

Watson Assistant Service

Deep Dive Hands on Technical Lab

AT&T

Learn - Day 1

November 12<sup>th</sup>, 2018

Watson

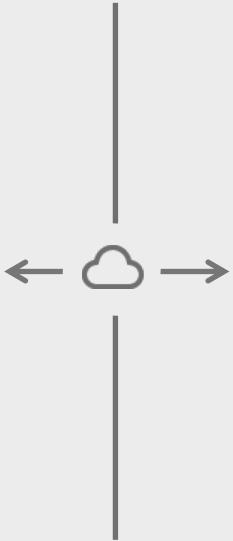
IBM

# Watson Assistant

# Fundamental Concepts

# IBM Cloud Spaces and Watson Assistant Skills

Lab Room



IBM Cloud “Dev Space”



# Intents

The action a user wants to take (verb).

A label for a group of examples of things that a user might say to communicate a specific goal or idea.

Deep Learning state of the art classifier for understanding the intent behind a user's utterance.

- Uses entities (synonyms) as training elements
- Absolute confidence values for intents
- Multiple end-user examples with the same intent should be clustered together.
- Poor clustering will cause problems with machine learning.

Identified in tooling via **#intent\_name**

- Allowed: Letters, numbers, hyphen, underscores, dashes and/or dot.
- Prohibited: Spaces, more than 128 characters.

19	<b>#capabilities</b> can I manipulate the
40	<b>#goodbyes</b> adieu
35	<b>#greetings</b> aloha
49	<b>#locate_amenity</b> Amenities
<b>#not_specified</b>	
⊕ Add a new user example...	
<input type="checkbox"/>	any
<input type="checkbox"/>	anything
<input type="checkbox"/>	doesn't matter
<input type="checkbox"/>	no preference
<input type="checkbox"/>	whatever

# Entities

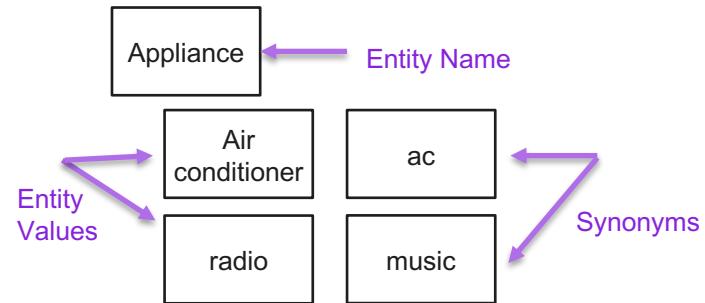
An Entity is a term or object which provides context for an intent

Entities are objects the user wants to take action on  
Entities alter the way Watson responds to the user's intent.

- “Where is the pool?” Entity = @Pool
- “I need to reset my password to my email” – Entity = @Email

Consist of a **name**, **value** and **synonym** or **pattern**

**Fuzzy matching** can be enabled for entities to allow for stemming, misspelling, and partial match



# Entities

The Patterns field lets you define specific patterns for an entity value.

Must be entered as a regular expression in the field.

Patterns may not contain:

- Positive repetitions (e.g.,  $x^+$ )
- Backreferences (e.g.,  $\backslash g1$ )
- Conditional branches (e.g.,  $(?<cond>)true$ )

Email example, e.g name@ibm.com

$\backslash b[A-Za-z0-9._%+-]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}\backslash b$

To store a Pattern entity value as a context variable, use advanced JSON:

```
{  
  "context": {  
    "email": "@contactInfo.literal"  
  }  
}
```

The screenshot shows the Entity configuration for '@ContactInfo'. It includes a 'Fuzzy Matching' section with a note about increasing Watson's ability to recognize misspelled entity values. Below this, there is a table with columns for 'Value' and 'Type'. The 'Value' column contains 'localPhone', 'fullUSphone', 'internationalPhone', 'email', and 'website'. The 'Type' column shows a dropdown set to 'Patterns' with a selected pattern '(d{3})-(d{3})-(d{4})'. There is also a 'Type' button icon. A 'Patterns' dropdown menu is open, showing four entries: '(d{3})-(d{3})-(d{4})', '^((?:+[0-9])?)?(0-9\_\\-\\ )\$', '\\b[A-Za-z0-9.\_%+-]+@[A-Za-z...]', and '(https?:\\/\\/)?([da-zl-\_-]+\\.)[a-z...]'.

# Entities

Identified in tooling with `@entity_name`.

- Names can contain letters, numbers, underscores, dashes.
- No spaces, limit to 64 characters, 128 example values and 32 synonyms

Import entity lists via CSV file:

- `<entity>,<value>,<synonyms>`

```
<entity>,<value>,<synonyms>
```

```
appliance,volume,sound  
appliance,lights,tubelights,tubelight,light,head lights  
appliance,ac,fans,air conditioning,air conditioners  
appliance,music,songs,song,radio  
appliance,fan,fans  
appliance,heater,heat  
appliance,wipers,wiper,windshield wipers,windshield wiper
```

# System Entities

Common entities available out of the box

- Must be enabled, but centrally maintained (can not be modified/extended).
- Will expand over time.

Numbers (sys-number) – detects numbers in numerical and textual form

@sys-number.literal – String text representation

@sys-number.numeric\_value – Number representation

Currency (sys-currency) – detects monetary currency values

@sys-currency.literal

@sys-currency.numeric\_value

@sys-currency.unit – Currency unit (Euro, USD, etc)

Percentage (sys-percentage) – detects percent numbers in numerical and textual form

@sys-number.literal – String text representation

@sys-number.numeric\_value – Number representation

# Dialog

**Dialog** uses the intents and entities that are identified in the user's input, plus context from the application, to interact with the user and ultimately provide a response

- Allows users to quickly and easily visualize and understand the conversational flow
- A branch can be created for each intent, and in each node, matching conditions are entered based on intents/entities
- Nodes can be added, moved, and referenced, and child nodes can be added to any node
- One or more responses can be provided based on conditions, context, or taken from a random or sequential set

Watson Conversation / Car Dashboard - Sample / Build

Intents Entities Dialog

Add node

Car Dashboard - Sample

Start And Initialize Context  
conversation\_start

1 Response / 8 Context set

Entry Point For On Off Command  
#turn\_on

0 Responses / 1 Context set / Jump to

#turn\_off

0 Responses / 1 Context set / Jump to

Add child node  
Add node above  
Add node below  
Move  
Jump to  
Delete

# Pretrained Content

To help get started, Watson Assistant is pretrained on common intents by industry. These intents can be added to your skill from the Content Catalog page. Select the industry that best fits your use case and click the “Add to skill” button next to the desired content.

