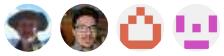
Tutorial: Create a weather bot with Composer

11/23/2021 • 17 minutes to read • 

In this article

APPLIES TO: Composer v2.x

Follow the instructions in this tutorial to build a weather bot using Bot Framework Composer and the OpenWeather Weather API. When finished, you'll know how to navigate Composer's UI, and have an understanding of the core elements of bot building, like: dialogs, triggers, interruptions, and cards.

In this tutorial, you'll learn how to:

- ✓ Create a bot from a template
- ✓ Modify the Greeting
- ✓ Add a new dialog
- ✓ Connect a trigger
- ✓ Prompt a user for input
- ✓ Validate user inputs
- ✓ Make an HTTP request to an external API
- ✓ Enable interruptions for help and cancel requests

- ✓ Add a card to display weather

Prerequisites

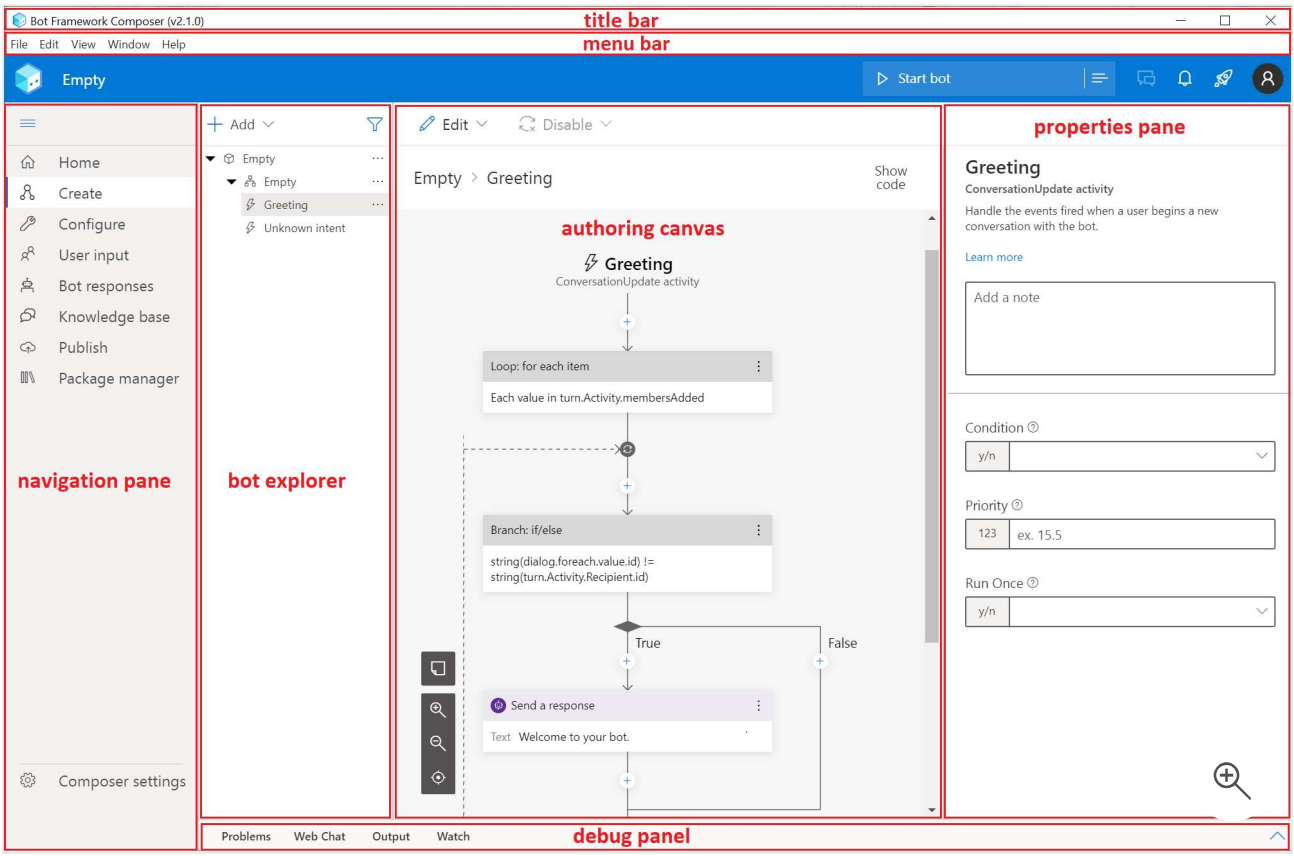
Before you get started, you'll need to:

- Download and install Bot Framework Composer for Windows, macOS, or Linux.
- Sign up for a free Weather API subscription with OpenWeather. After you create your account and subscribe, you can get your API key by clicking your user name from the navigation bar and selecting **My API keys**.

A quick tour of Composer

Before you jump in, it helps to understand which components make up the Composer's user interface. For this tutorial, you'll primarily use:

- The navigation pane: This is used to navigate Composer's options and features.
- The bot explorer: This displays the components that make up your project. This includes bots, dialogs, and the triggers associated with each dialog.
- The authoring canvas: This is where the logic for your bot lives. It shows all actions associated with the selected trigger.
- The properties pane: This is where you set properties for an explicit action, such as a prompt for text, sending an HTTP request, or clearing existing values from a group of properties.



Create an Empty Bot in Composer

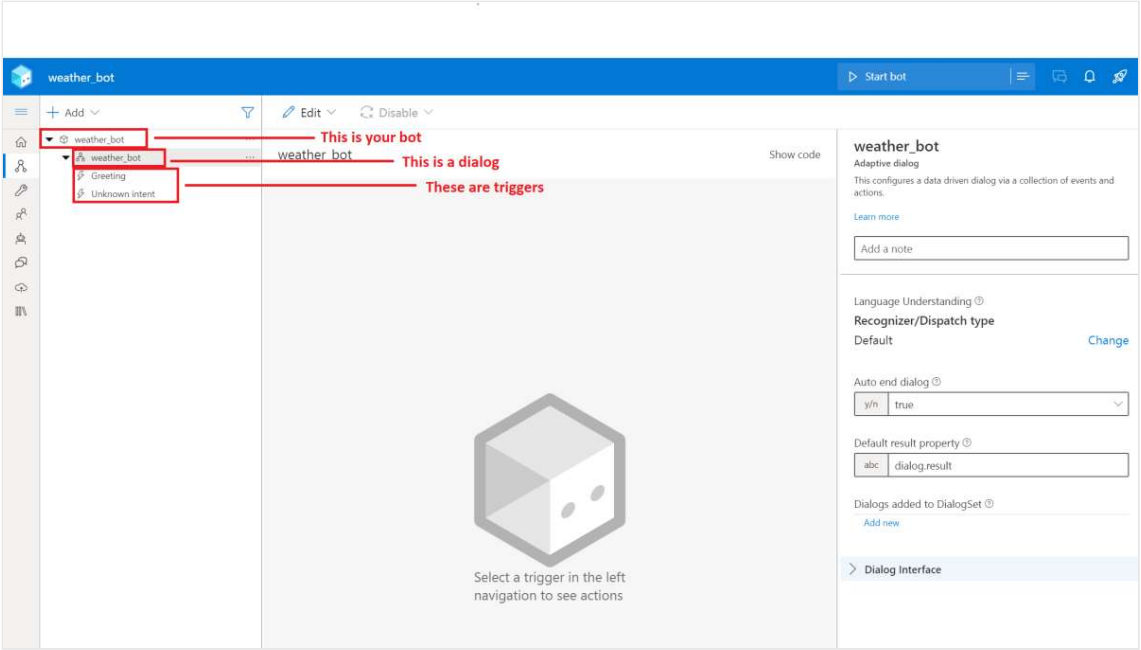
To create a weather bot, you're going to start with the **Empty Bot** template. The **Empty Bot** template includes one dialog with two triggers.

1. Open Composer.
2. Select + **Create new**.
3. Select **C#**, then select **Empty Bot**. Review the description, then select **Next**.
4. In the **Create a bot project** window, you'll need to provide some information about your bot. For this tutorial, you can use these values:

    - **Name**: `weather_bot`
    - **Runtime**: Azure Web App
    - **Location**: Select an existing folder, or create a new one to store your bot locally.

5. Select **Create**. After you select Create, Composer will get the template, create a project, build the runtime, and pull in any required packages. Setup can take a few minutes.
6. When finished, you'll have a basic bot skeleton, with a dialog named `weather_bot`, and two triggers: `Greeting` and `Unknown intent`.

