

Steps to recreate the scenario:

Prerequisites:

To create this scenario, follow these steps:

1. **Set up a bot on Azure AI Studio:** For this specific example, the [Chat Playground](#) has been used to create and test a bot. You can read more about it [here](#).
2. **Set up Microsoft Power Platform:** Create a M365 Developer account. You can read more about it [here](#).
3. **Set up SAP connection in Power Automate:** To connect Power Automate flows to your SAP system and make use of the SAP ERP connector for seamless data retrieval and updates, you must establish a connection between the Power Platform and your SAP system, below are the resources on how to do that.
 - [Power Platform + SAP - Installing the On-premises Data Gateway](#)
 - [Power Platform + SAP - Calling "Remote Function Call" \(RFC\) in SAP \(youtube.com\)](#)
4. **Set up / Sign up for Copilot studio.** Learn how to get started with it [here](#).

Now that all the prerequisites are in place, we can now begin to build the scenario.

Let us begin by getting started with the Copilot Studio:

The first step is to create a new copilot, to do this you can create a new copilot, go to the settings as shown below.

MSFT (default) NV

SAP Order Status Copilot Overview Knowledge Topics Actions Analytics Channels Published 5/21/2024 Publish Settings Test

Your copilot is ready! Here's what's next:

- Add actions so your copilot can do things for you
- Build topics to focus and guide how your copilot answers
- Publish your copilot so others can use it

Details Edit

Name
SAP Order Status Copilot

Description
None provided

Test your copilot

Go to the Generative AI section as shown below and enable dynamic chaining and Generative AI (this is what automatically detects which topic/flow to call based on the user prompt):

S

MSFT (default)

Settings

Copilot details

AI integration tools

Generative AI

Security

Entities

Skills

Languages

Language understandi...

Save

Generative AI

Generative AI is a premium feature and can be enabled or managed by your administrators. [See pricing tiers](#)

You consent to your data flowing outside your organization's compliance and geo boundaries. By proceeding you agree to the supplemental preview terms. [See preview terms](#)

[Learn more about responsible AI at Microsoft](#)

Use AI features in your copilot
Generating responses using AI doesn't guarantee accuracy or relevance.

How should your copilot decide how to respond? [Learn more](#)

☐ Classic - Build topics which are used to respond to user queries, and are matched to the example trigger phrases you have provided (in classic mode, actions can only be called by explicitly adding them to topics).

☒ Generative (preview) - Allow your copilot to use generative AI to identify the most appropriate combination of actions and topics to respond to a user, and provide a more natural conversational experience for end users.

Intelligent authoring with Copilot

Describe copilot topics you need, and Copilot will develop it. Access this intelligent authoring tool in user settings, unavailable in classic copilots.

[Go to user settings](#)

Copilot content moderation ⓘ
(You can override content moderation settings in the node)

High (default)
Copilot generates fewer answers, but responses are mor...

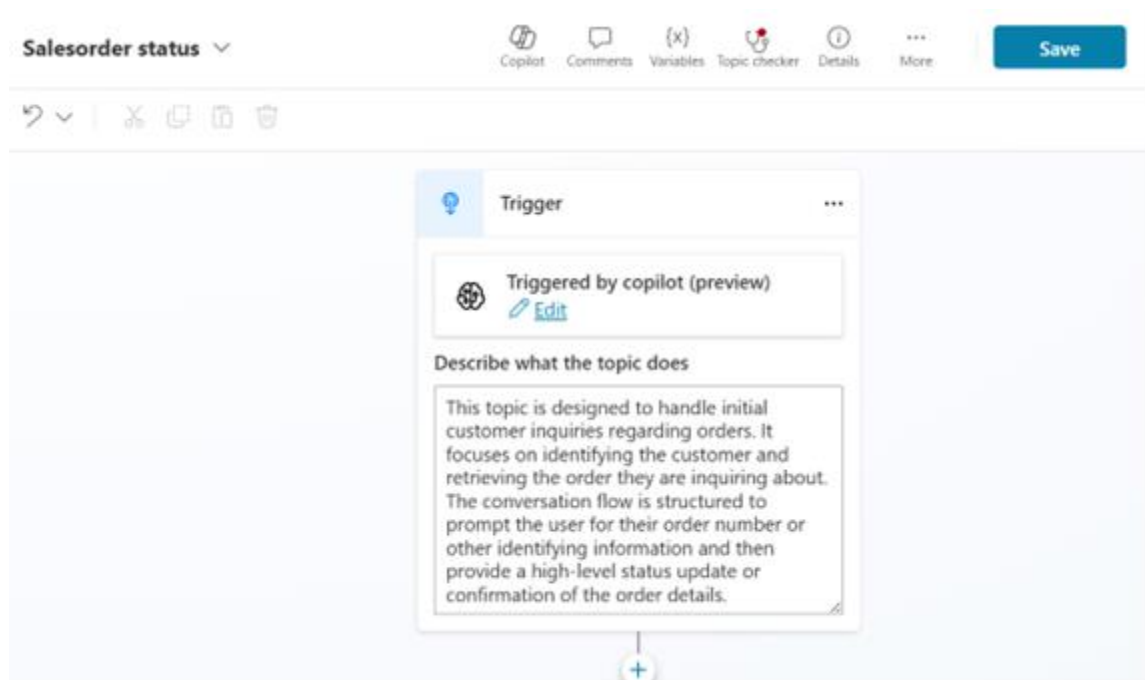
The next step is to create a new topic and the name is – “Sales order” – this is the topic that the salesperson will use to get the order status and information on the order.

The Copilot studio has unique functionality which figures out which topic/functionality to call based on the input text sent to the copilot/bot. This functionality is called “Dynamic Chaining” and uses AI to automatically pick the topic that fits the best based on the description of the topic provided while creating it.. This is essentially the equivalent of automatically knowing which function to called based on a sentence!

Therefore, the first step after creating a new topic is to write a description of the topic. This is what will help the AI identify what this flow is for.

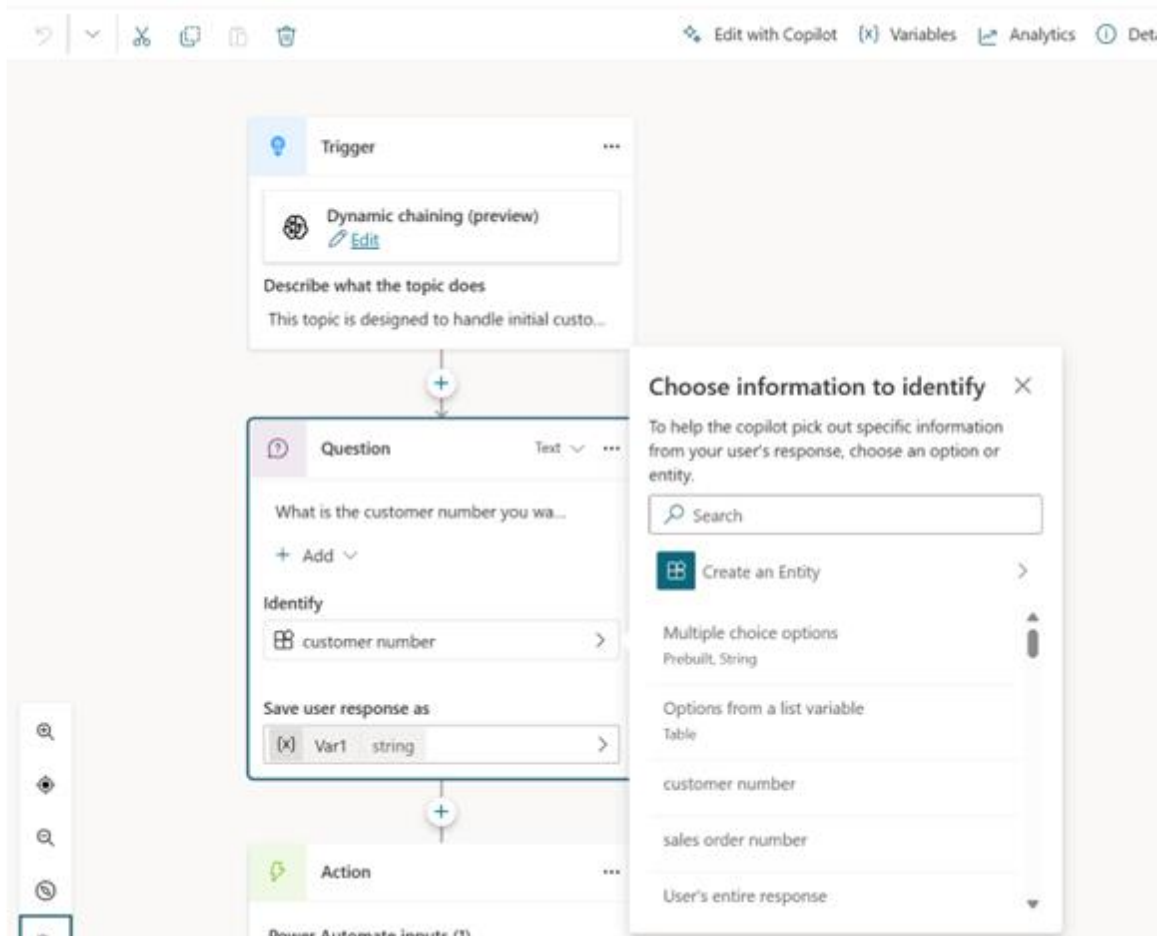
Here is a link to the guidelines to create the description for the topic that will help the AI best understand the intent behind what you want to build: [Use topics to design a copilot conversation - Microsoft Copilot Studio | Microsoft Learn](#)

The topic will be the one responsible for getting the order status for a customer. Here is the image of the topic description (Fun fact- Copilot was used to write the description)

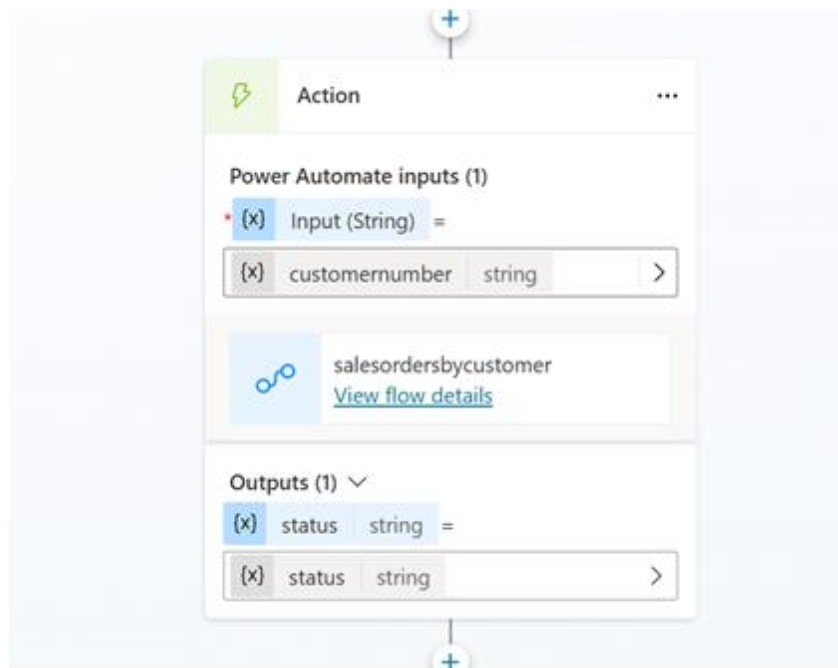


The next step is to add a node to ask a question to the user about what customer number they want this information for.

After the user responds, we want to identify the customer number from the user response, therefore in the question node, we will create an entity with the regex (regular expression) matching a customer number (this can be changed to whatever the most generalized pattern is for your system).

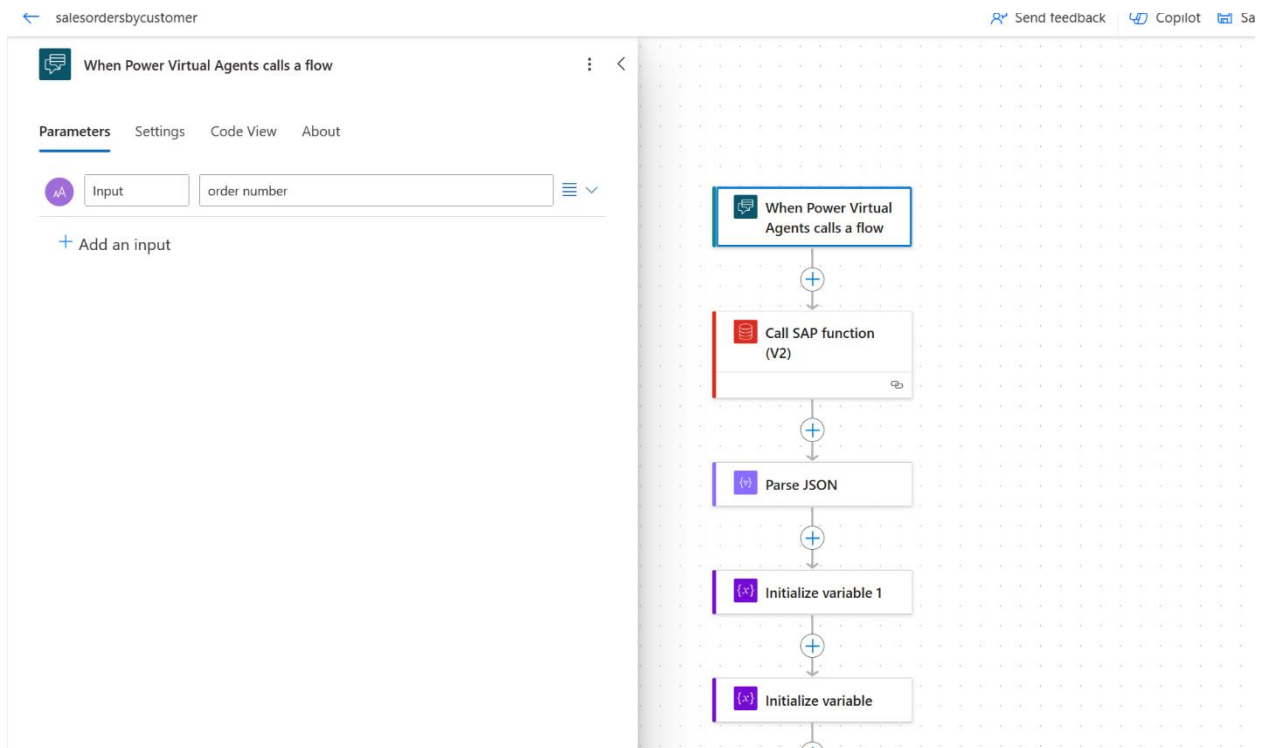


The next step is to now call an action and create a Power Automate flow that gets and parses this information from the SAP system.