The screenshot shows the Watson Assistant Content Catalog page. At the top, there are tabs for Skills, cognitive\_car-VictorP, Build, and a Try it button. Below the tabs, there is a heading "Content Catalog". A sub-instruction says: "Get started more quickly by adding existing intents from the content catalog. These intents are trained on common questions that users may ask." The main content is a table with three columns: Category, Description, and Intents. Each row has an "Add to skill" button. The categories listed are Banking, Bot Control, Customer Care, eCommerce, General, Insurance, Telco, and Utilities. The descriptions and intent counts are as follows:

Category	Description	Intents	Action
Banking	Basic transactions for a banking use case.	13	+ Add to skill
Bot Control	Functions that allow navigation within a conversation.	9	+ Add to skill
Customer Care	Understand and assist customers with information about themselves and your business.	18	+ Add to skill
eCommerce	Payment, billing, and basic management tasks for orders.	14	+ Add to skill
General	General conversation topics most users ask.	10	+ Add to skill
Insurance	Issues related to insurance policies and claims.	12	+ Add to skill
Telco	Questions and issues related to a user's telephony service, device, and plan.	21	+ Add to skill
Utilities	Help a user with utility emergencies and their utility service.	10	+ Add to skill

## Lab Use Case

The assistant flow you create in this lab exercise, demonstrates how the Assistant service uses Intents and Entities in a simple chat interface.

You are creating a car dashboard that allows a driver of a car to make control requests, or ask questions in natural language.

Intents and entities are extracted from those requests and presented as JSON data in the response from the Assistant API call.

An application using the Assistant skill, can take appropriate action based on that information. Usually, a combination of intent and entity results in a separate response or action.

# Lab Preparation

# Watson Assistant Service

# Create your Assistant Service Instance

An Assistant service instance is created on the IBM Bluemix console. The procedure is identical to that of creating other Watson services.

1. Access the IBM Cloud console by entering the URL and logging-in with your IBM Cloud ID <https://console.ng.bluemix.net/>
2. Access the **Catalog** link.
3. Locate **Assistant service** by starting to type “assist...” into the search field.
4. Select **Watson Assistant** (you will be prompted with pricing plans).
5. Select Lite
6. Enter a meaningful **Service name**, such as *Assistant-FirstNameFirstLetterOfLastName*, such as *Assistant-VictorP*.

# Create your Assistant Service Instance (cont.)

7. Click **Create**.
8. If prompted (because you already have an application), accept the default **Credential name** and under **Connect to**, leave the service unbound.

## Watson Assistant (formerly Conversation)

Add a natural language interface to your application to automate interactions with your end users. Common applications include virtual agents and chat bots that can integrate and communicate on any channel or device. Train Watson Assistant service through an easy-to-use web application, designed so you can quickly build natural conversation flows between your apps and users, and deploy scalable, cost effective solutions.

Service name: Assistant-VictorP

Choose a region/location to deploy in: US South      Choose an organization: IBM\_VP      Choose a space: Conversation (Primary)

**Images**

Click an image to enlarge and view screen captures, slides, or videos. Screen caps show the user interface for the service after it has been provisioned.



View Docs    Terms

AUTHOR IBM  
PUBLISHED 03/26/2018  
TYPE Service  
LOCATION

Need Help? [Contact IBM Cloud Sales](#)  Estimate Monthly Cost [Cost Calculator](#)

**Create**

# Create a Skill

A skill is a unique development and test environment where business and technical users can update the logic for their assistant system.

When calling the Watson Assistant Service API, you must specify a Skill ID. The service credentials for your service tile will remain across all **5 skills (20** skills for Standard plan) per tile.

1. Launch the Assistant Tooling

[Launch tool ↗](#)

2. Select the Skills tab



3. Click **Create new**.

4. Name it: cognitive\_car -

*FirstNameFirstLetterOfLastName, such as cognitive\_car-VictorP, click Create.*

## Skills

Develop powerful, natural language understanding for your Assistants. Leverage detailed analytics to improve conversational flow and customer engagement.

[Create new](#)

## Download the lab files

1. To prepare for the lab exercises, download the **Waston Assistant Lab.zip** from the Expert Services Learn Box folder.
2. Unzip the file into a directory on your local hard drive.
3. The lab file contains the following files:
  - Ground-truth.xls
  - Car\_entities.csv
  - Car\_intents.csv
  - Context.txt

# Ground Truth

# Question Collection

Question collection is the process of collecting utterances that represent the user scenario.

An **utterance** is any input a user provides when prompted. It could be a sentence, a question, a word, or nonsense. In our examples we will assume that the utterances are understandable words. Utterances can also be referred to as **questions**.

Collect utterances from real live examples; not made-up questions.

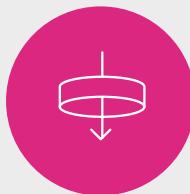
The training will make the language classifiers of the training service much more accurate in being able to detect intents from a user's utterance.

# Create Ground Truth



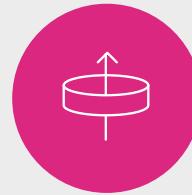
## Objective

- Establish Ground Truth by matching the representative questions to question intent groups.
- A baseline test will be run against all the questions in Ground Truth to determine how well Watson Assistant Service performs against questions it has been trained on. Additional tests will be run against subsets of the full set for regular performance testing.



## Inputs

- At least 10 representative questions for training an intent.
- Client domain SMEs to provide answers to questions



## Outputs

- The identification of question intent groups covered by the representative questions gathered;
- List of Entity classes including values and synonyms
- Answer content for the most common question intent groups that require a unique answer response.

## Guidance – Creating Ground Truth

**Should be gathered from real user data.**

- Caution against relying on SMEs, business users, other "proxy" users.

**The simpler your intents the more accurate your classifier will be.**

- #book\_room vs (#book\_single\_room and #book\_double\_room).
- Use entities/context for different answers.

**Follow naming conventions to create consistent intents.**

- Use "-" to separate multiple levels (Example : location-weather-forecast).
- Use "\_" to separate multiple word intents (Example : business\_center).
- Try for at least 10 variations to train each intent.

