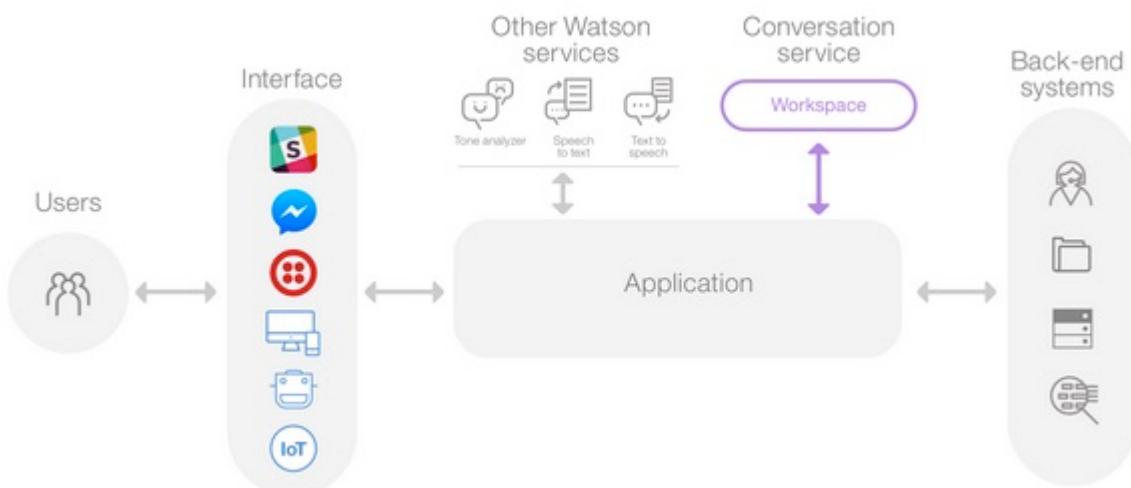


# Watson Conversation

## Overview

In this lab, you will create an Automated Robotic Concierge (ARC) application for a hotel. The application is a chat-bot which the user can interact with whilst in their hotel room. The bot allows simple functions such as controlling the appliances in the room – switching lights on and off for example, as well as more complex functions such as ordering a taxi – which require a conversation to take place. The core of this application is the Watson Conversation service, which has a tool to help you create and test the chat-bot logic. You will use this tool to create and train the ARC chat-bot. You will also build a simple Node-RED application that will provide an interface for the user to interact with the chat-bot. Note that in the full application other services may also be used such as consulting weather or traffic data.



Note: some of the tasks below contain a shortened link to the Bluemix dashboard that opens the US South Region. If you wish to use the UK region instead then substitute the link below

- US South: [http://bit.ly/bluemix\\_new\\_us](http://bit.ly/bluemix_new_us)
- UK: [http://bit.ly/bluemix\\_new\\_uk](http://bit.ly/bluemix_new_uk)

In Watson Conversation, chat-bots are trained using three key concepts:

**Intents.** Example user phrases are grouped into broad “intents” - this allows the chat-bot to understand the underlying meaning behind the users input. Intents are marked with a hashtag. Example:

- #turn\_off – example phrases include “Shutdown”, “I don't need you any more”, “Goodbye”, “I'm done”

**Entities** determine key pieces of information – categories of words and phrases that the chat-bot needs to understand and can use to change its response. Entities are marked with an ‘at’ symbol. Examples:

- @appliance: fan, heater, light, lock
- @genre: classical, pop, rock

**DIALOGS** are based on Intents and Entities and define the flow of the conversation. Dialogs are built using a tree of nodes – each node in the tree represents a single interaction with the user – that is an input from the user followed by a response by the bot:



Dialogs may give simple responses when specific Intents are detected (such as “Clean My Room”) and require no other information. Other responses require an Intent and an Entity – such as “turn on” – which would need to know which appliance to turn on. Dialogs may also form more complex interactions by requesting more information from the user.

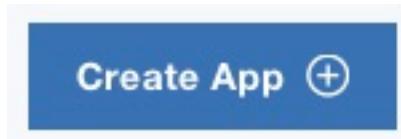
---

## Task 1. Deploy New Application

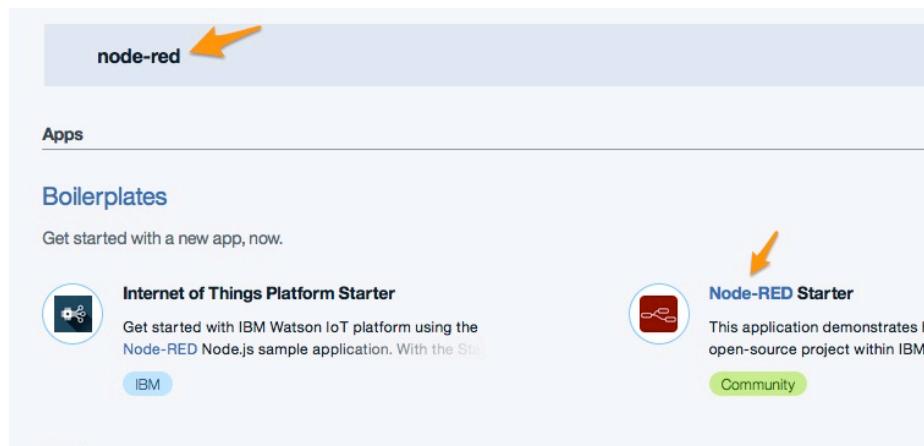
---

In this task, you will deploy a new Node-RED Starter application and add the Watson Conversation service.

1. Deploy new application:
  - a. Open your Bluemix dashboard: [http://bit.ly/bluemix\\_new\\_us](http://bit.ly/bluemix_new_us)
  - b. In the All Apps section, click **Create App**:



- c. In the **Search** bar, type node-red
- d. Click the **Node-RED Starter** Boilerplate:



- e. Give the new application a unique name

**TIP:** Suggestion: IoT-BB-Conversation+YourInitials+DOB

App name:

IoT-BB-Conversation-AL0505

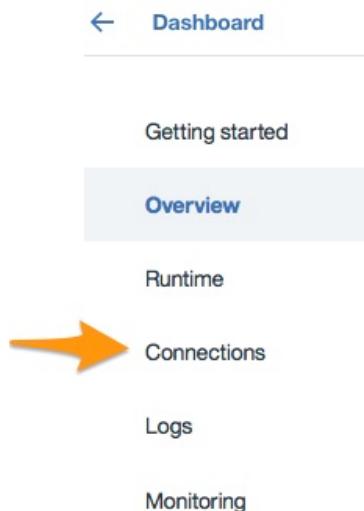
Host name:

IoT-BB-Conversation-AL0505

- f. Click **Create**
- g. Wait for the application to deploy

**TIP:** Check the logs and look for the message “Started Flows”

- 2. Connect a new Conversation Service
  - a. Click the **Connections** tab in the sidebar:



- b. Click **Connect New**
- c. In the sidebar, click **Watson**

The screenshot shows the Watson Catalog interface. At the top, it says "All Categories". Below that is a section titled "Apps" which includes "Mobile". Under "Services", there is a "Data & Analytics" section containing "Watson", "Internet of Things", "APIs", and "Storage". An orange arrow points to the "Watson" button. The "Watson" button is highlighted with a blue border and has a right-pointing arrow icon.

- d. Click the **Conversation** service:

The screenshot shows the "Conversation" service page. It features a circular icon with a hexagonal pattern inside. The title "Conversation" is displayed above the icon. Below the icon, the text reads: "Add a natural language interface to your application to automate interactions with your end users. Common". A blue "IBM" button is located at the bottom left of the card.

- e. Click **Create**:



- f. Click **Restage** to restage the application:

- g. Wait until the application has finished restaging and is running:

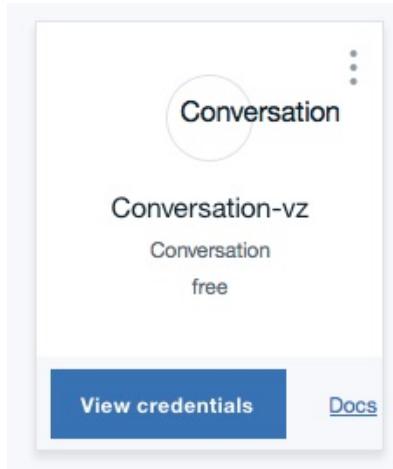
The screenshot shows a status message: "Status: ● Your app is running". The status is indicated by a green circle.

## Task 2. Define Intents

Training data and the flow of the dialog for chat-bots is kept in a **workspace**. A single conversation service instance may have multiple workspaces – one for each chat-bot. In this task, you will create a workspace for the ARC chat-bot and create the first of the three things you need to train the bot – **Intents**. An intent is a group of example phrases a user might use to communicate a specific goal or idea. This helps Watson understand the underlying *intent* behind the user's words.

To define an **Intent**, first identify something the user might want (for example they want the room cleaned or order a taxi) – then give a few examples of how they might express that need. In this first example, you will define two Intents – one that expresses a need for maintenance (something in the room isn't working) and another that will allow the user to control appliances in the room.

1. Open the Conversation Tool:
  - a. Click the **Conversation** service:



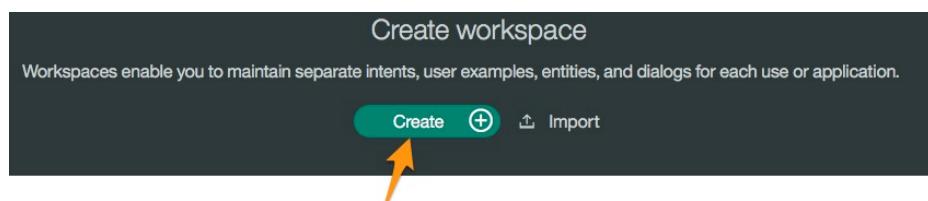
- b. Click **Launch Tool**:



- c. Click **Log in with IBM ID**:



2. Create a Workspace:
  - a. Click **Create**:



- b. In the **Name** field, type ARC:

## Create a workspace

Workspaces enable you to maintain se

Name	<input type="text" value="ARC"/>
Description	<input type="text"/>

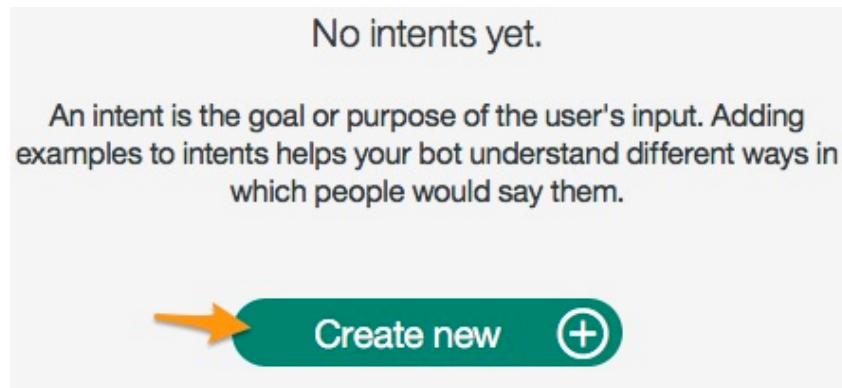
- c. Click **Create**

3. Create an Intent:

*TIP:* Note that the menu bar has tabs to define Intents, Entities and Goals. You are currently on



- b. Click **Create new**:



- c. In the **Intent name** field, type `maintenance`:

The "Intent name" field contains the text "#maintenance".

