Stage 1: Log in, create the service, and launch tooling

To begin the tutorial, you need to create new instance of the Conversation service and launch the tool.

1 Log in to IBM® Bluemix® and navigate to the Conversation service on IBM® Bluemix®:
- Don't have Bluemix account? Start free .
- Have a Bluemix account? Log in .

2 In the **Service name** field, specify `conversation-tutorial` .

3 Click **Create.**

4 On the "Service Details" page, click **Launch tool.**

Menu

Stage 2: Use workspaces

A *workspace* is a container for all the artifacts that define the behavior of your service.

You configure the Conversation service by using the tool within a workspace. When you click the tile and you don't have a workspace, you are prompted to create a workspace. Your workspaces are displayed as tiles on the Workspaces page.

Tip: Before walking through this tutorial, you might want to import a complete workspace for an existing demo app. The demo app can help you get a feel for the UI and the dialog flow.

If you import the demo workspace, make sure you also create an empty workspace for completing the tutorial.

1. On the Create workspace page, click **Create**.
2. In the **Name** field, type `Car tutorial`.
3. Select **Create**. The new workspace is displayed as a tile on your dashboard.

Menu

Stage 3: Add intents and examples

Add some intents on the Intents tab. Intents are used in your dialog flow.

1 On the Workspaces page, select the *Car tutorial* workspace tile that you created.

2 On the Intents tab, select **Create** and add an intent name:

```
turn_on
```

This intent indicates that the user wants to turn on an appliance such as the radio, windshield wipers, or headlights.

3 In the **User example** field, type an utterance and press Enter:

```
I need lights
```

4 Add these 4 more examples to help Watson recognize the #turn_on intent and hit Enter:

```
Listen to some music
Play some tunes
Air on please
Turn on the headlights
```

5 Repeat the process to create the `#greeting` intent with 5 examples:

```
Hello
Hi
Good morning
Good afternoon
Good evening
```

You defined two intents, `#turn_on` and `#greeting`, with example utterances. These examples help train Watson to recognize these intents in user input.

Menu

Stage 4: Add entities

An entity definition includes a set of entity *values* that can be used to trigger different responses. Each entity value can have multiple *synonyms*, which define different ways that the same value might be specified in user input.

Create entities to represent what the user wants to turn on.

1. On the Car tutorial workspace page, click the Entities tab. If **Entities** is not visible, use the ☰ menu to open the page.

2. On the Entities tab, click **Create** and add an entity name:

```
appliance
```

The @appliance entity represents an appliance in the car that a user might want to turn on.

1. Add a value in the **Value** field

```
music
```

   The value represents a specific appliance that users might want to turn on.

2. Add another way to specify the music appliance entity in the **Synonyms** field:

```
radio
```

3 Click the plus sign **(+)** to define additional values for @appliance.

- Value: `headlights`. Synonym: `lights`.
- Value: `air conditioning`. Synonyms: `air`.

4 Repeat the process to create the @ `genre` entity with 2 values and synonyms:

- Value: `classical`. Synonym: `symphonic`.
- Value: `rhythm and blues` Synonym: `r&b`
- Value: `rock`. Synonym: `pop`

You defined two entities: `@appliance` (representing an appliance the can turn on) and `@genre` (representing a genre of music the user can choose).

When the user's input is received, the Conversation service identifies both the intents and entities. You can now define a dialog that uses intents and entities to choose the correct response.