## Guidance – Creating Ground Truth

- Varies by number of total intents in model
- Try to avoid overlapping intents across examples.
- Use off\_topic, out\_of\_scope, Mark as irrelevant, and/or do\_not\_answer topics.
- Be cautious of exploding number of intents

Does everything need to be handled in conversation or is there a better strategy

# Intents Used in this Lab

Intents	Representative questions	Description
greetings	Hello, how are you?	Just speaking a greeting message
goodbyes	bye now	Just speaking a goodbye message
turn_on	turn my lights on or me	Request for turning on an equipment
turn_off	turn off my wiper please	Request for turning off an equipment
turn_up	can you increase the	Request to increase music volume
turn_down	turn it down	Request for decreasing music volume
hours	What are your hours	Request for hours when scheduling service
schedule_service	I want to schedule service	Request for scheduling a service appointment
weather	how cold is it out?	Question for weather
locate_amenity	i need to stop for gas	Request for finding local amenity
capabilities	what can you do for me	Question for dashboard's capability
off_topic	book a flight for NY on sunday	Request or question that this dashboard can not handle
out_of_scope	how do i adjust exterior mirrors	Request or question that this system is not designed to handle

# Entities used in this Lab

Entities	Values defined in the entity	Description
amenity	gas, restaurant, restroom	Amenity list that this dashboard can find.
appliance	ac, fan, heater, lights, music, volume, wipers	Appliance list that you can control through this dashboard.
service_type	Recall, oil change, inspection	Types of maintenance you can schedule
special_request	Car wash, loaner	Used for handling special requests received during the schedule service process
genre	Classical, jazz, Pop, rock	Music types that the dashboard like to play.
genre_bad	Blues, Country, Latin	Music types that the dashboard do not like to play.
option	closest, first, second, third, fourth, fifth	Optional preference on finding amenities.
phone	call, text	Type of phone call

# Ground Truth Lab

# Intent Lab

1. Open up ground-truth.xls from your lab files
2. Open the intent-exercise sheet
3. Review each utterance and classify with one of the provided intent names

Utterance	Intent Name
	capabilities, goodbyes, greetings, locate_amenity, navigation, phone, selections, traffic_update, turn_down, turn_off, turn_on,
Play something in car rdaio	
g'day	
what options do ypu have?	
lower ac	
adios	
I want the ac off	
place up	
how long it would take me get home	
House music	
play 21	
lower temperature	
go to the country	
enhance the music	

# Results - Intent Lab

Utterance	Intent Name
Play something in car rdaio	turn_on
g'day	greetings
what options do ypu have?	selections
lower ac	turn_down
adios	goodbyes
I want the ac off	turn_off
place up	turn_up
how long it would take me get home	traffic_update
House music	turn_on
play 21	turn_on
lower temperature	turn_down
go to the country	locate_amenity
enhance the music	turn_up
pls volume up	turn_up
I want to drink beer	locate_amenity
how hot	weather
AC low	turn_down
dial Home	phone
do you have a phone	phone
I'd like to call my mom	phone
stop my lights	turn_off
Can you call Eliza?	phone

# Using Pivot Table

Count of Utterance		
Intent Name	Utterance	Total
+capabilities		10
+goodbyes		8
+greetings		16
+locate_amenity		9
+navigation	can you navigate to my home	1
	Hi Watson! Please enable the GPS.	1
	how do i get from here to there	1
	How long does it take to get the first gas station?	1
	what is the best route to a gas station	1
navigation Total		5
+phone		9
+selections		6
+traffic_update		9
+turn_down		9
+turn_off		8
+turn_on		13
+turn_up		12
+weather		8
Grand Total		122

# Entity Lab

1. Open up ground-truth.xls from your lab files
2. Open the entity-exercise sheet
3. Review each utterance and identify possible entity names

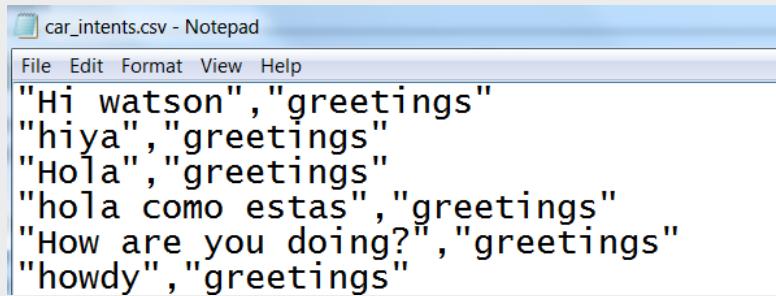
Intent	Utterance	Entity <b>What objects could be used as entities in the following utterances?</b>
turn_down	AC low	
turn_off	Turn off the air conditioning	
turn_down	lower ac	
turn_down	Reduce the fan speed	
turn_down	decrease the volume	
turn_down	Decrease volume	
turn_off	shut of the headlights	
turn_down	heat low	
turn_down	slow down the wipers	
turn_down	I need you to turn my music down	
turn_down	lesser the ac	
turn_off	Lights off	
turn_down	low AC	
turn_off	lights be off	
turn_off	turn off my radio please	

# Create an Intent Import File

To create an import file for intents, start with a spreadsheet (preferred method) or text editor

1. Organize intents as shown (no heading line!)
  - Utterance, intent
  - Utterance1, intent
  - Utterance 2, intent
2. Save it as a comma delimited file (CSV) with .csv extension example: car\_intents.csv
3. Open CSV file in a text editor to verify no special characters or symbols
4. Try it out – Open the **ground-truth.xls** and create an **intents.csv** from the **intent extract** sheet.

A	B
Hi watson	greetings
hiya	greetings
Hola	greetings
hola como estas	greetings
How are you doing?	greetings
howdy	greetings



The screenshot shows a Notepad window titled "car\_intents.csv - Notepad". The window displays a list of utterances and their corresponding intents in CSV format. The data is as follows:

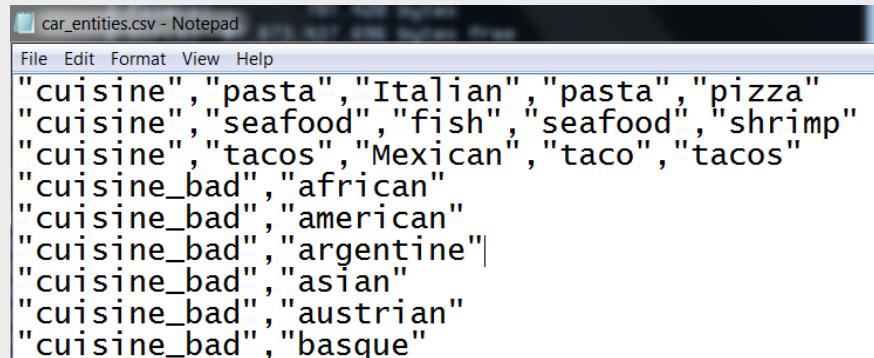
Utterance	Intent
"Hi watson"	"greetings"
"hiya"	"greetings"
"Hola"	"greetings"
"hola como estas"	"greetings"
"How are you doing?"	"greetings"
"howdy"	"greetings"

# Creating an Entity Import File

To create an import file for entities, start with a spreadsheet (preferred method) or text editor.