**TIP:** Note that an Intent is marked using a hash tag (#) – you will use the same notation later in Dialogs when referring to Intents

- d. In the **Add a user example** field, type `my kettle is broken` and then click **Add example**:

The "User example" field contains the text "my kettle is broken". An orange arrow points from the text to the "Add example" button, which is a green button with a white plus sign inside a circle.

**TIP:** Instead of clicking **Add example** you can simply press the **enter** key

- e. Repeat step d using the text `my tv doesn't work`

## #maintenance

User example

Add a user example...

my kettle is broken -my tv doesn't work -

- f. Repeat step d several times using other phrases and questions that could be categorized as relating to wanting room service
- g. When you have entered enough examples, click **Create**:

Intent name  
**#maintenance**

User example  
Add a user example...

my kettle is broken -

my tv doesn't work -

my window will not close -

my windows won't open -

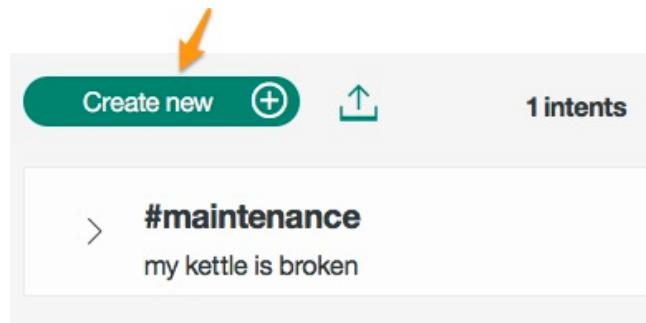
the chair in my room is damaged -

The fan is too noisy -

The shower is leaking -

**TIP:** The service requires at least five examples to begin training

4. Add a second intent:
  - a. Click **Create new**:



- b. In the **Intent name** field, type turn-on:

Intent name  
**#turn-on**

- c. As before, add at least five examples of questions or phrases that could be interpreted as turning on a device or appliance:

**#turn-on**

User example  
Add a user example... **+**

enable the air conditioning **-**

i need the heating on **-**

switch on the tv **-**

the room is too cold - fix that **-**

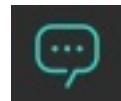
turn on the lights **-**

warm up the room **-**

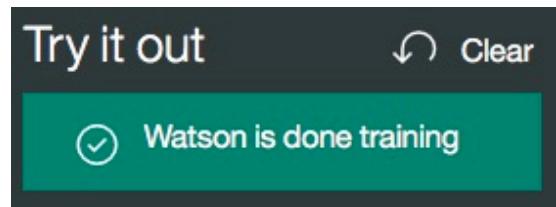
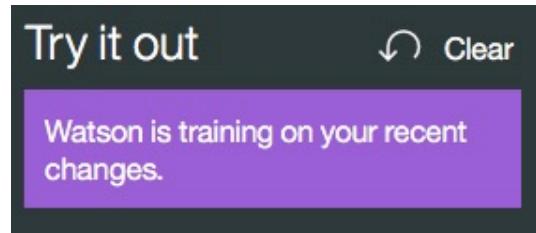
- d. Click **Create**

5. Test the Intents:

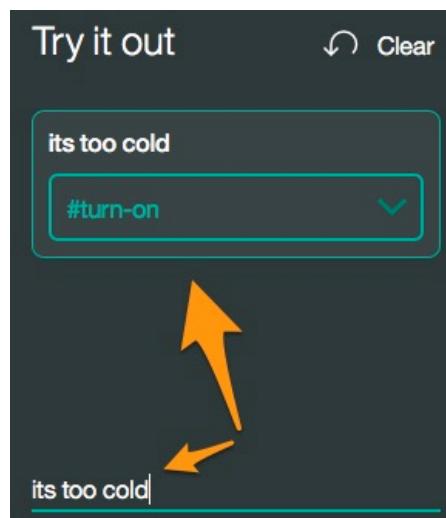
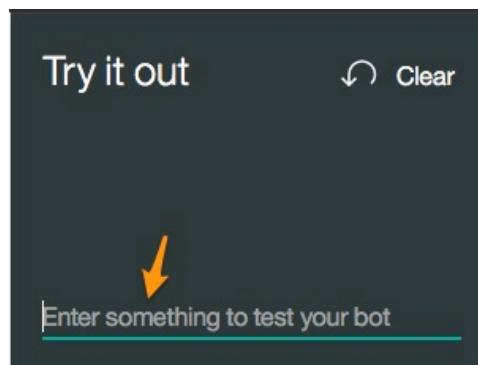
- a. In the top right-hand corner of the page, click the **chat** symbol:



- b. Wait for Watson to finish training itself on your changes:

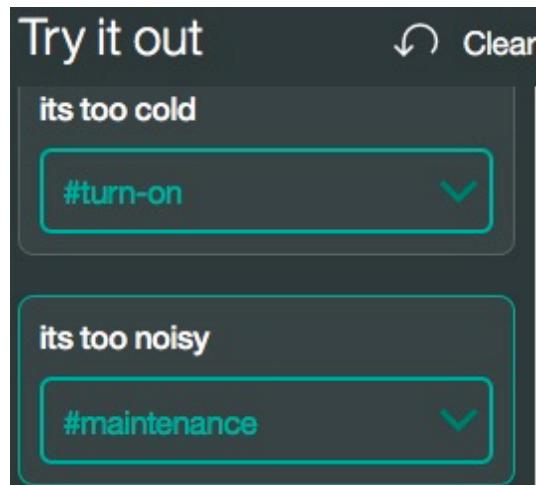


- c. In the Enter something to test your bot field, type `it's too cold` and hit the **enter** key.



**TIP:** If Watson gets it wrong, you can use the drop-down menu to categorize the statement into the correct intent – helping Watson to learn.

- d. Try *its too noisy*



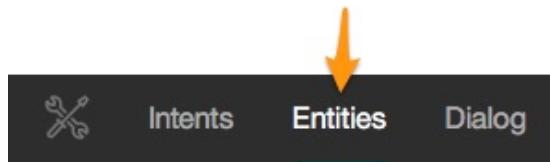
**TIP:** Try a few questions of your own. If Watson gets it wrong, then use the combo box to correct the response.

### Task 3. Define Entities

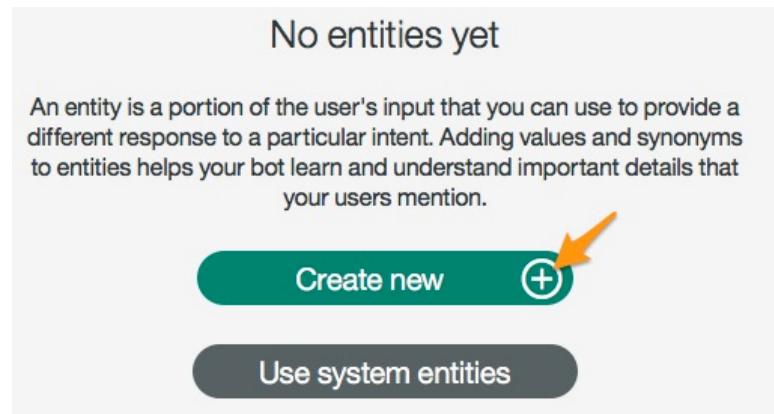
In this task, you will define **Entities**. An Entity represents a term or object (or more accurately the classification of such objects) in the user's input that Watson can use to slightly alter the way it responds to the Intent. Entities make it possible for a single Intent to represent multiple specific actions. For example, the user could turn off the TV, or the Lights in the room. The Intent would be *turn-off*, whilst the TV and Light represent what the user wants to turn off. TV and Light might then be both classified as an *appliance* Entity.

When the user's input is received, the conversation recognizes both Intents and Entities. Your dialog flow can then use these to provide the best answer. You should only create an Entity for **something that matters** in terms of **changing the way the application responds to an intent**. For example, in the case of a maintenance request – the application may not care what the problem is and will simply send someone to the room – in which case there is no need to define the myriad array of items that could be broken or damaged as Entities.

1. Create Entities:
  - a. In the menu bar, click **Entities**:



- b. Click **Create new**:



- c. In the **Entity** field, type **appliance**:

Entity  
**@appliance**

Value  
Add a value, for example, Cat

**TIP:** Note that an Entity is marked using an at sign (@) – you will use the same notation later in Dialogs when referring to Entities

- d. In the **Value** field, type **light**:

Entity  
**@appliance**

Value  
**light**

**TIP:** Use the **Tab** key to move quickly between fields

- e. In the **Synonyms** field, type **lamp** and then press the **enter** key

Value light	Synonyms lamp   X
<a href="#">Add synonyms...</a>	

- f. In the **Add Synonyms** field, type lighting and then press the **enter** key:

Value light	Synonyms lamp   X	lighting   X
<a href="#">Add synonyms...</a>		

- g. Add another synonym - *lights*

**TIP:** You may find <http://www.thesaurus.com> invaluable when adding synonyms ☺. Note that while Watson applies machine learning to Intents – allowing it to interpret what you've said without the need to explicitly define each possible phrase – the same is not true for Entities. If you do not explicitly include the synonym 'lights' then "turn on lights" will not work while "turn on light" will.

- h. When you are done adding synonyms, for this Entity click **Add new value**:

Entity <b>@appliance</b>	Value light	Synonyms lamp   X	lighting   X
<a href="#">lights</a>   X <a href="#">Add synonyms...</a> <span style="float: right; color: orange; font-size: 2em;">+</span>			

**TIP:** Add new value will add another entry into this Entity category.

2. Add another appliance

- a. In the **Value** field, type television:

Entity <b>@appliance</b>	Value <b>television</b>
<span style="color: orange; font-size: 2em;">→</span> light	

- b. In the **Synonyms** field, type tv and then press the enter key

The screenshot shows the 'Value' field containing 'television'. In the 'Synonyms' section, 'tv' is entered into a text input field, which is highlighted with an orange arrow. Below it, three buttons ('lamp', 'lighting', 'lights') are shown with a minus sign icon. A green plus sign button is located at the top right of the synonyms area.

- c. As before, add several more synonyms for 'television'
- d. When you have entered the synonyms for 'television' click **Add new value**:

The screenshot shows the 'Value' field containing 'television'. The 'Synonyms' section now lists 'tv', 'telly', and 'tv set', each with a red X icon. An orange arrow points to the green plus sign button at the top right of the synonyms area.

- e. Add one more Entity – heating with synonyms heat, heater and radiator:

The screenshot shows the 'Value' field containing 'heating'. In the 'Synonyms' section, 'heat', 'heater', and 'radiator' are listed, each with a red X icon. An orange arrow points to the green plus sign button at the top right of the synonyms area.