Menu

Stage 5: Create a dialog

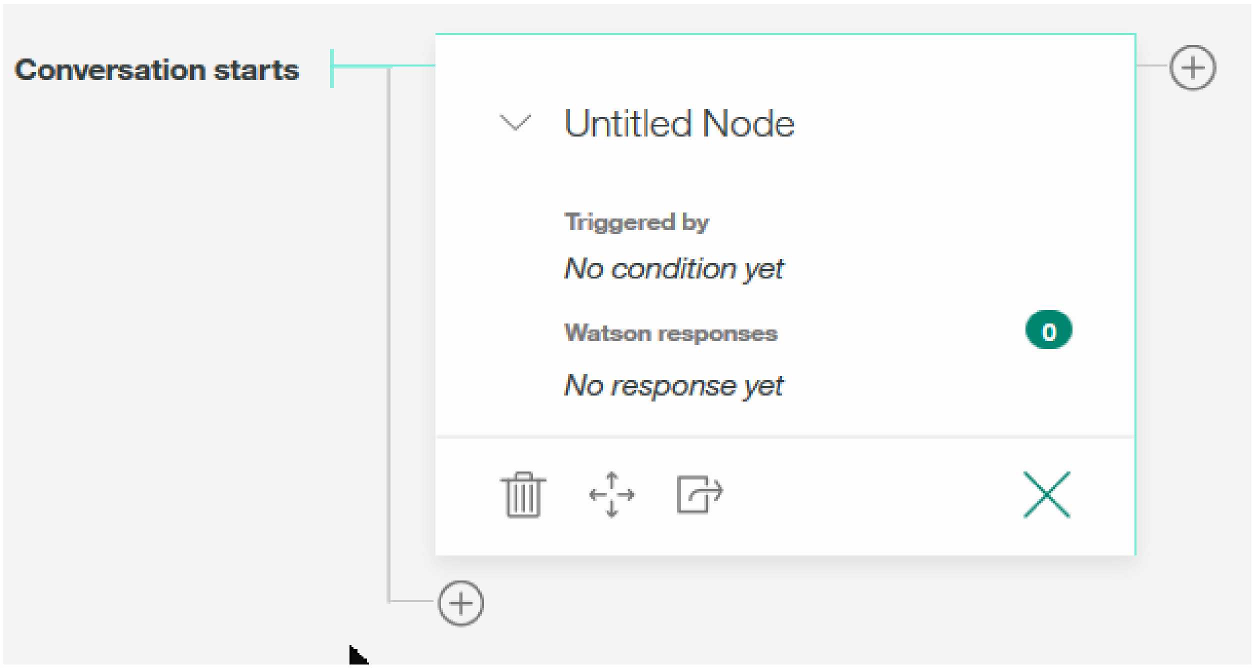
A dialog is a set of conversational nodes that are contained in a workspace. Together the set of nodes makes a dialog tree, on which every branch is a conversation that can be had with a user.

Start the dialog

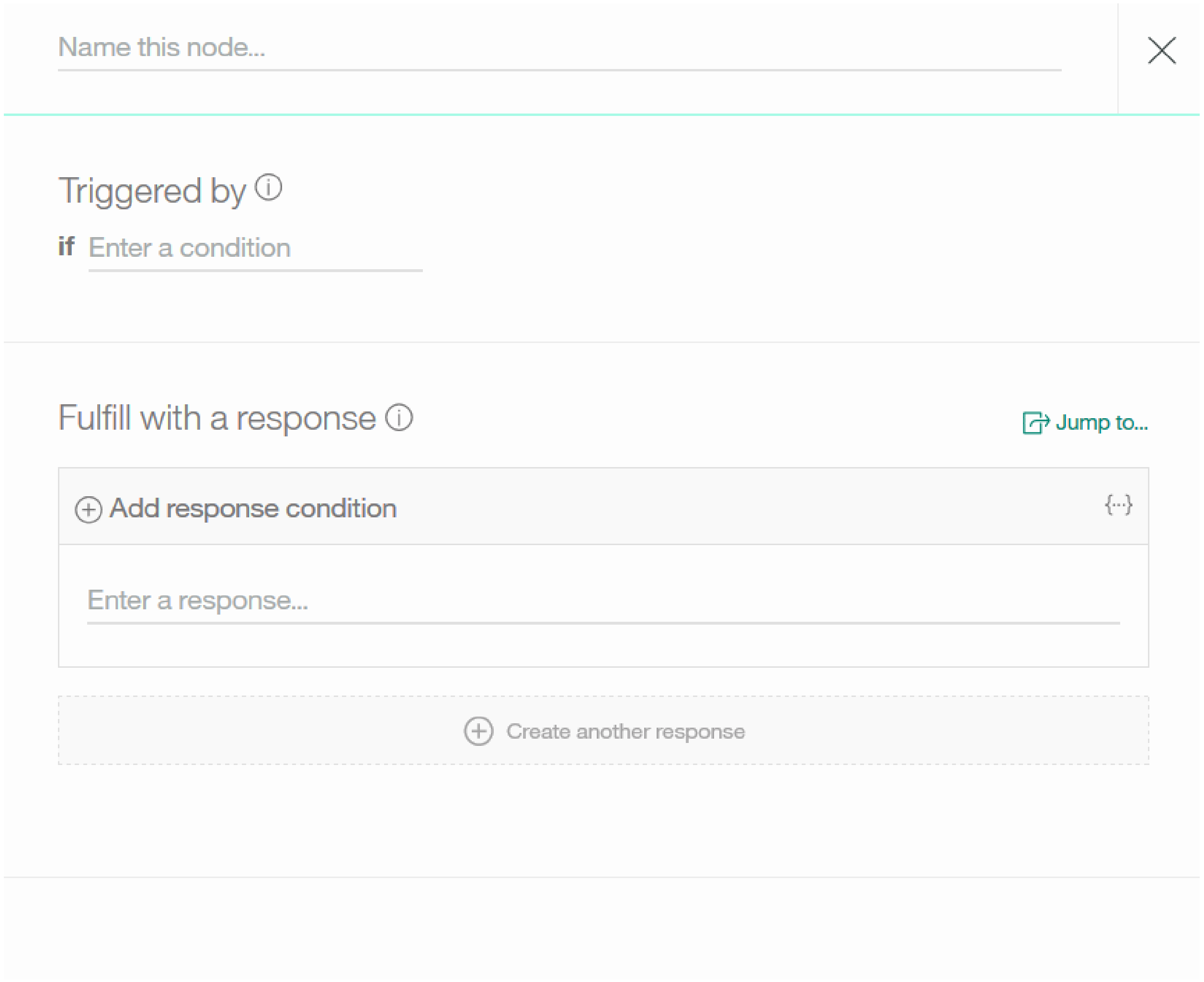
First we need to create a starting node for the dialog:

1 On the Car tutorial workspace page, select the Dialog tab.

2 Select **Create.** The dialog is created with a single node. This node is displayed in the dialog tree:



The edit view of the node is also displayed. This is where you modify the node.


Name this node...

Triggered by ⓘ

if Enter a condition

Fulfill with a response ⓘ

3 In the edit view, specify the condition and response for the starting node of the conversation:

a. In the **Triggered by** field, start typing `welcome` .

b. Select **welcome (create new condition)** from the list. This node is triggered at the beginning of the conversation.

   When you create the condition in your first dialog node, a node with the condition `anything_else` is created in the dialog tree. We'll look at this node soon.

c. Add the first response from your bot in the **Fulfill with a response** field:

```
Welcome to the car demo!
```

Your bot will issue this response when the specified condition is true (in this case, when the conversation starts).

d. Select the `anything_else` node that was created when you defined the conversation_start node.

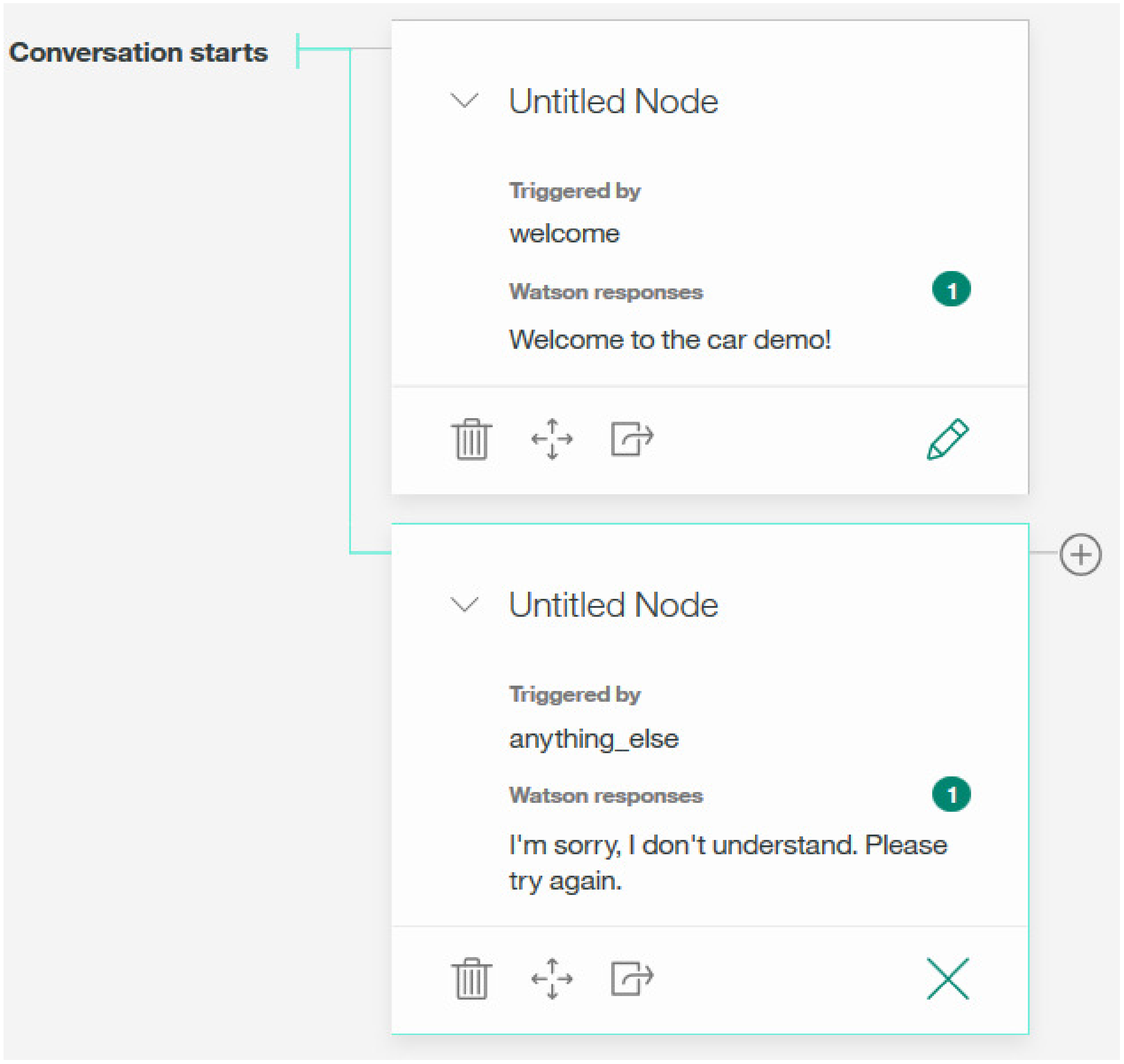
   The edit view now shows the `anything_else` node rather than the welcome node.