Below is the description of the first Power Automate flow to get all the orders for each customer with each step explained.

The input to the Power Automate flow will be the customer number extracted from the input from the user.



The next step will be to call the SAP BAPI (BAPI_ SALESORDER_GETSTATUS) for this you must provide the customer number as well as the sales organization number. You would also have to enter your SAP server credentials (as shown in the SAP set up videos linked above in the perquisites).

← salesordersbycustomer • Published Send feedback Copilot

Call SAP function (V2)

Parameters Settings Code View Testing About

SAP System *

RFC Name *

BAPI_SALESORDER_GETLIST

Advanced parameters

Showing 2 of 3 Show all Clear all

RFC Group Filter

Auto Commit

No

When Copilot Studio calls a flow

Call SAP function (V2)

Parse JSON

Once you call the SAP BAPI, you will get a JSON response. This response needs to be parsed and stored so that we can use the information to perform further tasks, To do this, you must provide a sample JSON. The JSON is available on the GitHub with the exported Power Automate flow.

Parse JSON

Parameters

Settings

Code View

About

Content *

SALES_O...

Schema *

```
1
{
  "type": "array",
  "items": {
    "type": "object",
    "properties": {
      "SD_DOC": {
        "type": "string"
      },
      "ITM_NUMBER": {
        "type": "string"
      },
      "MATERIAL": {
        "type": "string"
      },
      "SHORT_TEXT": {
        "type": "string"
      },
      "DOC_TYPE": {
        "type": "string"
      },
      "DOC_DATE": {

```

When Copilot Studio calls a flow

+

Call SAP function (V2)

+

Parse JSON

+

Initialize variable 1

The next step is to initialize two variables – one called “allorders” which have the type “array”, and its values will be the body of the JSON we retrieved previously.

Power Automate

Search

← salesordersbycustomer • Published

{x}

Initialize variable 1

Parameters

Settings

Code View

About

Name *

allorders

Type *

Array

Value

{v}

Body ×

The other variable is the string variable we will use to store all the parsed fields/information from the sales documents we need to show the user called "orderdetails."

Power Automate Search

← salesordersbycustomer • Published

{x} Initialize variable

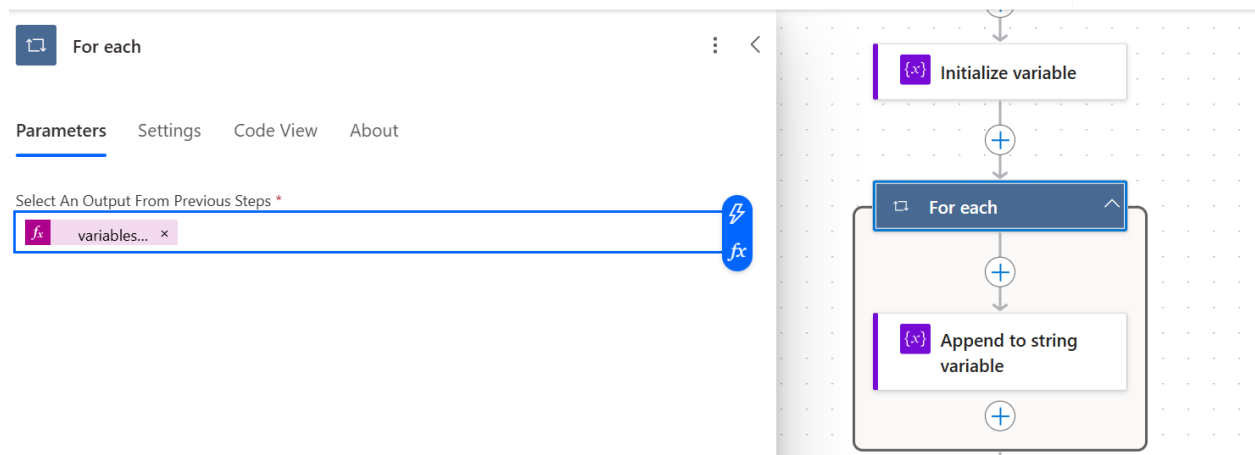
Parameters Settings Code View About

Name *
orderdetails

Type *
String

Value
Enter initial value

Next, we need to loop over all the sales documents found from the customer, go over each and then take all the fields we need from it and paste it in the response. For this, we will use the for each loop and go over the variable "allorders".



Inside the foreach loop, we will append the fields as shown below to the variable "orderdetails".

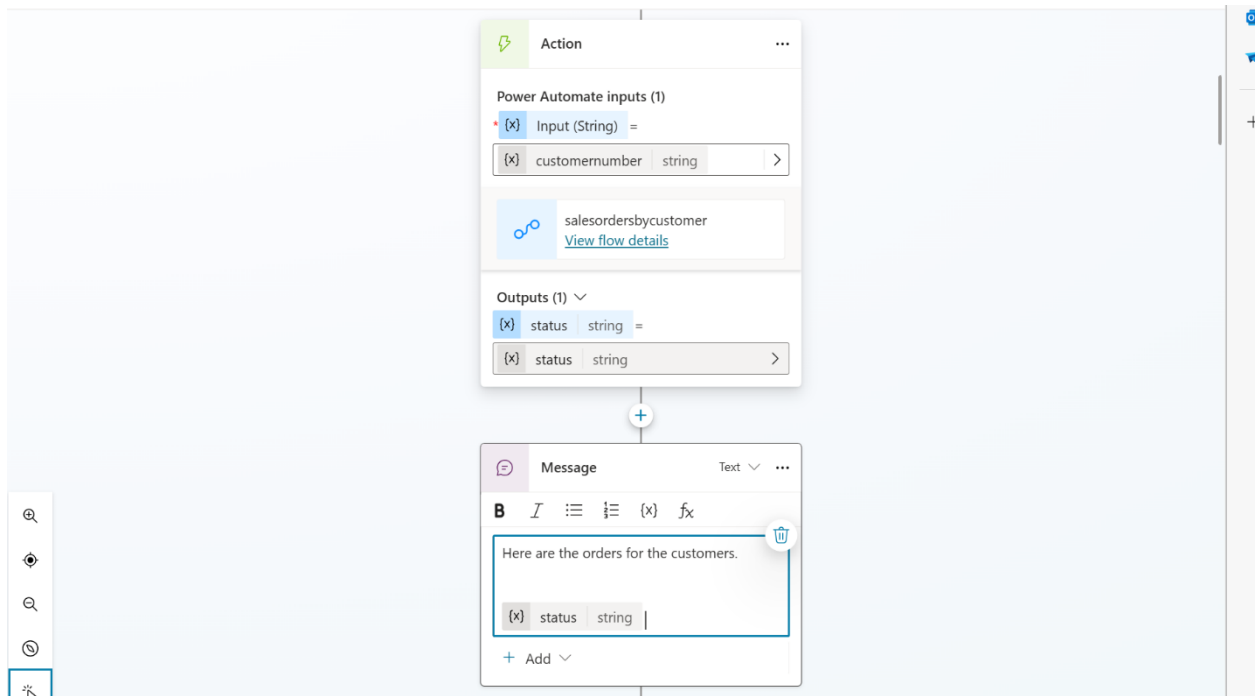
You can add these dynamic fields from the dynamic variables shown below.

The screenshot shows the 'Append to string variable' step in a Power Automate flow. The step is titled 'Append to string variable' and has a purple icon with a curly brace and an 'x'. The 'Name' field is set to 'orderdetails'. The 'Value' field is a dynamic expression containing the following fields: 'PO#:', 'PURCH_NO : Order #', 'SD_DOC : Product', and 'SHORT_TEXT'. A tooltip above the 'Value' field reads: 'Enter the data from previous step. You can also add data by typing the \'/ character.' The flow is part of a larger process named 'salesordersbycustomer' which is 'Published'. The flow steps visible in the background are: 'Call SAP function (V2)', 'Parse JSON', 'Initialize variable 1', and 'Initialize variable'.

Once the loop has completed and all the information regarding the documents has been appended to the string variable "orderdetails," we will now send that variable back to the Copilot Studio with all the orders for that customer.

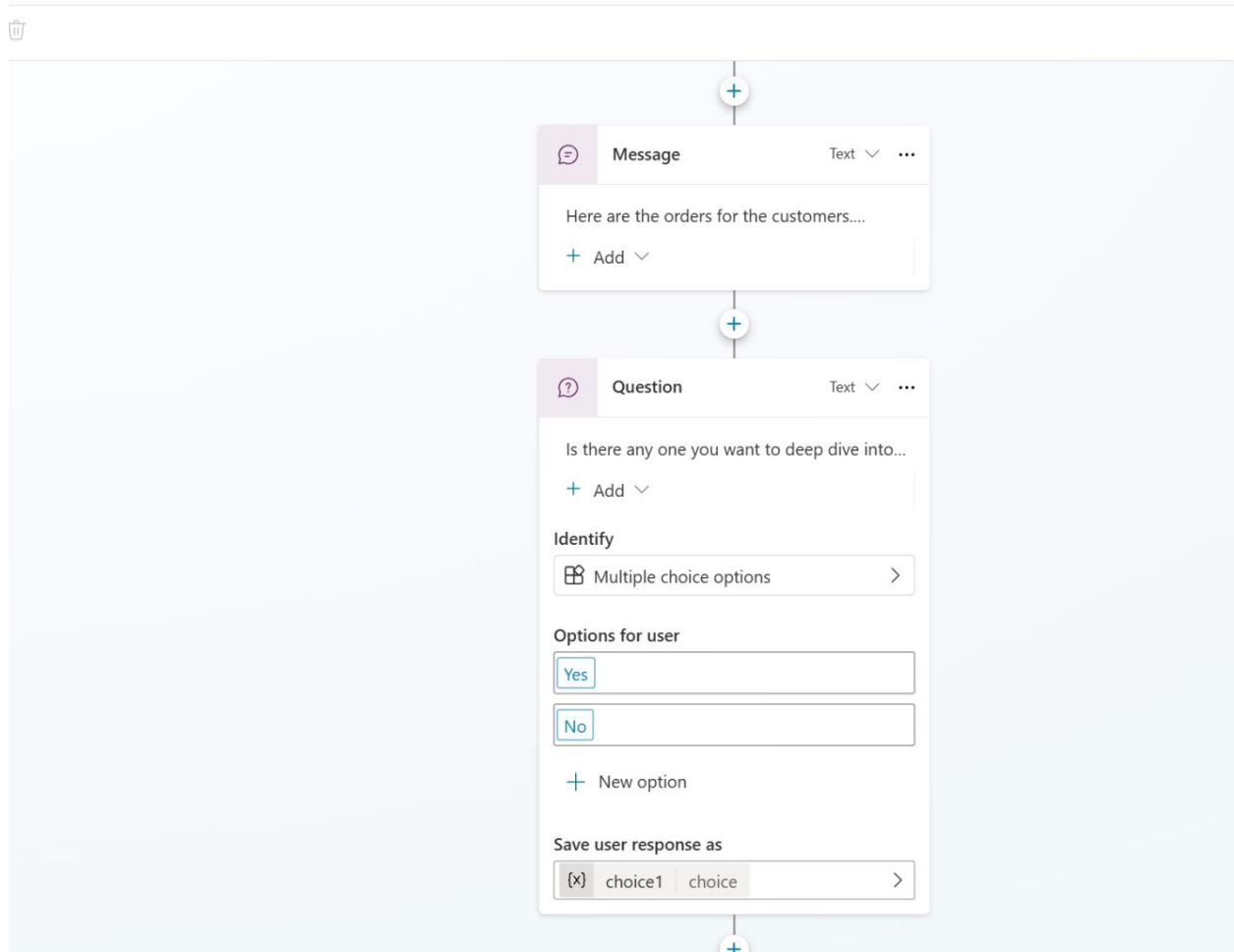
The screenshot shows the 'Return value(s) to Power Virtual Agents' step in a Power Automate flow. The step is titled 'Return value(s) to Power Virtual Agents' and has a dark blue icon with a speech bubble. The step is part of a larger process named 'salesordersbycustomer' which is 'Published'. The flow steps visible in the background are: 'Call SAP function (V2)', 'Parse JSON', 'Initialize variable 1', and 'Initialize variable'. The 'Parameters' tab is selected, showing a list of parameters: 'status' and 'orderdet...'. The 'Add an output' button is visible at the bottom.

Now back in the Copilot Studio we will append the response returned by the Power Automate flow as shown like this.