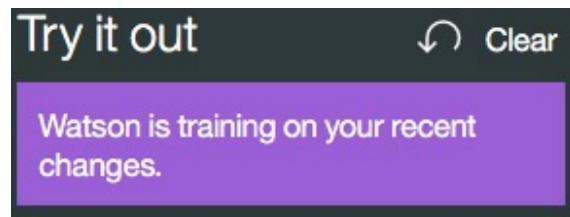
- f. Click **Create**

**Create**

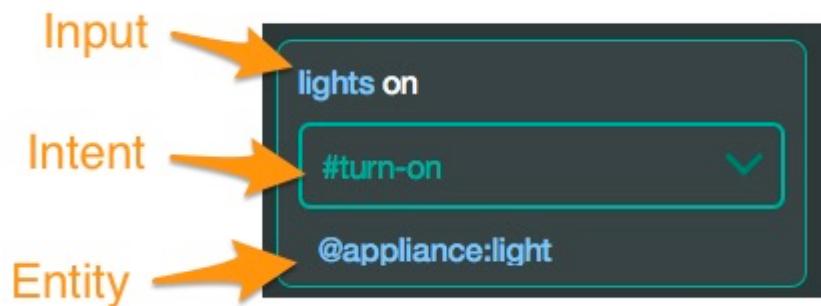
- g. Click the **@appliance** Entity to view it:

The screenshot shows the '@appliance' entity details. It includes a 'Create' button, a 'Delete' icon, and a 'Add a new value' button. The entity has three values: 'heating', 'light', and 'television', each associated with a list of synonyms: 'heat', 'heater', 'radiator'; 'lamp', 'lighting', 'lights'; and 'telly', 'tv', 'tv set' respectively. Each row has a '(3 Synonyms)' link.

- 3. Test the chat-bot:
  - a. Wait for Watson to finish training on your new changes:



- b. As before, type into the chat window to test Watson's responses:



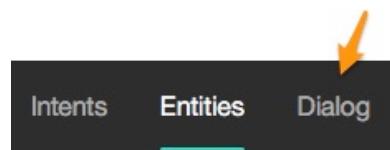
**TIP:** Click Clear to clear the chat window:

**TIP:** Click to close the chat window – you can re-open it at any time by clicking the chat icon:

#### Task 4. Build a Simple Dialog

Dialogs involve *interactions* – that is inputs from the user followed by responses from Watson. Each of those interactions is represented by nodes in a tree. When Watson receives an input from a user it tries to match the input against the interaction nodes in the tree (more on that later). Interactions may be simple or complex. In this example, matching a **#maintenance Intent** in an interaction node is enough information for the application to respond appropriately. In the case of a **#turn-on Intent** – an **Entity** is also required so the ARC application can switch on the appropriate appliance. Later you will see how even more complex interactions can occur – with Watson requesting additional information that it requires to progress the conversation.

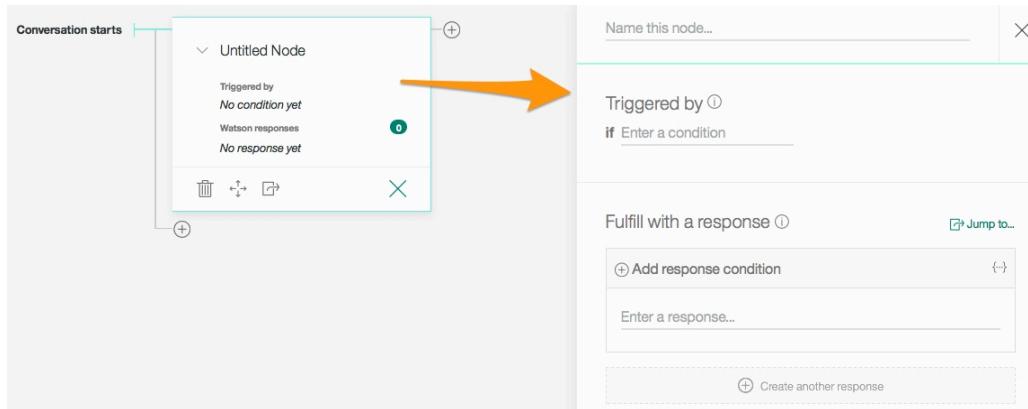
1. Create a Dialog:
  - a. In the menu bar, click **Dialog**:



- b. Click **Create**:



- c. Note that an initial node has been created and it is open for editing:



- TIP:** Dialogs are defined by creating a tree of nodes. When a user input is received, Watson tries to match it against the **child nodes** of the currently active node (the ‘context’) in the tree. At the start of the conversation the currently active node is the virtual “Conversation starts” node – in other words at the start of the conversation Watson tries to match against the top-level set of nodes. Checks are performed sequentially – in the order the nodes appear in the tree – that is top to bottom. As soon as a match is made – that node becomes the context (no more matching checks are made) and the conversation continues from there.
- TIP:** In text based chat-bots it is common to have a single root node that is automatically triggered when the application starts – usually displaying some sort of welcome message. In voice-controlled applications it is more common to wait for a trigger phrase – again this would be a root node in the tree. However, for simplification this example will skip a root ‘welcome’ node and we assume the application is always listening.

## 2. Create a Maintenance node:

- a. In the **Triggered by** field, type `main` and note the **auto-completion** options:

## Triggered by ⓘ

```
if main|
```

#maintenance

main (create new condition)

#main (create new intent)

@main (create new entity)

- b. Use the mouse or keyboard to select #maintenance:

## Triggered by ⓘ

if #maintenance



*TIP:* Remember # denotes an Intent – so this node will be matched if the user input is identified as being a #maintenance Intent

*TIP:* Conditions are expressed using the Spring Expression Language (SpEL):  
<http://ibm.biz/springexplang>

- c. In the **Enter a response** field, type OK I'll send someone right away:

## Fulfill with a response ⓘ

Jump to...

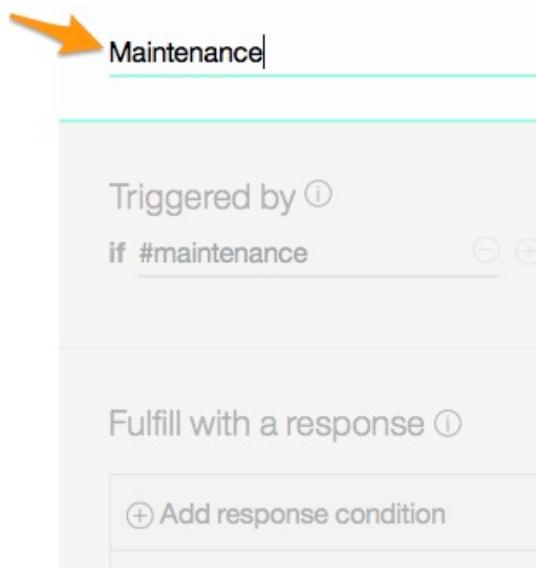
⊕ Add response condition

{...}

OK I'll send someone right away

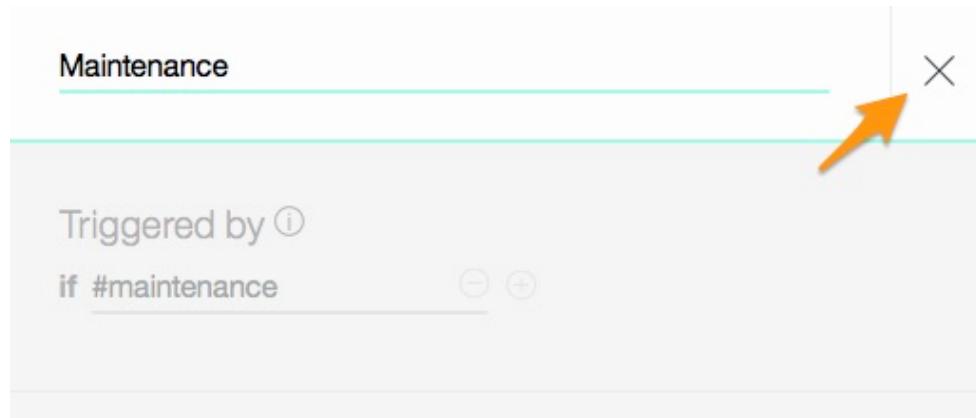


- d. In the **Name this node field**, type Maintenance:



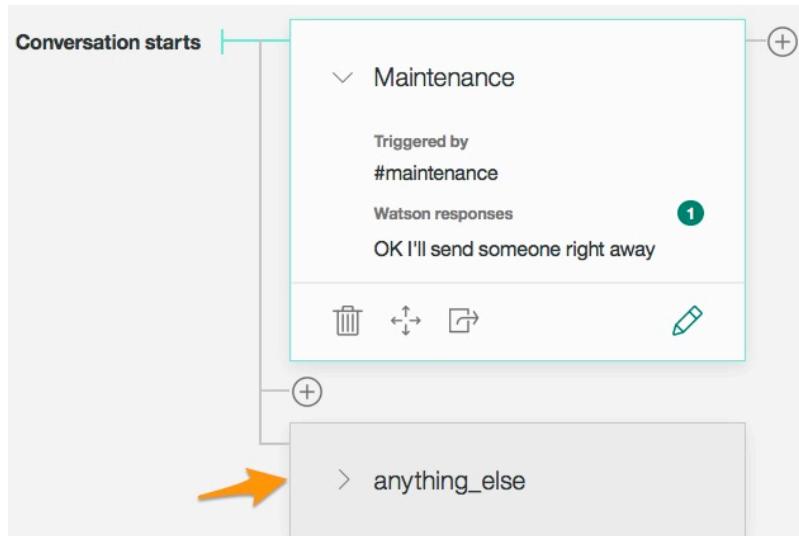
**TIP:** Naming nodes is optional but helps to identify them later.

- e. Click X to close the editor panel:



3. The anything\_else response:

- a. Note that an **anything\_else** node has been automatically added to the root:



- b. Click the **anything\_else** node to edit it
- c. In the **Enter a response** field, type **I'm sorry I didn't understand**:

Triggered by ⓘ

if anything\_else

Fulfill with a response ⓘ

Add response condition

I'm sorry I didn't understand

**TIP:** When user input is received, Watson first tries to match it against the Maintenance node. If that succeeds, then that node is processed – in this case by giving the user the response that someone is on their way. If it does not match, then it moves to the next node and tries to match that – in this case that would be the anything\_else node which has no condition and always matches. Once a ‘leaf’ node (that is a node that has no children) has been processed, the flow of the conversation returns to the root. Since both nodes you have right now are leaf nodes – as soon as either node is processed, the conversation resets.

#### 4. Add alternative responses:

- a. In the **Add a variation to this response** field, type **I don't know how to do that**:

## Fulfill with a response ⓘ

[Jump to...](#)[⊕ Add response condition](#)

1. I'm sorry I didn't understand



2. I don't know how to do that

[Add a variation to this response](#)Response variations are **sequential**. Set to random ⓘ

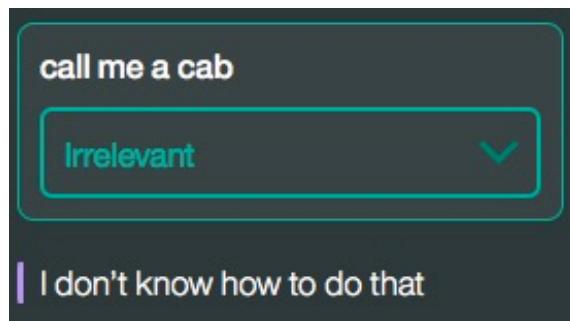
**TIP:** Variations in responses help to create more natural interactions by avoiding repetition. Response variations may be set to *sequential* or *random*.