e. Add a response in the **Fulfill with a response** field of the `anything_else` node:

   ```
   I'm sorry, I don't understand. Please try again.
   ```

   Your bot will issue this response when the user input doesn't match any other node.

f. Close the edit view. The tree view now shows the updates that you have made to these two nodes:



Conversation starts

Untitled Node

Triggered by

welcome

Watson responses 1

Welcome to the car demo!

Untitled Node

Triggered by

anything_else

Watson responses 1

I'm sorry, I don't understand. Please try again.

Test the initial conversation

1 Select the icon. In the chat pane, you should see the response, `Welcome to the car demo` .

2 Type anything and press Enter.

Because you haven't defined any other nodes, you should see the response, `I'm sorry, I don't understand. Please try again.`

3 Close the chat pane.

Create branches for intents

Now we can create dialog branches that handle the defined intents.

Create a branch to respond to #greeting

The #greeting intent requires a simple response, so the branch has a single node.

1 Select the **welcome** node.

2 Select the **+** icon on the bottom of the node to create a root-level node. Because this new node is a peer of the welcome node (rather than a child node), it represents an alternative conversation. The edit view is opened for the new node.

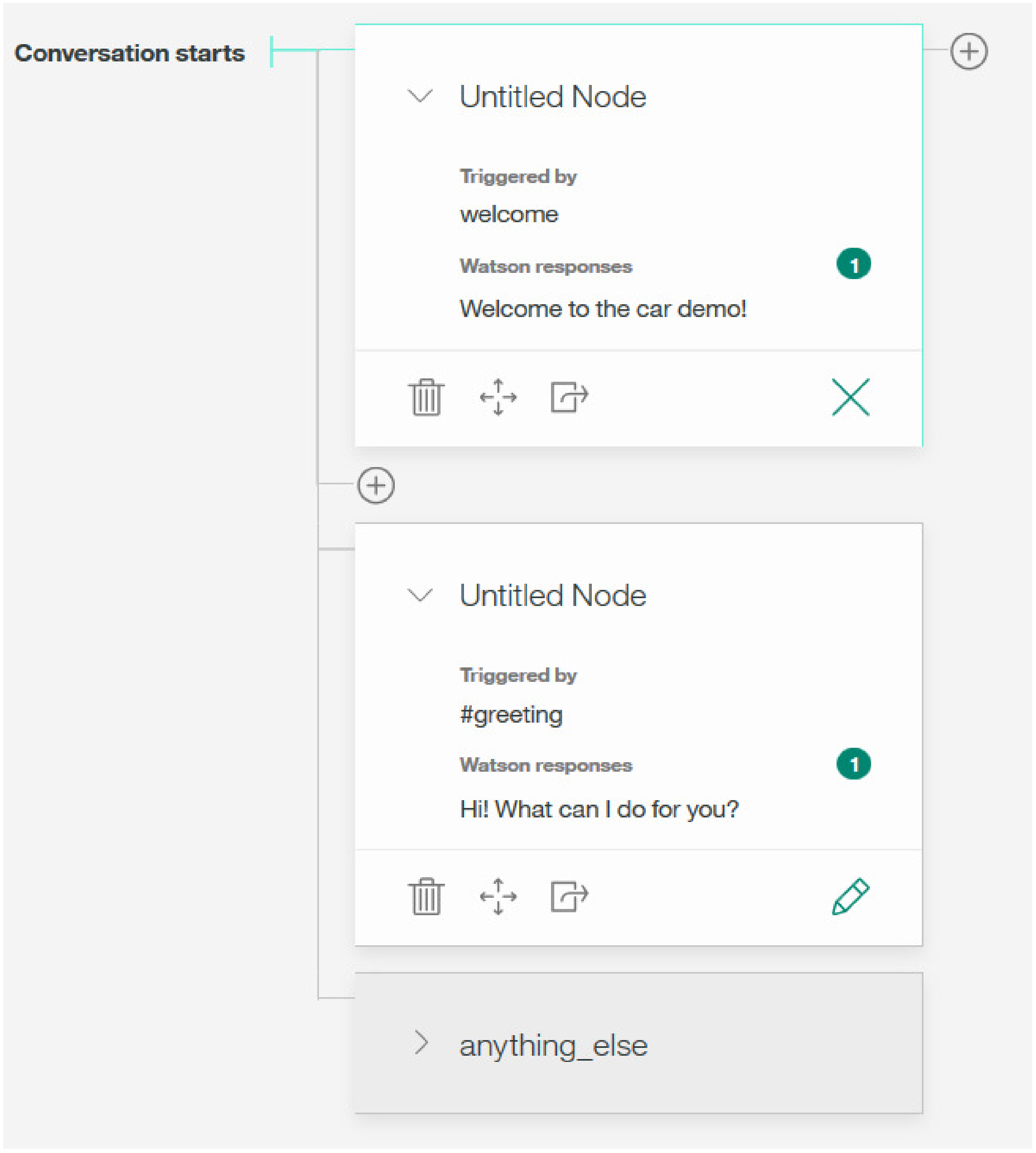
3 In the **Triggered by** field, start typing `#greeting` .