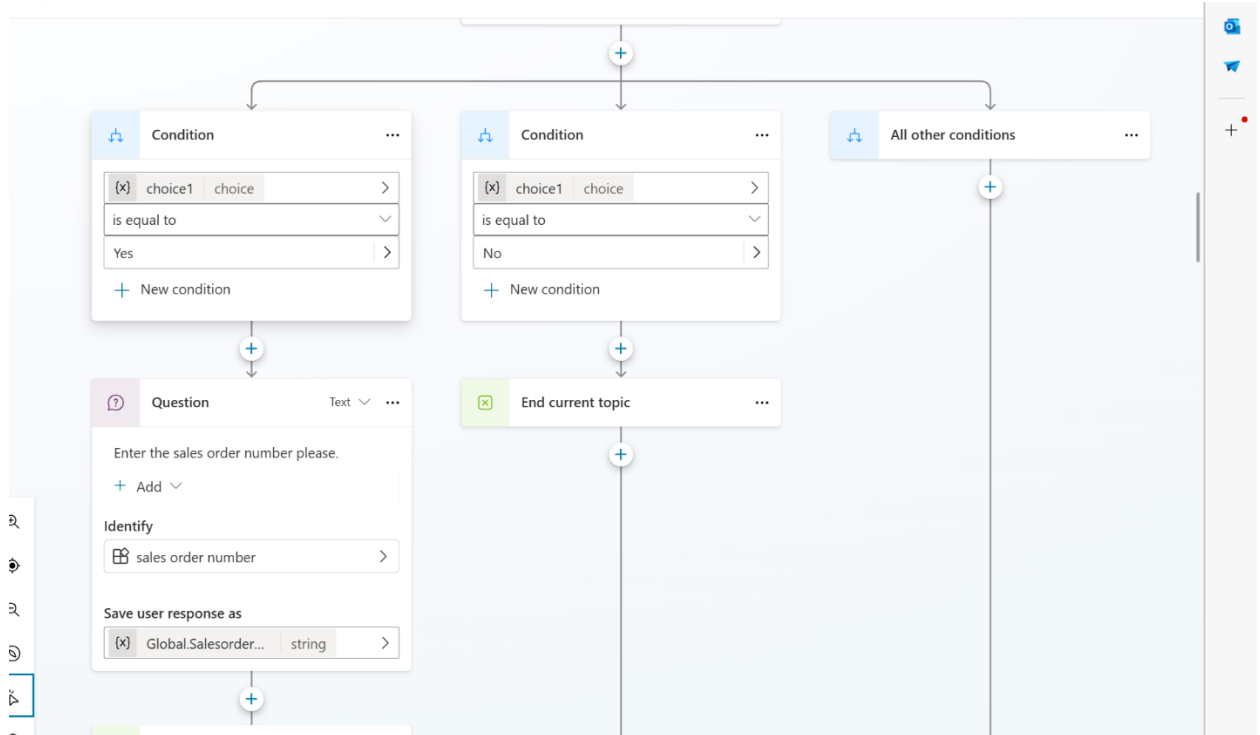


Once we have shown the salesperson all the customer's sales orders, they can now act on it and deep dive into the sales order they want to look into.

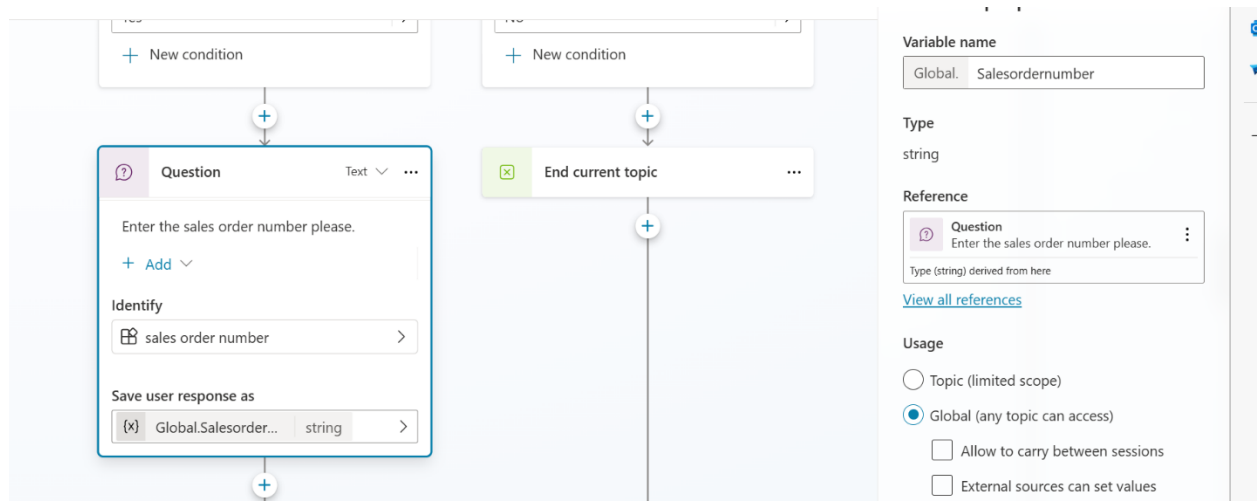
Note : In this demonstration one such action that the salesperson could take has been shown. The action we have selected for this demo is asking the salesperson if there is any particular problematic order they want to dive into and confirm the status of. However, you could create any such flow based on what you would like the functionality to be.



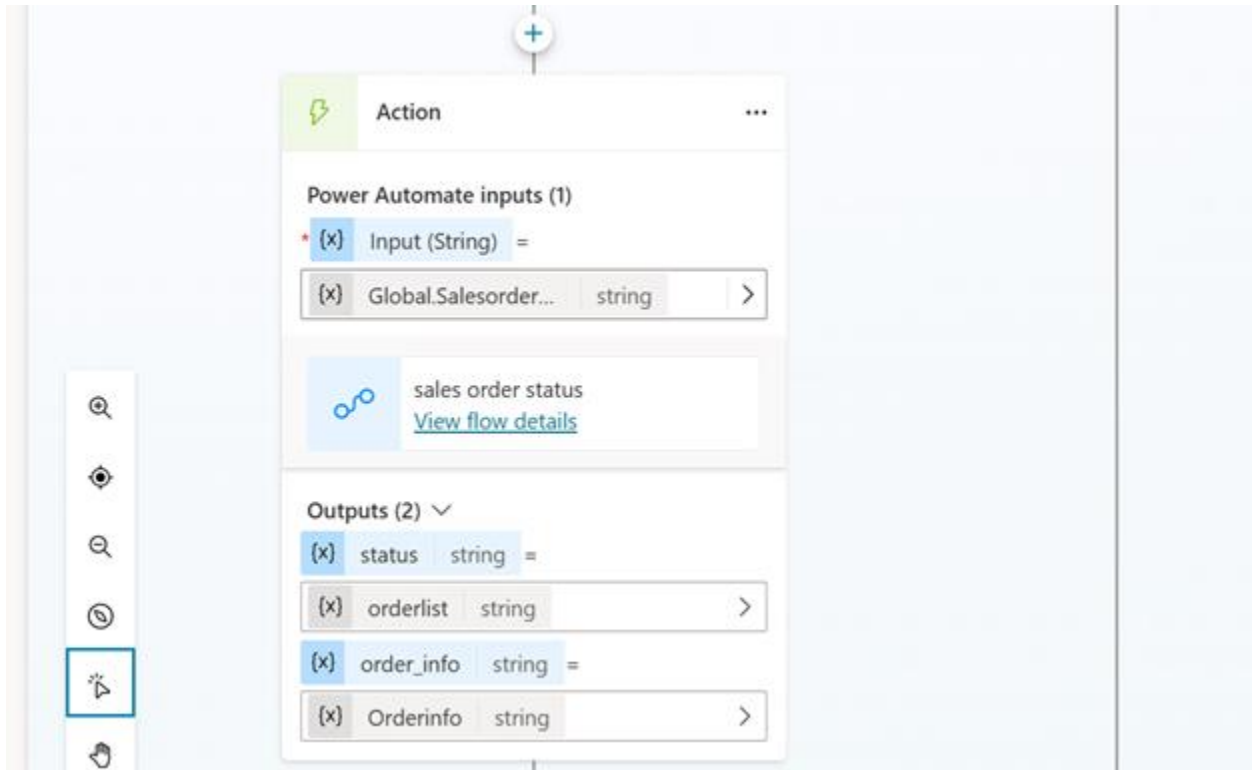
We will create a question node and ask the salesperson to enter the order number in the Copilot studio flow as shown and we will once again create a new entity with the regex in the format for the SAP sales order number (the same way we did it for the customer number above).



The sales order number the customer wants to know about will be a global variable as this may be valuable information across topics. You can make it a global variable by creating a new variable and modifying it in its settings.



Once we have the sales order number that the customer wants to use, we will now dive deep into this sales order and find out about its status from the SAP system – whether it is delayed or not.

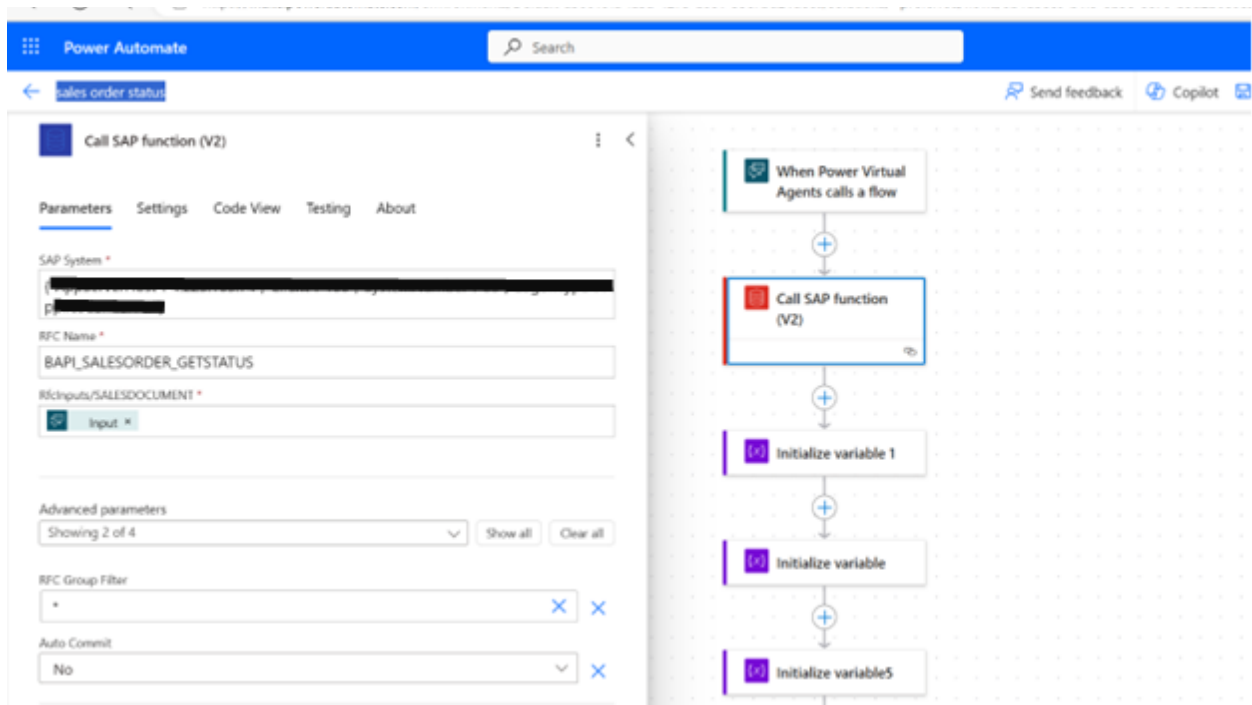
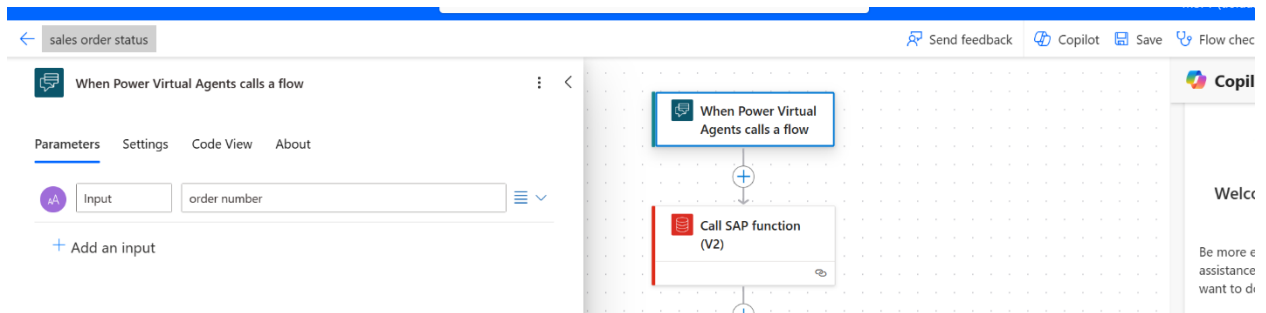


Therefore, the next Power Automate flow we need to call is one that gets the order status for the order the customer mentioned and returns a response on whether the order is delayed or on time.

The input to the flow will be the sales order number.

