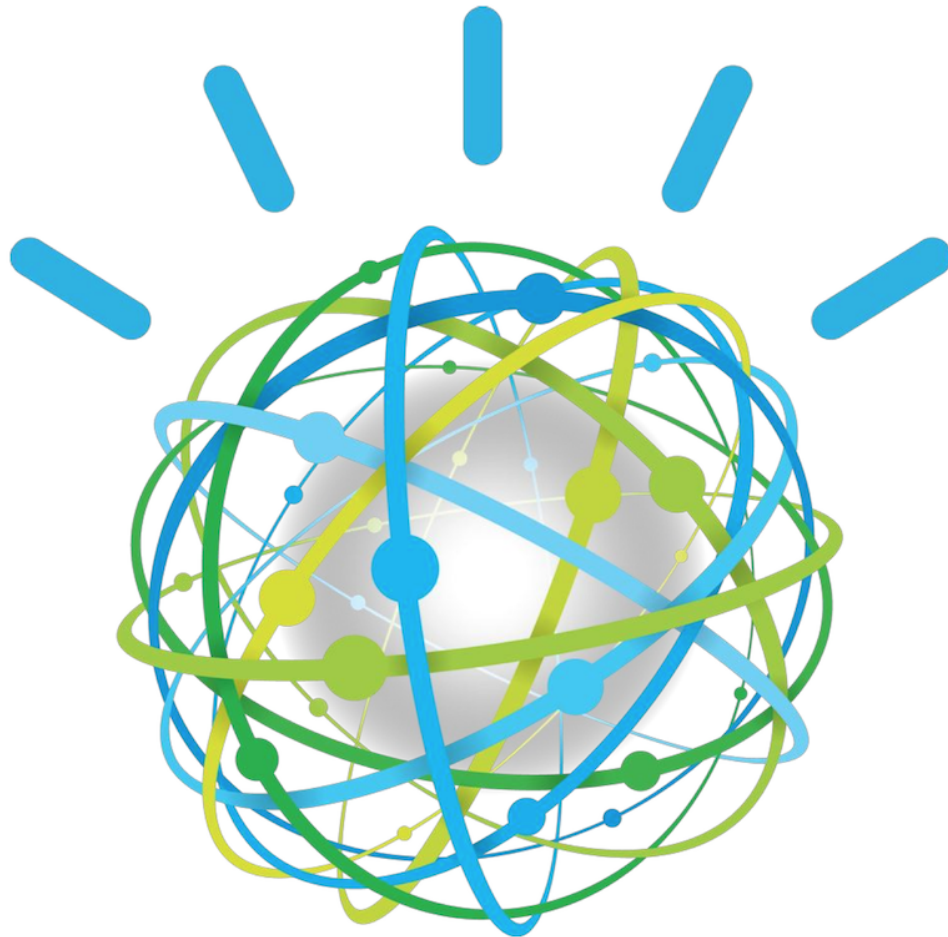


# **IBM Watson Solutions**

## **Business and Academic Partners**



## **Building Advanced Dialog in Watson Conversation Service**

**Prepared by Armen Pischdotchian**

**Version 1.0 November 2016**

# Overview

What is Bluemix you ask? [Bluemix](#) is an implementation of IBM's Open Cloud Architecture, leveraging Cloud Foundry to enable developers to rapidly build, deploy, and manage their cloud applications, while tapping a growing ecosystem of available services and runtime frameworks. You can view a short introductory video here:

<http://www.ibm.com/developerworks/cloud/library/cl-bluemix-dbarnes-ny/index.html>

Additionally, for our academic partners, there are no-charge 12-month licenses for faculty and no-charge 6-month licenses for students - all renewable and NO CREDIT CARD required!

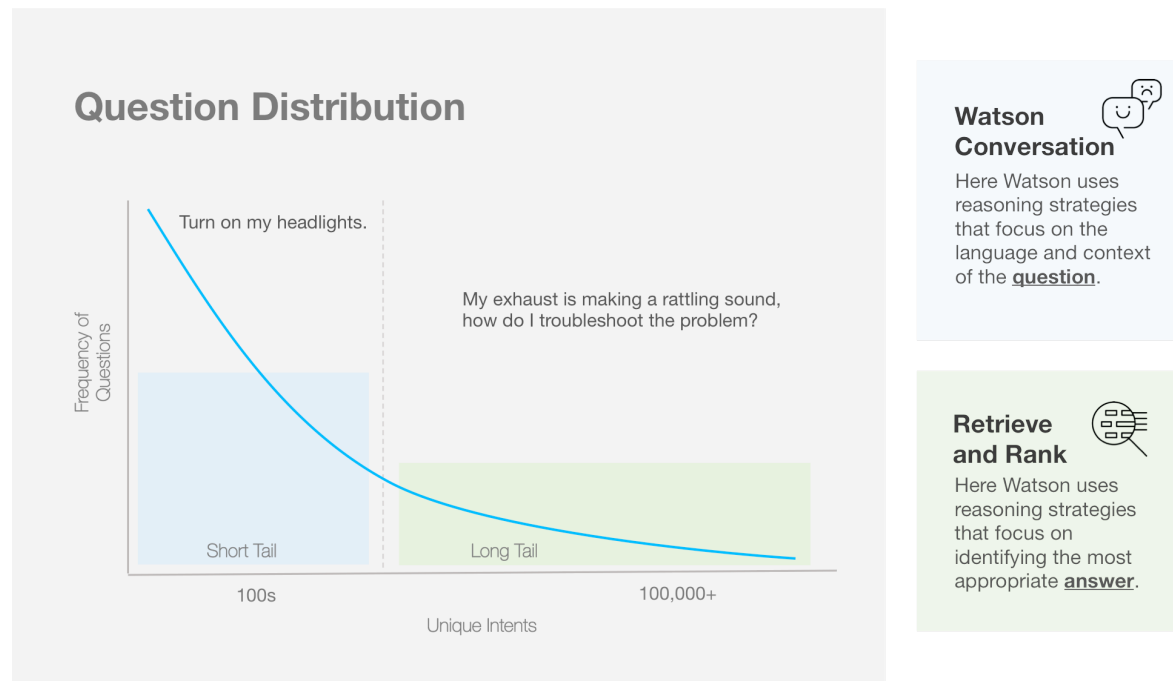
To get started, you will need to become an Academic Initiative member. Refer to this document for details:

[http://it.husc.edu.vn/Media/TaiLieu/IBM\\_Academic\\_Initiative\\_for\\_Cloud\\_Process.pdf](http://it.husc.edu.vn/Media/TaiLieu/IBM_Academic_Initiative_for_Cloud_Process.pdf)

The purpose of this guide is not to introduce you to Bluemix, that foundational knowledge is a prerequisite study on your part and you can obtain it from the links mentioned above. This guide is more of an instructional approach to working with the IBM Watson™ Conversation service where you can create virtual agents and bots that combine machine learning, natural language understanding, and integrated dialog tools to provide automated customer engagements. Watson Conversation provides an easy-to-use graphical environment to create natural conversation flows between your apps and your users. Creating your first conversation using the IBM Watson™ Conversation service entails the following steps:

1. Train Watson to understand your users' input with example utterances: Intents and Examples
2. Identify the terms that may vary in your users' input: Entities
3. Create the responses to your user's questions: Dialog Builder
4. Test and Improve

IBM Watson Conversation service is designed to answer the short tail of typical question distribution per below.