4 Select **#greeting (create new condition)** from the list. This condition is triggered by any input that matches the #greeting intent.

5 Add a response in the **Fulfill with a response** field:

```
Hi! What can I do for you?
```

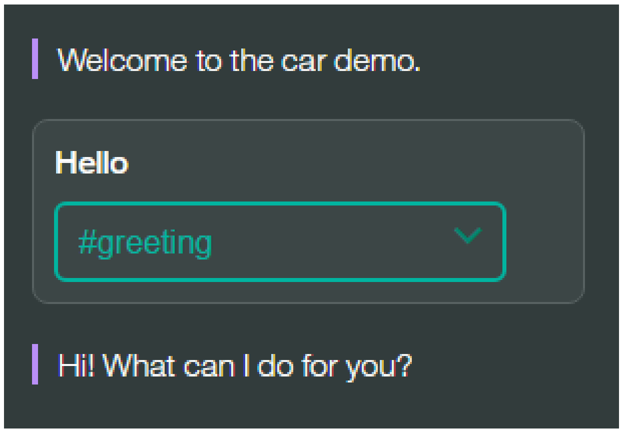
6 In the `anything_else` node, select the minimize icon. This unclutters the tree view by minimizing this node. Here is the tree view now:



Test the first intent branch

1 Select the icon to open the chat pane.

2 Type `Hello` and press Enter.


Welcome to the car demo.

Hello

#greeting

Hi! What can I do for you?

The output shows that the #greeting intent is recognized, and the appropriate response appears.

Add a root-level node for #turn_on

Create another dialog branch to respond to the #turn_on intent.

Because there are multiple possibilities for what the user might want to turn on, this branch represents a more complex conversation.

Start by creating the root-level node:

1. Select the **+** icon on the bottom of the `#greeting` node to create a root-level node. If the **+** icon isn't visible, select the `#greeting` node to put it into focus.
2. In the **Name this node** field, enter `turn on`. The title does not affect the processing of the node, but it makes it easier to find.
3. In the edit view, in the **Triggered by** field, start typing `#turn_on`.
4. Select **#turn_on** from the list. This condition is triggered by any input that matches the #turn_on intent.
5. Do not enter a response in this node.

Add multiple child nodes for #turn_on

The #turn_on intent requires additional processing, because the dialog needs to determine which appliance the user wants to turn on. To handle this, we create multiple responses based on additional conditions. There are three possible scenarios, based on the intents and entities that we have defined:

1. The user wants to turn on the music, in which case we need to ask for the genre.
2. The user wants to turn on any other valid appliance, in which case we simply echo the name of the requested appliance in a message that indicates that we're turning it on.
3. The user does not specify a recognizable appliance name, in which case we need to ask for clarification.

