topic ended.
- When the *"Measure Time Saved"* topic runs later, it would have no idea which team the user belongs to. All your feedback data would be anonymous and far less actionable. Global variables provide the "memory" that links the user's identity across the different functions of your agent.

3. System Variables

System variables are read-only values provided by the platform. You can't change them, but you can read them to make your agent's logic smarter.

- **When to Use**: Use system variables to make your agent's logic more dynamic and aware of the conversational context. For example, you can use them to check if a user is signed in or to access the exact text they just typed.

Use Case in the "Guideline Q&A" Topic

The "Guideline Q&A" topic uses a "Create generative answers" node. This node needs input to know what the user is asking.

The RIGHT Way to Use a System Variable:

- The input for the generative answers node is set to the **system variable** `System.Activity.Text`. This variable always contains the last thing the user typed.

When a user asks, "What is our rule for using ampersands?" this exact phrase is in `System.Activity.Text` and is fed directly into the generative AI model to find the answer from your knowledge base.

- This is the most direct way to handle Q&A.

Why This is CRUCIAL (The WRONG Way):

- A less efficient way of doing the same would be to create a **Question** node first, asking "What is your question?", and save the response to a `Topic.UserQuestion` variable.
- While this works, it's redundant. The user has *already stated their question*. Using `System.Activity.Text` allows you to immediately capture the user's intent from their trigger phrase, creating a more fluid and intelligent-feeling interaction without asking an unnecessary follow-up question.

4. Environment Variables

Environment variables store configuration details that change between your environments (e.g., Development, Test, Production). This is a critical practice for professional ALM (Application Lifecycle Management).

- **When to Use**: Use environment variables to manage external service endpoints, API keys, or any other configuration that needs to change without editing the agent itself. This prevents hard-coding values.

Use Case in the "Content Review and Revision" Topic

Your topic has a branch to handle content from a web URL. This requires a Power Automate flow to fetch the web page content. That flow might also call a Generative AI model endpoint.

The RIGHT Way to Use Environment Variables:

- **Define Endpoints**: Let's assume Progress's AI models may have different endpoints for testing and for production. You create an **Environment Variable** in your solution named `ComplianceAIEP`.
- **Configure in Each Environment**:
  - In your **Development** environment, you set `ComplianceAIEP` to `https://dev-ai.progress.com/api/v1/review`.
  - In your **Production** environment, an admin sets `ComplianceAIEP` to `https://prod-ai.progress.com/api/v1/review`.

- **Use in Power Automate**: The Power Automate flow that fetches the URL content and calls the AI model does **not** have a hard-coded URL. It dynamically reads the value of the `ComplianceAIEP` environment variable.

Why This is CRUCIAL (The WRONG Way):

- If you hard-code the development URL (`https://dev-ai.progress.com/...`) directly into your Power Automate flow, it will work perfectly during testing. But when you deploy the agent to your production environment, it will **continue to call the development AI endpoint**.
- Environment variables ensure that your agent automatically and safely adapts to the environment it's running in without requiring risky manual edits after deployment.

Supported Variable Data Types

The type of a variable determines what kind of values it can hold. A variable's type is set the first time a value is assigned to it

Data Type	Description	**Use Case Example**
String	A sequence of characters representing text.	Storing a customer's name or address.
Number	Any real number.	Storing a product quantity or price.
Boolean	A logical value that can only be `true` or `false` .	Storing whether a customer has consented to terms (`true`).
DateTime	A date, a time, or both.	Storing an appointment date for a product demo.
Table	A list of values, where all values must be of the same type.	Storing a list of product IDs returned from an inventory search.
Record	A collection of name-value pairs, like a single	Storing a complete customer record with

	row from a database.	fields for name, email, and phone number.
Choice	A list of string values with associated synonyms.	Capturing a selection from a list of options, like "Email", “Blog Post", “Others”
Blank	Represents "no value" or a null value.	Used to check if a variable has been initialized.

Managing the Variable Lifecycle: Best Practices

Properly managing how variables are created, populated, and cleared is crucial for preventing the issues you've described.

Variable Initialization

Variables are typically initialized when a user provides an answer in a **Question** node. A variable is also initialized when you manually assign it a value. If you use a global variable before it's initialized, the agent will automatically pivot to the topic where it's first defined, ask the question, and then return to the original topic. While seamless, it's a best practice to have a clear "Welcome" or "Setup" topic that gathers key global information upfront.

Variable Assignments

Use the **Set a variable value** node to explicitly assign or change a variable's value. You can set it to a literal value (like "123"), another variable, or a Power Fx formula. This is useful for transforming data or setting default values.

Clearing Variables

This is key to avoiding "corrupted" data between conversations.

- **Topic Variables**: These are automatically cleared when the topic ends.
- **Global Variables**: These persist for the entire user session. They are only cleared when the conversation is explicitly reset.
- **Best Practice**: The built-in **Reset Conversation** system topic uses a **Clear variable values** node to wipe all global variables. Always guide users to a "Start Over" or "Main Menu" option that redirects to this topic at the end of a major workflow (like a completed order). This ensures that the next time the user engages, they start with a clean slate, preventing data from a previous session from "corrupting" a new one.

Input/Data Validation

You can ensure the data users provide is correct using advanced validation.

- **Basic Validation**: When you create a Question node, you identify the type of data to expect (e.g., Number, Email). The agent automatically reprompts if the user's input doesn't match.
- **Advanced Validation**: In the node's settings, you can add **Additional entity validation**. This allows you to write a Power Fx formula to enforce more complex rules. For example, if you ask for a quantity, you can add a condition like `Topic.Quantity > 0` to ensure the number is positive.
- **Handling No Valid Entity**: Configure what happens if the user repeatedly fails to provide a valid answer. Instead of just ending the conversation, you can use the **No valid entity found** setting to set the variable to a default value, leave it blank, or escalate to a live agent.

By thoughtfully applying these variable scopes and management techniques, you can build complex, multi-topic agents that are reliable, maintainable, and provide a seamless experience for your users.

Further Reading

1. Variables overview - Microsoft Copilot Studio
2. Work with global variables - Microsoft Copilot Studio
3. Work with variables - Microsoft Copilot Studio
4. Configure context variables for agents
5. How to Create Variables in Microsoft Copilot Studio
6. How to Add Environment Variables in Copilot Studio
7. Mastering Variables in Microsoft Copilot Studio for Enhanced Interactivity

---

Understanding Topic Management (Agent's Nervous System)

If variables are the agent's "memory," then topic management is its "nervous system"—it controls the flow of conversation, dictates where the user goes next, and ensures the agent behaves logically. Mastering this is what allows you to build complex, multi-step workflows like your Content Compliance Agent.

Topic management nodes control the conversational path. They tell the agent when to start, stop, or switch between different topics. Using them correctly is critical for managing the "stack" of active topics and, by extension, the lifecycle of your variables.