1. Organize entities as shown (no heading line!)
  - entity\_name, entity\_value, synonym, synonym
  - entity\_name1, entity\_value1, synonym1, synonym2
2. Save it as a comma delimited file (CSV) with .csv extension example: car\_entities.csv
3. Open CSV file in a text editor to verify no special characters or symbols
4. Try it out – Open ground-truth.xls from your lab folders and create an entities.csv from the **entity extract** sheet.

A	B	C	D	E
cuisine	pasta	Italian	pasta	pizza
cuisine	seafood	fish	seafood	shrimp
cuisine	tacos	Mexican	taco	tacos
cuisine_bad	african			
cuisine_bad	american			
cuisine_bad	argentine			
cuisine_bad	asian			
cuisine_bad	austrian			
cuisine_bad	basque			



The screenshot shows a Windows Notepad window titled "car\_entities.csv - Notepad". The window displays a list of entity-value pairs separated by commas. The list includes both valid entries (like "cuisine", "pasta", "Italian") and invalid entries (like "cuisine", "bad", "african"). The invalid entries are likely placeholder values for the "synonym" field in the original table.

```
"cuisine","pasta","Italian","pasta","pizza"
"cuisine","seafood","fish","seafood","shrimp"
"cuisine","tacos","Mexican","taco","tacos"
"cuisine_bad","african"
"cuisine_bad","american"
"cuisine_bad","argentine"
"cuisine_bad","asian"
"cuisine_bad","austrian"
"cuisine_bad","basque"
```

# Import Intents in Watson Assistant Tooling

1. Login to IBM Cloud <https://console.ng.bluemix.net/>
  2. Select your assistant service
  3. Launch the tooling
  4. Select the **Skills** tab
  5. Select the previously created skill.
  6. On the Intents page - Click **Import intents**
  7. Click choose a file
  8. Navigate to your lab directory and select your **intents.csv** or the lab provided **car\_intents.csv**
  9. Click **Import**
- Should get a message the file was successfully imported
9. Click **Done**

The screenshot shows the Watson Assistant Tooling interface. At the top, a sidebar shows a skill named "cognitive\_car-VictorP" with a type of "Dialog" and no linked assistants. Below this, the main area has a title "Intents" with a circular icon showing two nodes connected by a line. The text "No intents yet." is displayed. A descriptive paragraph explains what an intent is: "An intent is the goal or purpose of the user's input. Adding examples to intents helps your virtual assistant understand different ways in which people would say them." Below this, a button labeled "Browse available intents to get started using the Content Catalog" is shown. At the bottom, there are three buttons: "Add intent" (highlighted in blue), "Browse content catalog" (in grey), and "Import intents" (in grey).

# Search

The Assistant allows to quickly find information within the Intent, Entities and Dialog pages.

1. To find an entry containing a specific string, click on the magnifying glass in the top right portion of the page.
2. Enter the search string and click Search
3. Review the results.

The screenshot shows the 'Intents' tab selected in the navigation bar. A search bar at the top right contains the text 'good bye'. Below the search bar, a message says 'Showing 1 intent containing: 'good bye''. A list item '#goodbyes' is shown with '1 match'. At the bottom, a 'User example' section displays the text 'good bye'.

**Note:** Search will not be available while the system is indexing the imported data.

# Import Entities in Watson Assistant Tooling

1. Switch to the **Entities** page.
2. On the Entities page - Click **Import**.
3. Click **Choose a File**.
4. Navigate to your lab directory and select your **entities.csv** or the lab provided **car\_entities.csv** Navigate to your lab directory and select **car\_entities.csv**.
5. Click **Import**.
6. Should get a message the file was successfully imported.
7. Click **Done**.

The screenshot shows the 'Entities' tab selected in the top navigation bar. Below it, the 'My entities' section is active, showing a circular icon with a network graph and the text 'No entities yet.' A descriptive paragraph explains what entities are and their purpose. At the bottom right are 'Import' and 'Add entity' buttons.

A modal window displays the import results:

- Import successful
- 10 new **entities** added
- 115 new **values** added
- 203 new **synonyms** added
- 0 new **patterns** added
- 0 duplicates ignored
- 0 lines need revision

# Enable System Entities

1. While on the Entities page
2. Switch to System entities
3. Switch radio button to On for all the entries to enable all system entities

The screenshot shows the 'System entities' tab selected in the Entities navigation bar. A descriptive message states: 'These are common entities created by IBM that could be used across any use case. They are ready to use as soon as you add them. \*System entities cannot be edited.' Below this, there is a table listing two entities:

Name (7) ▼	Description	Status
> <a href="#">@sys-currency</a>	Extracts currency values from user examples including the amount and the unit. (20 cents)	<input checked="" type="checkbox"/> On
> <a href="#">@sys-date</a>	Extracts date mentions (Friday)	<input checked="" type="checkbox"/> On

# Dialog

# Planning Responses (Using Core Intents and Entities)

A **response** is what the Assistant service returns to the end-user based on the intents and entities it recognizes in inputs. Not all answers are text; some are actions.

**Ground Truth** is the grouping of end-user examples with intents.

Watson Assistant service uses the Ground Truth mapping to train its cognitive models.

The **utterances** in Ground Truth are collected from end users in line with question collection best practices.

Intent	Entity Name	Entity Value	Response
greetings			
locate_amenity	amenity		
	amenity	gas	
	amenity	restaurant	
	amenity	restroom	
off_topic			
turn_on	appliance		
	appliance	ac	
	appliance	fan	
	appliance	heater	
	appliance	lights	
	appliance	music	
	appliance	volume	
	appliance	wipers	