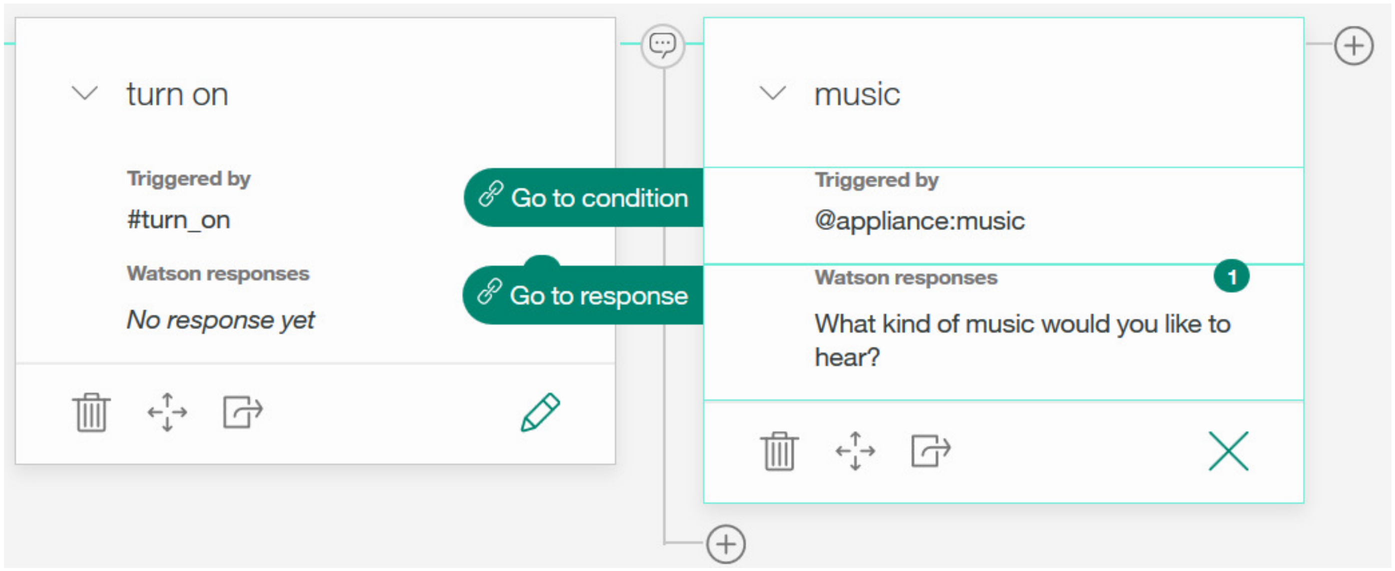
We'll check the conditions in this order. Determining the most efficient order in which to check conditions is an important skill in building dialog trees. If you find a branch is becoming very complex, check the conditions to see whether you can simplify your dialog by reordering them. It's often best to process the most specific conditions first.

To check the input, add a child node:

1. Select the **+** icon on the side of the `turn on` node to create a child node.
2. In the **Name this node** field, enter `music` .
3. Under **Triggered by**, enter `@appliance:music` . This condition is true if the value of the @appliance entity is `music` or one of its synonyms, as defined on the Entities tab.
4. Under **Fulfill with a response**, enter `What kind of music would you like to hear?`
5. Exit the edit view of this node.

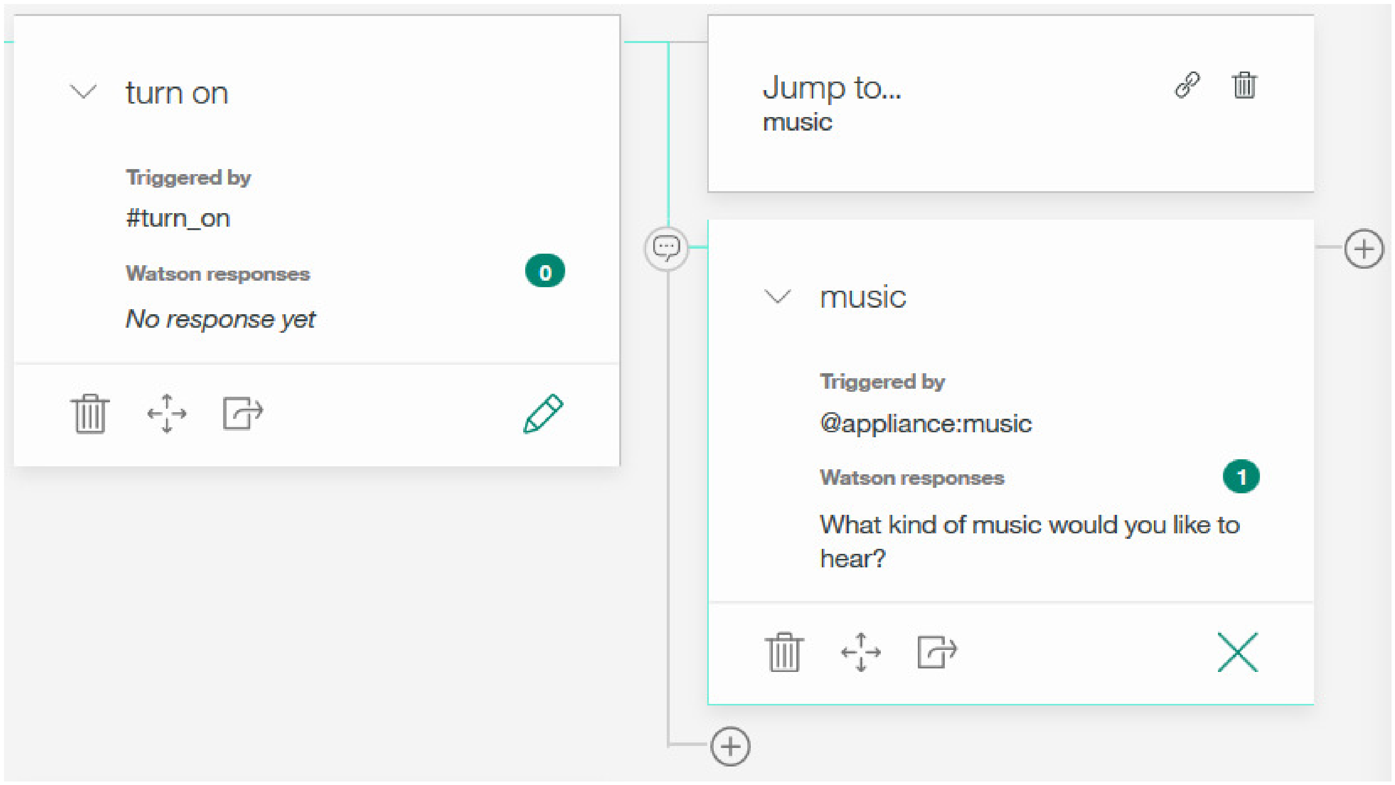
We want to jump directly from the `turn on` node to the `music` node without asking for any more user input. To do this, we use a **Jump to** action.