1. Go to another topic (Redirect)

- **What it is**: This node explicitly transfers the conversation from the current topic to a different one. Think of it as one topic "calling" another for help.
- **What happens under the hood**: The current topic is paused, and its state (including all its **topic variables**) is saved. The agent then starts the new topic. By default, once the new topic finishes, the conversation returns to the original topic, right after the redirect node.
- **When to use it**: Use this to create reusable, modular logic. If you have a set of steps that multiple topics need to perform (like asking for feedback or authenticating a user), you can build it once in a separate (shared) topic and have other topics redirect to it.

Best Practice & Connection to Variables:

- **Modular Design**: Break down large, complex topics into smaller, "bite-sized" ones.
  For example, we can break the "Content Review and Revision" topic is a perfect candidate. Instead of having one massive topic, you could have a main topic that redirects to smaller, specialized topics like "ProcessPastedContent," "ProcessFileURL," or "ProcessWebURL.
- **Passing Variables**: When you redirect, you can pass variables from the calling topic to the destination topic. This is essential for giving the new topic the context it needs.
  For example, our main "Content Review" topic could capture `Topic.ContentType` ("Blog Post") and then redirect to "ProcessPastedContent," passing `Topic.ContentType` as an input variable. This makes the "ProcessPastedContent" topic reusable for any content type.

Example in your Content Compliance Agent:
Our design already includes this! At the end of both the "Guideline Q&A" and "Content Review" topics, you use a **Go to another topic** node that redirects to your **"Measure Time Saved"** topic. This ensures that after the agent provides value, it consistently asks for feedback, making the feedback logic reusable.

2. Recognize intent

- **What it is**: This node acts like a mini-router inside your topic. It takes a variable (usually the user's last response) and tries to match it against all the trigger phrases of every topic in your agent.
- **What happens under the hood**: It reruns the agent's Natural Language Understanding (NLU) model on a piece of text you provide. If it finds a matching topic, it redirects the conversation there. If not, the conversation continues in the current topic.
- **When to use it**: Use this when you've reached the end of a task and want to let the user drive the next step with natural language, rather than forcing them to pick from buttons. It creates a

much more fluid, human-like conversation.

Best Practice: This is ideal for a "What's next?" prompt. After a successful content review, you might send a message: "Your content review is complete. What would you like to do next?" Then, instead of adding buttons, you add a **Recognize intent** node. If the user types "check another document," it will re-trigger the "Content Review" topic. If they type "what are our SEO rules?" it will trigger the "Guideline Q&A" topic.

Example in your Content Compliance Agent:
After the "Content Review and Revision" topic provides the compliance score, end with a message node: "I've sent the revised versions to you. Is there anything else I can help with today?" Immediately follow this with a **Recognize intent** node. This allows the user to naturally pivot to a new task (like asking a guideline question) without needing to be forced through a rigid menu.

3. End current topic

- **What it is**: This node explicitly stops the execution of the topic it's in.
- **What happens under the hood**: The topic is terminated. All of its **topic variables** are immediately destroyed. If this topic was called by another topic (via a redirect), the conversation returns to the parent topic. If it was a top-level topic, the agent will wait for the user's next message to trigger a new topic.
- **When to use it**: Use this for conditional exit points. It's perfect for when a user decides to cancel a process or when a certain condition means the topic's main goal can't be completed.

Best Practice & Connection to Variables: This is your primary tool for cleaning up **topic variables**. By ending the topic, you ensure that temporary data (like the `Topic.OriginalContent` from a review) is cleared, preventing it from being accidentally used later.

Example in your Content Compliance Agent:
In our "Content Review" topic, after you ask the user how they want to provide content (File, URL, Paste), they might choose "URL." You then ask for the URL. What if the user types "cancel" instead? You can have a condition that checks for the word "cancel." If true, that branch leads to a message ("Okay, I've canceled this review.") followed immediately by an **End current topic** node.
This cleanly exits the review process and wipes any data collected so far.

4. End all topics

- **What it is**: This is a more forceful stop. It ends not just the current topic, but also any parent topics that may have called it.
- **What happens under the hood**: The entire topic stack is cleared. All active **topic variables** from all active topics are destroyed. The agent effectively stops what it's doing and goes back to a "listening" state, as if the conversation just started.
- **When to use it**: This is for "hard reset" scenarios. Use it when the user wants to completely start over or when a final, concluding action has been taken and there's no logical path to return to.

Best Practice & Connection to Variables: This node **does not** clear **global variables**. This is a critical point. If you want to fully reset the session, you must first use a "Clear variable values" node and then use the **End all topics** node. The built-in "Reset Conversation" system topic is the perfect example of this pattern.

Example in your Content Compliance Agent:
Let's say a user finishes a content review and gives feedback in the "Measure Time Saved" topic.
At the very end of the *"Measure Time Saved"* topic, after thanking the user, you could use an **End all topics** node. This is because the task is completely finished. There's no "parent" topic to go back to. This provides a clean end to the workflow, waiting for the user to initiate something new.

5. End conversation

- **What it is**: This node sends a message to the user and then terminates the session from the agent's side. In many channels (like web chat), this will disable the input box for the user.
- **What happens under the hood**: It signals to the channel that the agent considers the conversation finished. All topics are ended, and all variables (both topic and global) are effectively cleared because the session is over.
- **When to use it**: Use this only when you are certain the interaction is 100% complete and you don't expect the user to say anything else. It's common after escalating to a live agent or after a final concluding statement.

Best Practice: Use this sparingly. In most cases, it's better to end the topics (`End all topics`) and let the user decide if they want to start a new interaction. A good place to use it is in a "Goodbye" topic triggered by phrases like "thank you, that's all."

Example in your Content Compliance Agent:
You could create a simple "Goodbye" topic. If a user says "thanks, bye," this topic is triggered. It could have one message node ("You're welcome! Have a great day.") followed by an **End conversation** node. This provides a definitive and polite end to the entire interaction.

By combining skillful variable management with precise topic management, you orchestrate a conversational flow that is logical, efficient, and robust. This ensures data is handled correctly and the user is guided seamlessly through the different capabilities of your agent.

Further Reading

1. Manage topics - Microsoft Copilot Studio
2. Trigger topics - Microsoft Copilot Studio
3. Copilot Studio Tutorial: Work with Topics
4. Enhancing Your Copilot Studio Agent with Topics and Actions
5. Topics in Copilot Studio - Microsoft Copilot Studio
6. Improving Topic Detection and Response Flow in FAQ Copilot Agents - Microsoft Q&A
7. Passing Variables from Topics to Actions in Copilot Studio

---

Activity

In Microsoft Copilot Studio, the `Activity` **object** is a fundamental concept that structures all communication between a user and an agent. It is part of the core **activity protocol**, which provides a standardized format for every interaction, ensuring seamless communication.

By understanding and utilizing the `Activity` object and its various types, developers can build more sophisticated and interactive agents that go beyond simple question-and-answer conversations.

The Activity Protocol

The activity protocol is the transport layer for communication in Copilot Studio. It categorizes interactions into different types of activities. The two most common types are `message` and `event` activities.