# Dialog Design

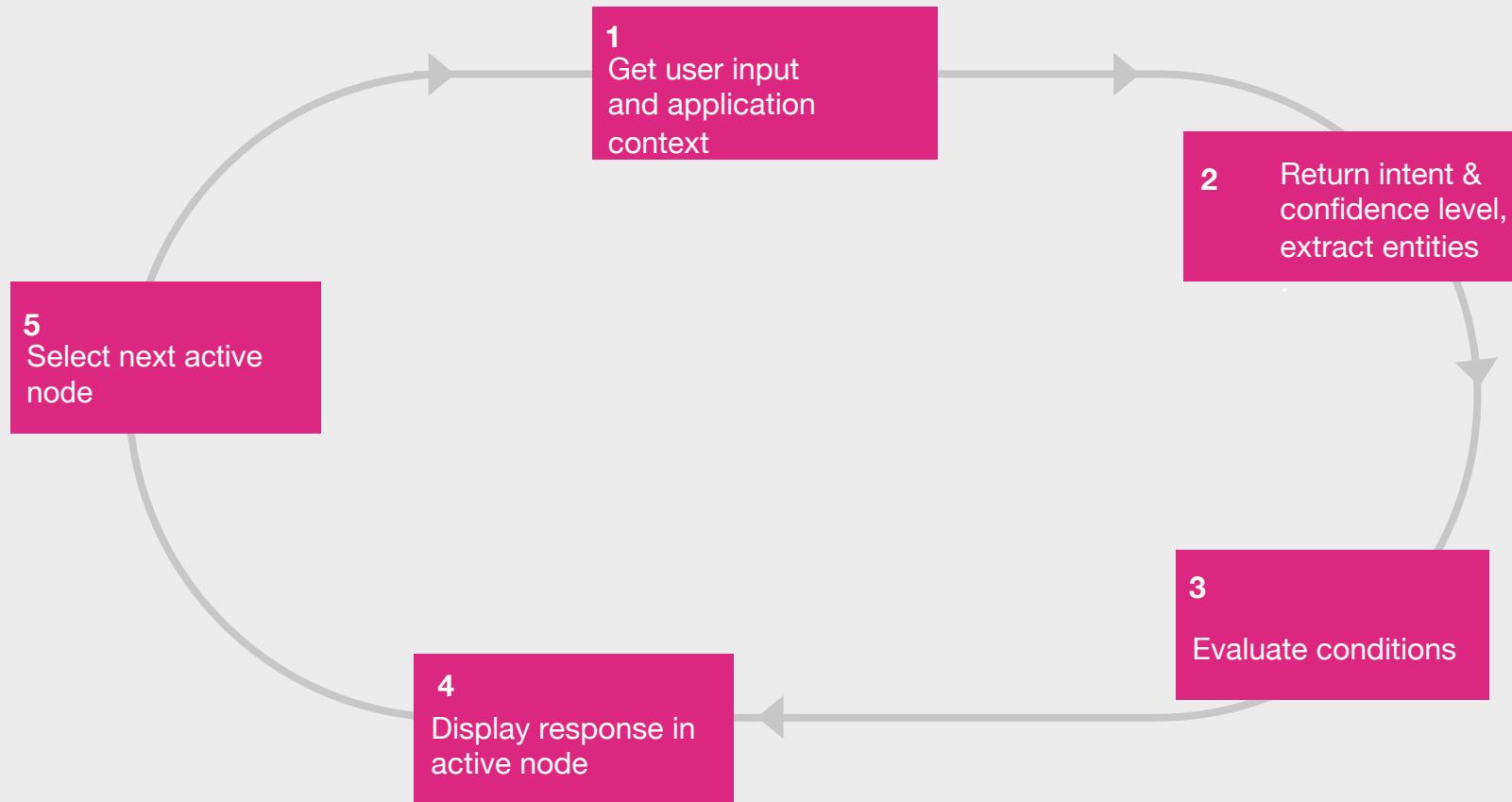
Good dialog design should reflect the solution's positioning to create a coherent and compelling solution that behaves as envisaged by you and the client.

If the dialog design does not consistently reflect the positioning, the solution is less likely to achieve its purpose, deliver value, or meet the client's expectations, so don't take the easy option!

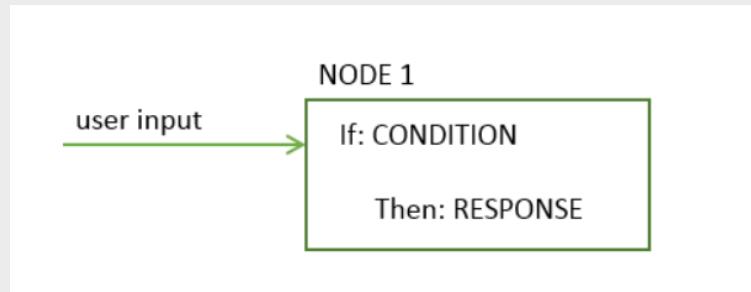
To Reflect the positioning, the dialog design must create behaviors that:

- Help achieve the defined **purpose**
- Consistently express the identified **viewpoint**
- Reflect the specified **proactivity**
- Reflect the **tone** and **personality**

## Dialog – Assistant turn



# Dialog Node



Name this node... Customize

If bot recognizes:  
Enter an intent, entity or context variable...

Then respond with:  
Enter a response...

And finally  
Wait for user input

# Conditions

You can use one or more of the following artifacts in any combination to define a condition:

**Intent:** The simplest condition is a single intent. The node is used if the user's input maps to that intent. Use the syntax `#{intent-name}`. For example, `#weather` checks if the intent detected in the user input is weather. If so, the node is processed.

**Entity:** The node is used when any value or synonym for the entity is recognized in the user input. Use the syntax `@{entity_name}`.

**Entity value:** The node is used if the entity value is detected in the user input. Use the syntax `@{entity_name}:{value}`. Specify a defined value for the entity, not a synonym.

**Context variable:** The node is used if the context variable equation that you specify is true. Use the syntax `$variable_name:value` or `$variable_name == 'value'`

**Special conditions:** `anything_else`, `conversation_start`, `false`, `irrelevant`, `true`, `welcome`

# Special Conditions

Condition Name	Description
anything_else	You can use this condition at the end of a dialog, to be processed when the user input does not match any other dialog nodes. The Anything else node is triggered by this condition.
conversation_start	Like welcome, this condition is evaluated as true during the first dialog turn. Unlike welcome, it is true whether or not the initial request from the application contains user input. A node with the conversation_start condition can be used to initialize context variables or perform other tasks at the beginning of the dialog.
false	This condition is always evaluated to false. You might use this at the top of a branch that is under development, to prevent it from being used, or as the condition for a node that provides a common function and is used only as the target of a Jump to action.
irrelevant	This condition will evaluate to true if the user's input is determined to be irrelevant by the Assistant service.
true	This condition is always evaluated to true. You can use it at the end of a list of nodes or responses to catch any responses that did not match any of the previous conditions.
welcome	This condition is evaluated as true during the first dialog turn (when the conversation starts), only if the initial request from the application does not contain any user input. It is evaluated as false in all subsequent dialog turns. The Welcome node is triggered by this condition. Typically, a node with this condition is used to greet the user, for example, to display a message such as "Welcome to our Pizza ordering app."