- b. Click X to close the editor
- 5. Test the bot:
  - a. If the chat panel is closed, then click the chat icon in the top right-hand corner of the page to open it:



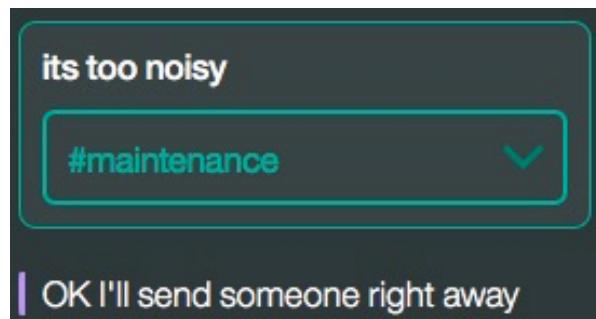
**TIP:** Note that the bot immediately responds with “I’m sorry I didn’t understand”. This is because upon starting, an initial empty input is created. Watson tries to match against the child nodes of “Conversation starts here”. The Maintenance node is not matched but the anything\_else node is. (This is typically why you would have a starting node with a welcome message – see the documentation on Bluemix for the various types of starting node you can use)

- b. In the chat window, type `call me a cab`



**TIP:** The user input does not match the Maintenance node so Watson tries the next node – the anything\_else – this does match and is processed. Since that node is a leaf then Watson returns to the root.

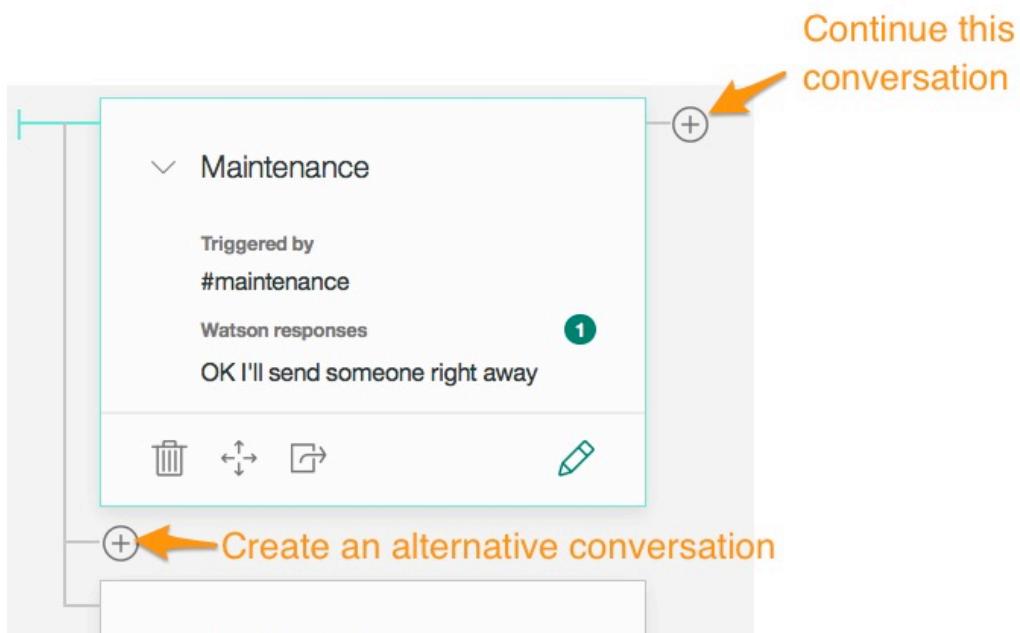
- c. In the chat window, type `its too noisy`



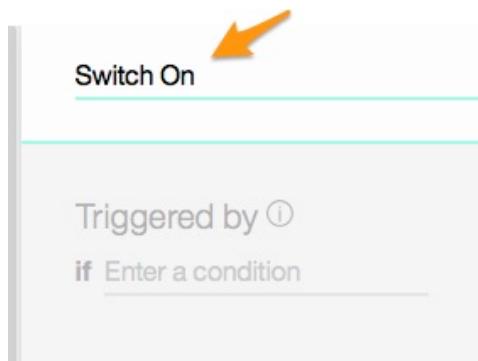
**TIP:** This time the Maintenance node does match and so Watson looks no further – that node is processed.

6. Add an Appliance Control Node:

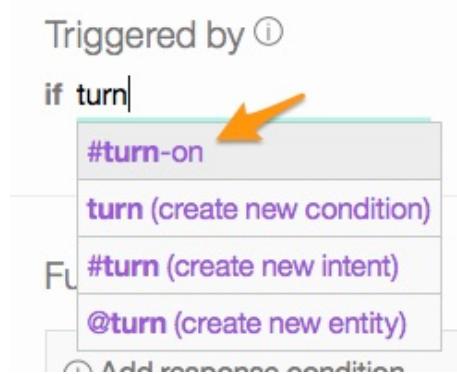
- a. Close the **chat** window
- b. Click the **Maintenance** node in the tree
- c. Click **Create an alternative conversation**:



- d. In the **Name this node** field, type `switch on`:



- e. Click away from the **Name this node** field to apply the change (or press the **enter** key)
- f. In the **Triggered By** field, type `turn`
- g. Select the `#turn-on` Intent:

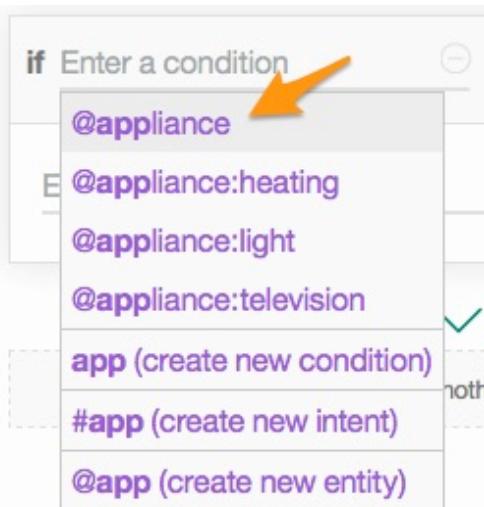


**TIP:** Sometimes when creating Dialogs, you find you need to create a new Intent – note that you can do that directly here (as well as Entities and Conditions). If you do – be sure to switch to the Intent later and provide the alternative phrases.

- 7. Add a Conditional Response:
  - a. Click **Add response condition**:



- b. In the **Enter a condition** field, type `app`
- c. Select the `@appliance` Entity:



- d. In the **Enter a response** field, type OK, turning on @appliance

Fulfill with a response ⓘ

if @appliance

1. OK, turning on @appliance

Add a variation to this response

**TIP:** Note that this response will be used when the input is matched against the #turn-on Intent AND the input is matched against any object in the @appliance Entity. Note that you could also add variation here to avoid repetitive responses.

8. Add Another Conditional Response:  
a. Click **Create another response**:

if @appliance

1. OK, turning on @appliance

Add a variation to this response

→ + Create another response

- b. In the new response, click **Add response condition**:

+ Add response condition

Enter a response...

- c. In the **Enter a condition** field, type app

- d. Select the **@appliance:heating** Entity:

if Enter a condition

@appliance

**@appliance:heating**

@appliance:light

@appliance:television

- e. In the **Enter a response** field, type `It is a little chilly - turning up the heating now`

The screenshot shows a single response card. At the top, there's a condition line: 'if @appliance:heating'. Below it is a text area containing the message: 'It is a little chilly – turning up the heating now'. To the right of the text area are three icons: a minus sign, a plus sign, a brace-and-dot icon, and a trash bin.

**9. Re-order conditional responses:**

**TIP:** Like nodes in the tree, responses are processed in order and as soon as a match is found no more matching takes place. Therefore, you should have the more generic responses **below** the more specific ones.

- b.** Click the **up arrow** above the '@appliance:heating' response to move it up the list:

The screenshot displays two separate sections of the Watson Conversation interface. The top section shows a response card for the condition 'if @appliance' with one response: '1. OK, turning on @appliance'. Below this response is a placeholder line: 'Add a variation to this response'. The bottom section shows a response card for the condition 'if @appliance:heating' with one response: '1. It is a little chilly – turning up the heating now'. Below this response is also a placeholder line: 'Add a variation to this response'. Annotations are overlaid on the bottom section: 'Move selected response up the list' points to the up arrow icon next to the response; 'Selected response' points to the '@appliance:heating' response card; and 'Move selected response down the list' points to the down arrow icon next to the response.

- c.** Note the order the responses will be checked:

**Check this first**

If no match  
then check  
this next

Fulfill with a response ⓘ [Jump to...](#)

if @appliance:heating ⊖ ⊕ ⋮ ⏚

1. It is a little chilly – turning up the heating now ⊖

Add a variation to this response

if @appliance ⊖ ⊕ ⋮ ⏚

1. OK, turning on @appliance ⊖

Add a variation to this response

#### 10. Add a Default Response:

**TIP:** What happens when the user tries to control a device or appliance we cannot identify? Watson tries to match in two stages. In stage 1 it tries to match against the child nodes of the current context. If no match is found, then it moves to stage 2 – it returns to the root and tries to match against the top-level set of child nodes. Having an *anything\_else* root node allows the chat-bot to have a generic ‘I didn’t understand’ type of response. However, we can add specific ‘anything else’ type nodes in the dialog to give specific responses when something is not recognized. Let’s do that now.

- b. Click **Create another response**
- c. In the **Enter a response** field, type **I’m sorry, I cannot control that appliance**:

+ Add response condition ⋯ ⏚

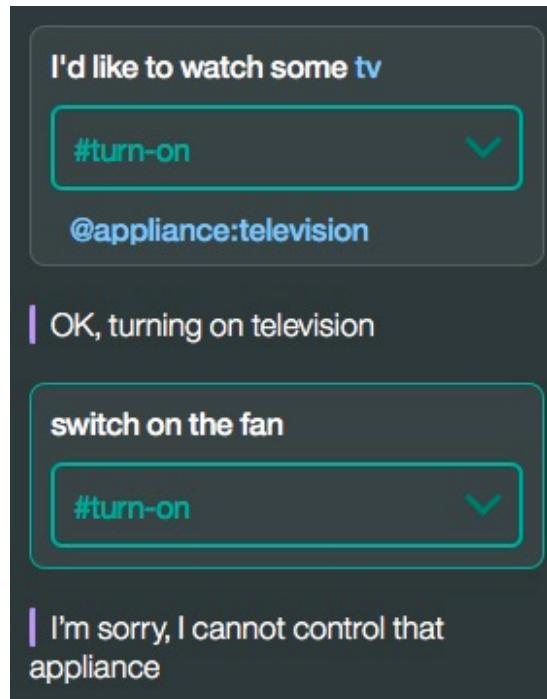
I'm sorry, I cannot control that appliance

**TIP:** Make sure this response is the last one in the list as it has no conditions and therefore will always match.

- d. Click **X** to close the editor

**11.** Test the bot:

- Click the **chat** button to open the chat panel (if it is not already open)
- In the chat window – try various ways of asking ARC to switch on appliances (including ones we haven't identified)



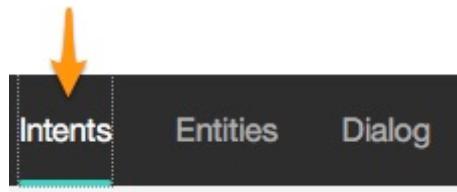

---

### Task 5. Order a Taxi

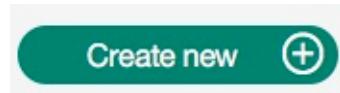
In this task, you will begin creating a more complex interaction and see how you can have the chat bot you have built interact with an application. You could build that application in Java, JavaScript etc. and use the APIs to communicate – see the Bluemix documentation for details. In this example, you will build a simple Node-RED flow to interact with the bot. You will build that flow a little later - in this task, you will extend the chat-bot to have a conversation with the user around ordering a taxi.