- **Message Activities**: These are the basic units of conversation, carrying the content of the interaction, such as text, media, or adaptive cards. They are used for direct back-and-forth communication where a user sends a query and the agent provides a response.
- **Event Activities**: These are used to communicate non-verbal information or system-generated updates. They enable asynchronous communication and can trigger workflows without direct user interaction. For instance, an event can notify the agent that a user has joined the conversation or that the context of the conversation has changed.

Based on the image and the context of Microsoft Copilot Studio, here is an explanation of the `Activity` system variable and `Activity.Text`.

Activity System Variable

The `Activity` variable is a **System variable**. System variables are automatically created with every agent and provide contextual information about the ongoing conversation or the user interacting with the agent. You don't create or define them yourself; they are always available for use in your topics.

The `Activity` system variable specifically holds information about the most recent interaction or "activity" that occurred in the conversation. An activity can be a message from the user, an event, or another type of interaction defined by the underlying Bot Framework protocol.

As shown in the "Select a variable" panel in your image, the `Activity` variable is an object with several properties, including:

- `Activity.Recipient.Name`: The name of the recipient of the activity.
- `Activity.Text`: The text content of the activity.
- `Activity.Type`: The type of the activity (e.g., 'message' or 'event').

Activity.Text

`Activity.Text` is a property of the `Activity` system variable that contains the textual content of the most recent message received from the user.

In the "Create generative answers" node shown in your image, `Activity.Text` is used as the **Input**. This means the agent will take the raw text from the user's last message and use it to generate a relevant answer from the specified data sources. This is a common pattern for building generative AI agents that need to respond directly to a user's query.

Can I use activity to access the entire conversation?

The `Activity` variable, and specifically `Activity.Text`, represents only the **last user message** received by the agent, not the entire conversation history.

Microsoft Copilot Studio does not provide a built-in system variable that contains the full transcript of an ongoing conversation. To access the complete history of the dialogue at runtime (for example, to create a summary or pass it to another service), you would need to implement a more advanced solution, such as using a Power Automate flow to call the Direct Line API and retrieve all activities for the current conversation ID.

How the Activity Object is Used?

The `Activity` object is used to send and receive different types of information between the agent and the channel it's running on (like a custom web chat or Microsoft Teams). Copilot Studio provides specific nodes in the authoring canvas to create and send these activities.

How can I send Events with the Event Activity node?

You can use the `Event activity` **node** to send custom events from your agent. When you send an event, you give it a name and a value. The value can be a simple text string, a variable, or a Power Fx formula. The channel that receives the event can then use this information to perform actions:

Common use cases for the `Event activity` node include:

- **Controlling a custom web chat**: An agent can send an event to the web page it's embedded in, instructing it to perform an action, like navigating to a different page or updating its UI.
- **Integrating with external services**: Events can be used to control third-party services. For example, in a voice-enabled agent, you could send events to start or stop call recording through a service like AudioCodes.
- **Executing client-side tools**: An agent can send an event to the client (e.g., a desktop application) to request the execution of a local function. The agent then waits for the client to perform the action and return the result in another event. For example, an agent could ask a PowerPoint add-in to retrieve the text from a specific slide.

What are other Activity Types?

Besides the `Event activity` node, you can use the `Invoke activity` **node** to send other types of activities defined in the Bot Framework schema. Some common types include:

- **Typing**: This sends a typing indicator to the user, showing that the agent is preparing a response.

- **Invoke**: This is often used for Microsoft Teams integrations, where a topic can be triggered by an `Invoke` from Teams, and the agent can send an `Invoke response` back.
- **Handoff**: This is used to transfer the conversation to another system, such as a live agent platform.

How can I reviewing Agent Activity?

Copilot Studio allows you to track and review the activities of your agent to understand its behavior and troubleshoot issues. This is done through the **Activity** page.

- **Activity Map**: For each session, Copilot Studio generates a visual `activity map` that shows the sequence of inputs, decisions, and outputs as a series of nodes. This helps you visualize the flow of the conversation and identify any errors or unexpected behavior.
- **Transcript**: You can also view a detailed transcript of the conversation, which includes user inputs, agent responses, and the payloads of any event triggers.

What the Activity Map is Used For?

It is a visual debugging and analytics tool that provides a step-by-step, graphical representation of an agent's session. It allows developers to see exactly how the agent processes inputs, makes decisions, and generates outputs, making it an essential feature for troubleshooting and optimizing agent performance.

Each interaction in a session—whether from the user, the agent, or an external event—is represented as a *node* in the map, creating a clear and easy-to-follow flowchart of the conversation

The primary purpose of the Activity Map is to provide deep insight into your agent's inner workings. It helps you:

- **Debug and Troubleshoot:** Quickly identify errors, such as missing or invalid parameters for actions, and understand why an agent isn't behaving as expected.
- **Visualize Conversation Flow:** See the exact path a conversation takes, including which topics are triggered and how the agent moves between them.
- **Analyze Performance:** Review how long each step takes to execute, helping you pinpoint and optimize slow-performing actions or integrations.
- **Verify Data Handling:** Inspect the inputs and outputs of each node to confirm that the agent is sending and receiving data correctly.
- **Understand AI Rationale:** For generative actions, you can view the AI's "rationale" to understand why it chose a specific tool or knowledge source to respond to a user's query.

How can I use Activity Map to improve my agent performance?

You review the historical Activity Maps for several slow sessions. You notice that a particular node, which calls a Power Automate flow to look up customer order data, consistently takes over 10 seconds to complete. This data indicates that the bottleneck is the external flow, not the agent itself. You can then focus your optimization efforts on improving the performance of that Power Automate flow to speed up the agent's response time.

1. Review agent activity - Microsoft Copilot Studio
2. Understand your agent's activity
3. Configure context variables for agents
4. Use entities and slot filling in agents - Microsoft Copilot Studio
5. Mastering Variables in Microsoft Copilot Studio for Enhanced Interactivity
6. Send an event or activity - Microsoft Copilot Studio
7. Accessing Copilot Studio Conversation Transcript at Runtime

---

Create generative answers

The `Create generative answers` **node** allows your agent to generate real-time, AI-powered answers from various knowledge sources. Instead of relying on predefined conversational paths within a topic, this node enables the agent to dynamically synthesize information to respond to a user's query.

This node takes an input—typically the user's latest query (using the `Activity.Text` system variable)—and uses it to search across a defined set of data sources. It then uses a LLM to generate a conversational, natural-language response based on the retrieved information.youtube

This node can be used in two main ways:

- **As a fallback:** When a user's question doesn't trigger any specific topic you've created, generative answers can be used to search your knowledge sources and attempt to provide an answer. This prevents the conversation from hitting a dead end.
- **Within a topic:** You can add the node to any topic to provide answers from specific knowledge sources relevant to that part of the conversation. Sources defined within this node override the agent-level sources, allowing for more targeted responses.

What data sources are available for create generative answers?