Very noteworthy mention is that this workshop is the brainchild of **Ashima Arora** a software engineer with Watson Dialog services.

It is strongly recommended that you watch this and related videos from the playlist:

<https://www.youtube.com/watch?v=A96nLYSMItA&index=1&list=PLZDyxLINKRY-YOxeg5Hv0IMgQfmKp8ZN4>

In this workshop, emphasizes building a robust dialog from scratch. You will become intimately familiar with the notion of Intent, entity and an in depth dialog tree that uses advanced conditional context within the node.

At the end of this workshop, spend some time and consider what other services you can use to augment a better approach for bringing further cognition to your app; for example, Retrieve and Rank to extract information from specific documents (the long tail), or, you may want to include Speech to Text upfront and Text to Speech for the returned responses. Enjoy the cognitive journey you are about to undertake.

## Prerequisite

For this workshop all you need is a Bluemix account; you will not be building an app, but working within the conversation service in Bluemix to build a detailed dialog tree.

- **Obtain Bluemix credentials:**

1. Direct your browser to the Bluemix home page: <https://console.ng.bluemix.net/home/>
2. Click **Sign Up** on the top right. If you are affiliated with a university, use your edu email.
3. Enter requested information: for **Org**, select the suggestions that are provided for you, for example, your email address; for **Space** name, you can also select the provided suggestions, such as dev; alternatively, you can specify your own values and click **Create Account**.
4. Look for the email confirmation. You will have to login once again into Bluemix after clicking the link from the email.

- **Download the Conversation\_dialog.pdf document from Github**

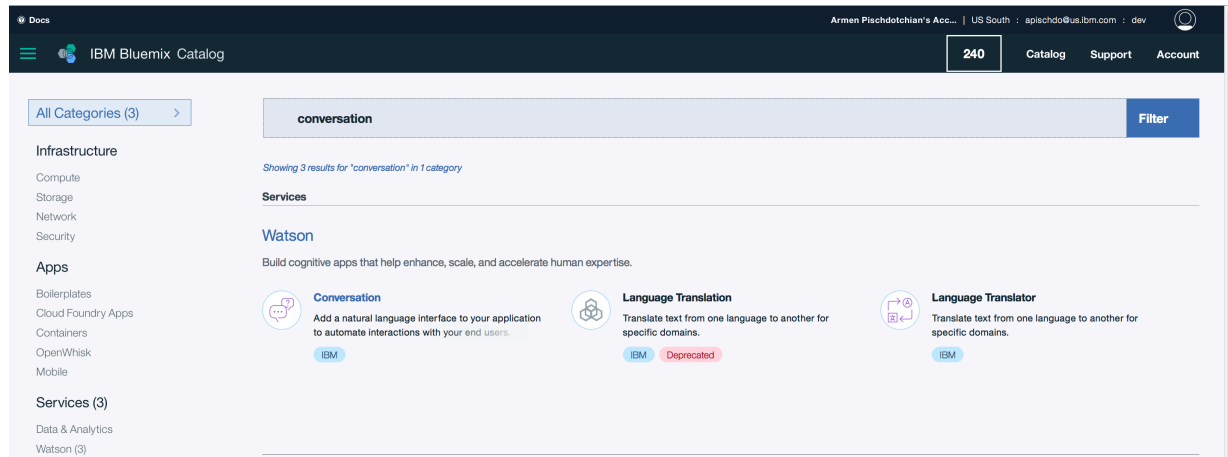
Since this workshop is an advanced approach to dialog building, you could start your journey with a more fundamental workshop that you can also access from the same Github repository per below, except that it is a zip file called ConversationMaster\_expedited.zip.

1. Direct your browser to GitHub (no need to sign up): <https://github.com/>
2. Search for **bluemix-workshop**.
3. Scroll down and select: *apischdo/Bluemix-workshop-assets*
4. Download just the **Conversation\_dialog.pdf** document.
5. Follow the instructions in that document.

# Using the Conversation service

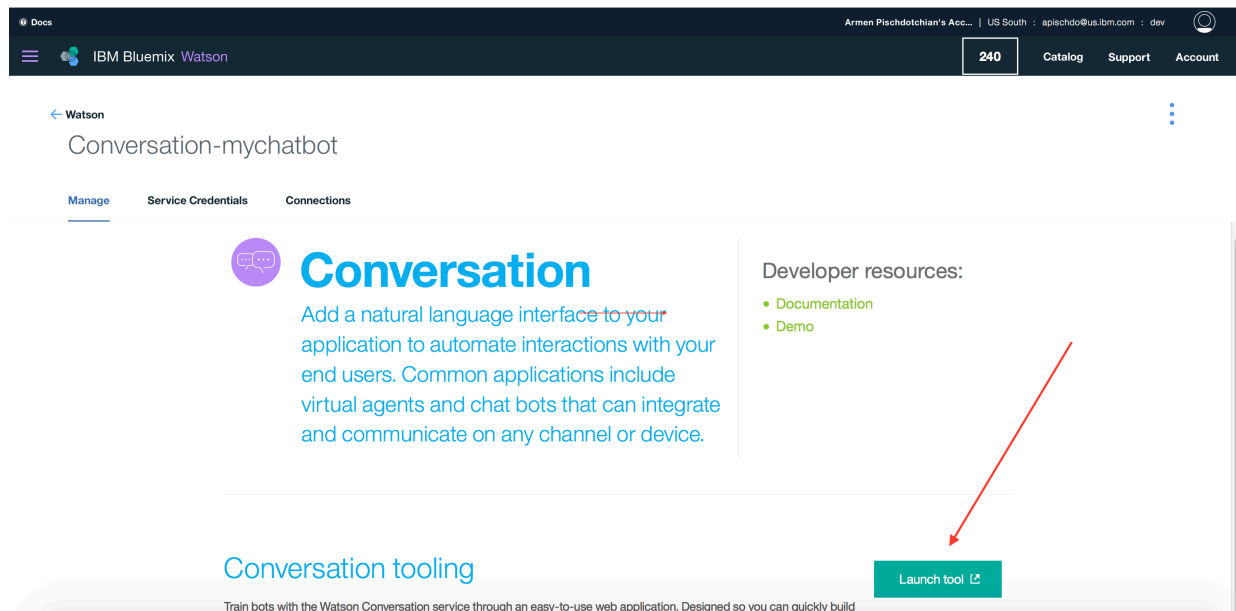
Let's begin our journey:

1. Login into Bluemix: <https://console.ng.bluemix.net>
2. Click the **Catalog** tab.
3. Search for the **Conversation** service and click that tile.