**1.** Add an Intent:

- In the menu bar, click **Intents**:



- Click **Create new**:



- c. In the **Intent name** field, type `taxi`

Intent name

`#taxi`

- d. In the **Add a user example** field, type `call me a taxi` and hit the **enter** key  
e. In the **Add a user example** field, type `i'd like a car to take me to work` and hit the **enter** key  
f. In the **Add a user example** field, type `i need to get to the airport` and hit the **enter** key  
g. In the **Add a user example** field, type `i would like to order a taxi` and hit the **enter** key  
h. In the **Add a user example** field, type `take me to the train station` and hit the **enter** key  
i. Click **Create**:

Create

A screenshot of a user interface for creating a new intent. At the top, there's a green "Create" button. Below it, the "Intent name" field contains "#taxi". Underneath, there's a section titled "Add a new user example..." with a plus icon. Five examples are listed, each with an empty checkbox:

- call me a taxi
- i'd like a car to take me to work
- i need to get to the airport
- i would like to order a taxi
- take me to the train station

Below this, there are two collapsed sections:

- > **#turn-on**  
enable the air conditioning
- > **#maintenance**  
my kettle is broken

2. Add Entities:

- a. In the menu bar, click **Entities**  
b. Click **Create new**  
c. Name the new Entity destination:

The screenshot shows the Watson Conversation interface. At the top, it says "Entity" and displays the name "@destination". Below this is a list of steps:

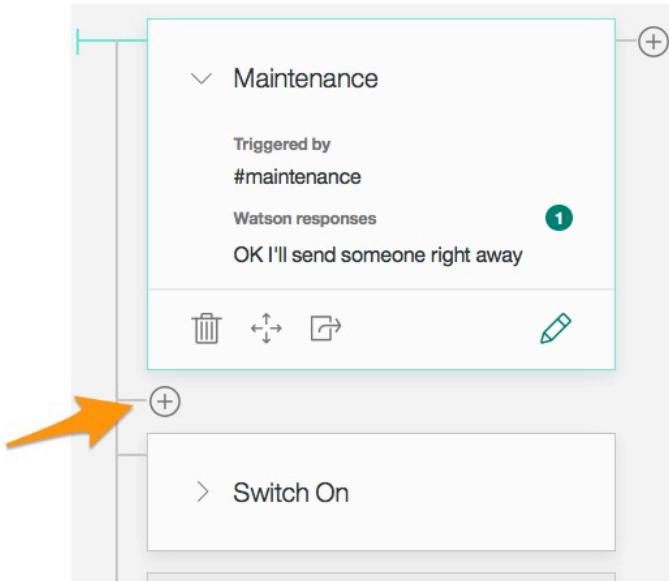
- d. In the **Value** field, type `airport`
- e. Add the **Synonyms**: `airfield, terminal`
- f. Click **Add new value**:
- g. In the **Value** field, type `railway`
- h. Add the **Synonyms**: `railway station, rail, train`
- i. Click **Create**:

Below these steps is a "Create" button in a green rounded rectangle.

After the steps, there is a list of entities under the heading "@destination":

- Add a new value**
- `airport` `airfield` `terminal` (2 Synonyms)
- `railway` `rail` `railway station` `train` (3 Synonyms)

3. Modify the dialog:
  - a. In the menu bar, click **Dialog**
  - b. Click the **Maintenance** node
  - c. Click **Create alternative conversation**:



**TIP:** This will insert a new node between Maintenance and Switch On. Remember nodes are processed in order from top to bottom. In this case, it doesn't really matter where the new node goes – so long as it is above the *anything\_else* node.

- d. In the **Name this node** field, type `Taxi` and then click away from the field to apply it (or press the **enter** key)
- e. In the **Triggered by** field, type `#taxi` and then select `#taxi`
- f. In the **Enter a response** field, type `Sure, for what time should I order a taxi ?`

**Triggered by** ⓘ

if #taxi

---

**Fulfill with a response** ⓘ

+ Add response condition

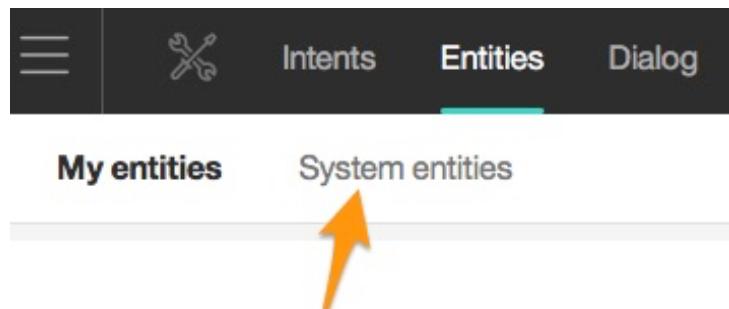
1. Sure, for what time should I order a taxi ?

Add a variation to this response

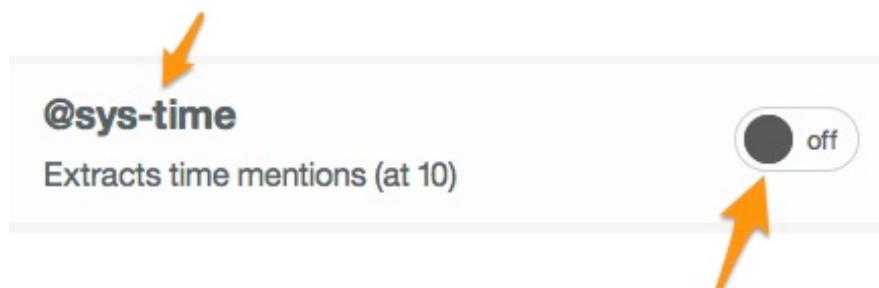
- g. Click **X** to close the editor
4. Add a System Entity:

**TIP:** The bot has now asked the user for more information - another input. Before you add nodes to the conversation to handle the next input, you need an entity to identify the various *time* responses. To do this you will use a built-in Entity that can do this automatically.

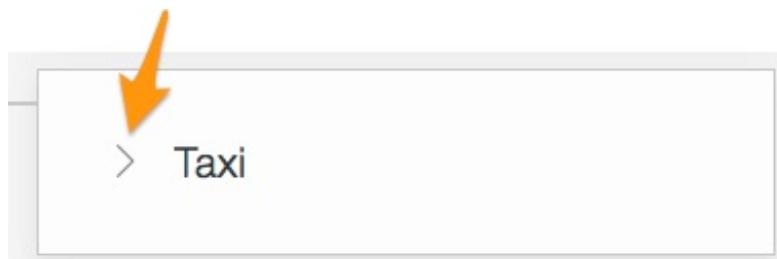
- b. In the menu bar, click **Entities**
- c. Click **System entities**:



- d. Click the **switch** next to **@sys-time** to give the bot the ability to extract times from user responses:

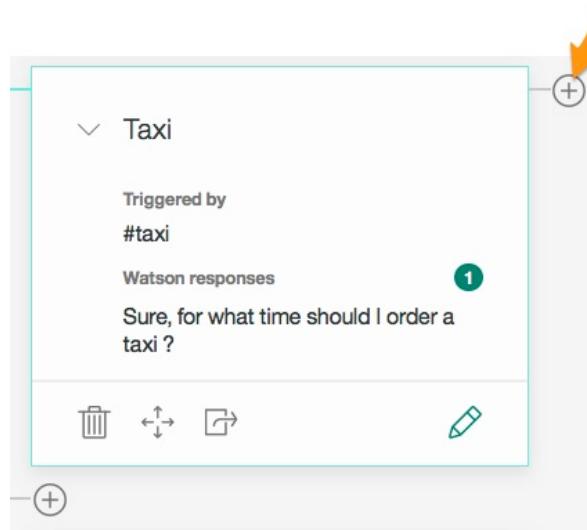


5. Continue the conversation:
  - a. In the menu bar, click **Dialog**
  - b. Click the > symbol in the **Taxi** node:



*TIP:* If you click anywhere else on the node, then it will open the editor which will be in the way of what you want to do which is add a new node. If this happens click the X to close the editor

- c. Click **Continue conversation**:



**TIP:** A new node is created which will continue the conversation after the Taxi node has been processed.

- d. In the **Name this node** field, type Pickup Time and then click away from the field to apply it
- e. In the **Triggered by** field, type sys and then select @sys-time
- f. In the **Enter a response** field, type No problem - where would you like to go ?

**TIP:** As well as responding to the user, you want the bot to store the time that was recognized in the response. *Context variables* are used to store information for use between nodes and between the bot and any application that interacts with it. You will now store the time in a context variable using the advanced response editor.

6. Store the time:
  - a. In the **Add response condition** header, click the {...} button:

## Fulfill with a response ⓘ

[Jump to...](#)

(+) Add response condition {...} 

---

1. No problem - where would you like to go ? (−)

Add a variation to this response

**TIP:** This expands to show the JSON object that is sent as a response. Note that it has an output, with a text object containing an array of values:

```
{
  "output": {
    "text": {
      "values": [
        "No problem - where would you like to go ?"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

- b. Modify the JSON by adding an extra attribute. The full text is shown below with the additional text in bold:

```
{
  "context": {
    "time": "@sys-time"
  },
  "output": {
    "text": {
      "values": [
        "No problem - where would you like to go ?"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

**TIP:** This will store the current value of the **sys-time** Entity into a context variable called **time**.

Refer to the documentation for other options:

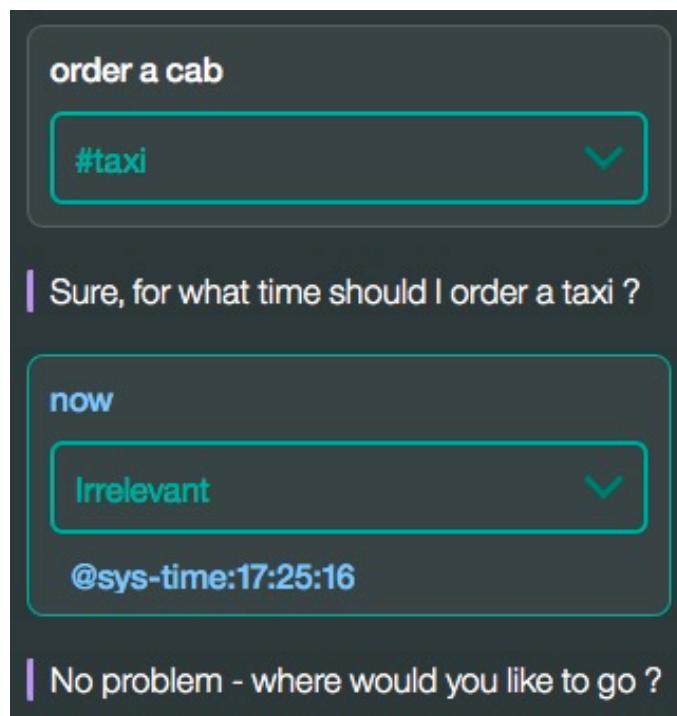
<https://www.ibm.com/watson/developercloud/doc/conversation/dialog-build.html#define-a-response>

**TIP:** Pay attention to the border of the advanced editor – if your JSON is incorrectly formatted the boundary will turn red – however this is VERY subtle and not at all obvious. Try temporarily deleting a comma or curly brace to see the effect.