You can configure the `Create generative answers` node to pull information from a wide range of internal and external sources, including:

- **Public Websites:** Search public web content using Bing Search.
- **SharePoint and OneDrive for Business:** Connect to your organization's internal documents and sites. Authentication ensures the agent only accesses content the user has permission to view.
- **Uploaded Documents:** Use files (such as PDFs, Word documents, or PowerPoint presentations) that you upload directly to the agent.
- **Azure OpenAI on your data:** Leverage your own data connected through the Azure OpenAI service.
- **Custom Data:** Provide your own data source, such as the output from a Power Automate flow or a Skill, allowing for dynamic, real-time information retrieval from other systems.

What are some common real-world use cases for create generative answers?

The `Create generative answers` node is versatile and can be applied in many scenarios to create more intelligent and helpful agents.

Human Resources and Internal Help desks

- **Use-Case:** An employee needs to know the company's policy on remote work.
- **Example:** An HR agent uses a `Create generative answers` node configured to search the company's internal SharePoint site where HR policies are stored. The employee asks, "What are the guidelines for working from home?" The agent finds the relevant policy document and generates a summary of the key points, such as eligibility requirements and communication expectations.

Customer Support

- **Use-Case:** A customer wants to troubleshoot a common issue with a product.
- **Example:** A support agent is connected to a public knowledge base of "how-to" articles. When a user asks, "How do I factory reset my device?", the `Create generative answers` node searches the articles, finds the correct procedure, and presents it to the user as a clear, step-by-step guide.youtube

Dynamic Information Retrieval

- **Use-Case:** A user wants to check the status of their flight.

- **Example:** The agent topic first asks for the flight number. This number is passed to a Power Automate flow that connects to the airline's API. The flow returns the flight's status, gate, and departure time as a block of text. This text is then fed into the `Create generative answers` node, which formats it into a user-friendly response like, "Your flight, AA123, is on time and scheduled to depart from Gate B5 at 3:45 PM".

Product Information and Sales

- **Use-Case:** A potential customer has questions about the technical specifications of a product.
- **Example:** A sales agent on a public website is configured with generative answers pointing to the product's official documentation and spec sheets. A user asks, "What is the battery life of the new laptop model, and does it support Thunderbolt 4?" The agent searches the provided sources and generates a concise answer detailing the battery capacity and port capabilities.

1. Add a generative answers node - Microsoft Copilot Studio
2. How to Use a Custom Data Source for Generative Answers? - Global SharePoint
3. Instruction guidance for generative orchestration - Microsoft Copilot Studio

In Microsoft Copilot Studio, a **topic** is the fundamental building block for creating conversations. Think of it as a detailed blueprint or a mini-script for a single, specific conversational path that defines how an agent should respond to a particular user request. Each topic is a self-contained unit designed to handle one specific task or subject, from start to finish.learn.microsoft+2

A topic consists of two main components:

- **Triggers:** These are the phrases or events that initiate the topic. For example, a user typing "I need help with my password" would trigger a "Password Reset" topic.youtube
- **Conversation Nodes:** These are the steps the agent takes once a topic is triggered. Nodes can send messages, ask questions, use logic to create branches, and call other tools or systems to get information.youtube

Breaking Down Complex Problems with Topics

Yes, a core principle of effective agent design in Copilot Studio is using topics to **break down complex problems into a series of simpler, manageable subproblems**. This modular approach makes the agent easier to build, test, and maintain.

Instead of creating one massive, complicated topic to handle a broad request, you design several smaller, focused topics. Each topic handles one piece of the larger task, and they can

redirect to one another to guide the user through a complete workflow.

For example, a user's request like "I'm a new employee and need to get set up" is a complex problem. A poorly designed agent might try to handle this in one giant topic. A well-designed agent would break it down into several distinct skills (topics):

- A main **"New Employee Onboarding"** topic that acts as a guide.
- A **"Benefits Enrollment"** topic to explain health insurance and retirement plans.
- An **"IT Equipment Setup"** topic to guide them through setting up their computer and accounts.
- A **"Company Policies Review"** topic to provide key HR documents.

The main topic would greet the new employee and then ask which area they'd like to start with, redirecting them to the appropriate sub-topic.

Examples and Use-Cases for Topic Design

Here are practical examples of how to design and build topics by breaking down common business scenarios.

Use-Case 1: IT Help Desk

- **Complex Problem:** A user says, "My computer is acting weird and I can't work."
- **Topic Breakdown:**
  - **Topic 1: "IT Support Triage"**
    - **Design:** This topic is triggered by general help phrases like "computer problem" or "IT help." It asks clarifying questions to diagnose the issue, such as "Is it a software problem or a hardware problem?" or "Are you getting an error message?" Based on the answers, it redirects to a more specific topic.
  - **Topic 2: "Application Troubleshooting"**
    - **Design:** This topic handles software issues. It asks for the application name and guides the user through common fixes like clearing the cache, restarting the application, or checking for updates.
  - **Topic 3: "Create Support Ticket"**
    - **Design:** If troubleshooting fails, other topics redirect here. This topic's skill is to gather the necessary information (user's name, description of the problem, and a screenshot if possible) and use a Power Automate flow to create a ticket in a system like ServiceNow or Jira.

Use-Case 2: Retail Customer Service

- **Complex Problem:** A customer wants to do something with their recent purchase.
- **Topic Breakdown:**
  - **Topic 1: "Manage My Order"**
    - **Design:** Triggered by phrases like "my order" or "I have a question about my purchase." It first asks for the order number and the customer's email to authenticate them. Then, it presents them with a choice: "What would you like to do? Check status, Modify order, or Start a return."
  - **Topic 2: "Check Order Status"**
    - **Design:** This topic takes the order number, calls an external shipping API to get the latest tracking information, and presents it to the customer in a clear, easy-to-read format.
  - **Topic 3: "Start a Return"**
    - **Design:** This topic guides the customer through the return process. It explains the return policy, asks for the reason for the return, and, if the criteria are met, generates a shipping label and emails it to the customer.

By designing your agent with these focused, single-purpose topics, you create a conversational experience that is both powerful and easy to manage.

1. Topics in Copilot Studio - Microsoft Copilot Studio
2. Create and edit topics - Microsoft Copilot Studio

Triggers

In Copilot Studio, a **Trigger** is what initiates a topic, determining when a specific conversational path should be executed. Triggers allow an agent to respond to a wide range of user inputs and system events, going beyond simple keyword matching.

There are two main categories of triggers: those based on user messages and those based on events.

User Message Triggers

These triggers fire in response to a user's typed or spoken message.

Trigger Type	Description	When to Use & Example

By agent	This is the default for agents using generative orchestration. The agent uses its natural language understanding (NLU) to match the user's intent with the topic's name and description.	**When to use:** In modern agents that leverage generative AI to understand user intent more broadly. **Example:** A topic named "Check Order Status" with a description "Allows users to check the status of their order by providing an order number" is triggered when a user says, "I want to know where my package is"
Phrases	The classic and most common trigger type. It fires when a user's message closely matches one of several predefined trigger phrases.	**When to use:** For topics with a clear and specific purpose that can be captured by a set of phrases. **Example:** A "Store Hours" topic is triggered if a user types "What time do you open?", "Are you open on Sundays?", or "Show me your business hours".

Event and Activity Triggers

These triggers fire in response to non-message activities, allowing the agent to react to system events or actions from the user interface.