# Responses

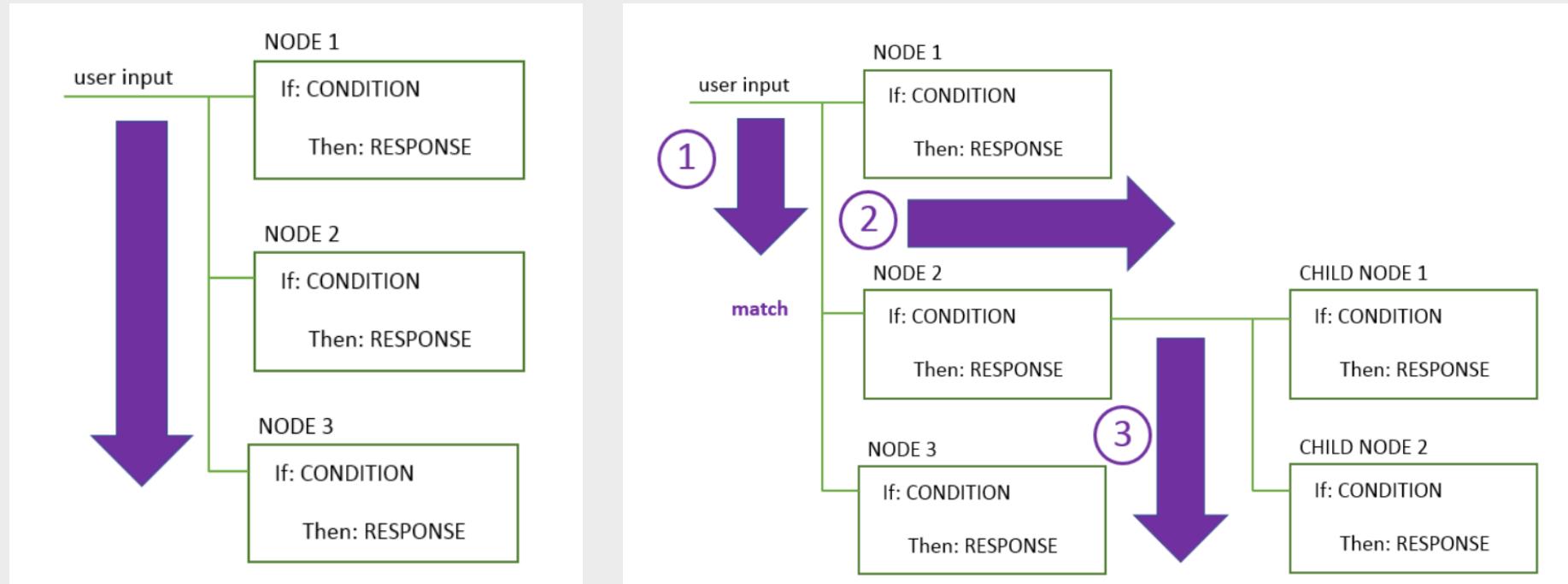
The dialog response defines how to reply to the user.

- **Simple text response** - text that the bot displays to the user
- **Multiple conditioned responses** - different responses, each one triggered by a different condition
- **Complex response** – use the JSON editor to specify the response in the "output":{} property

After making the specified response, you can instruct the Assistant to

- **Wait for user input** - the bot waits for the user to provide new input that the response elicits.
- **Jump to** - use if you wish to bypass waiting for user input and want the conversation to go directly to child nodes or to an entirely different dialog node.

# Dialog – Assistant flow



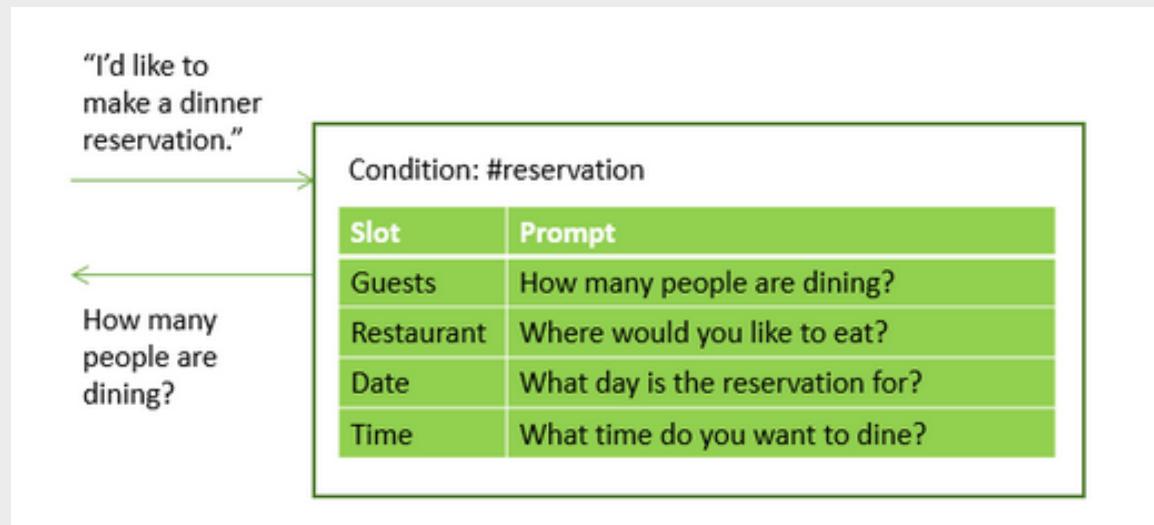
## Context Variables

A context variable is a variable that you define in a node, and optionally specify a default value for. Other nodes or application logic can subsequently set or change the value of the context variable.

- Pass information from the application to the dialog by setting a context variable and passing the context variable to the dialog.
- Pass context from node to node or to update the values of context variables.
- Define a context variable by adding a name and value pair to the {context} section of the JSON dialog node definition.

# Slots

Slots are a configuration option within dialog to collect multiple pieces of information as needed to complete a complex task for a user, such as making a dinner reservation.



# Dialog Lab

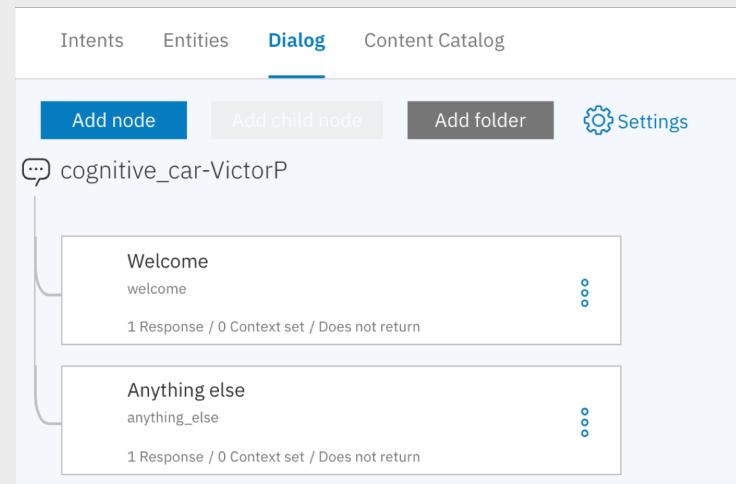
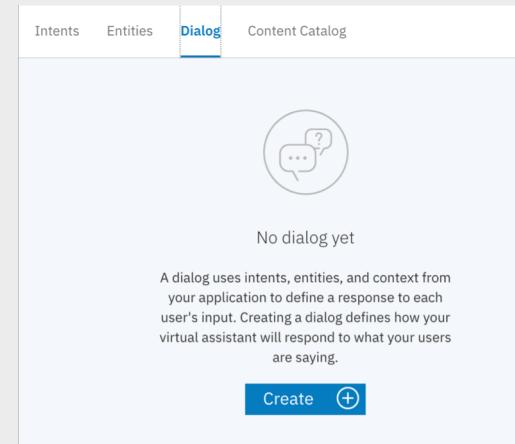
# Create Dialog