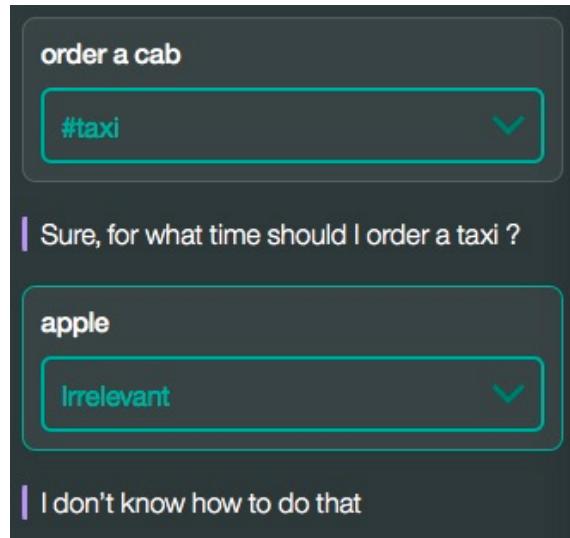
- c. Click the {...} button to close the advanced editor
- d. Click the X to close the node editor

7. Test the bot:

- a. Open the chat window if it is not already open
- b. In the chat window, type `order a cab`
- c. In the chat window, type `now`



- d. In the chat window, type `order a cab`
- e. In the chat window, type `apple`



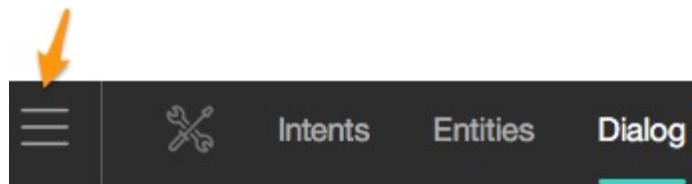
**TIP:** In this case the @sys-time was not recognized in the user response. Since you have no ‘anything\_else’ type node in this branch of the tree – no match was made and so Watson proceeded to Stage 2 – it returned to the root and tried to match there. In this case, it did not match the Maintenance node and so continued – matching against the anything\_else node. This is important to understand as you build your dialog – if you don’t add ‘catch all’ nodes in your conversation then Watson always returns to the start when an unrecognized input occurs.

## Task 6. Create a Node-RED Flow

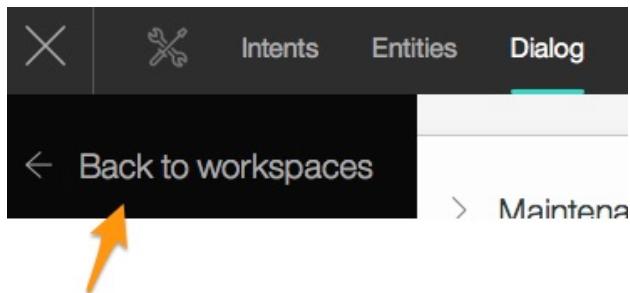
At this point you will create a simple flow in Node-RED and see how you can send and receive data from the bot. Once this is working you will complete the ARC application by asking the user where they need to go – and responding with information appropriate to their target destination (perhaps using other services like traffic or weather).

To connect your node-red application to the correct dialog (you could have several) you will first obtain the workspace ID for the ARC dialog

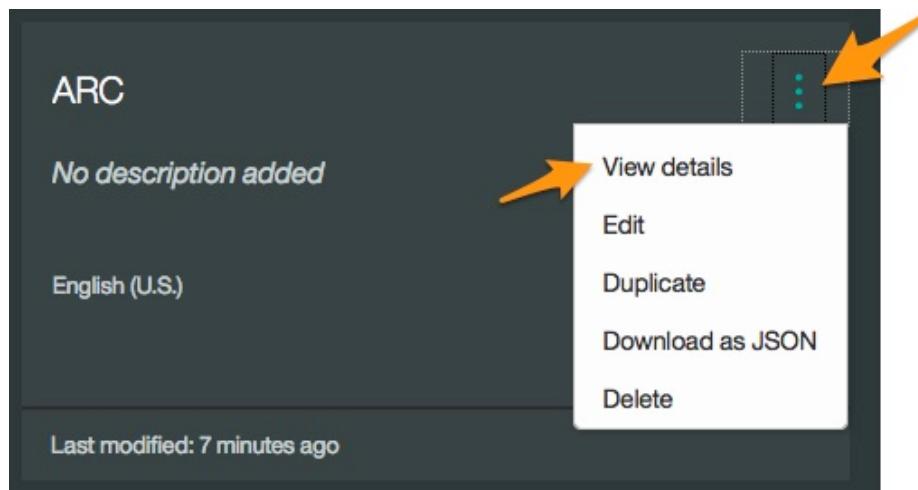
1. Obtain the workspace ID:
  - a. In the menu bar, click the ‘burger’ menu icon:



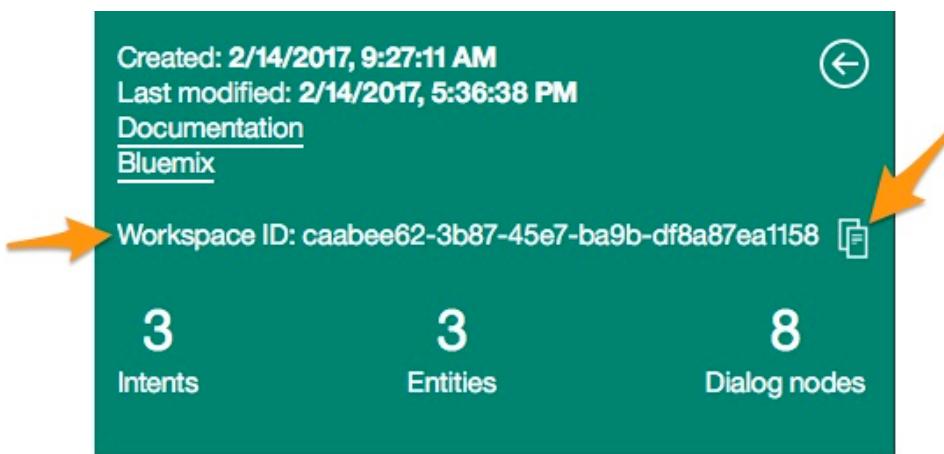
- b. Click **Back to workspaces**:



- c. Click the **menu** symbol in the ARC tile and select **View details**:



- d. Click the **copy** button next to the **Workspace ID** to place it in the copy buffer:



2. Open the Node-RED Application:
  - a. Switch back to your Bluemix web browser tab and open the Node-RED flow editor for the application
3. Connect to Watson Conversation:
  - a. From the palette, drag on a **conversation** node:



- b. Double-click the new node to edit it
- c. Paste the **workspace ID** you copied in the step above into the **Workspace ID** field
- d. Select **Save context**



**TIP:** Watson dialog itself retains no state information when accessed from an application. Unless you include the context when you send a request to the service – each request is treated as a brand-new conversation. If you write your own application in Java, JavaScript etc then you must manually include context when sending in requests.

- e. Click **Done**
- f. From the palette, drag on an **inject** and a **debug** node
- g. Connect the nodes together as shown below:



- h. Double-click the **inject** node to edit it
- i. Change the payload **type** to **string**
- j. In the **Payload** field, type Order a Taxi



- k. Click **Done**
- l. Make a copy of the inject node and change its payload to Now

- m. Connect the new node to the flow as shown below:



- n. Click Deploy

4. Test the flow:

- a. Activate the **Order a Taxi** inject node and observe the output in the debug panel:

```

object
  intents: array[1]
    0: object
      intent: "taxi"
      confidence: 0.8046794533729553
  entities: array[0]
  input: object
    text: "Order a Taxi"
  output: object
    log_messages: array[1]
      text: array[1]
        0: "Sure, for what time should I order a taxi ?"
    nodes_visited: array[1]
  context: object
    conversation_id: "7fad1f4c-5dee-40d6-a2b6-bca584a3a526"
  
```

identified Intent(s) ←

original input ←

bot response ←

conversation id ←

- b. Activate the **Now** inject node and observe the output in the debug panel:

```

msg.payload : Object
  ↘ object
    intents: array[0]
  ↘ entities: array[1]
    ↘ 0: object
      entity: "sys-time"
      ↗ location: array[2]
        value: "17:56:59"
      ↗ metadata: object
    ↗ input: object
    ↗ output: object
  ↘ context: object
    conversation_id: "7fad1f4c-5dee-40d6-a2b6-bca584a3a526"
  ↗ system: object
    time: "17:56:59"

```

**Stored context attribute** →

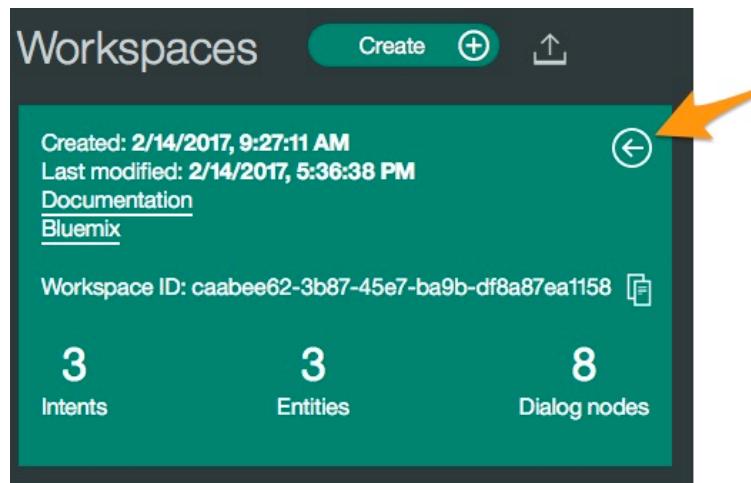
**TIP:** Note the stored context attribute. Note also that the conversation\_id is the same as the previous message – this is because the option to save the context was set on the conversation node.

## Task 7. Add More Complex Interaction

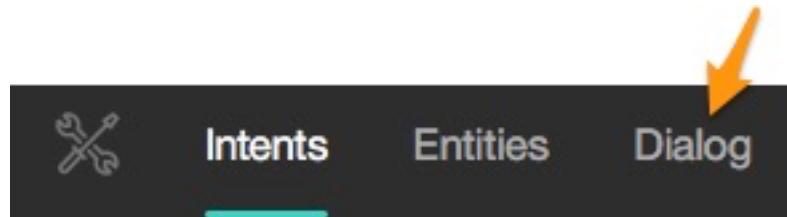
In this task, you will modify the bot to provide different responses depending on the user's choice of location. In this simple example if the user is headed to the airport then you will provide a simple 'bon voyage'. If they are not headed to the airport, then the application will check the weather and the bot will advise the user if they need an umbrella. Since this may involve further interactions – you will split the dialog at this point into two separate branches.

Note that in this simple example the node-red flow will simply assume it's raining – in a real system the app would perhaps ask the user if they would like the weather checked - and use the weather service to do it. The following steps serve to highlight the mechanism whereby the bot and the node-red flow can interact.