- `Greeting` - When a user connects to the bot, the user is sent a greeting.
- `Unknown intent` - When the user sends a message or makes a request that the bot doesn't recognize, the bot sends a message telling the user it didn't understand the request. Here's a quick look at how your bot should look in the Composer UI.



Check point: Did you create an Empty Bot?

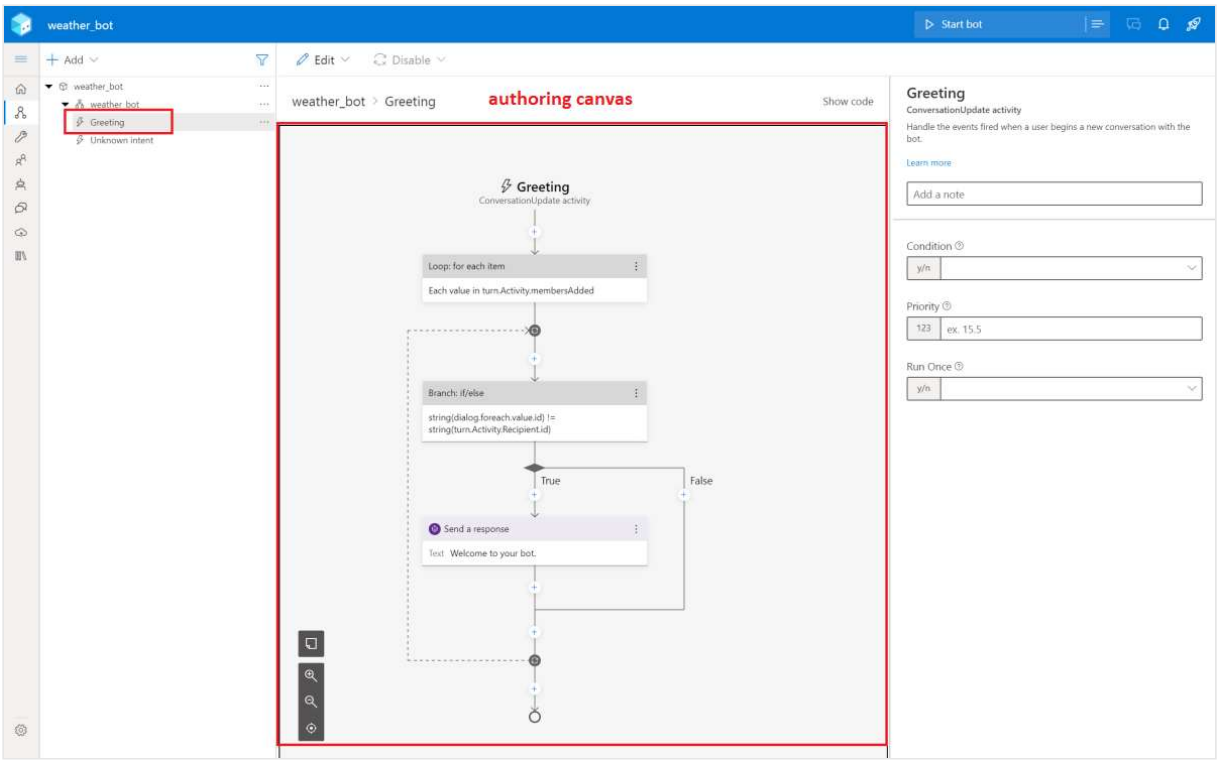
I created an empty bot | I ran into an issue

Modify the Greeting

Now let's update your bot's `Greeting` message. This is a great opportunity to give your users an idea of how to communicate with your bot. Here we're going to welcome users and tell them that they can start by typing **weather**.

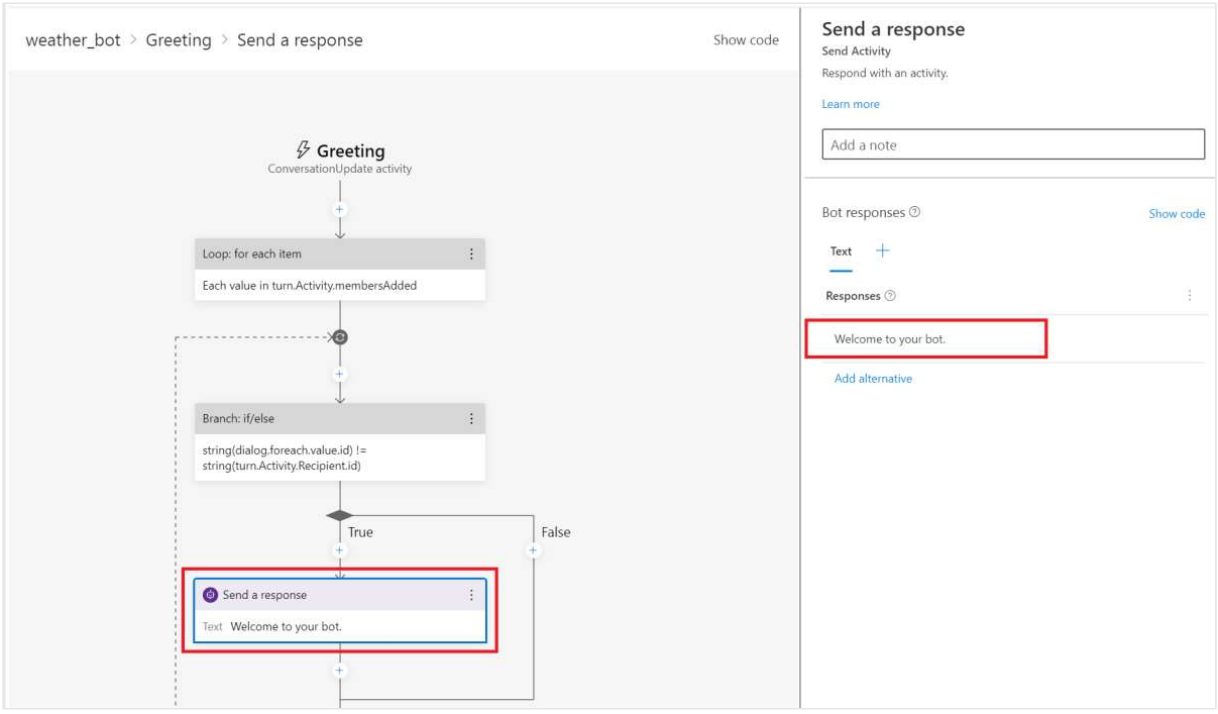
1. Select the `Greeting` trigger in the bot explorer.

2. After you select `Greeting`, you'll see the authoring canvas. This is where you add logic to your bot.



3. In the authoring canvas, locate and select the action named **Send a response**.

4. In the properties pane, you'll see a few options. Locate **Responses**, then select **Welcome to your bot**.



5. Replace the text with:

Bot response	Copy

```
Welcome to Weather Bot! Type "weather" to get started.
```

That's it, you've updated your bot's greeting message.

Check point: Did you change the Greeting?

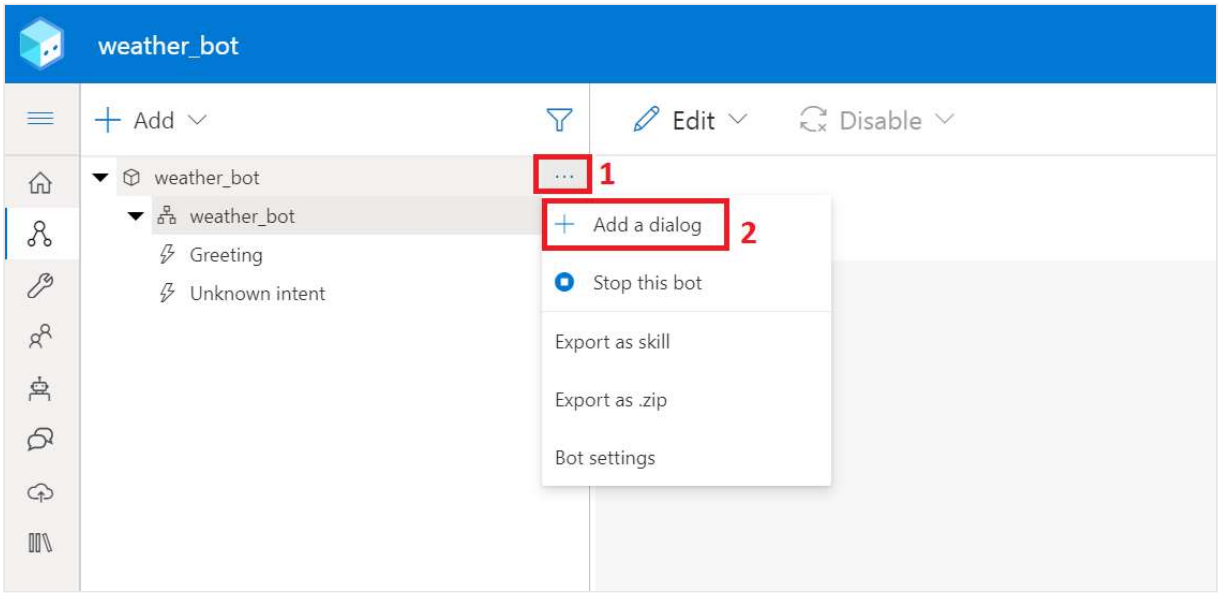
1. Select **Start bot** or **Restart bot** in the upper right of the Composer window.
2. In the Local bot runtime manager, select **Open Web Chat**.
3. When the bot starts, you should see the updated Greeting message: Welcome to Weather Bot! Type "weather" to get started.