Trigger Type	Description	When to Use & Example
Activity received	A general-purpose trigger that fires when an activity of *any* type is received. You can filter it to a specific activity type if needed.	**When to use:** To catch all incoming activities for logging purposes or to handle custom activity types not covered by other triggers. **Example:** Create a topic that logs every incoming activity to an external monitoring service.
Message received	Fires specifically when a `message` activity is received. This is the activity type for standard user chat messages.	**When to use:** When you want to execute a topic every time a user sends a message, regardless of its content. **Example:** A topic that checks every user message for profanity before the main NLU processing occurs.
Event received	Fires when an `event` activity is received. Events are used to send programmatic information to the agent without displaying it to the user.	**When to use:** To have the agent react to actions happening in the background or on the hosting webpage. **Example:** A web page sends an event named

		"user-logged-in" with the user's name. A topic with an "Event received" trigger captures this event and greets the user by name.
Conversation update received	Fires when a `conversationUpdate` activity is received, such as a user joining or leaving the chat.	**When to use:** To provide a welcome message or perform setup actions when a conversation starts. **Example:** In a Microsoft Teams chat, a topic with this trigger sends a "Hello and welcome! How can I help you today?" message as soon as the user is added to the conversation.
Invoke received	Fires when an `invoke` activity is received. This is common in channels like Microsoft Teams when a user interacts with an Adaptive Card button or a message extension.	**When to use:** To create interactive cards that trigger specific topics when a button is clicked. **Example:** An agent sends a card asking "Did this solve your issue?" with "Yes" and "No" buttons. Clicking "Yes" sends an `invoke` activity that is caught by a topic

		with an "Invoke received" trigger, which then closes the support ticket.
Inactivity	Fires when a user has not interacted with the agent for a configured period.	**When to use:** To prompt a user for a response or to gracefully end a conversation that has gone idle. **Example:** After two minutes of no response from the user, a topic with an "Inactivity" trigger sends the message, "Are you still there? I will end this conversation in one minute if I don't hear back."

Orchestration Triggers (Generative Orchestration)

These triggers are specific to agents using the generative orchestration model and allow you to hook into the agent's AI-driven planning process.

Trigger Type	Description	When to Use & Example
Plan complete	Fires after the agent has finished executing all the steps it planned to take in response to a user's query.	**When to use:** To perform cleanup actions or log the final outcome of the agent's turn. **Example:** After the agent has successfully

		answered a user's question by calling two different actions, a topic with this trigger logs a "turn successful" metric.
AI response generated	Fires after the agent has generated its response but *before* it is sent to the user. This gives you a chance to inspect or modify the response.	**When to use:** For compliance, moderation, or to add extra information to the agent's generated response. **Example:** A topic with this trigger intercepts the agent's response, checks it for sensitive keywords, and if none are found, adds a standard legal disclaimer before sending it to the user.

1. Mastering Topic Triggers in Copilot Studio
2. Understanding Generative Orchestration Topic Triggers in Copilot Studio | Digital Bricks
3. Trigger topics - Microsoft Copilot Studio
4. Event triggers overview - Microsoft Copilot Studio

Entities

In Microsoft Copilot Studio, an **entity** is a crucial building block that represents a unit of information, allowing an agent to recognize and extract specific types of data from a user's conversation. Think of entities as categories or labels for real-world subjects like a date, a phone number, a color, or a product name. By using entities, your agent can move beyond simple keyword matching to intelligently understand what the user is talking about, making conversations more efficient and natural.

How Topics Use Entities

Entities are the key to making topics dynamic and intelligent. Within a topic, you use entities in **Question** nodes to identify and save specific pieces of information from a user's response into variables.

A powerful feature called **slot filling** allows an agent to proactively capture this information. If a user provides multiple pieces of information at once, the agent can "fill in the slots" for several questions simultaneously, skipping them and making the conversation faster .

Sharing and Reusability

Entities are designed to be **reusable and sharable across all topics** and components within your agent. You define an entity once, and it becomes part of the agent's core knowledge. Any topic that needs to understand that specific type of information can then reference it.

This centralized approach offers two major benefits:

- **Consistency:** All topics will recognize and handle the same type of data (like a "Product ID") in the same way.
- **Maintainability:** If you need to update the definition of an entity (for example, by adding new products to a "Product" entity), you only have to do it in one place, and the change is immediately available to every topic that uses it.

Examples and Use-Cases for Designing with Entities

Understanding how to use entities is fundamental to designing effective agents. Here's how you can apply them in practical scenarios.

Use-Case 1: Ordering a Pizza

- **Complex Problem:** A user wants to order a pizza with various customizations.
- **Entity Design:** Before building the topics, you would create several **custom entities**:
  - **Pizza Size (Closed list):** A list with items like "Small," "Medium," and "Large," and their synonyms (e.g., "personal" for Small, "family" for Large) .
  - **Toppings (Closed list):** A list of available toppings like "Pepperoni," "Mushrooms," "Olives," etc.
  - **Crust Type (Closed list):** "Thin Crust," "Deep Dish," "Stuffed Crust."
- **Topic and Entity Interaction:**
  - A single **"Order Pizza"** topic is created.
  - Inside the topic, you add a series of **Question** nodes:
    i. "What size pizza would you like?" (Identifies the `Pizza Size` entity).

    ii. "What toppings would you like?" (Identifies the `Toppings` entity).
    iii. "What type of crust do you want?" (Identifies the `Crust Type` entity).
  - **Slot Filling in Action:** A user might say, "I want to order a large pizza with pepperoni on thin crust." The agent, using the entities, recognizes all three pieces of information at once. It saves "Large" to the `PizzaSize` variable, "Pepperoni" to the `Toppings` variable, and "Thin Crust" to the `CrustType` variable, and then completely skips all three questions, moving directly to confirming the order.

Use-Case 2: A Banking Agent

- **Complex Problem:** A user needs to perform various banking transactions.
- **Entity Design:** You would use both prebuilt and custom entities:
  - **Transaction Type (Closed list):** A custom entity with items like "Check Balance," "Transfer Funds," and "Pay Bill."
  - **Account Type (Closed list):** A custom entity with items like "Checking" and "Savings."
  - **Money (Prebuilt):** The built-in entity to recognize currency values like "$50.25" or "one hundred dollars".
  - **US Social Security Number (Prebuilt regex):** A prebuilt entity that uses a regular expression to identify a nine-digit SSN for identity verification.
- **Topic and Entity Interaction:**
  - **Topic 1: "Main Menu"** - Asks the user what they want to do and uses the `Transaction Type` entity to route them to the correct topic.
  - **Topic 2: "Transfer Funds"**
    i. Asks, "How much would you like to transfer?" (Uses the prebuilt `Money` entity).
    ii. Asks, "From which account?" (Uses the `Account Type` entity).
    iii. Asks, "To which account?" (Reuses the same `Account Type` entity).
  - **Sharing in Action:** The `Account Type` entity is defined once but is reused in multiple topics, such as "Check Balance," "Transfer Funds," and "View Transaction History," ensuring consistency across the entire agent.