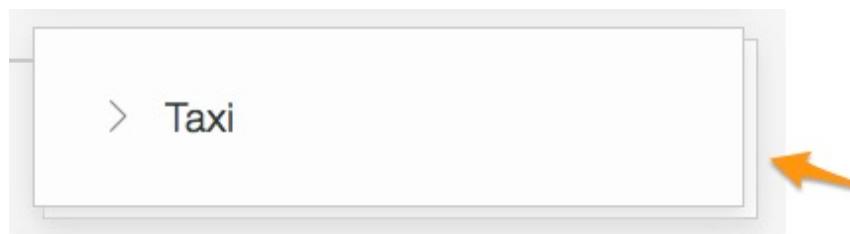
1. Modify the Bot Interaction:
  - a. Switch back to the Watson Conversation tab
  - b. Click the **back** button in the ARC tile:



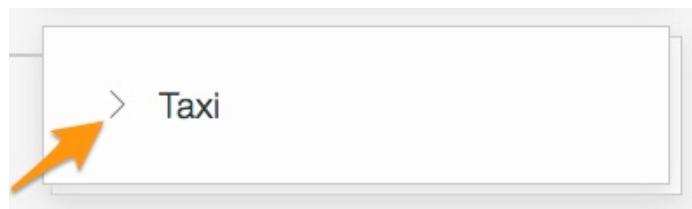
- c. Click the **ARC** tile to edit the workspace
- d. In the menu bar, click **Dialog**:



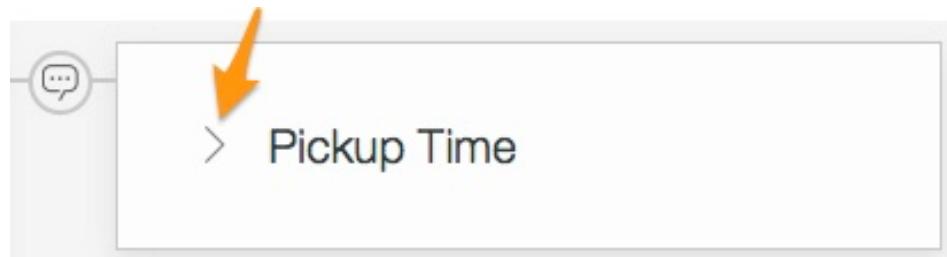
- e. Note that the Taxi node icon reflects the fact that it has **child nodes**:



- f. Click the > symbol in the **Taxi** node:



- g. Click the > symbol in the **Pickup Time** node:



h. Click **Continue conversation:**

A screenshot of the Watson Conversation interface showing the "Pickup Time" node expanded. The node has a green border and a downward-pointing arrow icon. Inside the node, there is a list of items: "Triggered by @sys-time", "Watson responses" (with a count of 1), and the response text "No problem - where would you like to". In the top right corner of the node's container, there is a small circle with a plus sign (+) inside, which is highlighted with an orange arrow.

i. Name the new node **Airport**

j. In the **Triggered by** field, type dest and select **@destination:airport**

A screenshot of the Watson Conversation interface showing the configuration for the "Airport" node. The node name "Airport" is at the top. Below it is a "Triggered by" section with the condition "if @destination:airport". There are minus (-) and plus (+) buttons to the right of the trigger condition.

k. In the **Enter a response field**, type Booking your cab for \$time. Have a good trip !

+ Add response condition

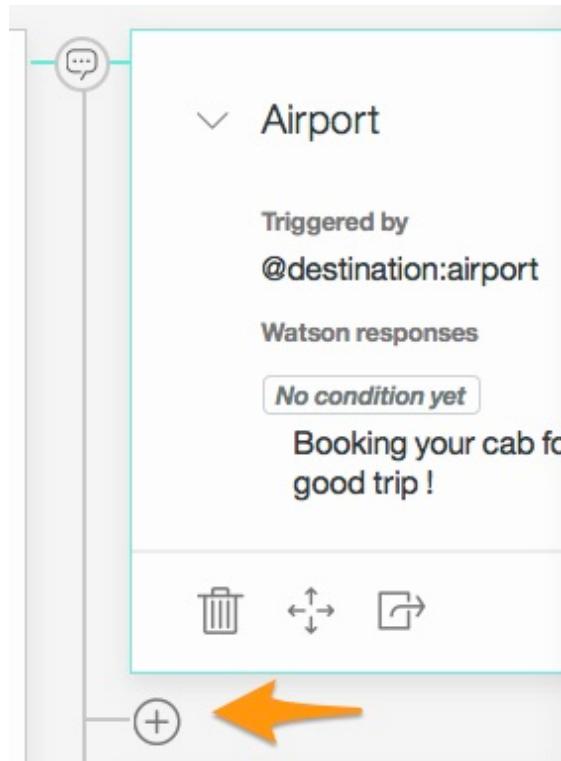
1. Booking your cab for \$time. Have a good trip !

Add a variation to this response

- I. Click X to close the editor

**TIP:** If the user is headed to the airport then the bot gives a simple response and ends the conversation

2. Add an alternative branch:
  - a. Click Create alternative conversation:



**TIP:** If the icon does not show, click > on the Airport node to collapse it and then click it again

- b. Name the new node **Other Destination**
- c. In the **Triggered by** field, type `dest` and select **@destination**

**TIP:** If the user is headed somewhere other than the airport, then your node-red application will ‘check the weather’ for the area and advise them whether they need to take an umbrella. Note that this is of course a simple example but will serve to illustrate how your bot can exchange information with a connected application.

- In the **Fulfill with a response** section, click **Add response condition**:

### Fulfill with a response ⓘ

**(+) Add response condition**



- In the **Enter a response field**, type **Checking Weather**
- Click the advanced editor button **{...}**

- Modify the JSON to add a new action attribute into the output – the full text is shown below with the modifications in bold:

```
{
  "output": {
    "action": "checkWeather",
```

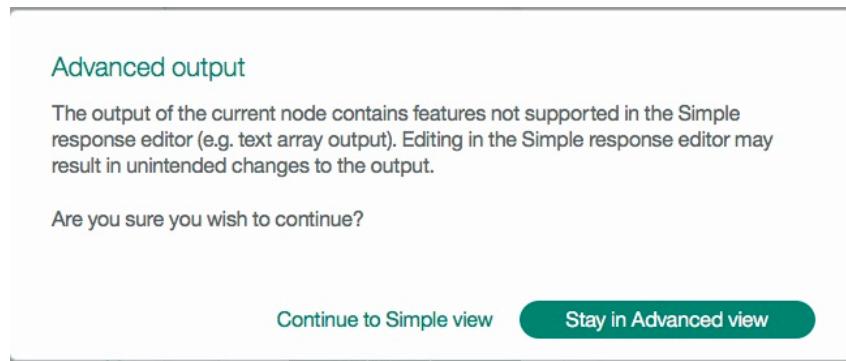
```
    ],
    "selection_policy": "sequential"
}
}
```

Fulfill with a response ⓘ

```
{
  "output": {
    "action": "checkWeather",
    "text": {
      "values": [
        "Checking Weather"
      ],
      "selection_policy": "sequential"
    }
  }
}
```

**TIP:** Note that context attributes persist – attributes in the output do not – this is useful for only triggering actions in the connected application at certain times in the conversation.

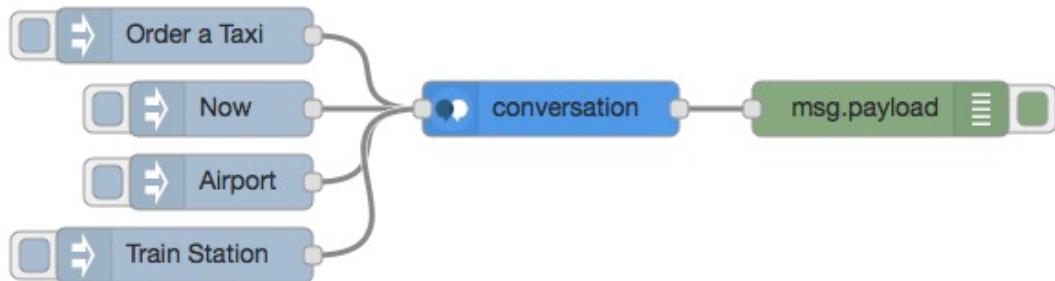
- h. Click the {...} button to close the editor. Note that while you can switch back to the simple editor this could cause unexpected side effects:



**TIP:** In this case the changes we have made to the JSON are minimal and switching back to the simple view will not cause any problems.

- i. Click **Continue to Simple View**
  - j. Click **X** to close the editor
3. Modify the Node-RED Flow:
  - a. Switch back to the Node-RED flow editor
  - b. Make a **copy** of one of the **Inject** nodes and modify it to send a payload of **Airport**

- c. Make another **copy** of one of the **Inject** nodes and modify it to send a payload of **Train Station**
- d. Connect the new nodes to the flow as shown below:



- e. Click **Deploy**

4. Test the Flow:
  - a. Activate the **Order a Taxi** inject node
  - b. Activate the **Now** node
  - c. Activate the **Train Station** node
  - d. Note the output in the debug panel:

```

15/02/2017, 08:22:19  node: 24bb2a8a.cb4df6
msg.payload : Object
  ▼ object
    ▶ intents: array[1]
    ▶ entities: array[1]
    ▶ input: object
    ▶ output: object
      ▶ log_messages: array[1]
      ▶ text: array[1]
      ▶ nodes_visited: array[1]
      action: "checkWeather" ←
      ▶ context: object
  
```

5. Handle Actions in the Flow:
  - a. From the palette drag on a **switch** node:



- b. Double-click the new node to edit it
- c. In the **Name** field, type **Check for Actions**
- d. In the **Property** field, type **payload.output.action**
- e. In the comparator field, select **is not null**:

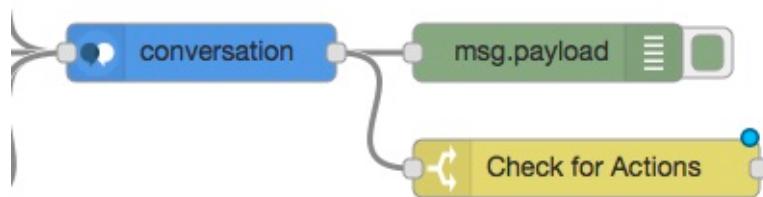
**Name** Check for Actions

**Property** msg.payload.output.action

**is not null**

→ 1

- f. Click **Done**
- g. Connect the switch node to the conversation output as shown below:



**TIP:** This part of the flow will only come into play if the bot includes an action in its response

- h. Make a copy of the switch node and double-click the new node to edit it
- i. In the **Name** field, type **Actions**
- j. In the comparator field, select **==** and type **checkWeather**