I updated the Greeting message | I ran into an issue

Create a new dialog

Every bot is built from a series of components called dialogs. Each dialog encapsulates some bot functionality, such as sending a response, prompting a user for text, making an HTTP request, or maybe all of these. In this section, you're going to create a dialog to get the weather.

1. Locate the `weather_bot` bot in the bot explorer. Select **...**, then select **+ Add a dialog**.



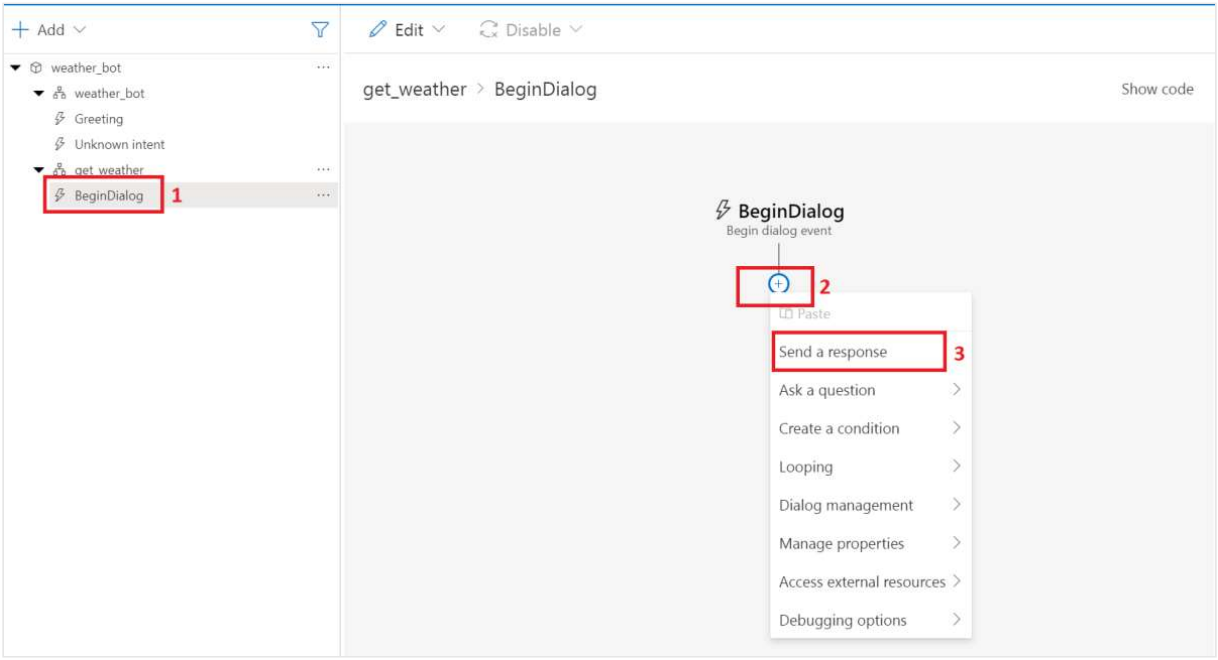
2. **Create a dialog** will appear. When prompted, enter the following:

   - **Name -** `get_weather`
   - **Description -** `Get the current weather conditions.`

3. Select **OK**.

4. Locate the `get_weather` dialog in the bot explorer, then select the `BeginDialog` trigger.

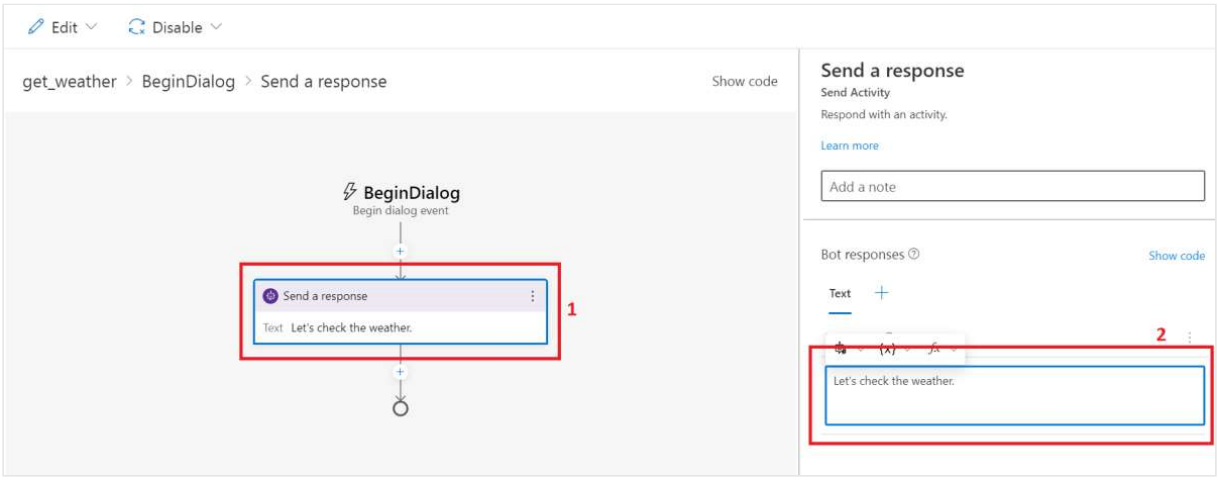
5. In the authoring canvas, select **+**, then select **Send a response**.



6. You should see a new activity in the authoring canvas called **Send a response**.

7. In the properties pane on the right, locate the empty text box and enter the following text:

Bot response	Copy
`Let's check the weather.`	



Let's recap what you just did. You created a new dialog called `get_weather`. This dialog has a trigger called `BeginDialog`, and within `BeginDialog`, you've added an **activity** called **Send**

a response. This response is what is sent to users when the `get_weather` dialog is activated.

Before you can test your new dialog, you need to create a trigger in the main `weather_bot` dialog, which allows you to start `get_weather`.