1. Switch to the Dialog page.

2. Click Create.

Two nodes will automatically be created for you

- Welcome
- Anything else



# Welcome Node

The welcome node is the first node in the dialog flow – typically a greeting message.

1. Add a second greeting messages to your welcome node

The screenshot shows the Dialogflow interface with a node titled "Welcome". The node configuration includes:

- If bot recognizes:** A condition "welcome" with edit (+) and delete (-) buttons.
- Then respond with:**
  1. Hello. How can I help you?A "Delete" button (⊖) is located to the right of the response list. Below the responses is a placeholder "Add a variation to this response".
- And finally:** A "Wait for user input" button with a dropdown arrow.

# Results – Welcome Node

The screenshot displays a conversational AI builder interface with two main sections: a left sidebar and a right configuration panel.

**Left Sidebar:**

- Buttons: Add node, Add child node, Add folder, Search icon.
- Section: Cognitive Car
- Node List:
  - Welcome**:
    - welcome
    - 1 Response / 0 Context set / Does not return
  - Anything else**:
    - anything\_else
    - 1 Response / 0 Context set / Does not return

**Right Configuration Panel:**

- Welcome**:
  - If bot recognizes: welcome
  - Then respond with:
    1. Hello. How can I help you?
    2. Hi, I'm here to help. What can I do for you?
  - Add a variation to this response
  - Variations are **sequential**. Set to random ⓘ

# Anything\_Else Node

The anything\_else node gets hit if the system has low confidence in the intent, or if no other dialog conditions matched.

1. Add a low-confidence message to your Anything else node
2. Set your response variations to random.

The screenshot shows the Dialog builder interface with the 'Dialog' tab selected. On the left, under 'cognitive\_car-VictorP', there are two nodes: 'Welcome' (white background) and 'Anything else' (light blue background). The 'Anything else' node is currently selected. On the right, the configuration pane shows the node name 'Anything else' and its condition 'If assistant recognizes: anything\_else'. Below that, under 'Then respond with:', there is a 'Text' dropdown containing four response variations: 'I didn't understand. You can try rephrasing.', 'Can you reword your statement? I'm not understanding.', 'I didn't get your meaning.', and 'Enter response variation'. A note at the bottom states 'Response variations are set to sequential. Set to random' with a help icon.

# Results – Anything\_Else Node

The screenshot shows a conversational AI configuration interface. On the left, a tree view displays two nodes under a root node named "Cognitive Car". The first node is "Welcome" (welcome), which has 1 Response / 0 Context set / Does not return. The second node is "Anything else" (anything\_else), which also has 1 Response / 0 Context set / Does not return. Both nodes have three-dot ellipsis icons for more options.

On the right, the "Anything else" node is selected, revealing its response editor. The title bar says "Anything else". The editor contains a single response: "anything\_else" followed by a minus sign (-) and a plus sign (+). Below this, the text "Then respond with:" is followed by five variations:

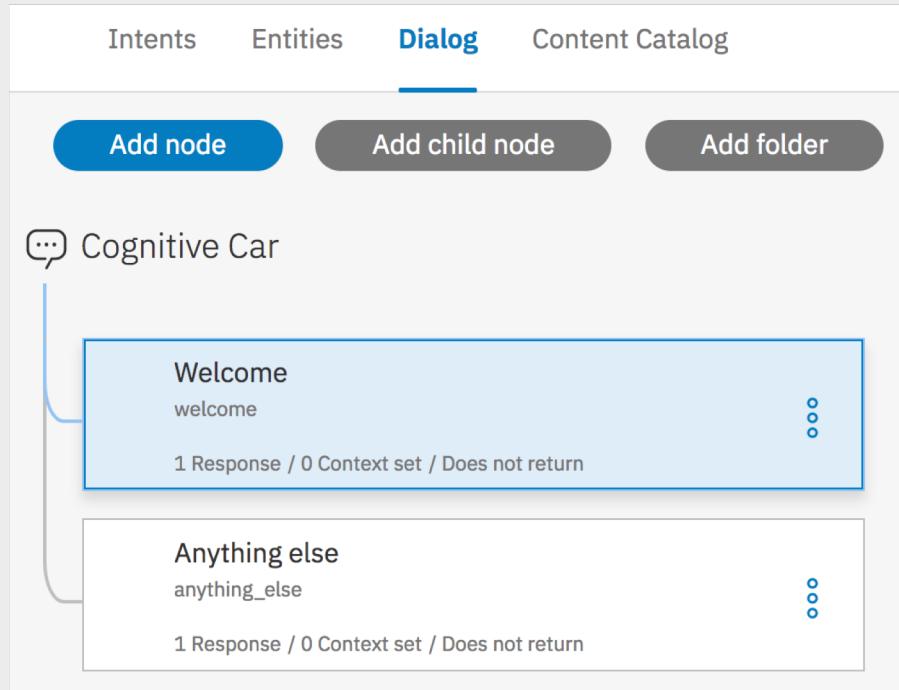
- I didn't understand. You can try rephrasing. (minus sign -)
- Can you reword your statement? I'm not understanding. (minus sign -)
- I didn't get your meaning. (minus sign -)
- Can you please try again? (minus sign -)
- Add a variation to this response

At the bottom, a note states "Variations are set to random. Set to sequential ⓘ".

# Create Weather Node

Add a node under the welcome node

1. Select the welcome node
2. Select Add node
3. Name the node weather
4. Condition on the #weather intent
5. Provide the response “Unfortunately I don't know much about the weather. I'm still learning.”