1. In the `turn_on` node, select the **Jump to** icon ↪.
2. Select the `music` node, and then select **Go to condition**. We want to process the condition of the `music` node.



Note that you had to create the target node (the node to which you want to jump) before you added the **Jump to** action.

After you select **Go to condition** you see a new box in the tree:



Now we need a node to process the type of music that the user requests.

1 Select the **+** icon on the right side of the `music` node to create a child node. This child node is evaluated only after the user has responded to the question about the type of

music. Because we need a user input before this node, there is no need to use a **Jump to** action.

2 Under **Triggered by**, enter `@genre` . This condition is true whenever a valid value for the @genre entity is detected.

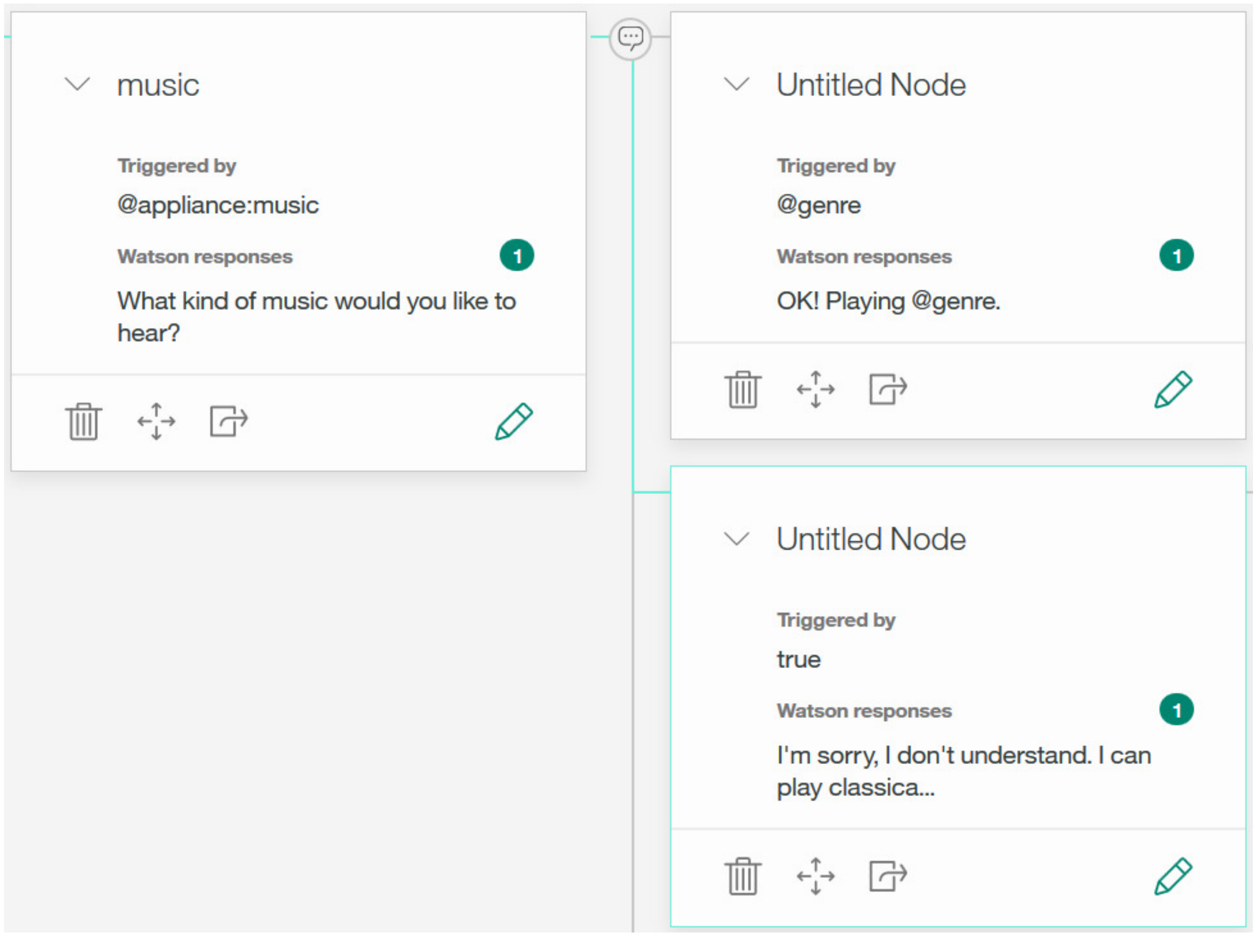
3 Under **Fulfill with a response**, enter `OK! Playing @genre.` This response uses the value that the user entered.