Start a dialog from a trigger

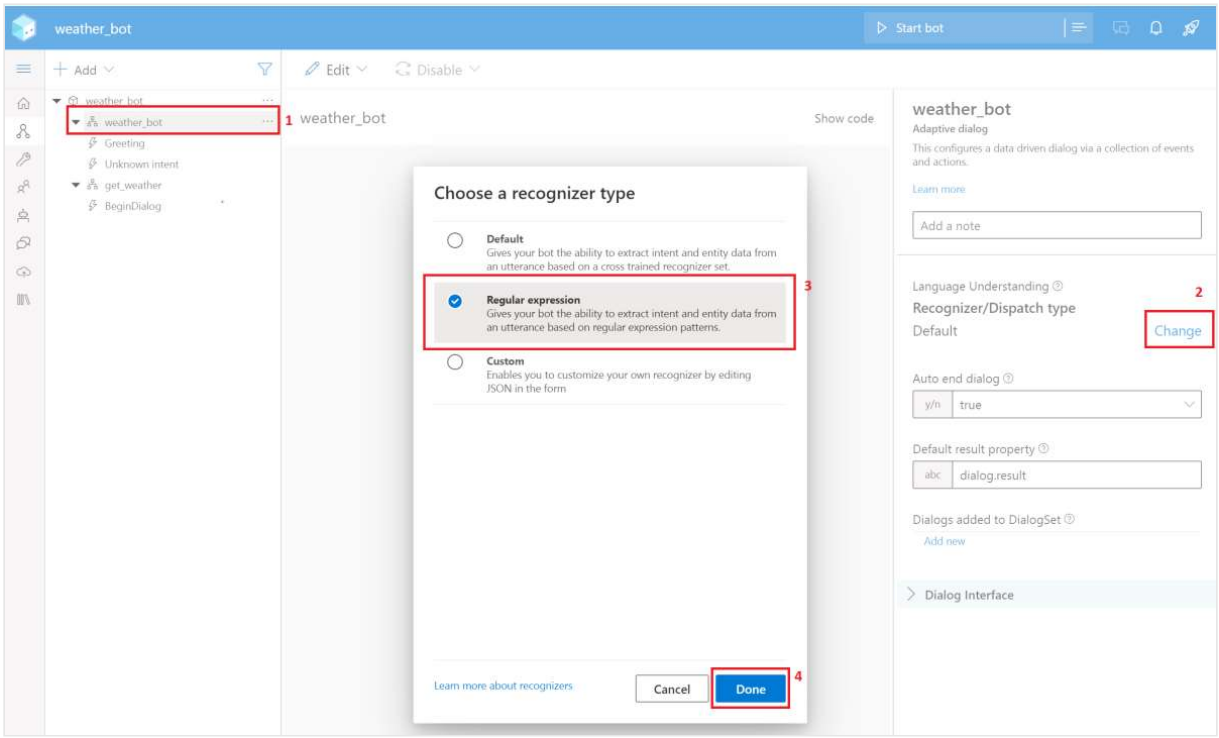
The conversation flow with a bot is broken down into different dialogs. In this section, you'll learn how to connect dialogs, specifically, how to connect the `get_weather` dialog to the `weather_bot` dialog (or main dialog for your bot).

To learn more about dialog flow and design recommendations, see Best practices for building bots using Composer.

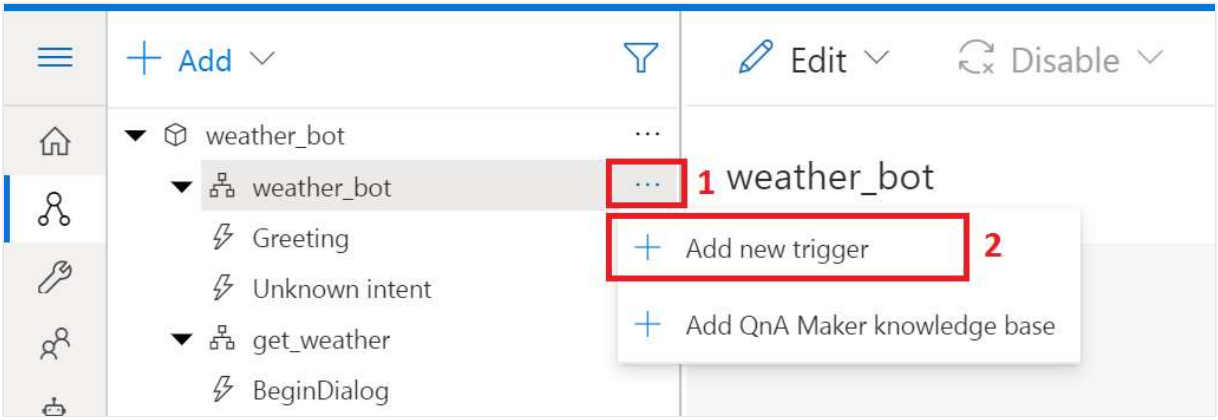
1. Select the `weather_bot` dialog in the bot explorer.

2. Then, in the properties pane on the right, locate **Recognizer Type/Dispatch type** and select **Change**.

3. A window will appear, select **Regular expression**, then select **Done**.



4. Locate the `weather_bot` dialog and select the **...** to expand the menu, then select **+ Add new trigger**.

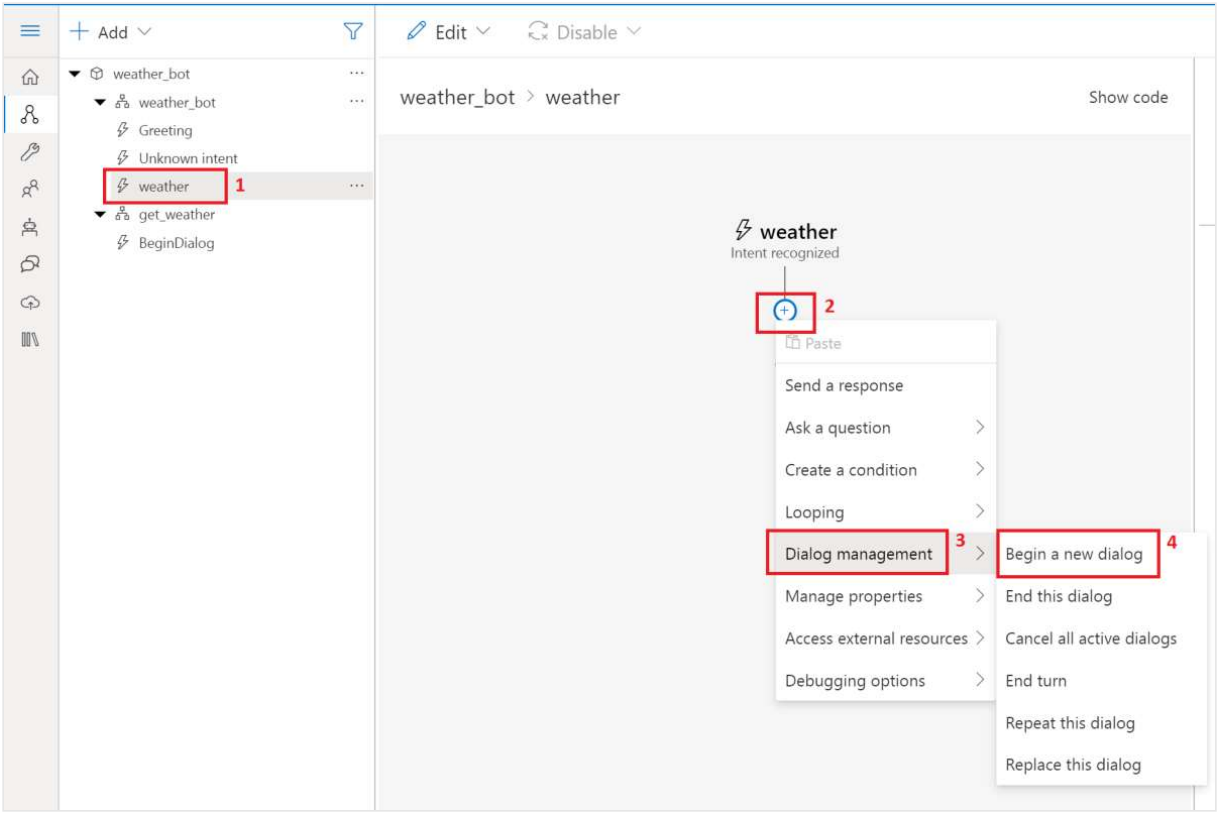


5. When the **Create a trigger** menu appears, enter the following information and select **Submit**:

   - **What is the type of this trigger? - Intent recognized**
   - **What is the name of this trigger? -** `weather`
   - **Please input the regEx pattern -** `weather`