By first defining the key pieces of information (entities) your agent needs to understand, you can then build modular, intelligent topics that create smooth and efficient conversational experiences.

1. Use entities and slot filling in agents - Microsoft Copilot Studio
2. Using Entities in Copilot Studio For Teams - Power Platform for Educators (Ep. 8)

3. What is Entity in Microsoft Copilot Studio?
4. Topic authoring best practices - Microsoft Copilot Studio
5. Work with entities and variables in Microsoft Copilot Studio - Training
6. Microsoft Copilot Studio | Entities and Variables Explained

Understanding Generative Orchestration Topic Triggers in Copilot Studio | Digital Bricks

---

Knowledge

- Knowledge sources are documents or information bases that the agent can draw from to answer questions. For example, you can connect your agent to FAQs, product manuals, web pages, or enterprise data.
- With Generative Answers, the agent can search across these knowledge sources and generate an answer on the fly, even if you didn't script a specific topic for the question. Knowledge acts as the agent's extended memory, allowing it to provide relevant info to users from your content.

Actions

- Actions let your copilot do things – they are integrations or operations the agent can execute.
- Actions can range from looking up data in a database, creating a support ticket, sending an email via a Power Automate flow, or calling a REST API.
- In Copilot Studio, you can add prebuilt or custom actions (formerly known as plugins) to extend your agent's capabilities. Each action has a name, description, and input/output parameters.
- In classic (non-generative) mode, you would call actions explicitly within a topic.
- With generative orchestration turned on, the agent's AI planner can automatically decide to use an action when it's relevant, based on the action's description.
- Copilot Studio makes sure that if an action requires certain inputs (like a customer ID or date), the agent will ask the user for that info or pull it from context automatically.

Analytics

- Once your copilot is up and running, the Analytics section of Copilot Studio helps you understand how well it's performing.

- Analytics provide dashboards and metrics about user engagement, conversation sessions, resolution rates, escalation rates, abandonment, and which topics are being used. In short, it shows you what your users are asking, how the bot is handling those queries, and where there's room for improvement.
- By reviewing analytics, you can iterate on your topics and knowledge to make the agent more effective.

Channels

- Channels are the platforms or interfaces through which users interact with your agent.
- Copilot Studio allows you to deploy your chatbot to multiple channels such as a website chat (a live or demo web site), a mobile app, Microsoft Teams, Facebook Messenger, or even as an integration with Microsoft 365 Copilot.
- You can publish your agent to one or more channels so that your customers or employees can reach it in their preferred medium. For instance, you might embed it on your company's support webpage and also make it available as a Teams bot internally. The conversation logic remains the same, but Copilot Studio handles the connectors to these various channels.

All of these components work together to form a complete conversational experience. For example, imagine a user asks a question on your website (Channel). The agent looks at the Topics it has and Knowledge sources to figure out the best answer. It might use Variables to remember details from the user's previous messages, and even call an Action to fetch some data. Finally, you can review Analytics later to see how that interaction went. Copilot Studio provides an end-to-end toolkit to design, power, and refine such experiences.

Generative Orchestration: The AI Planner in Action

One of the most powerful features of Copilot Studio is generative orchestration.

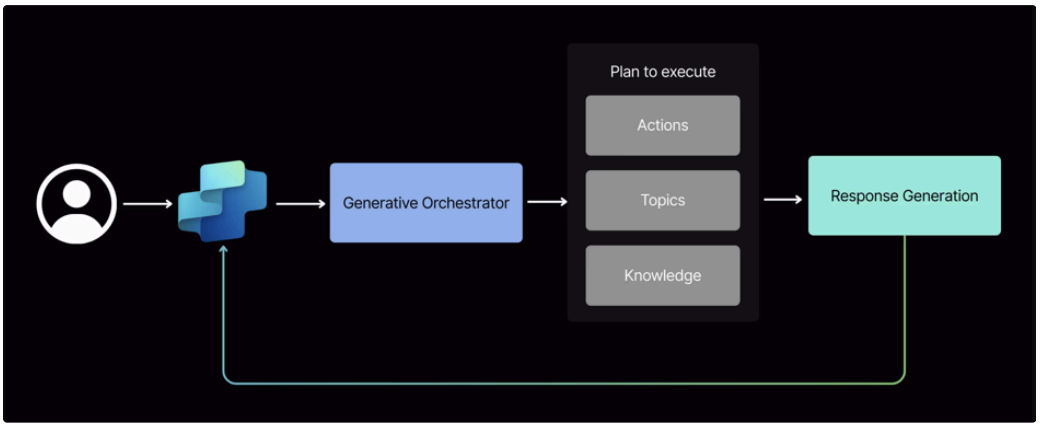
This is an AI-driven orchestration layer (often called the *planner*) that changes how the agent decides what to do when a message comes in.

In traditional chatbots (and in Copilot's "classic" mode), the agent would try to match the user's message to a single topic based on trigger phrases or keywords, then follow that topic's script. Generative orchestration, on the other hand, uses a large language model (LLM) to understand the user's request in context and dynamically plan the best way to respond.

Think of the generative orchestrator as an intelligent coordinator living inside your agent. Every time a user says something (or an event triggers the agent), this AI planner asks: *"What is the user really asking, and what combination of my tools (topics, actions, knowledge) can best fulfill that?"* **Instead of simply picking a pre-written topic by keyword match, it can reason about the user's intent and even break down complex queries into multiple steps if needed.**

When you enable Generative Orchestration, you unlock the power of an LLM to guide interactions dynamically. Rather than relying solely on predefined trigger phrases or static paths, the LLM becomes responsible for:

- **LLM Intent recognition**
- **LLM Entity extraction**
- **Dynamic Chaining**
- **Knowledge invocation**
- **Conversation Context Aware**
- **Follow Up Questions**



User sends a message to the Copilot agent. The Generative Orchestrator evaluates the request and available capabilities. It constructs a dynamic plan: choosing which topics to trigger, actions to invoke, or knowledge to consult. Response is generated based on that plan, regardless of how many components were involved.The response is returned to the user, and if there's a follow-up question, the context-aware orchestrator picks up the conversation seamlessly.

Overall, generative orchestration makes the agent more autonomous and flexible in handling conversations.

It’s powered by a large language model “planner” that interprets user inputs and decides how to achieve the user’s goal using all the pieces you’ve given it (topics, actions, knowledge, etc.).

As the maker, your job shifts more towards *configuring* the right pieces and providing good instructions/descriptions, rather than writing linear scripts for every scenario.

The better you describe your topics and actions, the better the AI planner can utilize them appropriately.

In the next section, we’ll walk through how exactly the orchestration planner processes a user query step by step.

Throughout this process, the generative orchestrator (the AI planner) is doing a lot of heavy lifting: understanding intent, choosing topics/actions, asking clarifying questions, and composing the answer.

As a Copilot Studio maker, you mostly see the results of this in the testing pane or live chat – you’ll notice the agent might jump between topics or call actions without explicit hard coding.