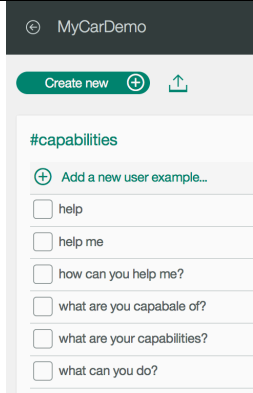
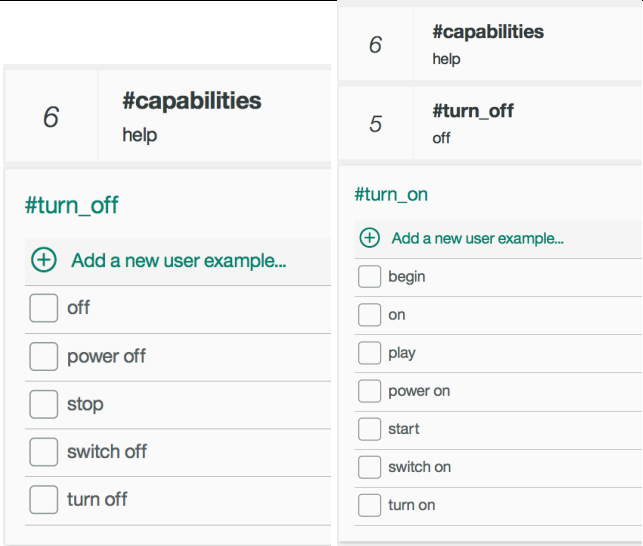
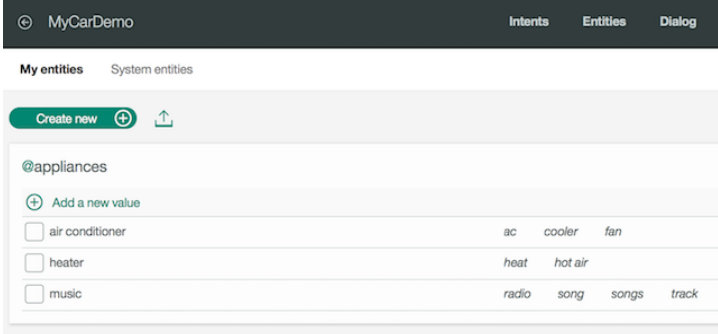
Edit the Service name to something meaningful to you (for example: Conversation-mychatbot) and click **Create** (If you have just created your account and accessed it from the confirmation email, you may need to log into Bluemix once again, then you can see the Create button in the bottom right corner).


4. Click the **Launch Tool** button.



5. Login once again using the same credentials that you use to login Bluemix.
6. Click **Create** to create a new workspace and give it a name, in this example: MyCarDemo
7. Click **Get Started**.
8. Click **Create New**.

Let's use a table format now to capture the steps.

Steps	Example screen capture
<p>9. Add a new <b>Intent</b> and name it <b>capabilities</b>.</p> <p>10. Add a few examples of how you might ask your car what are its capabilities; for example, <i>help me</i>, <i>what can you do?</i> And so forth. Minimum of 5. The more the better. After all, machine learning is about building patterns.</p> <p>11. Click <b>Create</b>.</p> <p>Next you will create two new intents of just what it is that the car can do; for example, turning things on and off is a good place to start.</p>	
<p>12. Add a new <b>Intent</b> and name it <b>turn_off</b> (no spaces).</p> <p>13. Add a few examples of how you might ask your car to turn off certain things; for example: <i>off</i>, <i>power off</i>, and so forth. Minimum of 5. The more the better.</p> <p>14. Click <b>Create</b>.</p> <p>15. Add another Intent and name it <b>turn_on</b> (no spaces).</p> <p>16. Add a few examples of how you might ask your car to turn on certain things; for example: <i>begin</i>, <i>on</i>, <i>play</i>, <i>power on</i>, and so forth. Minimum of 5.</p> <p>17. Click <b>Create</b>.</p> <p>You are now ready to create Entities. Intents are about “what can the system do,” Entities are about what “things” can be done now that the system knows what to do. If the system, knows that your intention is to turn on something, then the entity specifies what those things are: heater, engine, air conditioner, music, all sorts of things that you can turn on or off in a car.</p>	
<p>18. Click the <b>Entities</b> tab and click <b>Create new</b>.</p> <p>19. Type <b>appliance</b>.</p> <p>20. Add new value of <b>air conditioner</b>, plus synonyms such as <i>ac</i>, <i>cooler</i>, <i>fan</i>.</p> <p>21. Add another value of <b>heater</b> and add synonyms such as <i>heat</i>, <i>hot air</i> and so forth.</p> <p>22. Repeat the above steps and add another value of <b>music</b> with synonyms such as <i>radio</i>, <i>song</i>, <i>songs</i>, <i>track</i> and so forth.</p> <p>23. Click <b>Create</b>.</p>	

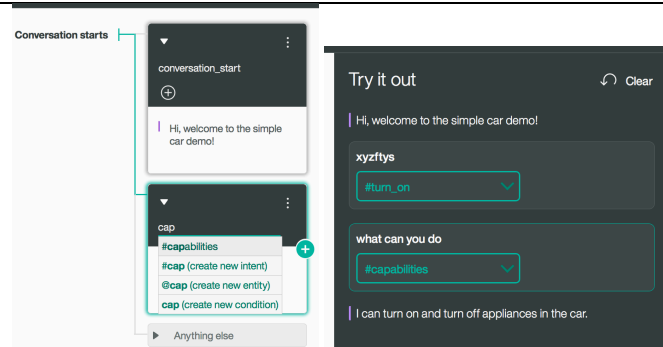
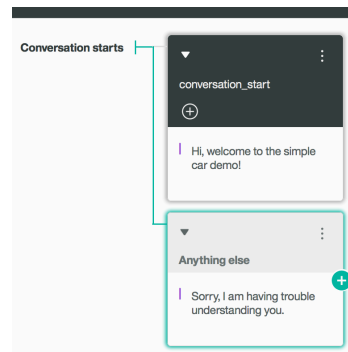
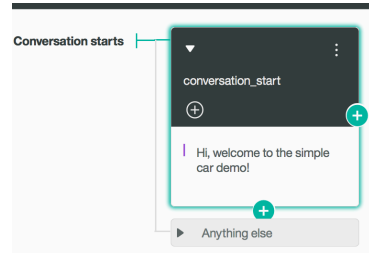
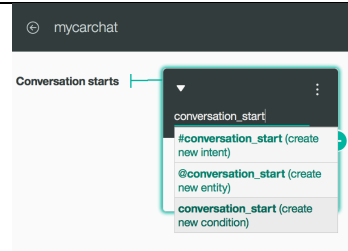
24. You are now ready to create the Dialog tree. Click the **Dialog** tab.
25. Click **Create**. This is a Dialog node, the top half is merely a Boolean condition: true or false. The bottom half is the response that has a simple and an advanced view (where you can include JSON objects to add dynamic output and contextual variables for multi-turn conversations).
26. Specify a new condition (the third option from the drop down) after you type **conversation\_start**. *Anything else* appears by default. Close that description box.
27. In the context section, type: Hi, welcome to the simple car demo.
28. Let's test the node. In this lab, each time you add a node, habitually click the chat icon in the top right corner  to test the conversation.
29. Type something, just bunch of letters to see what happens.
30. So not much happens, because you don't have any other dialog nodes. But the Anything else box opens. That is a catch all node and is always set to true incase the input is not understood. Enter the following text in the Anything else node: **Sorry, I am having trouble understanding you.**