We also need a node to respond when the user does not specify a recognized value for @genre.

1 Select the **+** icon on the bottom of the `genre` node to create a peer node.

2 Under **Triggered by**, enter `true` . This condition specifies that if the dialog flow reaches this node, it should always evaluate as true. (If the user specifies a valid @genre value, this node will never be reached.)

3 Under **Fulfill with a response**, enter `I'm sorry, I don't understand. I can play classical, rhythm and blues, or rock music.`


music

Triggered by
@appliance:music

Watson responses 1
What kind of music would you like to hear?

Untitled Node

Triggered by
@genre

Watson responses 1
OK! Playing @genre.

Untitled Node

Triggered by
true

Watson responses 1
I'm sorry, I don't understand. I can play classica...

That takes care of all the cases where the user asked us to turn on the music.

Test the dialog for music

1 Select the icon to open the chat pane.

2 Type `Play music` . The bot recognizes the #turn_on intent and the @appliance:music entity, and it responds by asking for a musical genre.

3 Type the name or a synonym for a valid @genre value (for example, `pop`). The bot recognizes the @genre entity and responds appropriately.

4 Type `Play music` again, but this time specify an invalid response for the genre.

The bot responds that it does not understand.

Create nodes for other appliances

Next, we'll create a node that is used when the user specifies any other valid value for @appliance. For these other values of @appliance, we don't need to ask for any more input. We just give a positive response.

1. Select the `music` node, so that the options to create child and peer nodes are displayed.
2. Select the **+** icon on the bottom of the music node to create a peer node.
3. Under **Triggered by**, enter `@appliance` .
   This condition is triggered if the user input includes any recognized value for the @appliance entity, except music.
4. Under **Fulfill with a response**, enter `OK! Turning on the @appliance.` This response uses the value that the user entered.

Now add a peer node that will be triggered if the user input did not specify a valid appliance:

1. Select the **+** icon on the bottom of the `@appliance` node to create a peer node.
2. Under **Triggered by**, enter `true` . This condition specifies that if the dialog flow reaches this node, it should always evaluate as true. (If the user specifies a valid @appliance value, this node will never be reached.)
3. Under **Fulfill with a response**, enter `I'm sorry, I don't know how to do that. I can turn on music, headlights, or air conditioning.`

Test the dialog with other appliances

1. Select the 💬 icon to open the chat pane.

2 Type `lights on` .

The bot recognizes the #turn_on intent and the @appliance:headlights entity, and it responds with `OK, turning on the headlights` .

3 Type `turn on the air` .

The bot recognizes the #turn_on intent and the @appliance:(air conditioning) entity, and it responds with `OK, turning on the air conditioning.`

4 Try variations on all of the supported commands based on the example utterances and entity synonyms you defined.

If the bot fails to recognize the correct intent, you can retrain it directly from the chat window. Select the incorrect intent and type the correct one in the field.

Tip: Don't include the `#` character when you type the intent name.

What to do next

Now that your bot is complete, you can experiment by enhancing it with new functions. For example:

- Define entities for additional appliances and musical genres
- Add synonyms for entities
- Add a new intent to turn off appliances
- Add capability for turning on music and specifying a musical genre with a single command

APIs
Docs
Developer Tools