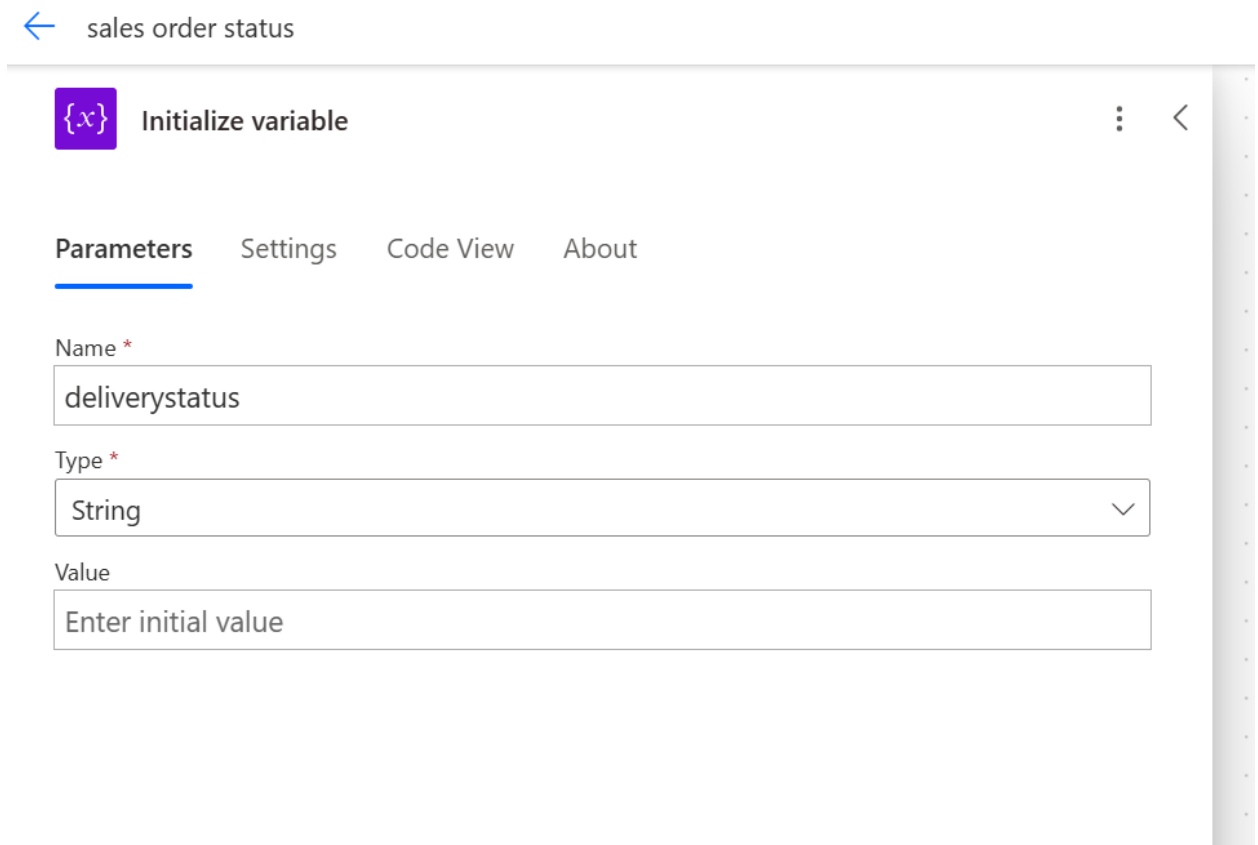
Here are the steps for this Power Automate flow "sales order status":

We will first take the order number as an input to the Power Automate flow, then next we call the BAPI_SALESORDER_GETSTATUS – this is to get the order status information for the sales order.



Next, we initialize three variables.

“salesdocs” – of type array and stores the sales document response from the SAP system.



"deliverystatus" of type string – which will store the delivery status of the order.

← sales order status

{x}

Initialize variable2

Parameters

Settings

Code View

About

Name *

orderinformation

Type *

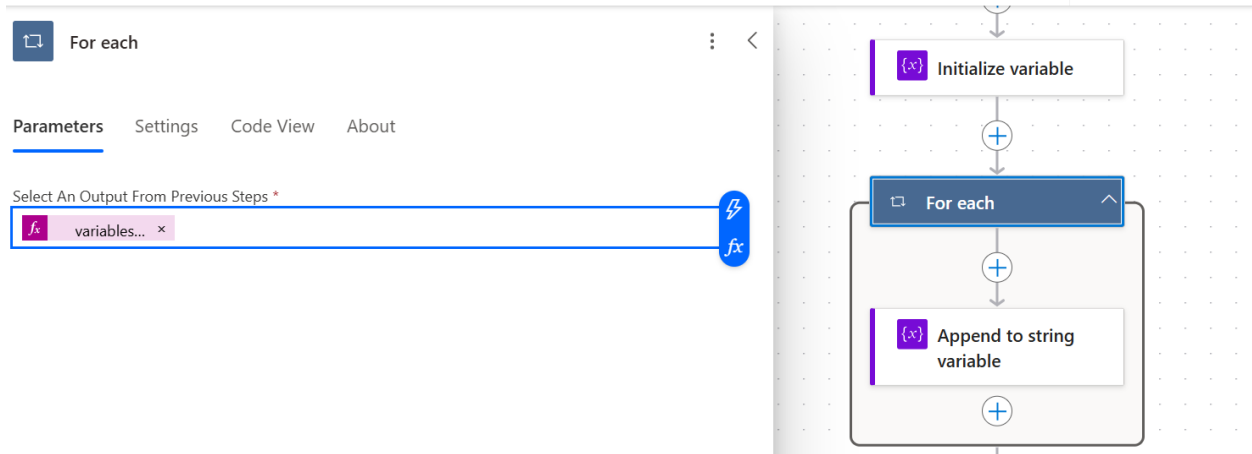
String

Value

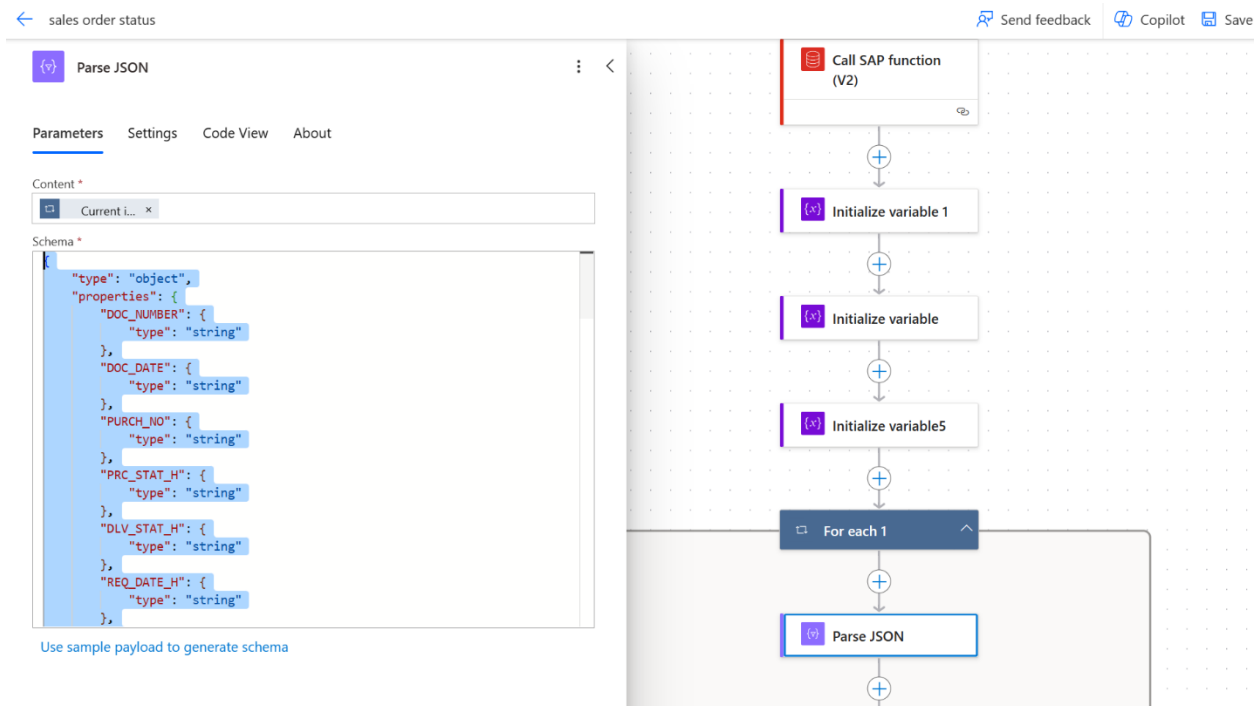
Enter initial value

And the variable "orderinformation" which will store other information about the order.

Next, we need to create a for loop for each that would parse all the sales documents stored in the variable "salesdocs" (Sometimes an order has > 1 materials and therefore has two documents associated with the same order number). From the dynamic variables, pick the "salesdocs" variable to loop over.

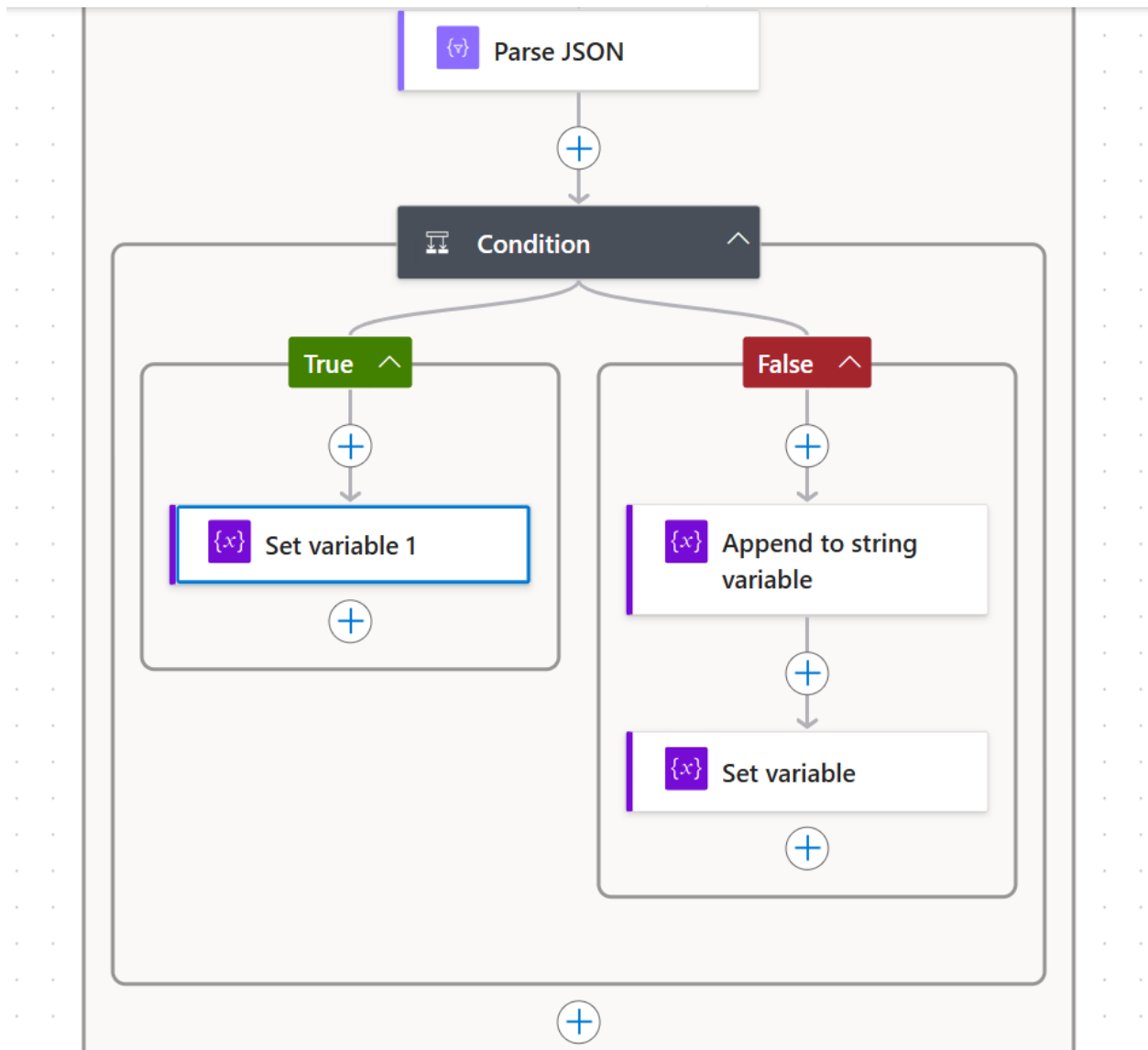


Next, like the previous Power Automate flow, we must parse the JSON response.

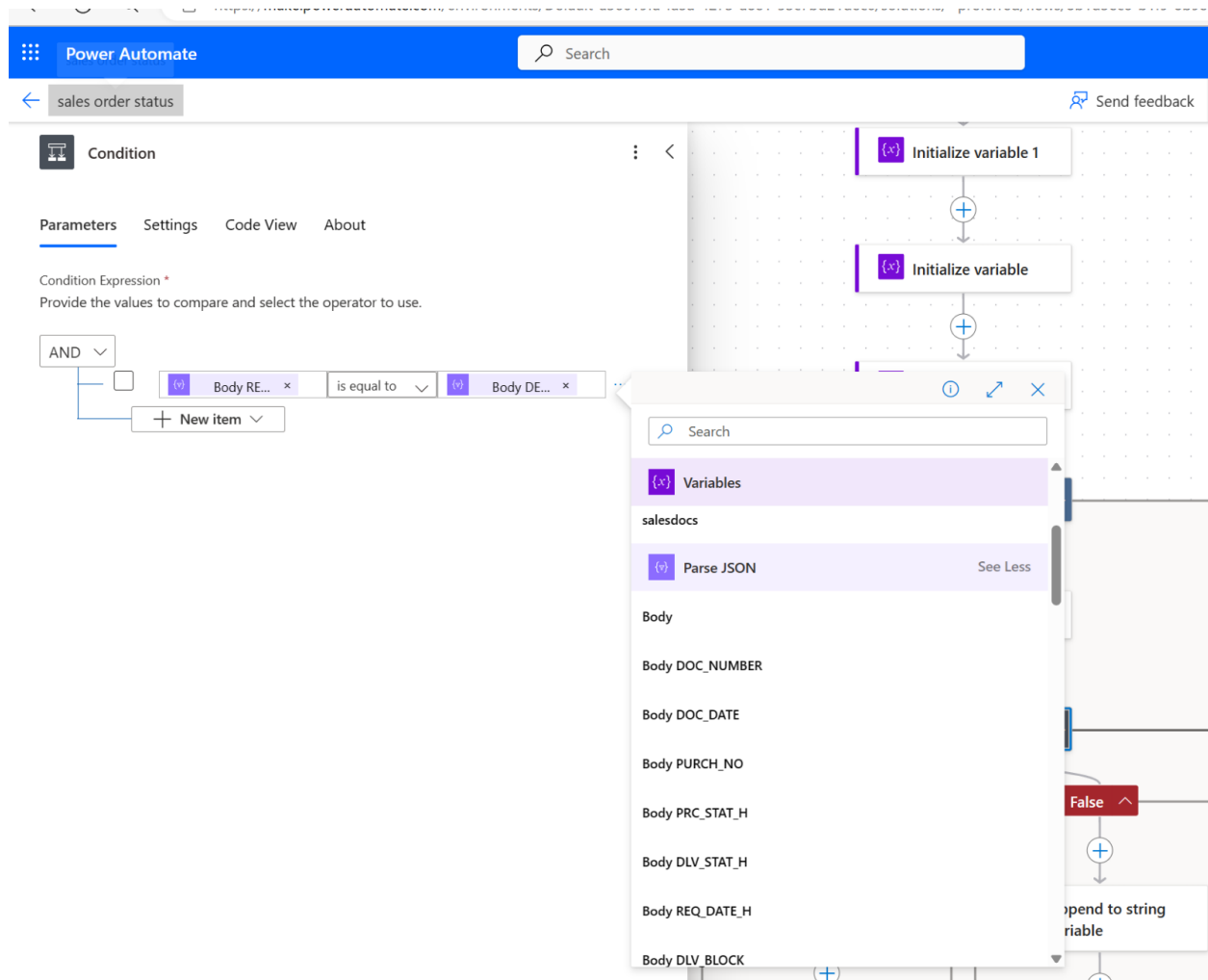


The JSON used for this is available in the GitHub with the exported Power Automate flow.