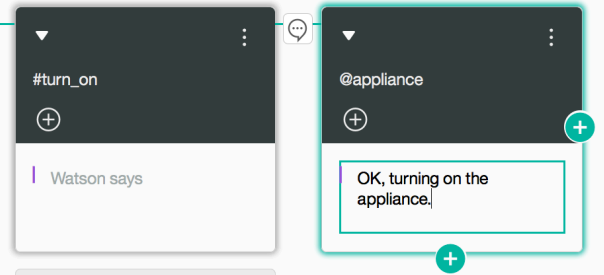
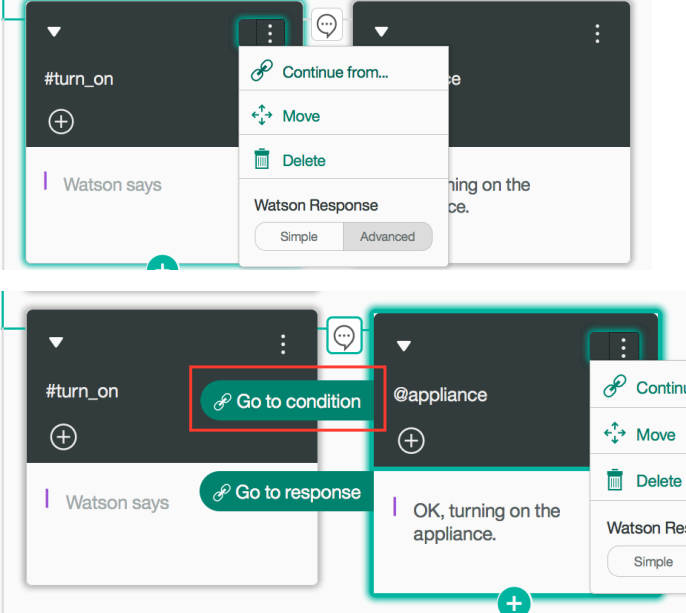
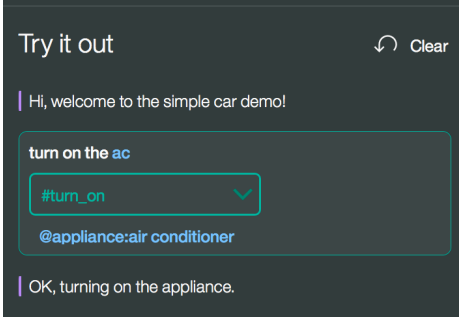
OK, so let's add more Dialog nodes. You will now add sibling nodes (sits underneath, different conversation) and children nodes (goes across, has to do with the same conversation).

31. Create a sibling node (click the plus sign underneath) and type **capabilities**. If you just type *cap...* the rest appears in the drop down and select the first.
32. In the context section, type: **I can turn on and turn off appliances in the car.**
33. Open the chat window and ask the car: **what can you do?**

So when the response came in, the first node *conversation\_start* was false so it jumped to the next node; the *capabilities* node had a condition true, so it executed that condition, hence the response that you got.

34. Clear the chat box and close it.

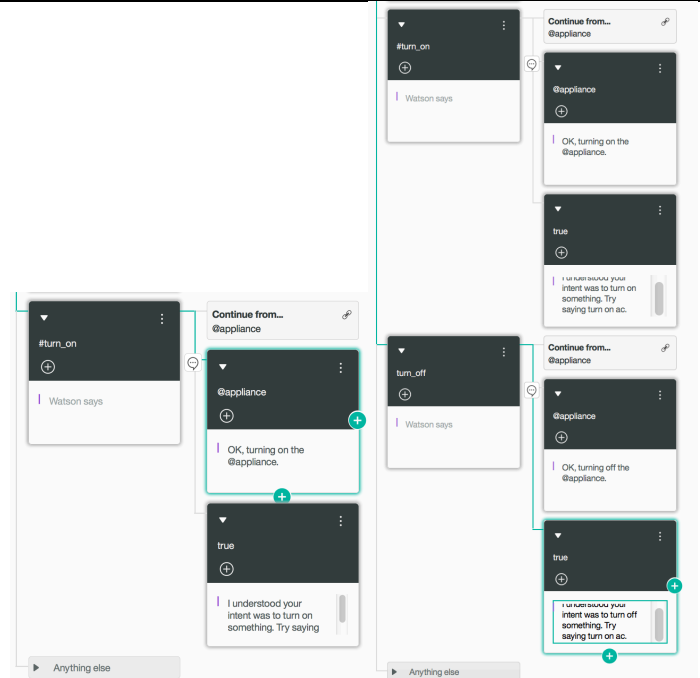


<p>35. Let's build on this dialog by clicking the plus sign under the capabilities node and specify the <b>turn_on</b> condition (the second choice from the drop down).</p> <p>36. Click the plus sign to the left--a child node, because you want to specify turn on which appliance. Because it might have an Intent but not an Entity, so let's condition it on appliance. Pick the first one (just appliance).</p> <p>37. So if there is an intent on an appliance, you asked it to turn on something, then type in the system response in response section: <b>OK turning on the @appliance.</b></p>	 <p>the screen capture is missing the @ in front of appliance. Include the shorthand notation so it knows the value of the appliance.</p>
<p>38. Test the dialog by invoking the chat window. Type: <b>turn on the ac.</b> Notice it gets the intent and the entity right, but no response.</p> <p>The reason is that there is an implicit <i>wait for user input</i> (see the little chat icon in between the two nodes). This means, once the <i>turn_on</i> node is evaluated for true, then it is going to wait for the user input before moving to the child node (<i>@appliance</i>). You overcome this problem by adding a <i>continue from</i>. This is very useful when you need to move to different location in the dialog tree.</p> <p>39. Click the three dots inside the <i>#turn_on</i> node and click <b>Continue from...</b></p> <p>40. So you either continue from a <i>condition</i> or from <i>response</i>; click the three dots in the <i>@appliance</i> node and select <b>Go to condition</b>.</p> <p>Ok let's take a moment here: you can continue from the condition part of a dialog node (the top half) or the response part of the dialog node (the bottom half) as well as continuing from a user input. This means you can first evaluate the condition of appliance and give a response from that, or bypass the condition and just give the response that is typed on the bottom half.</p>	
<p>41. Let's test it. Click the chat icon and type: <b>turn on the ac.</b></p> <p>42. Notice the response comes back as expected.</p> <p>So consider this: What will happen if there is no appliance entity? It will evaluate the <i>turn_on</i> node, then be forced to fall back to the root node and if nothing else matches the <i>turn_on</i> condition, then the <i>Anything else</i> node will capture it.</p> <p>So let's add a default dialog of our own as a sibling to the <i>@appliance</i> dialog, such that if nothing matches, it will capture it with better relevance and not go back to the root.</p>	