Copilot Studio provides an “Activity map” or similar debugging view when testing, so you can actually inspect what the agent decided to do at each step. This can be very insightful: you might see that the planner chose Topic X and Action Y for a given user query. If that choice was suboptimal, you can adjust your topic descriptions or add a new topic to handle that case.

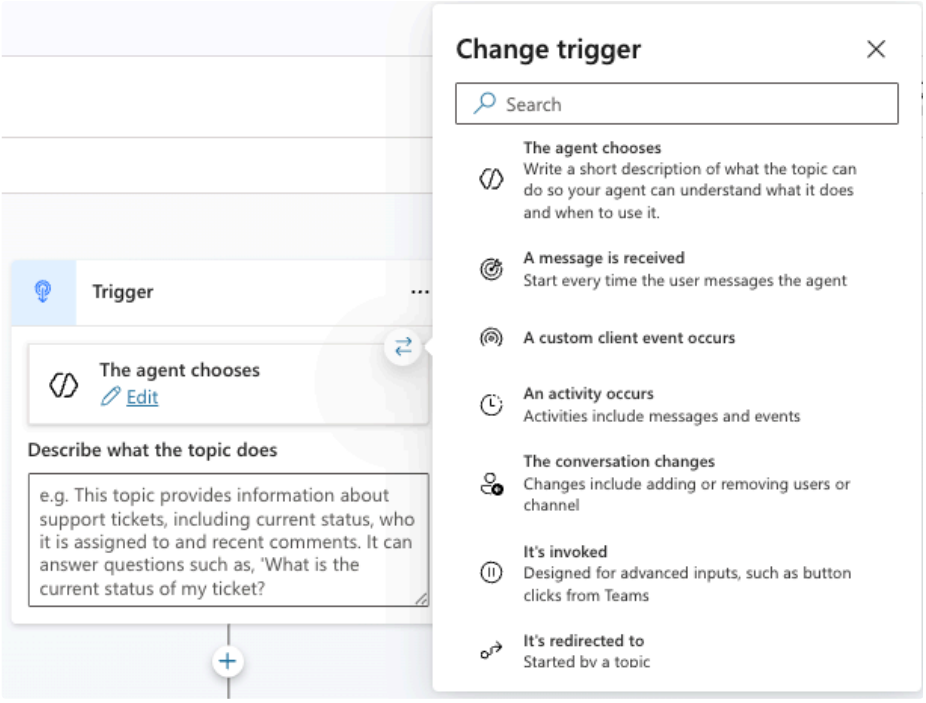
In essence, you configure the building blocks and the AI orchestrator intelligently strings them together during conversation.

Now that we have a clear picture of how generative orchestration works, let’s talk about the special orchestration topic triggers that Copilot Studio provides. These triggers allow you, as the bot maker, to hook into this orchestration process at specific points and insert custom logic or messages. They are powerful for tailoring the behavior of your copilot even further.

Understanding Topic Triggers in Generative Orchestration

Now that you understand the planner's role, let’s explore the three powerful Topic Triggers available when generative orchestration is turned on. **These allow makers to hook into different points of the orchestrator lifecycle to customize how their agent behaves.**

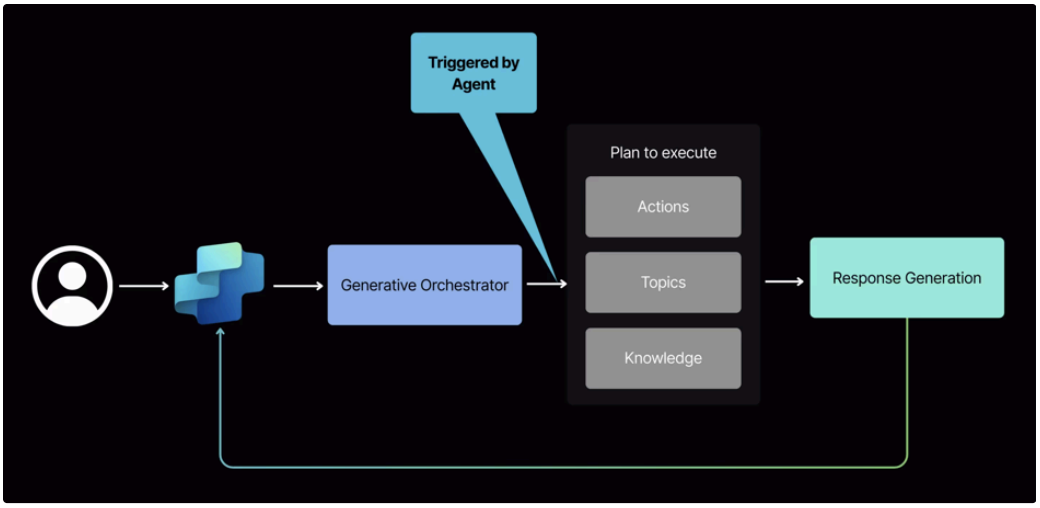
When you're in Copilot Studio and creating a topic, you can hover over the trigger node and click the two arrows icon to see the available trigger options. When the generative orchestration layer is enabled, you'll notice several new trigger types become available for use.



1. **Triggered by Agent**

When it fires: Right after the orchestrator receives the user's input.

Why it's powerful: This allows the orchestrator to evaluate this topic as part of its planning process. You can also provide orchestration instructions to fine-tune when this topic should be invoked.



Where 'Triggered by an agent' fires.

Virtually all standard Q&A or task-oriented topics in a generative agent use the “Triggered by Agent” mechanism.

Anytime you want the bot to handle a particular kind of request (checking store hours, booking an appointment, resetting a password, troubleshooting a product, etc.), you’d implement that logic as a topic and let the AI trigger it.

The maker’s job is to clearly define what the topic does (in description and in its node logic).

The AI’s job is to route the conversation into that topic when the user’s input aligns with it. For example, a Conversation Start system topic (the greeting message when a chat begins) is triggered by the agent automatically at the start of a session.

Other system topics like “End of Conversation” or “Escalate” (to a human agent) might also be internally triggered by the system when conditions are met. All of these can be thought of as agent-initiated triggers.

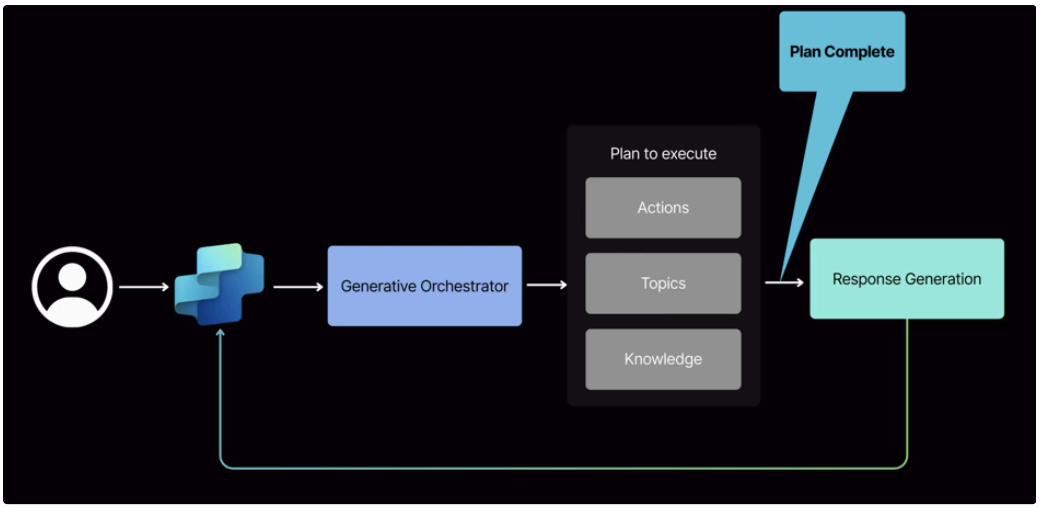
In summary, “Triggered by Agent” is the backbone of generative orchestration – it’s how the AI uses your authored topics as tools to fulfill user needs.