Next to check if the order is delayed, we will check if the delivery date of the order is same as the requested date if it not, then the order is delayed.



To include this logic in the Power Automate flow, we will add a condition that the field Body.REQ_DATE is EQUAL to DELIV_DATE.



If this condition is true:

The text: "on time" would be appended to the variable "deliverystatus".

If the condition is not true and the order is delayed, the variable "deliverystatus" will be set to "delayed" the text as shown below will be appended to the variable "orderinformation".



Set variable



Parameters

Settings

Code View

About

Name *

deliverystatus



Value *

delayed



Append to string variable



Parameters

Settings

Code View

About

Name *

orderinformation

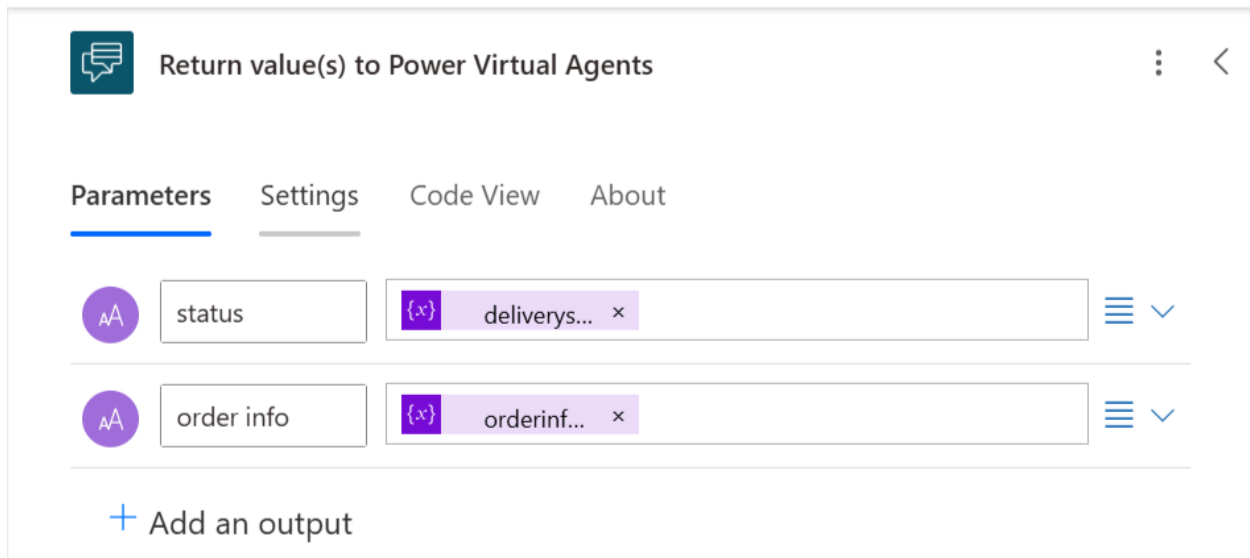


Value *

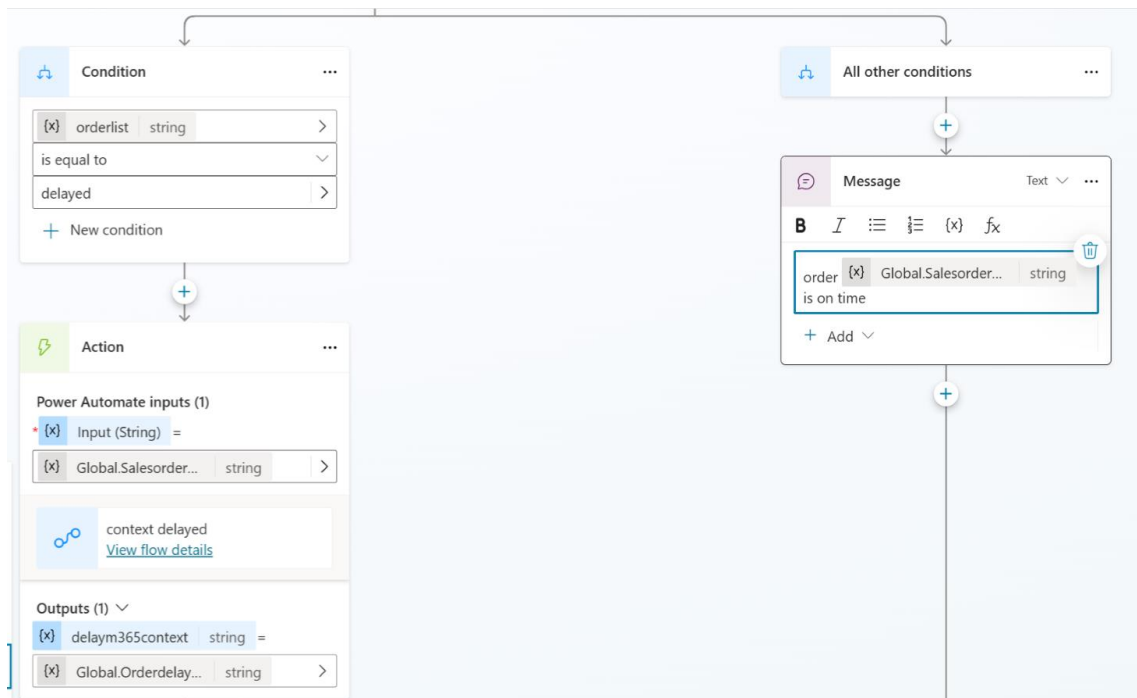
Customer order #  Body DO... × for  Body RE... ×  Body SH... × with
a desired delivery date of  Body RE... × is still not delivered.

The values can be found in the flow on the GitHub link.

Both the variables "deliverystatus" and "orderinformation" will be returned to the Copilot Studio as shown in the Copilot Studio.



Now in the Copilot Studio we will deal with both cases, whether the order is delayed or not but adding a condition node.



If the order is delayed, the next step is to harness the power of AI and the M365 data for conversation context (emails and Teams' chats on why the order is late). This is done by querying the M365 graph for Outlook emails and Teams' chats related to the order and then passing those emails and messages to Azure OpenAI. Azure OpenAI will then summarize the content and finally give the user direct insights which saves the salesperson the time they would have taken to go over all the messages and emails manually.

Here is the description and steps for that Power Automate flow, the name of this flow is "context delayed":

The input for this flow will be the sales order number.

We will define two variables: "messages" and "emails"

← context delayed

Initialize variable

Parameters Settings Code View About

Name *
messages

Type *
String

Value
Enter initial value

When Power Virtual Agents calls a flow

Initialize variable

Initialize variable 1

Power Automate

Search

← context delayed

Initialize variable 1

Parameters Settings Code View About

Name *
emails

Type *
String

Value
Enter initial value

When Power Virtual Agents calls a flow

Initialize variable

Initialize variable 1

Next, we will also make use of the Team's connector as shown below, however teams connector does not yet allow you to query specific messages, therefore there will be a bit more processing on that front as shown below, for now we will just fetch all the messages.

Power Automate

context delayed

Get messages

Parameters Settings Code View Testing About

Team *
SAPCHATBOTRIAL

Channel *
General

Connected to Microsoft Teams Default-7e2e2. [Change connection reference](#)

Initialize variable 2

Get messages

Get emails (V3)

Next, we will fetch all the outlook emails related to the order using the input order number as the search query .

The screenshot displays the Power Automate interface. On the left, the 'Get emails (V3)' action is configured with the following parameters:

- Advanced parameters: Showing 7 of 13 (dropdown), Show all, Clear all
- Importance: Any (dropdown)
- Only With Attachments: No (dropdown)
- Folder: Inbox (dropdown)
- Fetch Only Unread Messages: No (dropdown)
- Include Attachments: No (dropdown)
- Search Query: Input (text box)

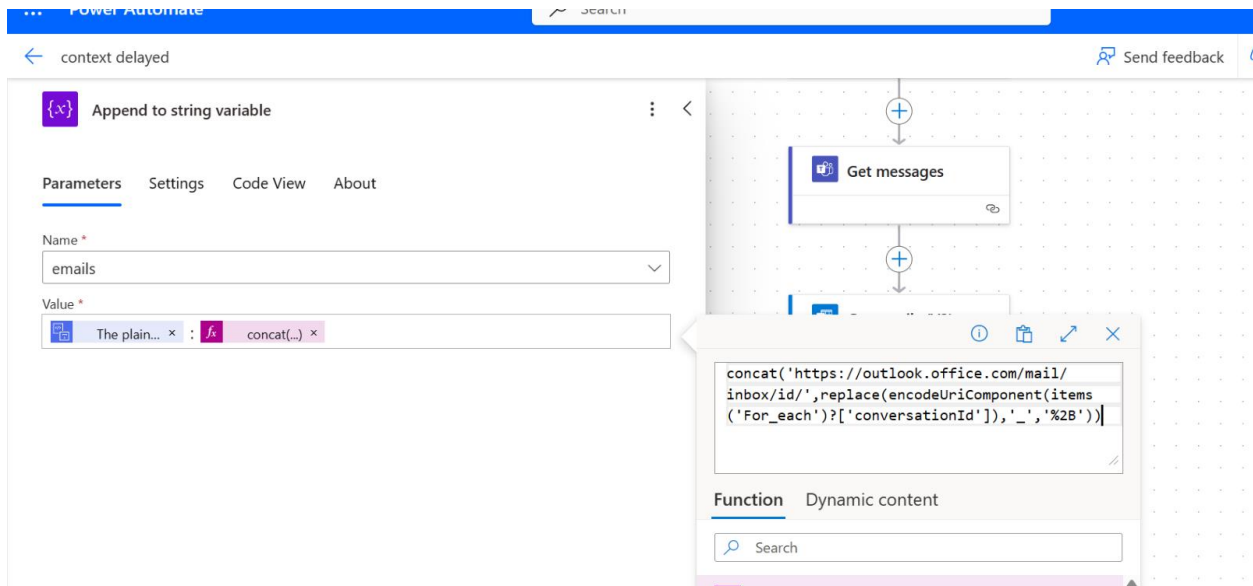
On the right, a flow diagram is shown with the following steps:

- Initialize variable Z
- Get messages
- Get emails (V3)
- For each (loop)
- Html to text
- Append to string

The emails will be appended to the string “emails” variable along with the URL for each email – this is helpful to verify the original source of the email and to ensure that the AI can reference the email it got the insights on the order delay from.

We will create a foreach loop where we will go through each Outlook message, as the outlook email is returned in HTML, we will use the action “HTML to text” to parse the HTML and return plain text in response. The input to the “Html to text” action will be the current item in the foreach loop.

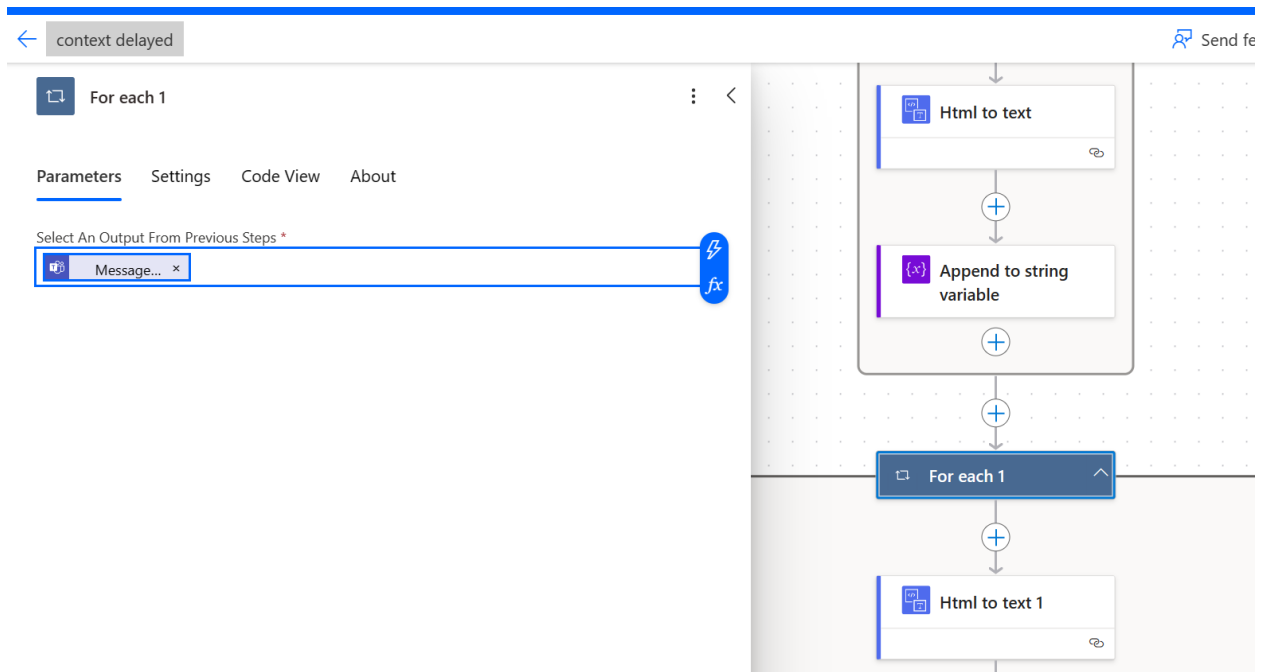
After we get the plain text, we will append it to the email's variable along with the URL of the email, follow the format of the URL as below or refer to the exported flow on GitHub.