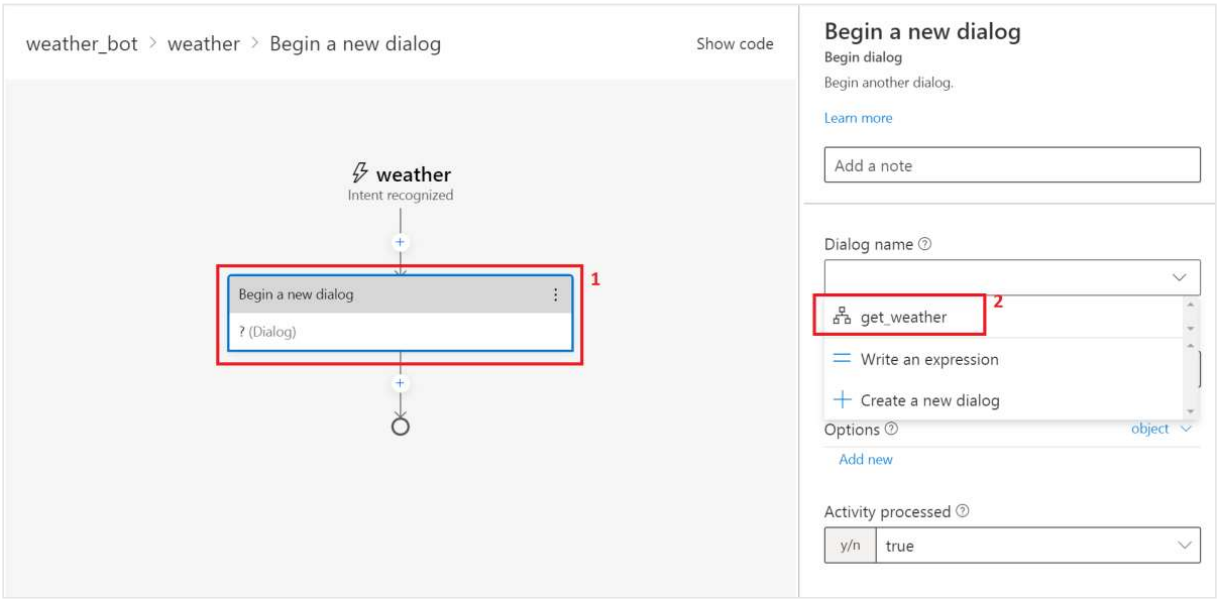
   This trigger tells your bot to look for the word *weather* in any incoming message. In most cases, regular expressions (regEx) are more complex, however, in this tutorial, the goal is to demonstrate how it works. To learn more about pattern matching and built in evaluations, see Adaptive expressions. If you'd like to learn more about improving a bot with natural language processing (NLP), see Natural language processing in Composer.

6. Make sure the `weather` trigger is selected. Then, in the authoring canvas, select **+**, then select **Begin a new dialog** from the **Dialog management** drop down menu.



7. In the authoring canvas, select **Begin a new dialog**.

8. In the properties pane on the right, locate **Dialog name** and select **get_weather** from the drop-down menu.



Let's recap what you accomplished. You've told your `weather_bot` to recognizer regular expressions. You created a trigger for `weather` and set its **Trigger phrase** to **weather**. Finally, you connected the `weather` trigger to the `get_weather` dialog.

Check point: Did you connect a trigger to get_weather?

1. Select **Start bot** or **Restart bot** in the upper right side of the Composer window.
2. In the Local bot runtime manager, select **Open Web Chat**.
3. After your bot has greeted you, type "weather". You should get this response: "Let's check the weather".

I created a new dialog and set up a trigger | I ran into an issue

Prompt a user for input

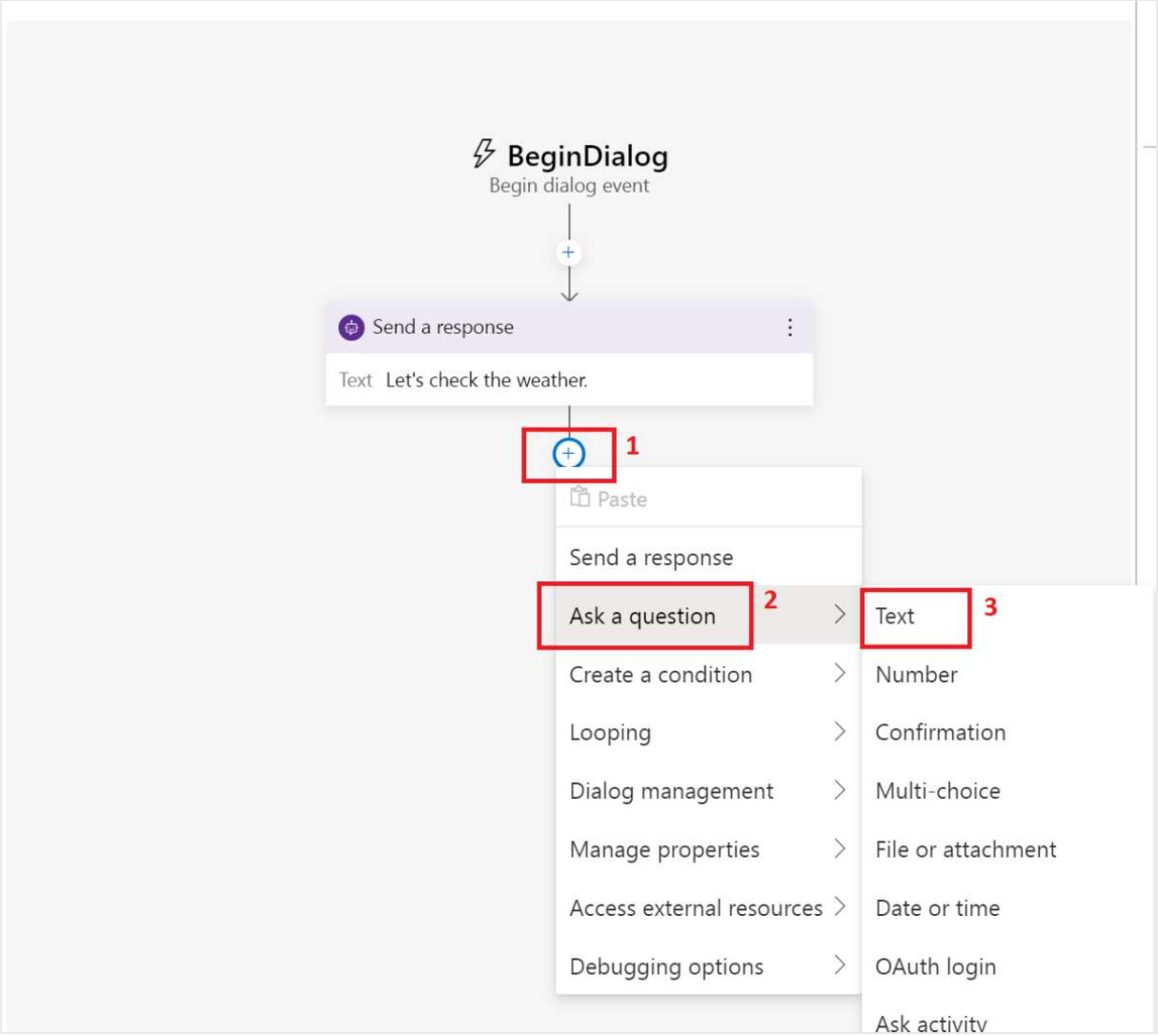
Before your bot can retrieve the weather, it needs the postal code for the location forecast. You can prompt a user for input with the **Text input** action.

1. Locate `get_weather` in the bot explorer, then select `BeginDialog`.

2. In the authoring canvas, below the **Send a response** action, select **+**.

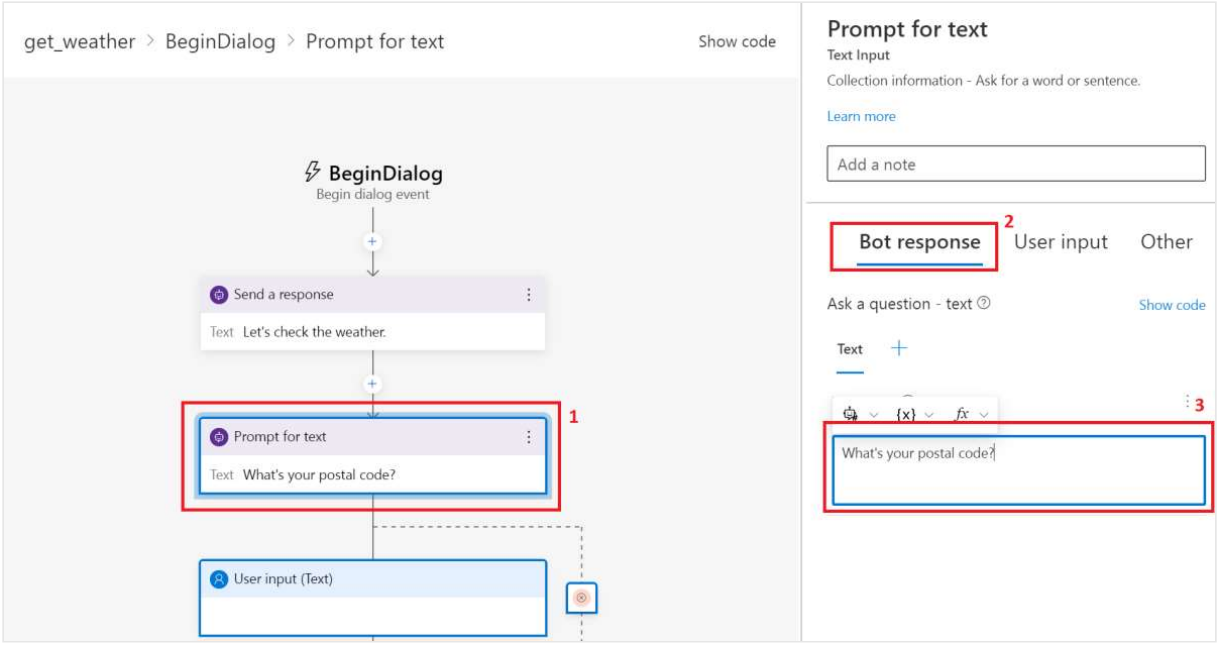
3. Select **Ask a question**, then select **Text**. Two nodes should appear in your authoring canvas: **Prompt for text** and **User input**. Each of these nodes corresponds to a tab in the properties pane, specifically **Bot response** and **User input**.