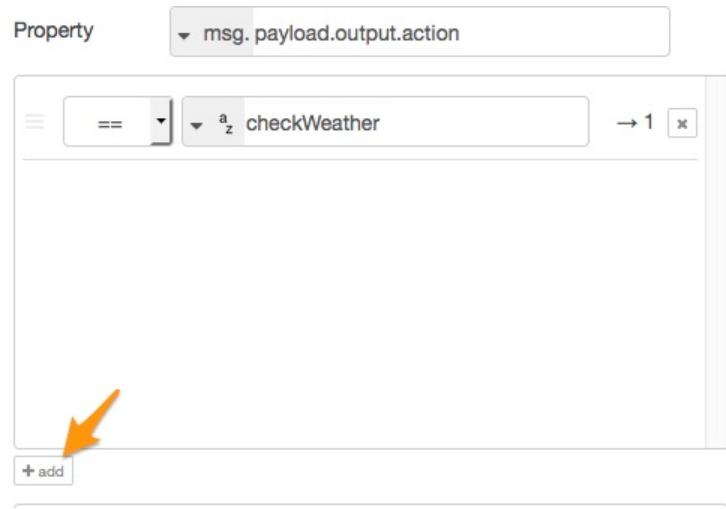
**Name** Actions

**Property** msg.payload.output.action

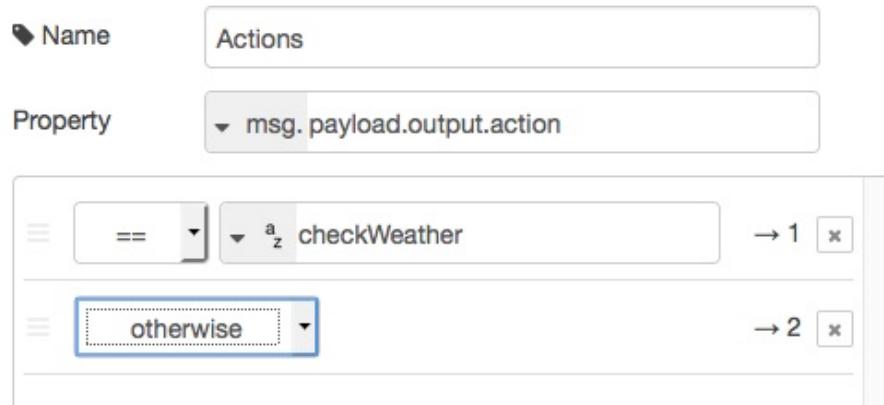
**==** **a\_z checkWeather**

→ 1

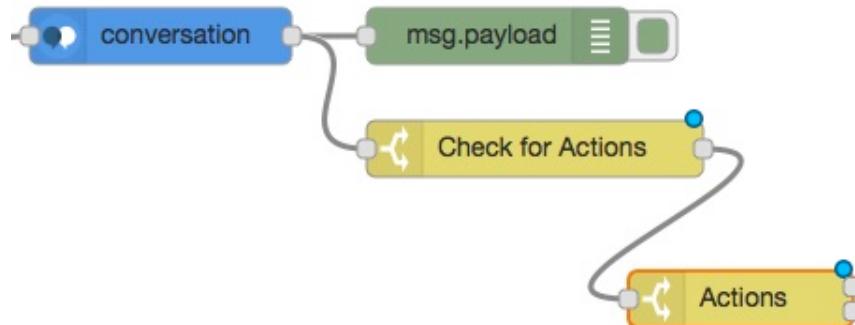
- k. Click **add** to add a second condition:



- I. In the **comparator** field, select **otherwise**:



- m. Click **Done**
- n. Connect the new switch to the existing switch as shown below:



6. Send a request with the weather data:
  - a. From the palette, drag on a function node:

© Copyright IBM Corp. 2017

Watson Conversation - 49



- b. Double-click the new node to edit it
- c. In the **Function** field, enter the following code:

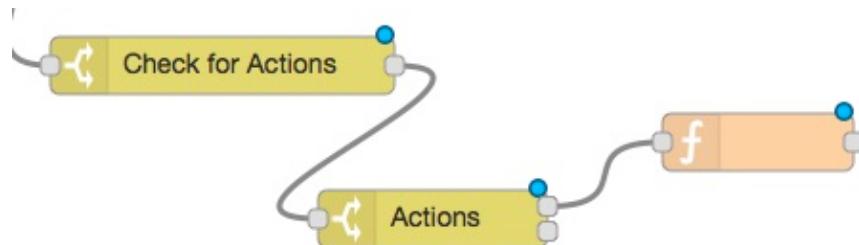
```
msg.payload = "weather response"
msg.additional_context = {'localweather' : 'rain'}
return msg
```

**Function**

```
1 msg.payload = "weather response"
2 msg.additional_context = {'localweather' : 'rain'}
3 return msg
```

**TIP:** Although we are not actually making a request – and therefore do not need a payload – the conversation node requires it and will not work without some sort of payload. The bot you are about to modify however will ignore the payload and focus on the context attributes. The *additional\_context* attribute allows you to insert new data into the context easily. In this case the node-red flow has added an attribute called **localweather** – you will modify the bot to recognize that attribute and deal with it.

- d. Click **Done**
- e. Connect the function node to the first switch output:



- f. From the palette drag on an **input link** and an **output link**:



- g. Double-click the **input link**
- h. In the **Name** field, type `Send Request`

The screenshot shows the 'Edit link in node' dialog. At the top are 'Delete', 'Cancel', and 'Done' buttons. Below is a 'Name' field containing 'Send Request'. An orange arrow points to this field. Below the field is a table with columns 'name' and 'flow'. The first row shows '25cb5970.713066' and 'Flow 2'.

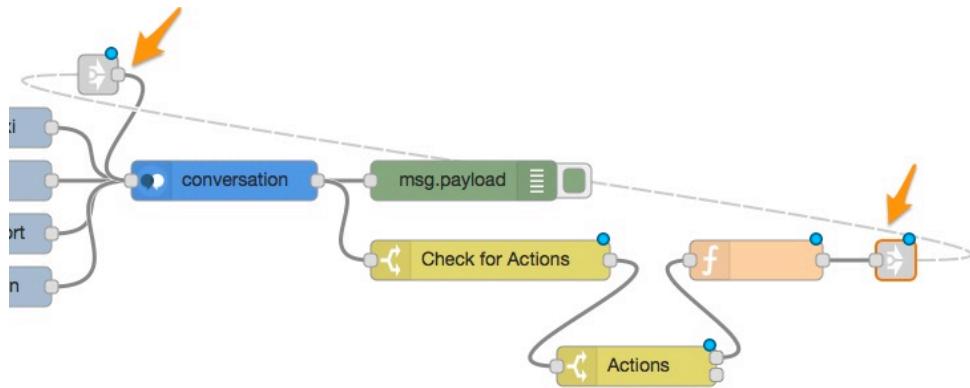
name	flow
25cb5970.713066	Flow 2

- i. Click **Done**
- j. Double-click the **output link**
- k. Select the **Send Request** link:

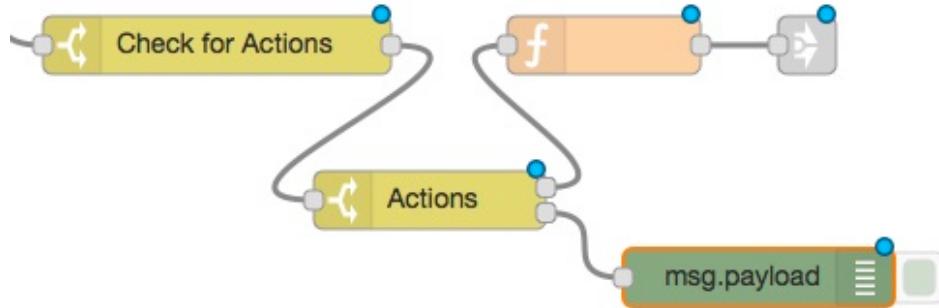
The screenshot shows the 'Edit link out node' dialog. At the top are 'Delete', 'Cancel', and 'Done' buttons. Below is a 'Name' field containing 'Name'. An orange arrow points to this field. Below the field is a table with columns 'name' and 'flow'. The first row shows 'Send Request' with a checked checkbox and 'Flow 2'.

name	flow
<input checked="" type="checkbox"/> Send Request	Flow 2

- l. Click **Done**
- m. Connect the **input link** to the **conversation** node and the **output link** to the **function** node as shown below:

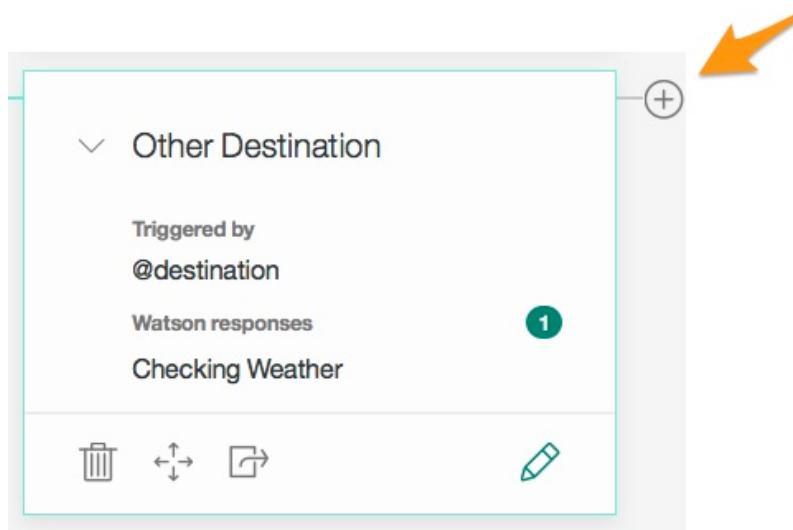


- n. Drag on a **debug** node and connect it to the second switch output as shown below:



**TIP:** This allows you to recognize when actions come in that have not been accounted for in the flow

- o. Click **Deploy**
- 7. Modify the Bot to Handle the Weather Data:
  - a. Switch back to the Watson Conversation tab
  - b. Click **Continue conversation** on the Other Destination node:



- c. In the **Triggered by** field, type \$localweather and select **\$localweather (create new condition)**

Triggered by ⓘ  
if Enter a condition  
\$localweather (create new condition)

**TIP:** You cannot use auto-complete here since the localweather attribute is a context variable that is dynamically being added to the context by the Node-RED flow. The \$ symbol refers to a context variable.

- d. In the **Fulfill with a response** field, click **Add response condition**:

Fulfill with a response ⓘ  
⊕ Add response condition  
Enter a response...

- e. In the **Enter a condition** field, type \$localweather:rain and select **\$localweather:rain (create new condition)**

if \$localweather:rain  
\$localweather:rain (create new condition)

- f. In the **Enter a response field**, type Booking your taxi for \$time - Better take a brolly !

if \$localweather:rain  
Booking your taxi for \$time - Better take a brolly !

8. Add another response:

- a. Click **Create another response**:

if \$localweather:rain  
1. Booking your taxi for \$time - Better take a brolly !  
Add a variation to this response

✓  
 Create another response

- b. In the **Enter a response field**, type Booking your taxi for \$time - Have a great day !

- c. Click X to close the editor
- 9. Test the application:
  - a. Switch back to the Node-RED Flow Editor

**TIP:** To make life simpler you will now modify the flow to extract only the bot responses

- b. From the palette, drag on a function node
- c. Double click the new node to edit it
- d. In the **Name** field, type `Get Response Text`
- e. In the **Function** field, enter the following code:

```
array = msg.payload.output.text

if (array.length > 0)
  msg.payload = msg.payload.output.text[0];
else
  msg.payload = "No Response";

return msg;
```

**Name** Get Response Text

---

**Function**

```

1 array = msg.payload.output.text
2
3 if (array.length > 0)
4   msg.payload = msg.payload.output.text[0];
5 else
6   msg.payload = "No Response";
7
8 return msg;

```

f. Click **Done**

g. Drag on a new Debug node and connect the nodes together as shown below:



h. Click **Deploy**

i. Activate the Inject Nodes to have a conversation with the bot

### Task 8. Add a User Interface

In this unguided task, you will build a chat interface to let the user interact with the bot using speech or a text interface.

#### Tips:

- To input speech, use the node-red extension node-red-contrib-browser-utils which has a microphone node and use Watson speech to text to convert it
- Add node-red-dashboard for a user interface and audio out nodes

If you want to get clever – add Watson language translation and add the option into the bot to change languages ☺