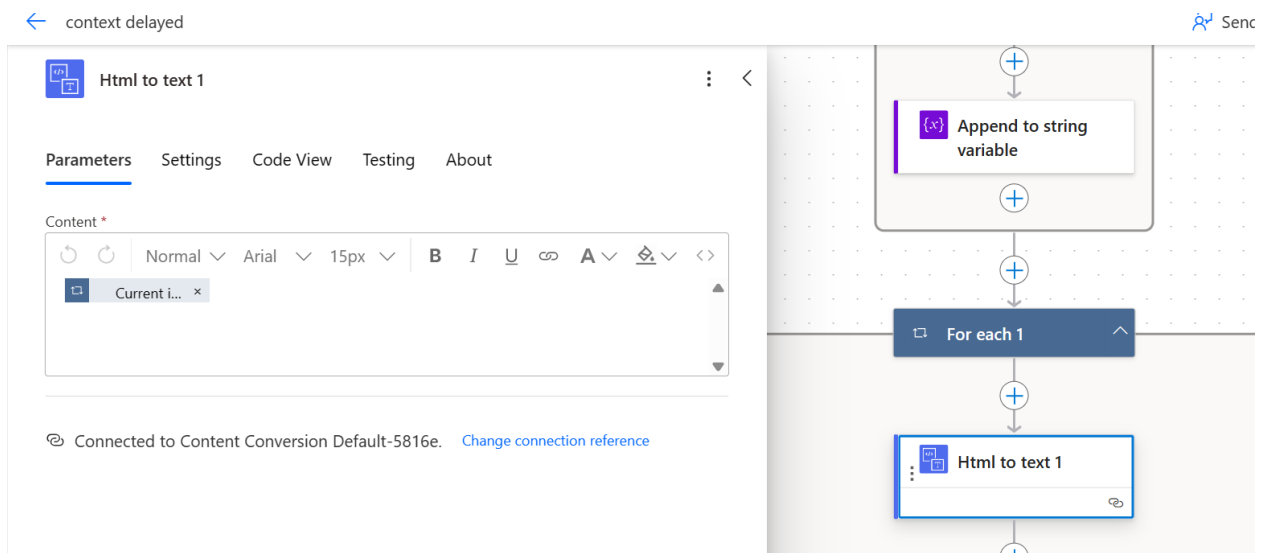
Once we have gotten the emails, we will now work on getting the Teams' messages.

We will create a "For each" action and loop over each message in the for loop, we will check if each message is related to the order by searching for the order number in the Teams message as the Teams connector to fetch messages does not yet have the feature to query for a particular text/string.

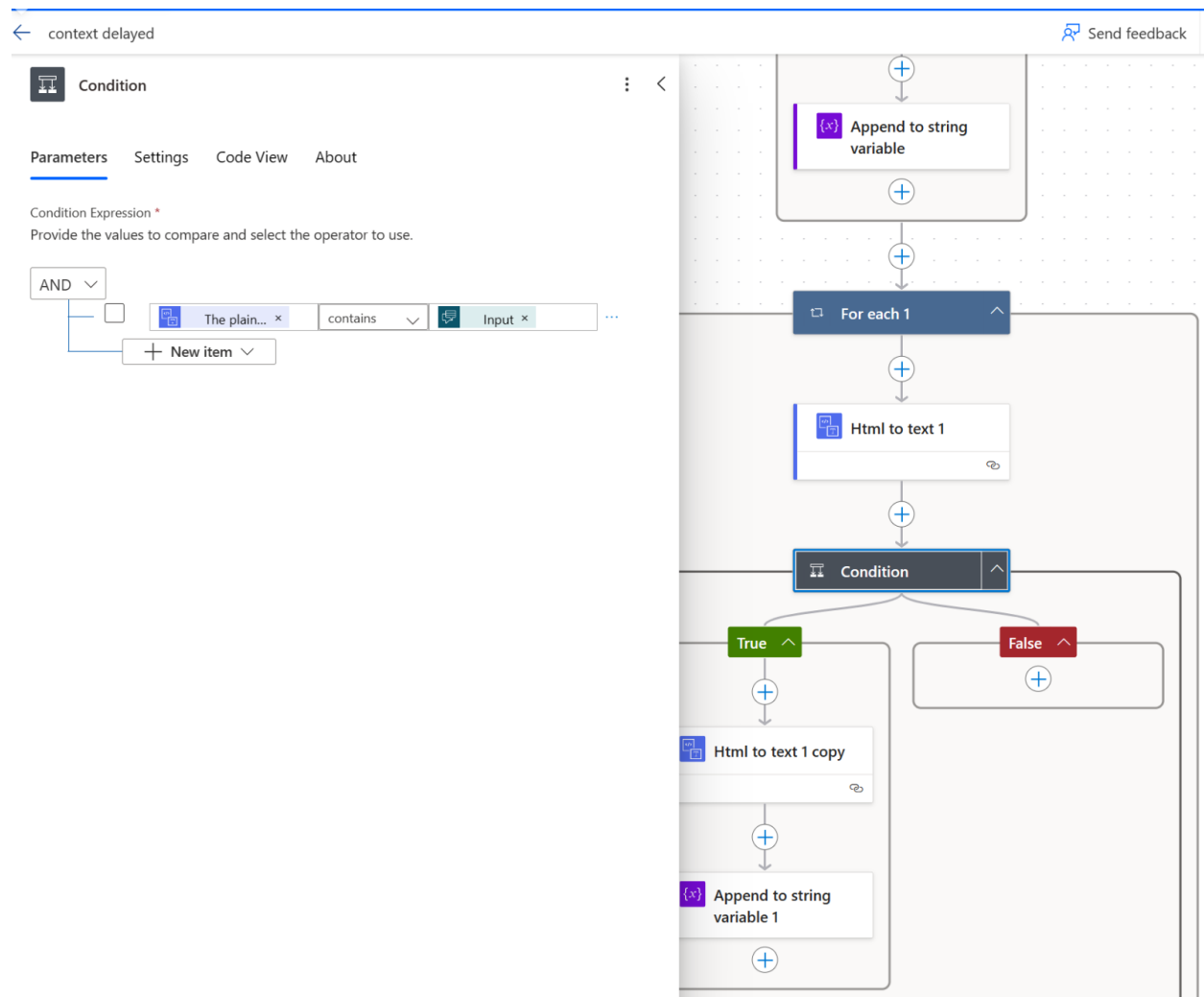
To do this we will loop over all Teams' messages:



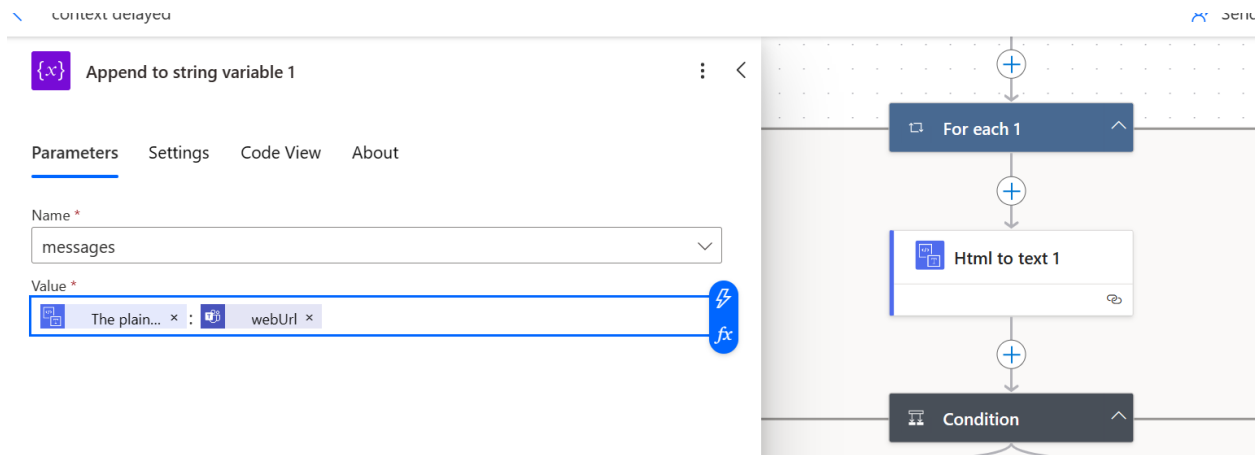
Next we will convert the response from the HTML format to text using the “Html to text” action.



Once we have the Teams message in plain text, we will use the “Condition” action and see if the content contains the order number (gotten from the input variable).



If it does, we will append the message as well as the WebUrl of the Teams message (gotten from the dynamic variables)



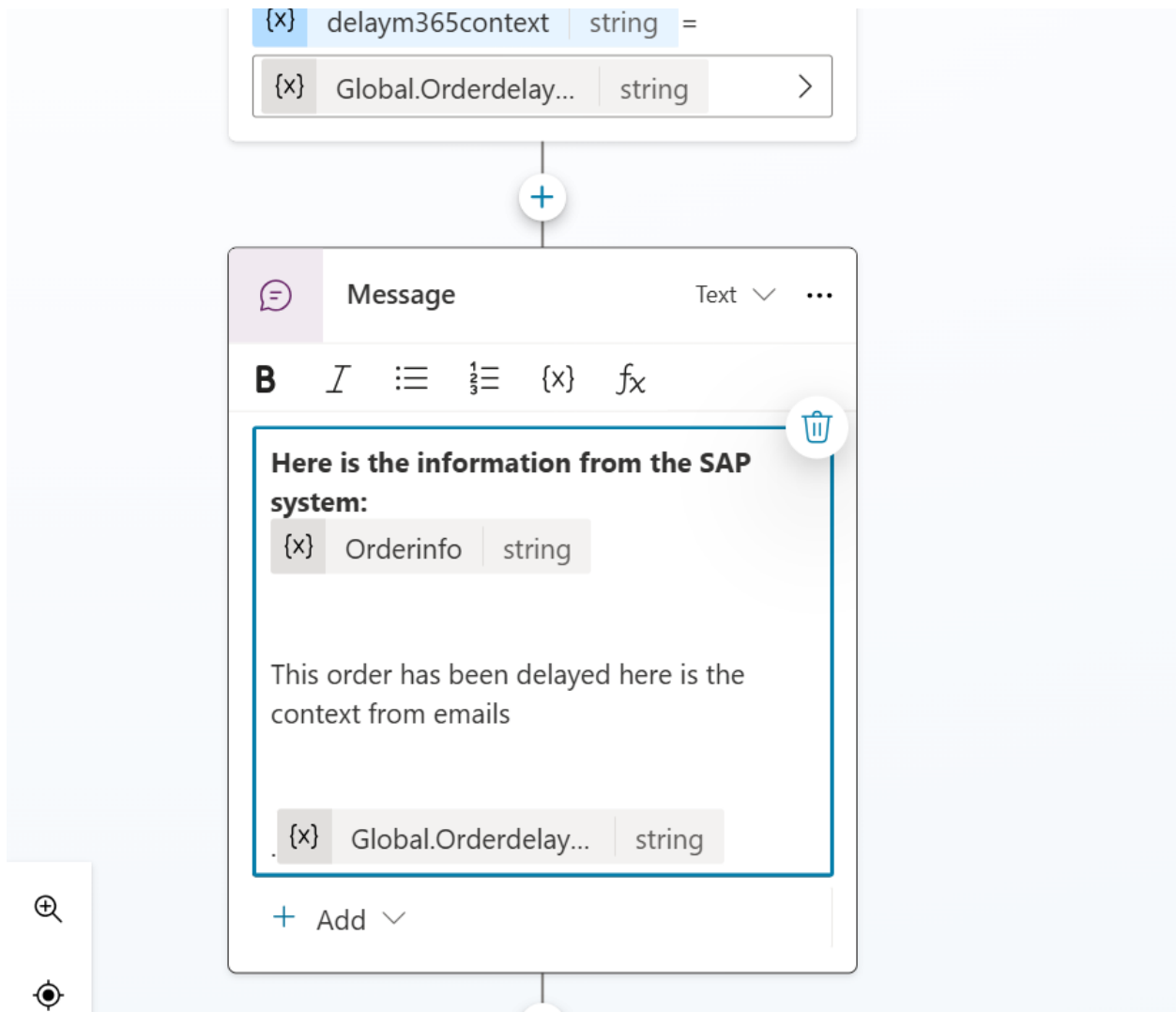
After the relevant team's messages and the URLs are sent to outlook, we will now send this as an input to Azure OpenAI to go through as context and summarize and give the salesperson insights on why the order is late. For this we will make an HTTP request to Azure OpenAI as shown below.