2. **Plan Complete**

When it fires: After the planner has constructed and executed the plan—but *before* the response is generated.

Why it’s useful: A Plan Complete trigger gives you a chance to inject custom logic or interaction at the end of the agent’s action sequence. Some reasons you might use it:

Use case: A great opportunity to introduce middleware logic. For example, you could enrich the planned response, log specific steps, or alter the outcome before the final user-facing response is assembled.



Where 'Plan Complete' fires.

You could use a Plan Complete topic to perform some behind-the-scenes action.

For example, log the conversation outcome to a database, update a CRM record (“case closed”), or call an analytics/event tracking action. Since Plan Complete happens after the main work is done, it’s a good time to fire off any “cleanup” or logging actions without delaying the main answer to the user.

You might use this trigger to append something to the response or adjust it.

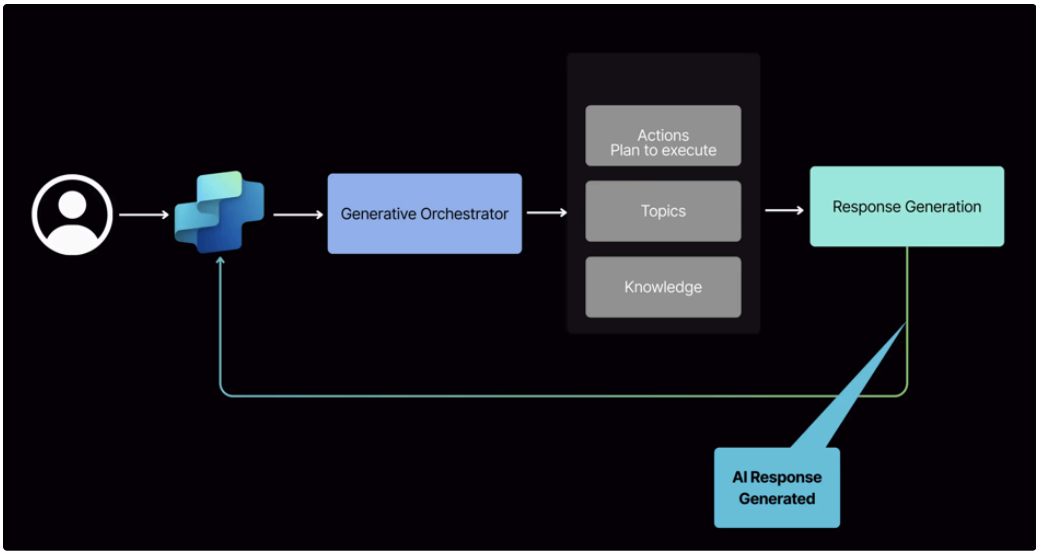
For instance, maybe you have a topic that checks the user’s profile or preferences and then adds a personalized note. Although note that if you want to actually alter the final message content, the next trigger type (AI Response Generated) might be more appropriate.

Plan Complete is slightly earlier in the sequence.

3. **AI Response Generated**

The “AI Response Generated” trigger fires at an even later stage in the cycle, essentially when the AI has generated a response for the user, but before that response is sent out. This gives you one last interception point – after the AI has formulated what it wants to say. If Plan Complete is for doing something after the plan’s done, *AI Response Generated* is specifically for reacting to or modifying the final message.

Why it’s useful: This is the most fine-grained control point you have in the conversation.



Perhaps you want to ensure the final answer doesn't violate some policy or contains certain disclaimers.

With this trigger, you could inspect the generated response (it might be available as an input variable to the topic) and then modify it or log it.

For instance, if the AI happened to produce a URL or a piece of text that you want to redact, you could do that here.

Or if you have compliance requirements (say, the agent sometimes gives financial info and you need to append "This is not financial advice."), you can detect the context and append such a statement.

Suppose your agent's primary language is English (as generative orchestration currently works in English best), but you have users who prefer Spanish.

You might let the AI generate the answer in English (since it has the most data for that), and then use an AI Response Generated trigger to call a translation **action** or service to convert that answer to Spanish before sending.

In this way, the user still gets a response in their language. This is a bit advanced but shows the power of intercepting the final answer.

Building Smarter Copilots with Orchestration Triggers

Generative orchestration in Copilot Studio unlocks a new level of intelligence and flexibility for chatbot agents.

The AI planner can understand user requests more deeply, chain together various capabilities (topics, actions, knowledge), and maintain context to carry on natural conversations.

By leveraging the **orchestration topic triggers** – Triggered by Agent, Plan Complete, and AI Response Generated – you as a maker can blend this AI-driven behavior with your own custom logic and personal touches at just the right moments.

Using these triggers wisely can make your copilot feel both smart *and* tailored.

The generative AI provides the smarts to handle unpredictable user questions, while your triggered topics provide the guardrails and custom actions that align the bot with your business needs and user experience goals. For example, you can have a friendly greeting at conversation start, detailed multi-step answers in the middle, a consistent sign-off at the end, and proactive check-ins – all orchestrated seamlessly.

In a friendly analogy, think of the generative orchestrator as the talented chef, and these triggers as the seasoning and timing you control. The chef (AI) will cook up the main dish (the answer to the user's query), but you can decide when to taste it, when to add a pinch of salt (extra info), or when to serve the next course. Together, it results in a well-rounded meal (conversation) for the user.

As you build and refine your agent, experiment with these triggers.

Start simple: let the agent trigger topics on its own and see how it performs.

Then maybe add a Plan Complete topic to inject a common behavior across all interactions.

Finally, if needed, use an AI Response Generated trigger for fine-tuning.

Always test the conversation to ensure it flows naturally – the Copilot Studio test chat and analytics will be your friends for this. You'll quickly see the value in being able to intercept the conversation at these key points.

By understanding and utilizing generative orchestration and its topic triggers, you can create a chatbot copilot that not only answers questions accurately, but also feels more engaging, proactive, and personalized.

It's the combination of powerful AI and your creative design that leads to the best outcomes. So go ahead and try these features in Copilot Studio – with generative orchestration and topic triggers in your toolkit, you're well on your way to building a smarter copilot that delights your users!