43. Add a sibling node to the appliance node, set the condition to true (type it and select the third option of *condition*) and type the following in the response section: **I understood your intent was to turn on something. Try saying turn on the ac.**

This is just some text that the car provides when the user does not provide an appliance to do something with.

44. Create the **turn\_off** condition node underneath the turn\_on node
- Create a child node of **@appliance**
  - Ensure it is *Continued from the condition* part of the parent node (Steps 39 and 40).
  - Add a **true** sibling condition node.
  - Copy paste the responses from the turn on node, except have it say turn **off** instead of turn on.
45. Test the dialog and ask to **turn off the ac**. Expect to see *OK, turning off the air conditioner*.

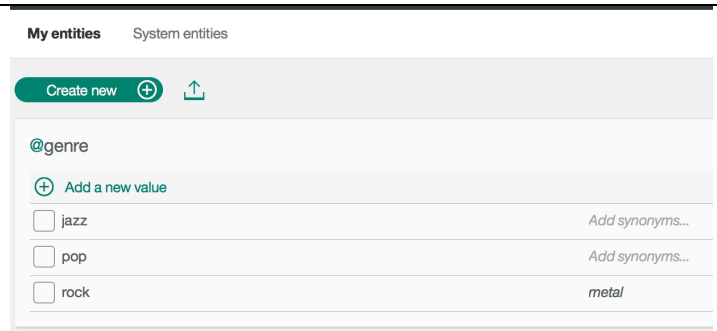


Above you are creating the **turn\_on**....and now the **turn\_off**

Now let's ask the user for a preferred genre of music. This is a special case and we want to handle it differently. Let's say that we have jazz, rock and pop as music genre. Let's go back to Entities.

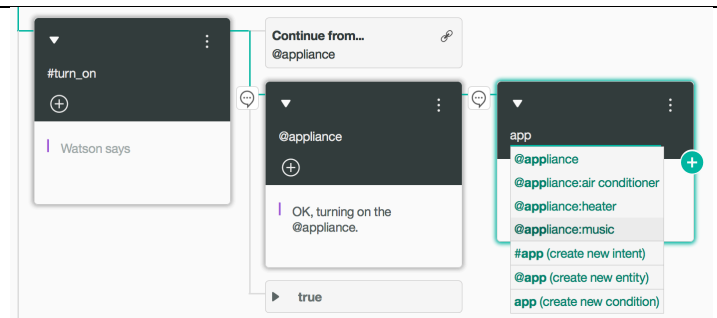
46. Click **Entities**.
47. Add a new Entity of genre with values of **jazz**, **pop** and **rock**; specify some synonyms, for example, *metal* for rock.
48. Click **Create**.
49. Click the chat icon and notice that it is training on the new changes. It will turn green shortly.

Yeap, this is neural nets, deep learning and the name of the algorithm is back propagation. The fact that it takes a few seconds to build patterns and use models (including Support Vector Machines) on a distributed scale is cutting edge technology.

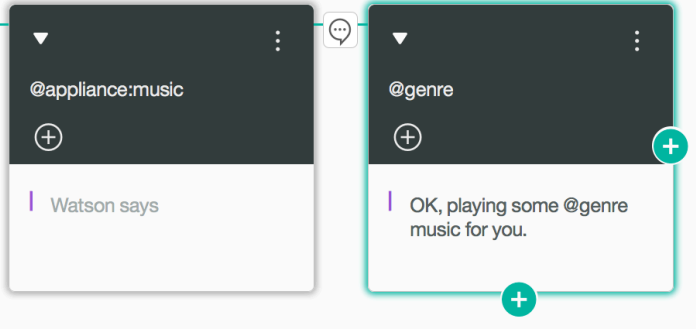
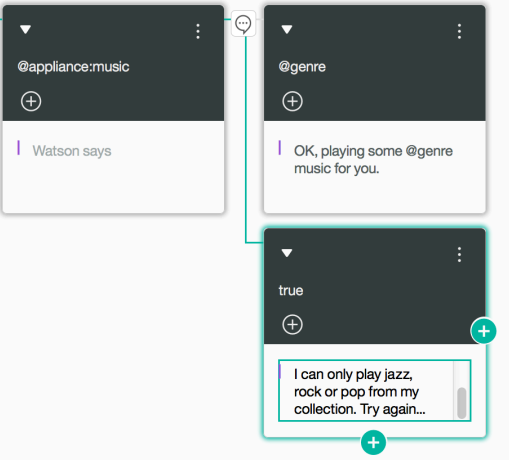
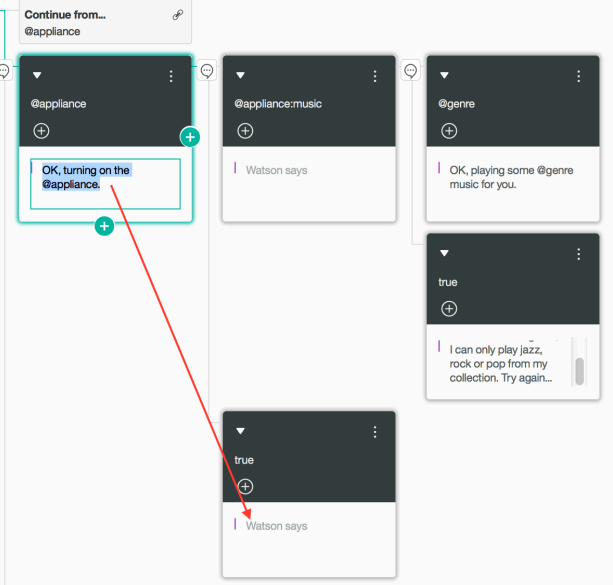