    - **Bot response -** The prompt from the bot asking the user for an input.
    - **User input -** Let's your bot assign the user input to a property that is saved in memory and can be used by your bot for additional processing.
    - **Other -** Offers options for validation and responses to user inputs.



4. In the properties pane, locate **Bot response** and select **Add alternative**. Then enter the following text.

Bot response	Copy
`What's your postal code?`	



This prompt is sent to users when the `get_weather` dialog is started.

5. Next, select **User input** in the properties pane.

6. Locate **Property** and set the value to:

```
user.postalcode
```

7. Locate **Output format**, then update the value with this adaptive expression. trim() is a prebuilt adaptive expression that removes preceding and trailing spaces from a value, this ensures that the value is accepted if someone accidentally adds a space before or after the postal code.

```
=trim(this.value)
```

Validate user inputs

Now that you've set a prompt and created a property to save the user's response, let's learn how to set validation rules. Since you'll be using postal code to look up weather forecasts, you'll need to make sure that the user input is five characters (US postal codes). When the user provides a postal code greater or fewer than five characters, your bot

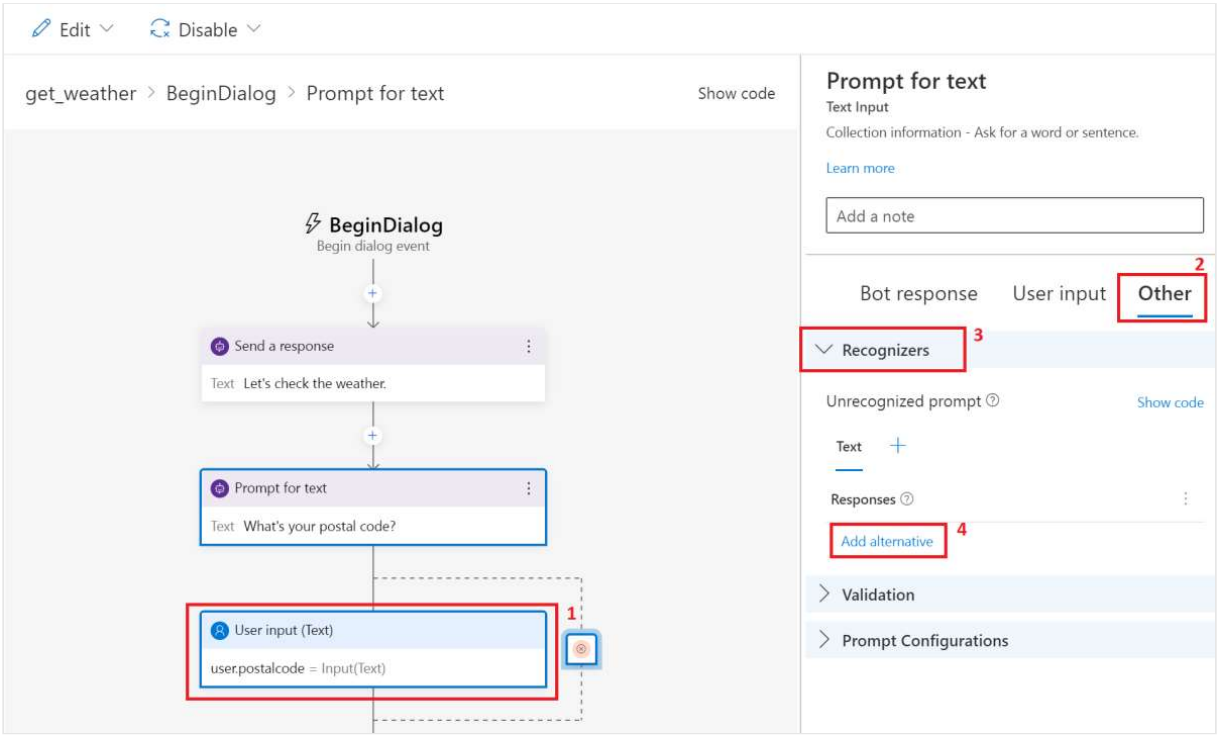
should respond with an error message. In this section, you'll learn how to validate user input and respond with error messages.

1. In the authoring canvas, make sure that **User input** is selected. Then select **Other** in the properties pane.

2. Expand **Recognizers** and select **Add alternative**. Then enter the following text:

Bot response	Copy

```
Sorry, I do not understand '${this.value}'. Please specify a 5 digit
postal code in the format 12345.
```



3. Expand **Validation**, then locate **Validation rules**. Select **Add new**, then from the drop-down menu select **Write an expression**. Enter the following text. This ensures that the user input equals five characters:

> ⚠ **Warning**
>
> If Composer adds an `=` character to your expression, **delete** it. The expression should match what's in the example below.

text	Copy

```
length(this.value) == 5
```

4. In the same menu, locate **Invalid prompt**. Below **Responses**, select **Add alternative**, then enter the following text. This is the message sent to the user if the postal code isn't equal to five characters:

Bot response Copy

```
Sorry, '${this.value}' is not valid. I'm looking for a 5 digit number as
postal code. Please specify a 5 digit postal code in the format 12345.
```

5. The last thing you'll want to do for validation, is set a default value for the postal code. Expand **Prompt configuration** and locate **Default value**, then enter the following text:

text Copy

```
90210
```

Now, whenever a user types **weather**, they'll be prompted for their postal code, and that value will be stored in `user.postalcode`. If the user enters a postal code that isn't equal to five characters, an error message will be sent to the user.

Check point: Did you validate user inputs?

1. Select **Start bot** or **Restart bot** In the upper right of the Composer window.
2. In the Local bot runtime manager, select **Open Web Chat**.
3. At the bottom of Composer, locate and select **Watch**. Then select **Add property**.
4. Locate **Name** and select `user.postalcode`.
5. In Web Chat, follow the prompts. When you enter a postal code, the value will be recorded in the **Watch** panel.

I prompted for text and validated the input | I ran into an issue

Make an HTTP request

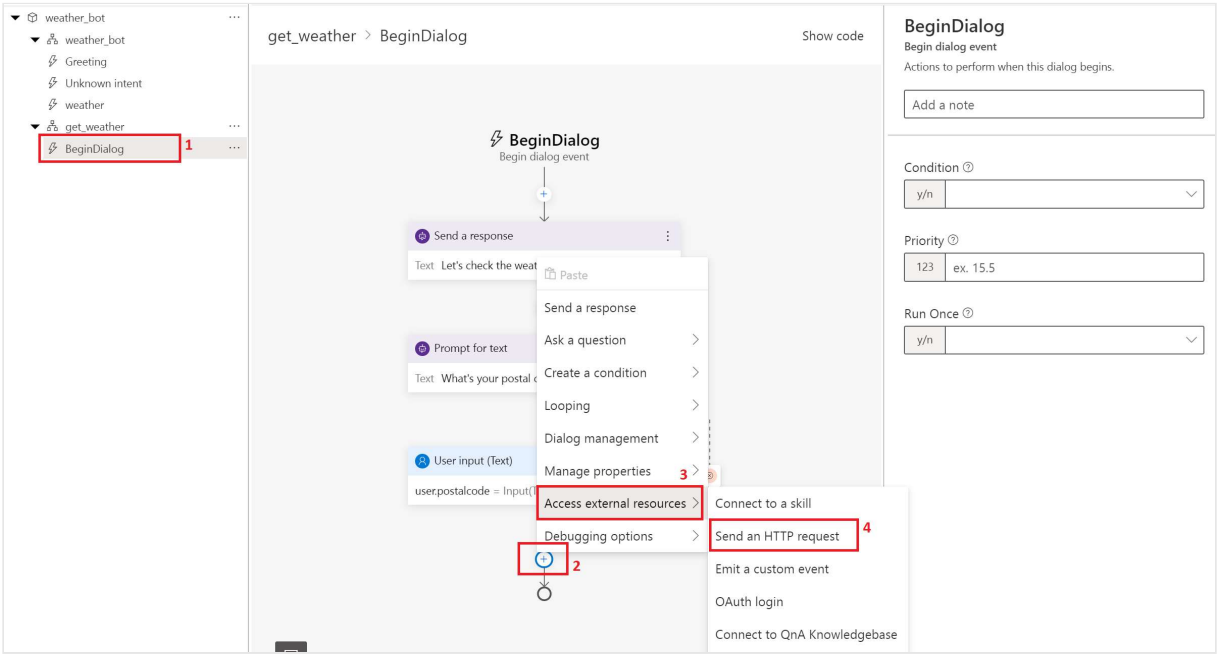
Your bot is configured to prompt the user for their postal code, and send an error message when an invalid postal code is provided. In this section, you'll learn how to make an HTTP