The screenshot displays the Power Automate interface for a flow named "context delayed". On the left, the "Parameters" tab for an "HTTP" action is visible. The "URI" field contains a redacted URL. The "Method" is set to "POST". The "Headers" section shows "content-type" as "application/json" and "api-key" as a redacted value. The "Body" section contains a JSON object with a "messages" array. The "Body" text is:

```
{
  "messages": [
    {
      "role": "system",
      "content": "here are some emails and teams messages with information on a order, summarize mentioning which are teams messages and which are emails the reason for the delay: here are the emails: [emails] and here are the messages: [messages], mention the email as well as the link to the email and if there are no teams messages or no emails mention that Make the URL embedded in a short form text call it link. don't add any of your personal inferences. Also in terms of timestamp, the emails and messages are from new to old. add a blank between each email/team message and seperate them by headings of team message and emails respectively"
    }
  ]
}
```

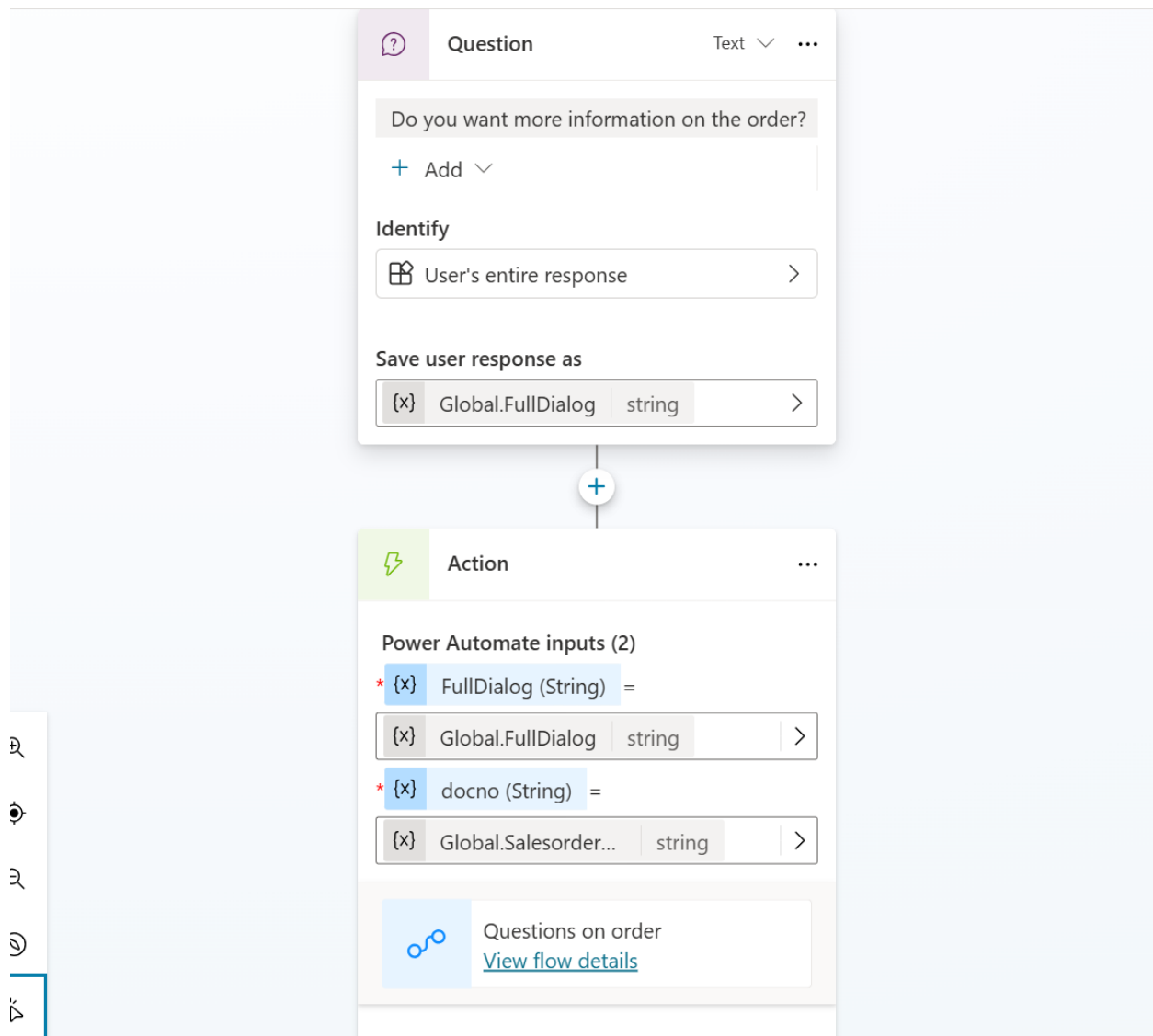
 On the right, the flowchart shows a "Condition" action branching into "True" and "False" paths. The "True" path contains an "Html to text 1 copy" action followed by an "Append to string variable 1" action. The "False" path is empty. Both paths merge and lead to an "HTTP" action, which then leads to a "Return value(s) to Power Virtual Agents" action.

Finally, we will send the response back to Copilot Studio and present this information to the user as shown – the variable "orderinfo" is the output from the "get order status" step.



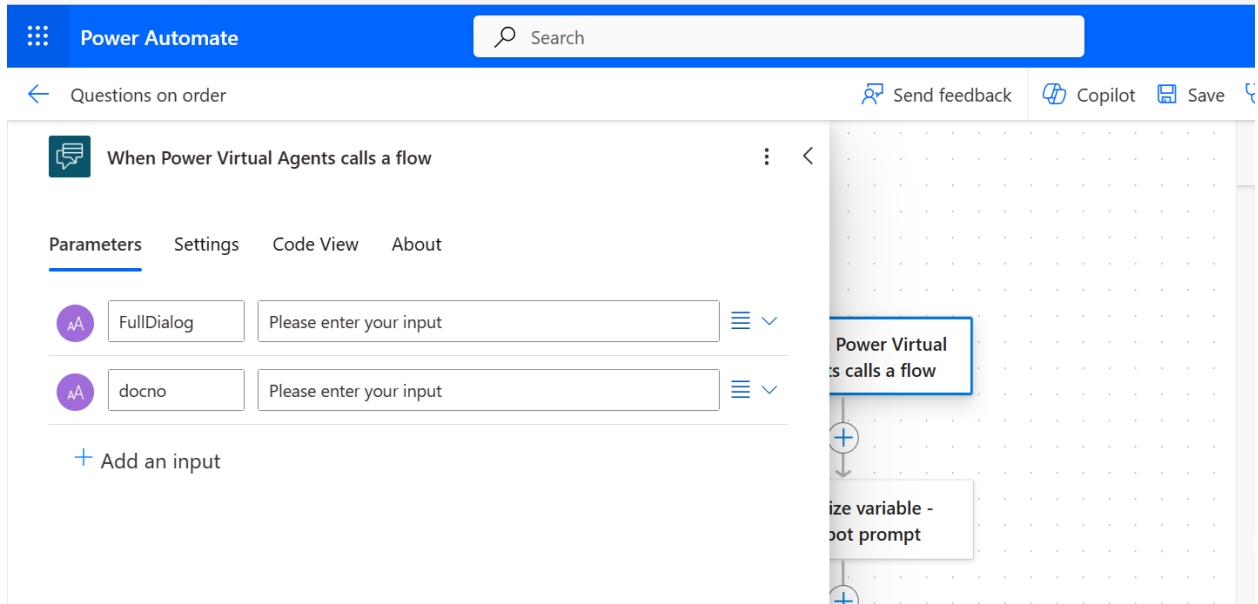
The next step after giving the order context, is to allow the user to ask the bot any particular question they have about the order on the SAP data .

For this we will create a Power Automate flow that gets the order status from the SAP system and then feeds it into Azure OpenAI along with the question the user has and Azure OpenAI will interpret the JSON response and answer the question.

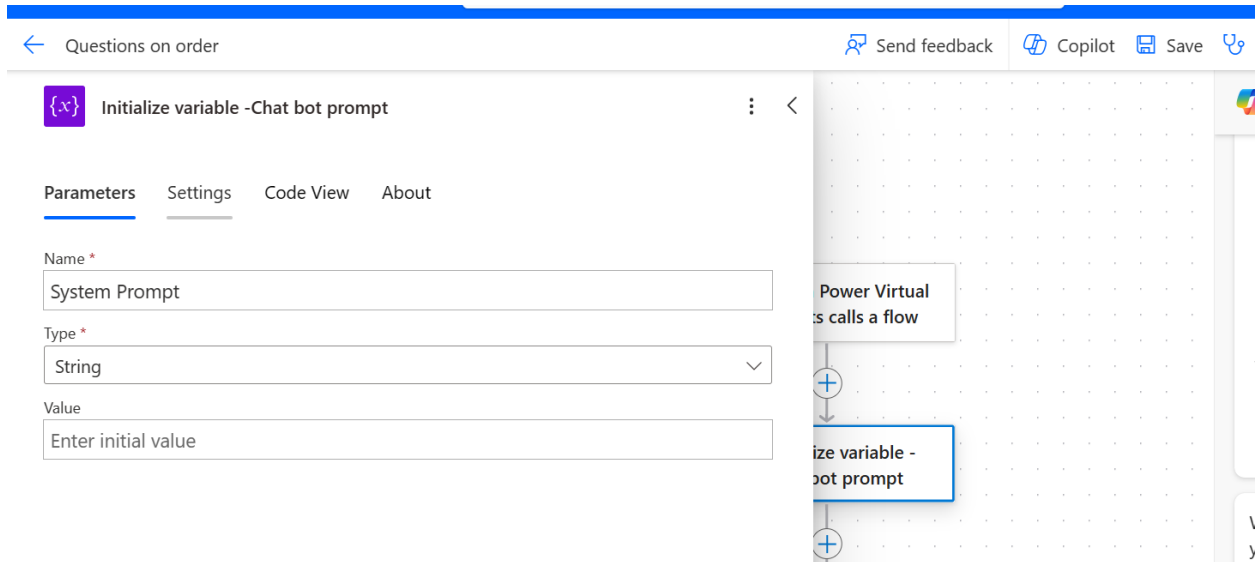


Here is the Power Automate flow for this:

The first step is to add two inputs to this the document number the order status information is needed for as well as the question that the user is asking.



The next step is to initialize the prompt variable that we will send to Azure OpenAI.



The next step is to call the SAP function to get order status.

Call SAP function (V2)

ParametersSettingsCode ViewTestingAbout

SAP System *

[REDACTED]

RFC Name *

BAPI_SALESORDER_GETSTATUS

RfcInputs/SALESDOCUMENT *

docno

Advanced parameters

Showing 2 of 4

Show all

Clear all

RFC Group Filter

*

Auto Commit

No

Power Virtual Agents calls a flow

Size variable - Add data to chatbot

Call SAP function

Power Automate

Questions on order

Send feedback Copilot Save

Set variable - Add SAP data to chatbot prompt

Parameters Settings Code View About

Name *
System Prompt

Value *
here is the data on the sales document answer any questions on the status information by understanding the JSON given here and any follow up questions the user has as well.
Here is the information.
`string(...)`
Dont mention it is from JSON data but mention the data is from the SAP System.

Power Virtual Agents calls a flow

```
string(body('Call_SAP_function_(V2)')?['STATUSINFO'])
```

Function Dynamic content

Search

String functions See more

Next, we will add the user's question and the SAP JSON response to the HTTP Request to Azure OpenAI.

Power Automate

Questions on order

Send feedback Copilot Save Flow checker Test New de

Initialize variable for request to bot

Parameters Settings Code View About

Name *
messages

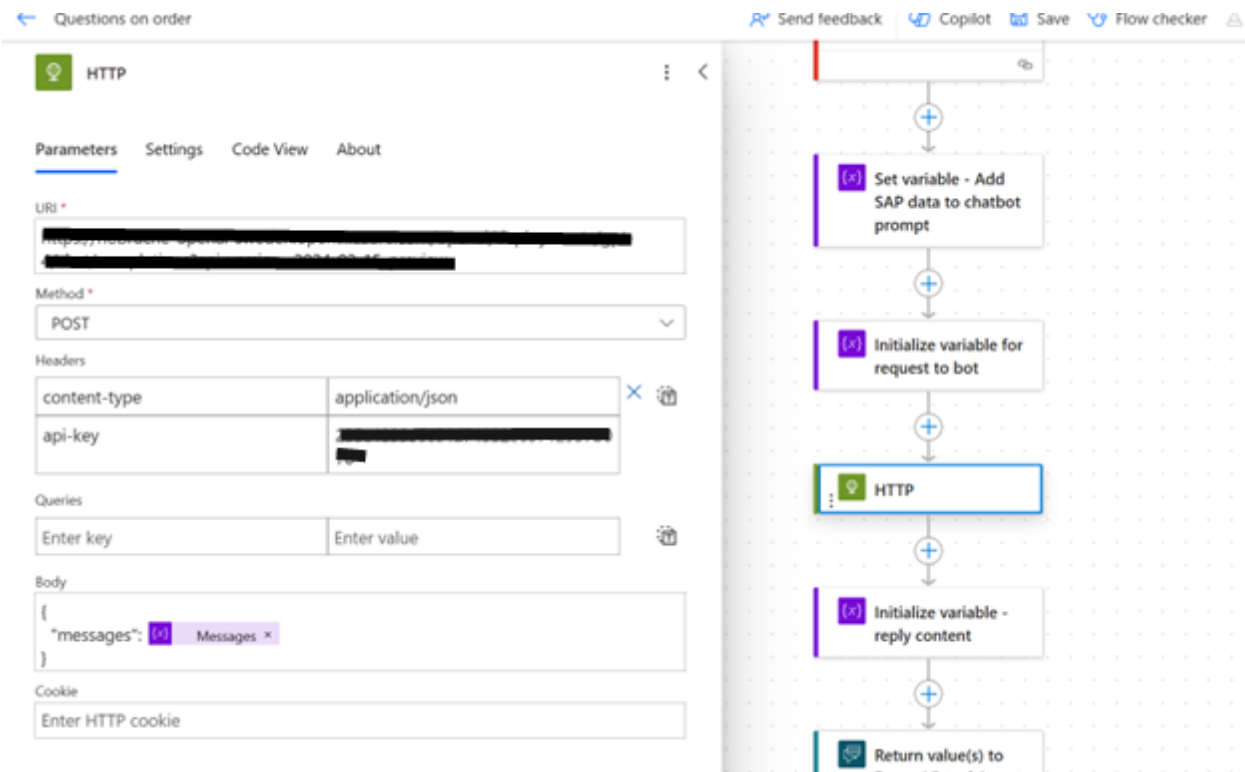
Type *
Array

Value

```
[  
  {  
    "role": "system",  
    "content": "System P..."  
  },  
  {  
    "role": "user",  
    "content": "FullDialog"  
  }  
]
```


Flowchart:

```
graph TD; A[Initialize variable for request to bot] --> B[Set variable - Add SAP data to chatbot prompt]; B --> C[Initialize variable for request to bot]; C --> D[HTTP]; D --> E[Initialize variable - reply content];
```



Lastly, we will process the response, store it in a variable, and send it back to the Copilot Studio – this variable will be called "Generatedanswer".