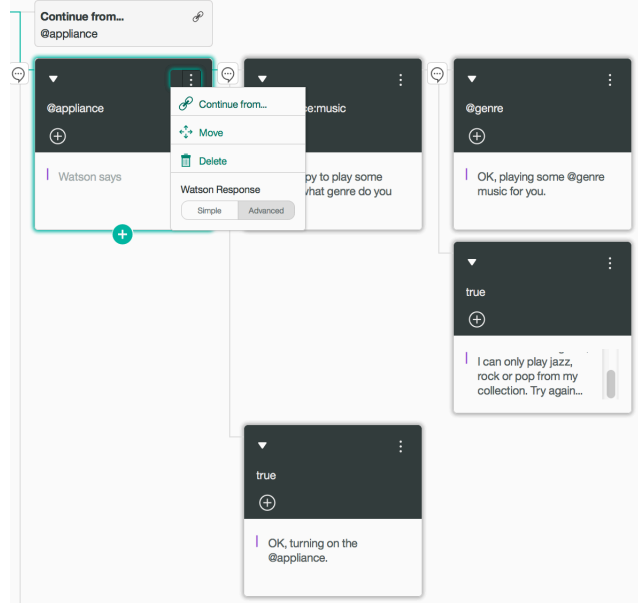
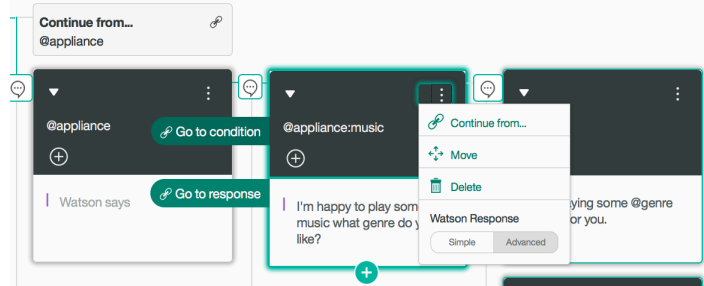
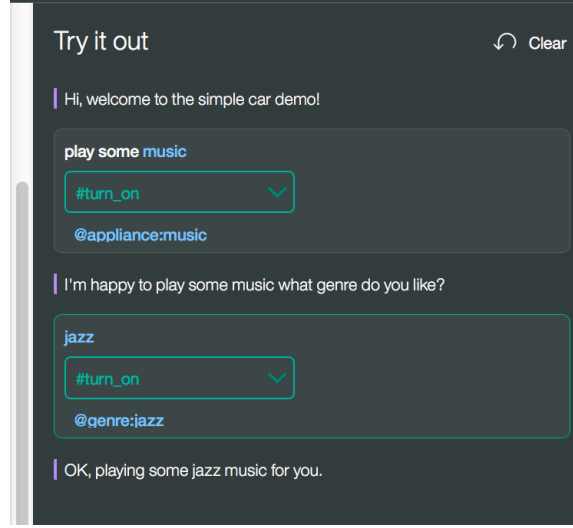
We now go back to the Dialog tab. Since you know that you want to turn something on and that is an appliance (music), then let's begin from the turn\_on->appliance node.

50. From the turn\_on -> appliance node add a child node and specify the condition of **@appliance:music**.





<p>51. And if the condition music is true, then add another child to specify the condition genre by adding a child node to the <code>@appliance:music</code> node and type <code>@genre</code>.</p> <p>52. Add the following response to the genre condition: <b>OK, playing some @genre music for you.</b></p> <p>This is just a dummy message, but this is where you use some JSON code to interact with the client app and play actual music of a certain genre from Spotify or Amazon Music and so forth.</p>	
<p>53. Similar to Step 43, if the genre is not understood by the system and the condition is true, then you want to capture a poignant message here.</p> <p>54. Add a sibling node of condition <code>true</code> and include the following response: <b>Sorry, I don't understand that genre, I can only play jazz, rock or pop from my collection. Try again...</b></p>	
<p>55. Repeat the above step and create another <code>true</code> condition for the <b>Appliance:music</b> node.</p> <p>This means, that if the appliance:music is true and the appliance is not music it reaches the <code>true</code> node you just created. Since the intent was to turn on something, (maybe you want to turn on the ac or the heater) then let's move the response from the appliance node to the <code>true</code> condition node.</p>	

<p>56. Cut (not copy) and paste the <i>response</i> from the appliance node to the <b>true</b> node that is a sibling of the <b>appliance:music</b> node.</p> <p>57. And if the appliance:music is true and the system understood (that is, we asked for the correct genre), then the system response for the appliance music node is: <b>I'm happy to play some music what genres do you like?</b></p> <p>58. From the appliance node, click the three dots and add the <b>continue from...</b></p> <p>This is so once the appliance is captured we want to present a prompt to the user. Continued from where you ask?</p>	
<p>59. Click the three dots from the appliance:music node and select the <b>Go to Condition</b> part of this node.</p> <p>This is because we know there was an appliance and we want to condition on what type of appliance was asked by the user.</p>	
<p>Let's test our work.</p> <p>60. Click the chat icon and ask the system to: <b>play some music</b>.</p> <p>It understood our intent of wanting to turn on an appliance and that appliance is music. It is going to prompt us for specific genre of music.</p> <p>61. Type: <b>jazz</b></p> <p>And so it does, well, at least it says it would.</p>	

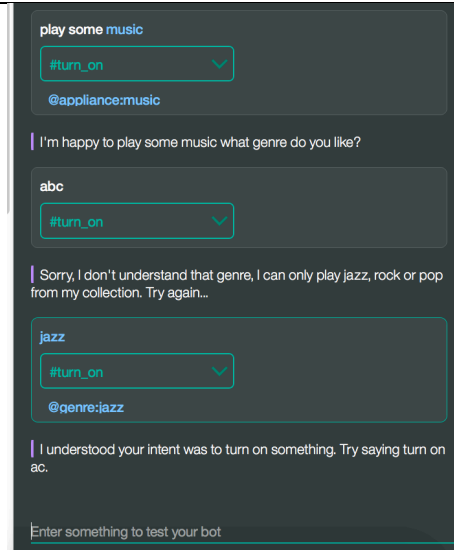
Let's try a slightly different flow.

62. Type: **play some music**.

63. For subsequent response, type: **abc**

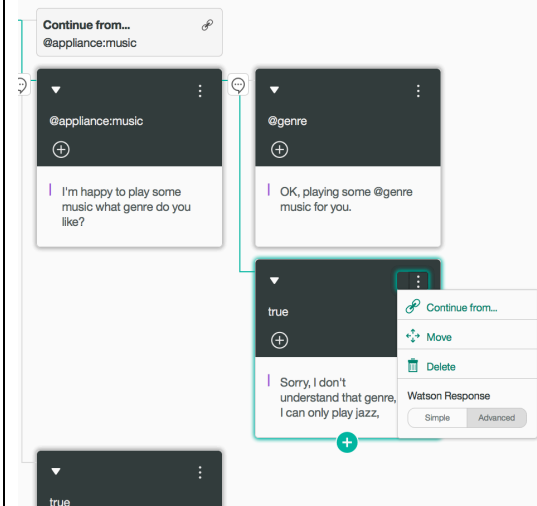
64. Since *abc* is not in it's collection, then type: **jazz**

Notice, it got that the intent right, which is to turn on something, but it lost the context. It went to the leaf node *genre* -> *true* and by default it went back to the root node of *turn\_on*. So you want to include a **Continue from** (from the *true* leaf node) and you want it to continue from *user input* (the bottom half, not the condition).