request to retrieve the current weather forecast from OpenWeather .

Before you continue, you're going to need your API key for the Weather API. If you don't have an API key, please follow the instructions in prerequisites to get one.

1. Make sure that you have `BeginDialog` selected in the bot explorer.

2. In the authoring canvas, select the + beneath all existing actions, then select **Send an HTTP request** from the **Access external resources** drop down menu.



3. Locate **HTTP method** in the properties pane and select **GET**.

4. Locate **Url** and set the value to the following text. Make sure to replace `YOUR_API_KEY` with your key from OpenWeather.

text Copy

```
http://api.openweathermap.org/data/2.5/weather?
zip=${user.postalcode},us&appid=YOUR_API_KEY
```

The reference to `${user.postalcode}` will be replaced by the value from the bot's `user.postalcode` property.

5. The **Result property** is where the response of your HTTP request is stored. The response from OpenWeather will include the following response values: `statusCode`, `reasonPhrase`, `content`, `headers`. Locate **Result property** in the right panel, then enter

the following value. We are storing the response in `dialog.api_response`. `dialog` is a scope that retains its properties until a dialog is ended.

```
text
dialog.api_response
```

You are storing the response in `dialog.api_response`. `dialog` is a scope that retains its properties for the duration of a dialog. To learn more about scopes and how properties are stored in memory, see Conversation flow and memory.

6. In the same menu, set **Response type** to **json**.

7. Next, you'll want to add some logic to your bot. You can do this with an **If/else branch**, using the HTTP response's `dialog.api_response.statusCode`. If the `statusCode` is `200`, you'll return the weather report. If the `statusCode` is anything else, you'll return an error message.

   In the authoring canvas, select **+**, then locate **Create a condition** and select **Branch: If/else**.

8. In the authoring canvas, locate and select **Branch: If/else**.

9. In the the properties pane, locate **Condition**, and from the drop-down menu select **Write an expression**. Enter the following text:

```
text
=dialog.api_response.statusCode == 200
```

10. Locate the **True** branch, then select **+**. Select **Set properties** from the **Manage properties** list.

11. Locate **Assignments** in the properties pane. For each property/value pair in the table below, select **Add new** and enter the following text:

Property	Value
`dialog.weather`	`=dialog.api_response.content.weather[0].description`
`dialog.icon`	`=dialog.api_response.content.weather[0].icon`

Property	Value
`dialog.city`	`=dialog.api_response.content.name`
`dialog.country`	`=dialog.api_response.content.sys.country`
`dialog.kelvin`	`=dialog.api_response.content.main.temp`
`dialog.celsius`	`=round(dialog.kelvin-273.15)`
`dialog.fahrenheit`	`=round((dialog.celsius * 9/5) + 32)`

For more information about the Weather API, see Current weather data .

12. In the **True** branch, below the **Set properties** action, select +, then select **Send a response**.

13. Locate **Responses** in the properties pane, then select **Add alternative**. Enter the following text:

Bot response Copy

```
The weather is ${dialog.fahrenheit}F or ${dialog.celsius}C and
${dialog.weather}.
```

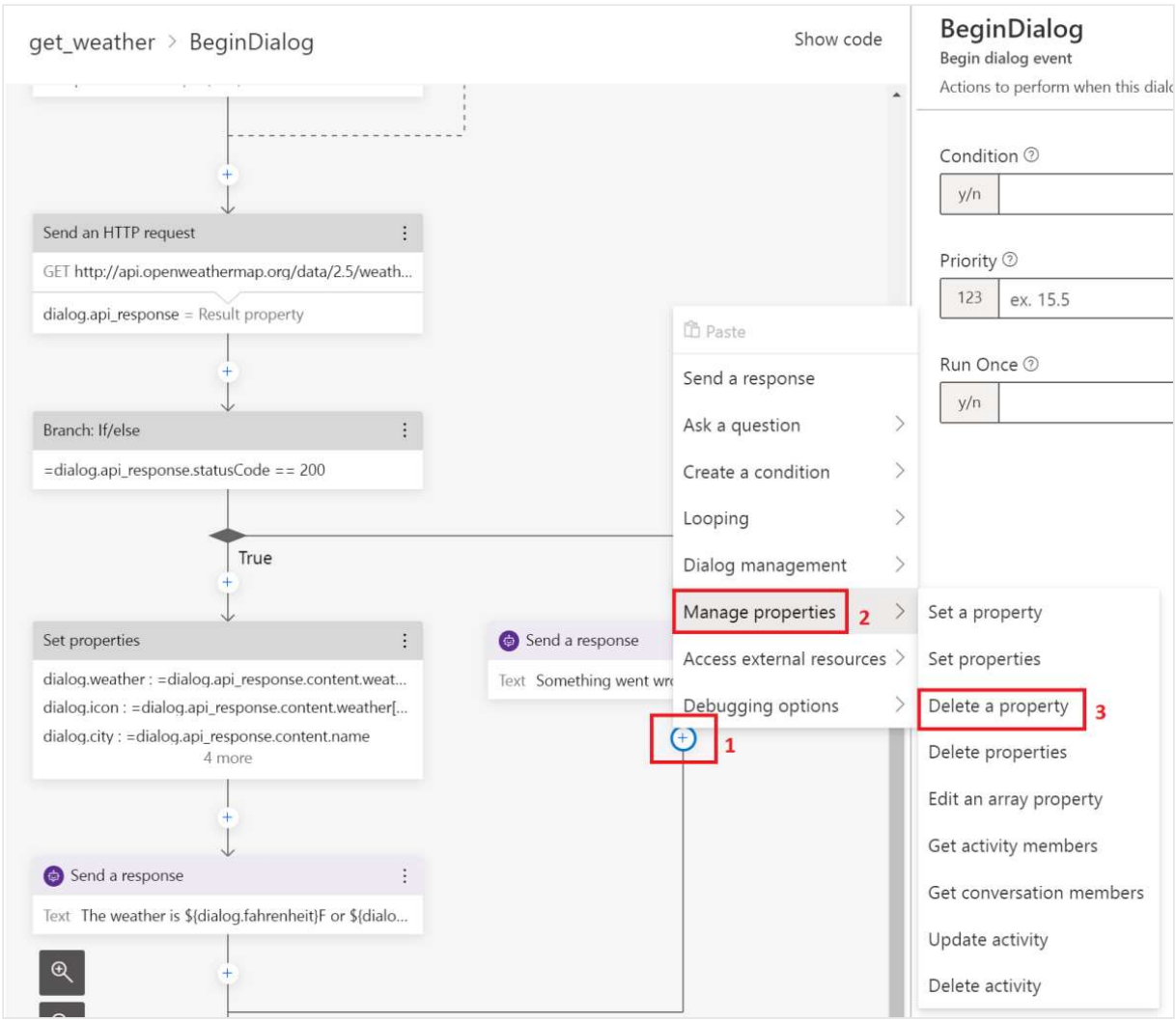
14. Now, let's tell your bot what to do if the `statusCode` isn't `200`. In the **False** branch, select +, then select **Send a response**.