The other variable we will send back is the order status information we got from the SAP System. This variable could be used by other potential topics, and we save it as a global variable in the Copilot Studio so that it can be used across topics.









 Questions on order



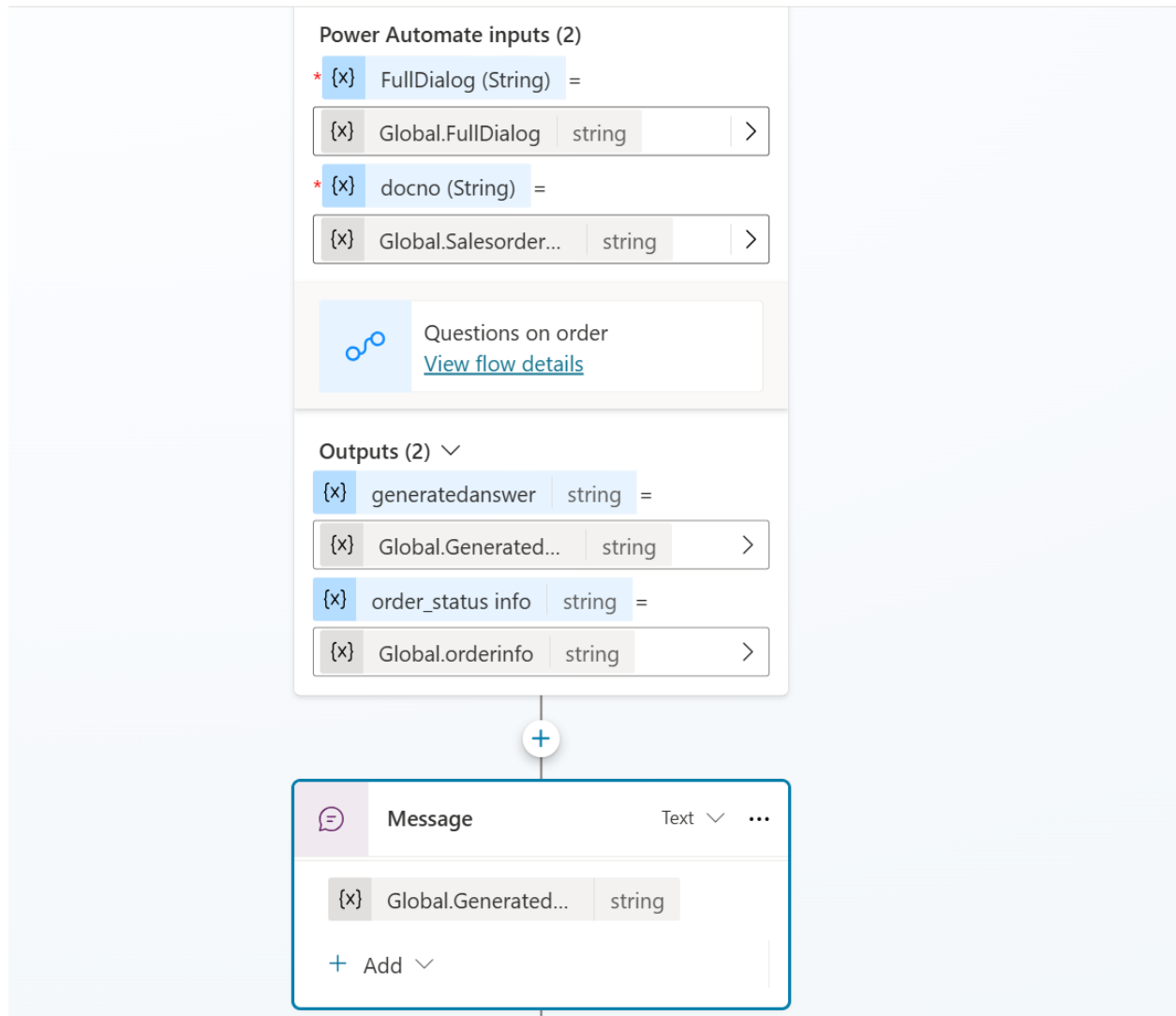
Return value(s) to Power Virtual Agents copy



Parameters Settings Code View About

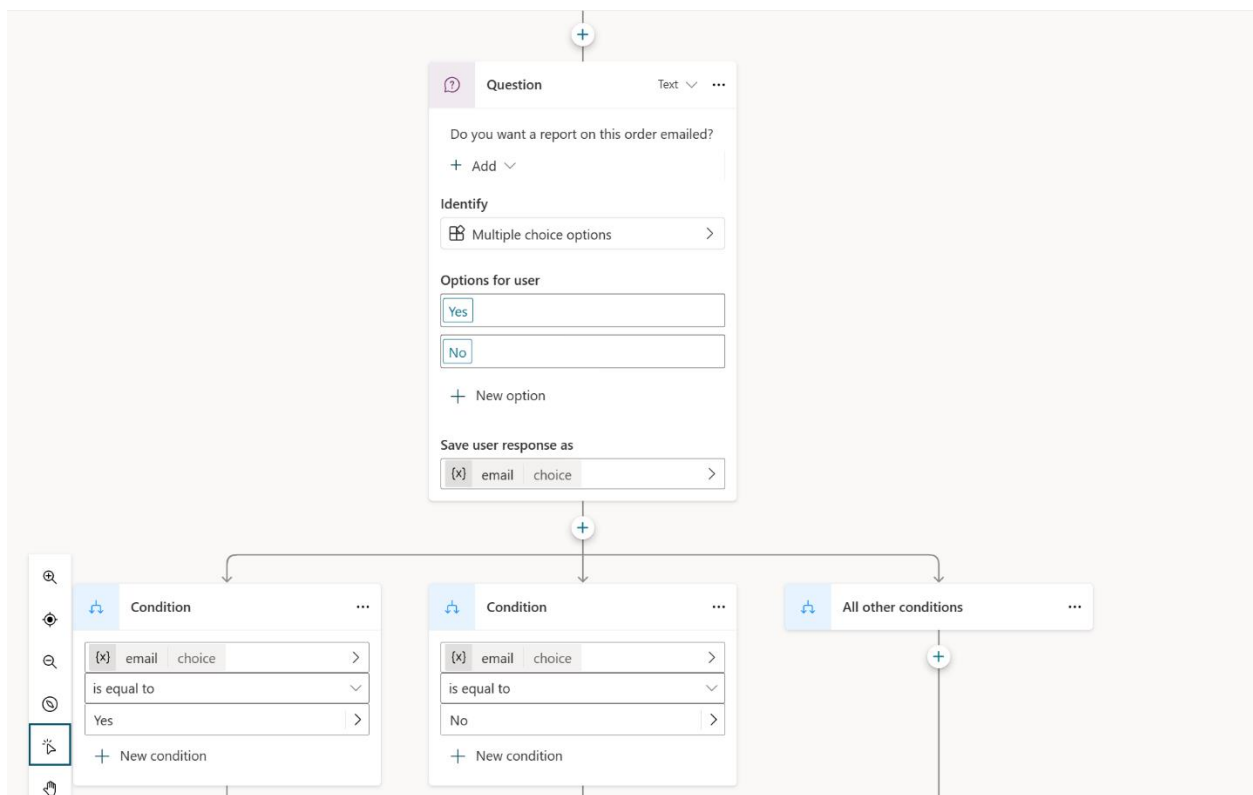
	Generatedan...	 body(...) ×	 
	order status i...	 string(...) ×	 

 Add an output



Lastly, in case the salesperson wants to get back to the customer with the analysis, we will ask the salesperson if they want the entire conversation/ questions they asked about the order to be summarized by AI and put into a report to be sent to the customer (or anyone via Outlook).

For this back in the Copilot Studio, we will ask the user a question as shown here:



If the user responds that they want to send a report of the email, we will ask them for the email ID they want to send this to, for this, we need to ask another question and select the ready made email entity as the response (as we did for customer number and sales order number earlier).

Finally after that, we will call the final Power Automate flow with the variables shown below.

This flow will combine the responses from the "context delayed" flow (the summary of the Teams' chats and Outlook emails) as well as the recent response from Azure OpenAI and send a request to Azure OpenAI to respond in an email format.

↶

✓

✂

📄

📄

🗑

+

↓

?

Question

Text

⋮

What email ID do you want to send th...

+ Add

⌵

Identify

📧

Email

>

Save user response as

{x}

EmailId

string

>

⬇

+

⬆

⚡

Action

⋮

Choose information to identify

✕

To help the copilot pick out specific information from your user's response, choose an option or entity.

🔍

Search

📦

Create an Entity

>

Duration

Lengths of time, extracted as a string, in standard TimeSpan format

Email

Email addresses, extracted as a string

Event

Event names, extracted as a string

Language

Language names, extracted as a string

The screenshot displays the configuration for a Power Automate flow. It is divided into three main sections: 'Power Automate inputs (4)', a connector section, and 'Outputs (1)'. The inputs section contains four required inputs, each with a blue selection box, a text label, an equals sign, and a dropdown menu. The first input is 'Delay context (String)' with the value 'Global.Orderdelay...'. The second is 'order info SAP (String)' with 'Global.Generated...'. The third is 'order no (String)' with 'Global.Salesorder...'. The fourth is 'email id (String)' with 'EmailId'. All dropdowns show 'string' as the type. Below the inputs is a connector icon and the text 'report email' with a link to 'View flow details'. The outputs section shows one output with the label 'email_has been se...' and the value 'Emailhasbeensent', both with 'string' as the type.

Power Automate inputs (4)

- * {x} Delay context (String) =
{x} Global.Orderdelay... string >
- * {x} order info SAP (String) =
{x} Global.Generated... string >
- * {x} order no (String) =
{x} Global.Salesorder... string >
- * {x} email id (String) =
{x} EmailId string >

report email
[View flow details](#)

Outputs (1) ▾

- {x} email_has been se... string =
{x} Emailhasbeensent string >

The steps for this flow are:

The first step is to define the 4 variables we need. The summary of Outlook + Teams messages, the response to the previous question asked from Azure OpenAI, the order number and finally the email ID we want to send the email to – this would be the customers email ID.

Power Automate

report email

Send feedback Copilot

When Power Virtual Agents calls a flow

Parameters Settings Code View About

Delay context Please enter your input

Report Please enter your input

order no 1 Please enter your input

email id Please enter your input

+ Add an input

When Power Virtual Agents calls a flow

HTTP

```
graph TD; Trigger[When Power Virtual Agents calls a flow] --> Join((+)); Join --> HTTP[HTTP]; HTTP --> Exit(( ));
```

Next we combine all these inputs and send a request to Azure OpenAI to generate an email out of it with formatting we want as shown here:

The screenshot displays the Power Automate interface. On the left, the 'Parameters' tab for an 'HTTP' action is open. It shows the following configuration:

- URI:** [Redacted]
- Method:** POST
- Headers:**

content-type	application/json
api-key	[Redacted]
- Queries:** A table with columns 'Enter key' and 'Enter value'.
- Body:**

```
{
  "messages": [
    {
      "role": "system",
      "content": "Take this information from there inputs and create formal business letter formatted professionally in HTML formatting on why the order was delayed .Also format the letter in a professional manner and return the response in bullet format and add an empty line between each bullet point in HTML formatting. The heading should be in bold in html formatting. Start the email off with dear sir/Madam and no text/description above it. and then start the rest of the email off with heading in bold with HTML formatting. Here is the input (do not include links) Delay co... + order inf... . For the signing off , just end with Thank you, Noopur "
    }
  ]
}
```
- Cookie:** Enter HTTP cookie

On the right, a flow diagram is visible, showing the sequence of actions:

- When Copilot Studio calls a flow
- HTTP (configured as shown in the left pane)
- Send an email (V2)
- Return value(s) to Power Virtual Agents

Next we use the outlook send email function to send the email to the customer and populate the fields as shown below.

report email

Send an email (V2)

Parameters Settings Code View Testing About

To *
email id x

Subject *
Report on order order no x delay

Body *
body(...) x

Advanced parameters
Showing 1 of 7 Show all Clear all

Importance
Normal

When Power Virtual Agents calls a flow

HTTP

Send an email (V2)

Return value(s) to Power Virtual Agents

The body of the email here is the response from Azure OpenAI (which is the generated email in this case) - this is the expression for it as used before body
“('HTTP')?['choices']?[0]?['message']?['content']”.

Finally we send a response back to the Copilot Studio stating that the email as been sent.

Power Automate

report email

Return value(s) to Power Virtual Agents

Parameters Settings Code View About

AA Email has be... Email has been sent

+ Add an output

And the chat bot sends a final response to the sales person informing that the email has been sent.