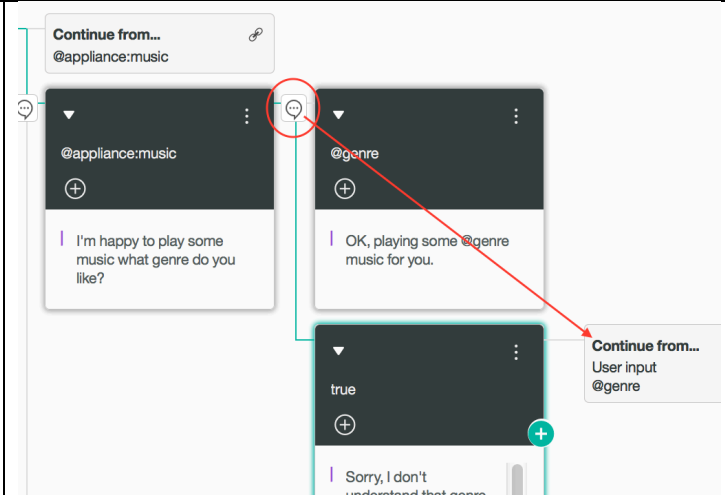


65. From the *true* node, click the three dots and select **Continue from...**

You want to continue from a user input. This means that the system has prompted you for user input and the *continue from* user input means that the system wants to come back for additional user input and condition the dialog at this granular level not from the root node.



66. Click the user input icon between the *appliance:music* and the *genre* dialog nodes.



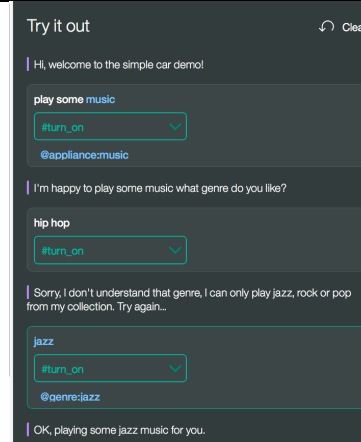
Test the chat flow with similar dialog:

67. Type: **play some music**

68. Type: **hip hop**

69. Type: **jazz**

Notice that it did not loose the context of the conversation and was able to capture the genre condition and not go all the way back to the *turn\_on* condition (node) and get stuck there.



Now let's explore another user input:

70. Clear the chat window.

71. Type: **play music**

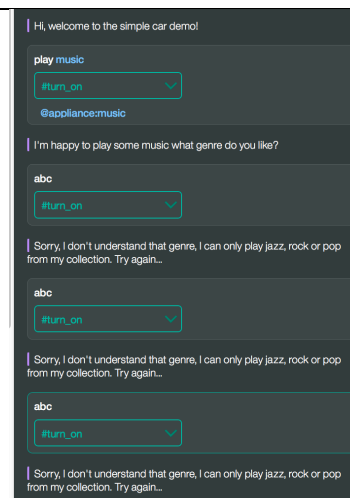
72. Type: **abc**

73. Again, type: **abc**

Notice that it is stuck in a loop. Let's fix that. The best practice to remove this loop is to add a context variable.

We are going to include a context variable that measures how often does the genre -> **true** node variable get's hit.

We want to initialize the **true** node one level higher before it is called by the condition.



74. Click the three dots inside the appliance:music node (this is the one level up from the *true* node to it's lower right) and click **Advanced**.

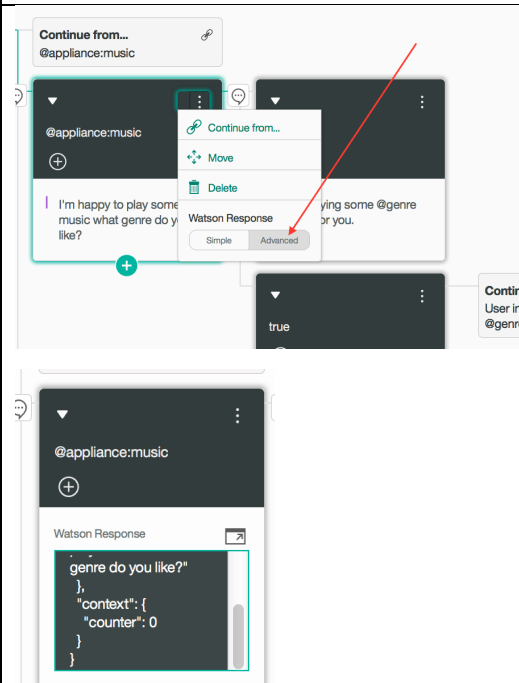
75. Edit the current response syntax:

```
{
  "output": {
    "text": "I'm happy to play some music what genre do you like?"
  }
}
```

to the following syntax (and don't forget that comma):

```
{
  "output": {
    "text": "I'm happy to play some music what genre do you like?"
  },
  "context": {
    "counter": 0
  }
}
```

Super important: ensure that your double quotes are san serif, straight up and down, not curved.



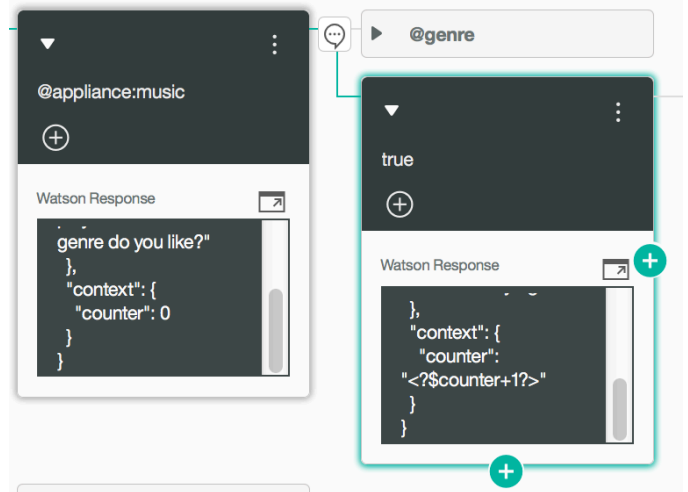
And now you must update the JSON response part in the true node.

76. Open the **Advanced** editor of the true node response.

77. Edit the current response to look like this:

```
{
  "output": {
    "text": "Sorry, I don't understand that genre, I can only
play jazz, rock or pop from my collection. Try again..."
  },
  "context": {
    "counter": "<?$counter+1?>"
  }
}
```

The "<?\$counter+1?>" is the syntax inside a JSON editor where you want to evaluate some expression. This means inside the context variable, set the variable to counter plus one.