15. In the properties menu, locate **Responses**, then enter the following text:

Bot response Copy

```
Something went wrong: ${dialog.api_response.content.message}.
```

16. For the purpose of this tutorial it's assumed that if you are in this branch, it's because the postal code is invalid. If the postal code is invalid, it should be removed so that the invalid value doesn't persist in the `user.postalcode` property. In the **False** branch, below the **Send a response** action, select +. Then select **Manage properties** and **Delete a property**.



17. In the properties pane on the right, update the value for **Property** with:

text	Copy
`user.postalcode`	

Let's recap what you just completed. You created an HTTP request to the Weather API, set up conditional branching, which uses the `statusCode` of the HTTP response, and added responses for a successful and failed request.

Check point: Did you make an HTTP request?

1. Select **Start bot** or **Restart bot** in the upper right of the Composer window.
2. In the Local bot runtime manager, select **Open Web Chat**.
3. When prompted to enter a postal code, type: "98052". This is the postal code for Redmond, WA.
4. You should see a message like this: "The weather is 61F or 16C and partly cloudy".

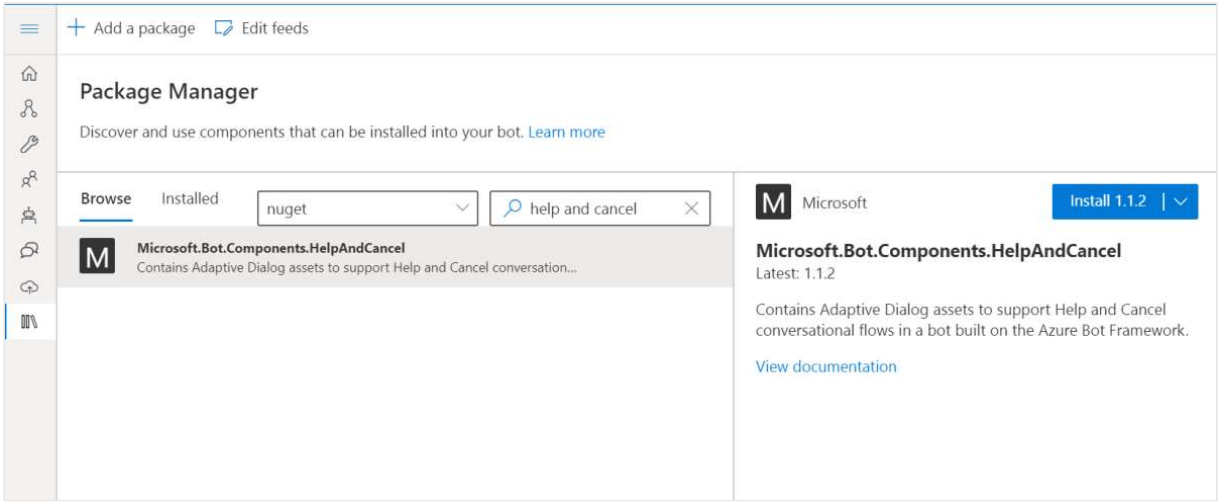
I made an HTTP request and got the weather | I ran into an issue

Add interruptions to the conversation flow

When building a bot, it's a good idea to allow your users to interrupt the conversation flow. Common interruptions include requests for help or to cancel the current dialog. In this section, you'll learn how to add help and cancel interruptions to your bot using **Package Manager**.

1. Select the **Package Manager** in the navigation pane. Make sure **nuget** is selected, then type **help** into the search bar and press **Enter**.

2. Select `Microsoft.Bot.Components.HelpAndCancel` in the package list. In the properties pane on the right, make sure the latest stable version is selected, as `rc` versions of packages may not function correctly. Then click **Install**.



3. Composer will let you know that the package is being installed. Installation may take a few minutes. When the **Project Readme** appears, review the content, then select **OK**.

4. Select the **Create** icon in the navigation pane to get back to the authoring canvas. You'll see two new dialogs: `CancelDialog` and `HelpDialog`. Each with a `BeginDialog` trigger.

CancelDialog

1. Locate the `weather_bot` dialog and select **...**, then select **+ Add a new trigger**.

2. When the **Create a trigger** window appears, enter the following values, then select **Submit**:

   - **What is the type of this trigger? - Intent recognized**
   - **What is the name of this trigger (RegEx) -** `cancel`
   - **Please input regEx pattern -** `cancel|stop|quit`

   This is a simple example of a regular expression that will recognize the cancel trigger if a user responds with: cancel, stop, or quit.

3. In the authoring canvas, select **+**.

4. Locate **Dialog management**, then select **Begin a new dialog**.

5. In the properties pane, locate **Dialog name**, then from the drop-down menu, select **CancelDialog**. Now the `cancel` trigger is connected to the `CancelDialog`.

HelpDialog

1. Locate the `weather_bot` dialog and select **...**, then select **+ Add a new trigger**.

2. When the **Create a trigger** window appears, enter the following values, then select **Submit**:

   - **What is the type of this trigger? - Intent recognized**
   - **What is the name of this trigger (RegEx) -** `help`
   - **Please input regEx pattern -** `help|support|advice`

   This is a simple example of a regular expression that will recognize the help trigger if a user responds with: help, support, or advice.

3. In the authoring canvas, select **+**.

4. Locate **Dialog management**, then select **Begin a new dialog**.

5. In the properties pane, locate **Dialog name**, then from the drop-down menu, select **HelpDialog**. Now the `help` trigger is connected to the `HelpDialog`.

Enable interruptions

The `get_weather` dialog knows how to get the weather forecast, but it doesn't know how to respond to requests for help. To enable the bot to respond to requests for help while in the `get_weather` dialog, you'll need to configure it to handle interruptions. This will let you call the new help functionality from the `get_weather` dialog, and then once done the conversational flow will seamlessly return to where it left off.

1. Locate the `get_weather` dialog in the bot explorer, then select the `BeginDialog` trigger.
2. In the authoring canvas, locate and select the **Prompt for text** action.
3. In the properties pane, select **Other**, then expand **Prompt Configurations**.
4. Locate **Allow interruptions** and select **true**.

Enabling interruptions in the `get_weather` dialog ensures that your bot consults the parent dialog's recognizer, which allows your bot to respond to help and cancel requests even when in the `get_weather` dialog.

Check point: Did you enable interruptions?

1. Select **Start bot** or **Restart bot** in the upper right of the Composer window.
2. In the Local bot runtime manager, select **Open Web Chat**.
3. Type "Cancel" or "Help".

I enabled help and cancel interruptions | I ran into an issue

Display weather information with cards

Now, let's take the information that you're pulling from Open Weather and build a card to display the current weather in your bot.

1. Locate the `get_weather` dialog in the bot explorer, then select the `BeginDialog` trigger.
2. Locate the **True** branch in the authoring canvas, then select **Send a response**.
3. In the properties pane on the right, select +, then select **Attachments**.
4. You should see a new tab called **Attachments**.

5. Select **Add a new attachment**. In the drop-down menu, select **Create from template**, then select **Thumbnail card**.

6. In the code editor that appears, replace the content with the following text:

Bot response | Copy

```
[ThumbnailCard
   title = Weather in ${dialog.city} in ${dialog.country}
   text =  It is "${dialog.weather}" in ${user.postalcode} and the temper-
ature is ${dialog.fahrenheit}&deg;F or ${dialog.celsius}&deg;C. Have a
nice day.
   image = http://openweathermap.org/img/wn/${dialog.icon}@2x.png
]
```

Check point: Did you add a weather card?

1. Select **Start bot** or **Restart bot** in the upper right of the Composer window.
2. In the Local bot runtime manager, select **Open Web Chat**.
3. When prompted to enter a postal code, type: "98052".
4. You should see a card with the city, country, weather report, and icon for current weather conditions.

I created a weather card | I ran into an issue

Next steps

- Improve your Composer bot's intent recognition with Natural Language Understanding (NLU)

Recommended content

Build Bot Framework Composer from source (Node.js)

Follow this guide to build the Bot Framework Composer from source. When finished, you'll be able to run the Bot Framework Composer locally as a web application.

Natural language processing in Bot Framework Composer

This article describes the use of natural language processing in Bot Framework Composer.

Configure a proxy server - Bot Framework Composer

How to configure a proxy server to access external resources from Composer.

Tutorial to answer questions with the Bot Framework SDK and QnA Maker - Bot Service

Learn how to add question-and-answer support to bots. See how to use QnA Maker and a knowledge base with a bot so that the bot can answer questions.

Show more