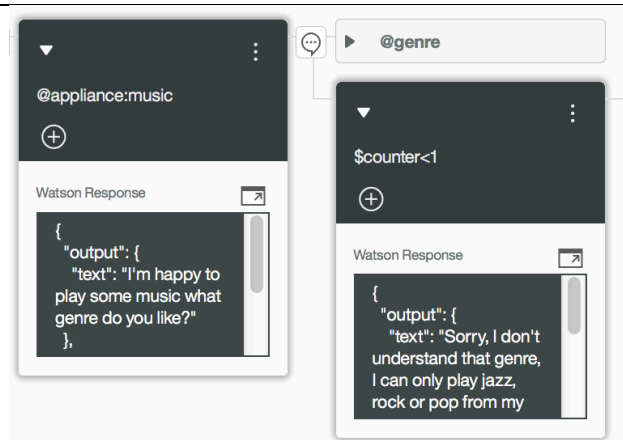
Watch out for the double quotes; they have to be straight up and down!

The expression: "counter": "<?\$counter+1?>" is one line.

Not quite done yet. You must now edit the true variable as follows:

78. Click the true variable text and change it to \$counter<1

This means, the first time the condition hits the appliance:music node it will set the variable to zero and the subsequent times it will increment it by one. We want to have a count of how many times we hit the genre node with the same "not true" response so it does not get stuck in a loop....it has a count of our hits; less than one means if I type genre music that is not in the collection over and over again, then do not fall for my trap again and again.



Let's test the dialog flow again:

79. Type: **play some music**

80. Type: **any**

81. Type: **jazz**

OK so that was normal operation, now let's make it fall into the loop trap by keep typing irrelevant genre of music.

82. Clear the chat and type: **play music**

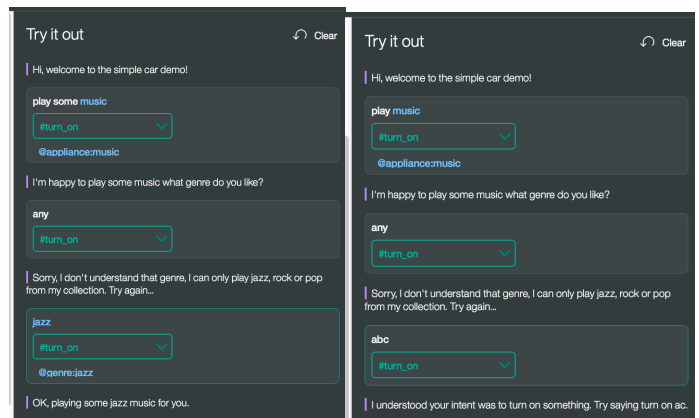
83. Type: **any**

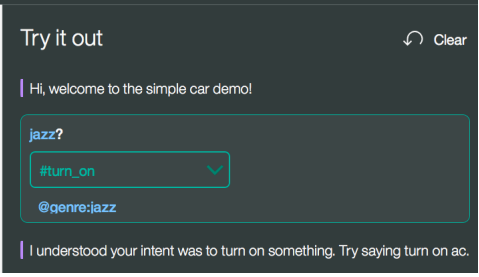
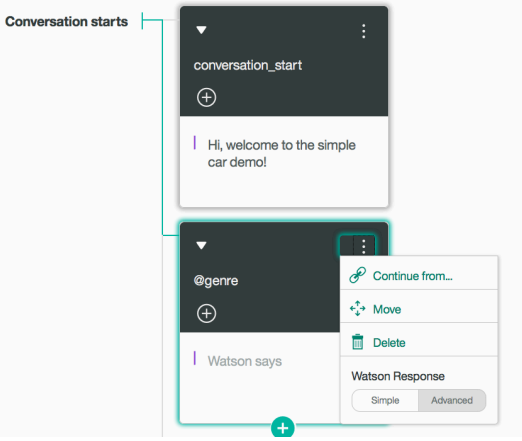
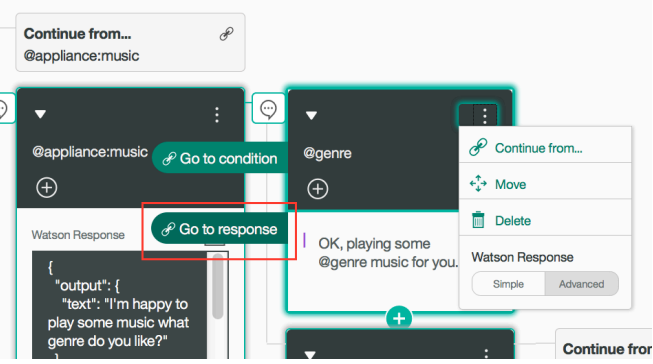
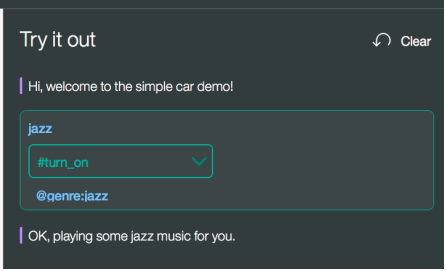
84. Type: **abc**

No more loop right? Otherwise, check your JASON code inside the response sections of both nodes you edited. Usually the biggest gotchas are the curvy double quotes.

This is normal conversation

and this is the loop test



<p>85. Let's try one more scenario. Clear the chat window and at the very start of the conversation type <b>jazz</b>.</p> <p>Notice that it understood the classification (Intent) and got the correct Entity, but what if you want it to play jazz from your first response. In another words, can you start the conversation by conditioning on an Entity instead of Intent first? The answer is YES!</p>	
<p>86. Create a sibling node underneath the <b>Start_conversation</b> node and set the condition to the Entity (not Intent, that we have been doing at the top level) by typing <b>@genre</b>.</p> <p>87. Click the three dots and click <b>Continue from...</b></p> <p>Continue from where you ask?</p>	
<p>88. Open the <b>@genre</b> node all the way from the <b>turn_on</b> parent node and click the three dots.</p> <p>89. Click <b>Go to response</b>.</p> <p>Because you want to display the “OK, playing some @genre music for you” from the very start.</p>	
<p>Test it.</p> <p>90. Clear the chat window and from the start of the conversation type: <b>jazz</b></p> <p>Notice that it jumps straight to the Entity (it is conditioned on Entity instead of Intent).</p>	
<p>Congratulate yourself. In 90 steps you performed some advanced dialog creation techniques.</p>	

And here is the answer to a burning question that you may have: How do I handle chitchat: greetings and good byes and off topic conversation?

The best way to do that is to have a separate workspace that just understands chitchat and understands related intents and responds accordingly and then have the client app to talk to both workspaces; if the main workspace is not able to answer the question, then it can point to the adjacent workspace.