# Results – Weather Node

Add node   Add child node   Add folder  

Cognitive Car

Welcome  
welcome  
  
1 Response / 0 Context set / Does not return

weather  
#weather  
  
1 Response / 0 Context set / Does not return

Anything else  
anything\_else  
  
1 Response / 0 Context set / Does not return

weather

If bot recognizes:  
`#weather`

Then respond with:

1. Unfortunately I don't know much about the weather. I'm still learning.

Add a variation to this response

And finally

Wait for user input

# Multi-conditional Response Node

All of your answers for a given intent (regardless of # of entities) can usually be stored in a single dialog node.

Multi-conditional responses are part of the standard dialog node

You can continue to add more conditional responses for all intent + entity combinations

Conditional responses can also leverage context conditions

Advanced Response functionality is available for each response (set context per slot)

# Condition Operators

Operator	Description
:	Fuzzy match query operator. Prefix the query term with : if you want to match any value that contains the query term, or a grammatical variant of the query term. Fuzzy matching is available for user input text, response output text, and entity values.
::	Exact match query operator. Prefix the query term with :: if you want to match only values that exactly equal the query term.
:!	Negative fuzzy match query operator. Prefix the query term with :! if you want to match only values that do <i>not</i> contain the query term or a grammatical variant of the query term.
::!	Negative exact match query operator. Prefix the query term with ::! if you want to match only values that do <i>not</i> exactly match the query term.
<=,>=,>,<	Comparison operators. Prefix the query term with these operators to match based on arithmetic comparison.

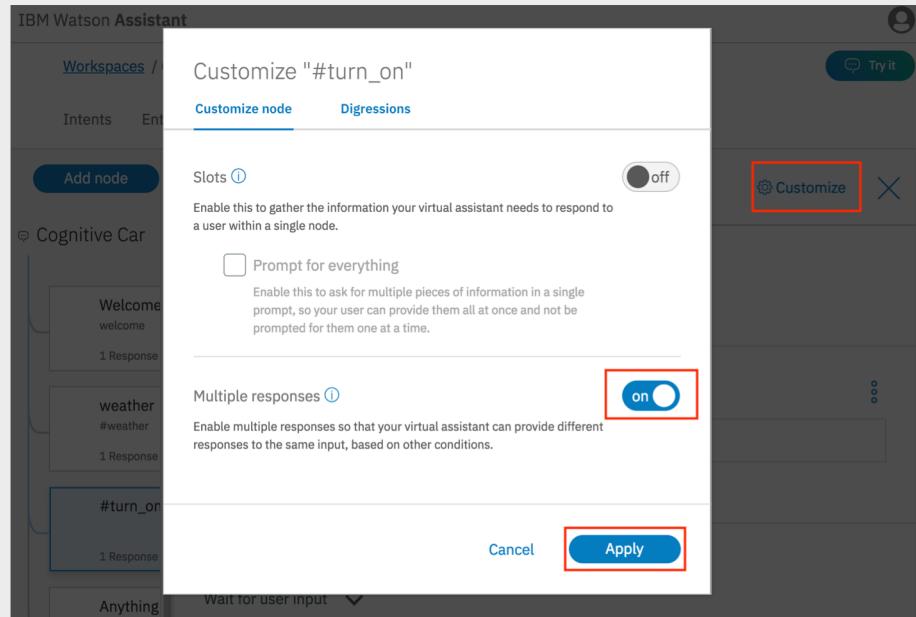
\	Escape operator. Use in queries that include control characters that would otherwise be parsed as operators. For example, \!hello would match the text !hello.
""	Literal phrase. Use to encloses a query term that contains spaces or other special characters. No special characters within the quotation marks are parsed by the API.
~	Approximate match. Append this operator followed by a 1 or 2 to the end of the query term to specify the allowed number of single-character differences between the query string and a match in the response object. For example, car~1 would match car, cat, or cars, but it would not match cats. This operator is not valid when filtering on log_id or any date or time field, or with fuzzy matching.
*	Wildcard operator. Matches any sequence of zero or more characters. This operator is not valid when filtering on log_id or any date or time field.
( ), [ ]	Grouping operators. Use to enclose a logical grouping of multiple expressions using Boolean operators.
	Boolean or operator.
,	Boolean and operator.

# Condition Operator Examples

Description	Query
The date of the response is in the month of July 2017.	<code>response_timestamp&gt;=2017-07-01, response_timestamp&lt;2017-08-01</code>
The timestamp of the response is earlier than 2016-11-01T04:00:00.000Z.	<code>response_timestamp&lt;2016-11-01T04:00:00.000Z</code>
The user input text contains the word "order" or a grammatical variant (for example, orders or ordering).	<code>request.input.text:order</code>
An intent name in the response exactly matches place_order.	<code>response.intents:intent::place_order</code>
An entity name in the response exactly matches beverage.	<code>response.entities:entity::beverage</code>
The user input text does not contain the word "order" or a grammatical variant.	<code>request.input.text:!order</code>

# Create a Multi-conditional Response Node

1. Add a node under the weather node
2. Name the node turn on
3. Condition on the #turn\_on intent
4. Click on 
5. Enable Multiple responses
6. Click Apply



# Create a Multi-conditional Response Node Continued

## 1. Add conditional responses

- lights @appliance:lights
- wipers: @appliance:wipers

The screenshot shows a configuration interface for a multi-conditional response node. At the top, there is a title bar with the text "Turn on" and a "Customize" button with a gear icon. To the right of the title bar is a blue "X" button.

Below the title bar, the text "If bot recognizes:" is displayed, followed by a condition entry "#turn\_on". There are three buttons next to it: a minus sign (-), a plus sign (+), and a refresh/circular arrow icon.

Underneath this section, the text "Then respond with:" is shown. This leads into a table-like structure where conditions and responses are mapped:

If bot recognizes	Respond with
1 @appliance:lights	ok turning on the lights for you. <span style="float: right;">⚙️ 🗑️</span>
2 @appliance:wipers	ok turning on the wipers for you. <span style="float: right;">⚙️ 🗑️</span>

At the bottom left of this section is a button labeled "+ Add response".

# Test it Out!

Open your Try it out panel, and click clear.

Ask a question that should match to that intent node and provide the conditional response.

– I.e. "Turn on the Wipers"

The screenshot shows the 'Try it out' panel from a conversational interface. At the top right are 'Clear' and 'Manage Context' buttons, with a notification badge showing '1'. The main area has a dark background with light-colored text boxes for messages and code snippets.

**Intent Node 1:**

- Text: 'lights on'
- Code: '#turn\_on' (highlighted with a blue border)
- Code: '@appliance:lights' (with a dropdown arrow icon)

**Response 1:**

Hi, I'm here to help. What can I do for you?  
ok turning on the lights for you.

**Intent Node 2:**

- Text: 'turn on the wipers'
- Code: '#turn\_on' (highlighted with a blue border)
- Code: '@appliance:wipers' (with a dropdown arrow icon)

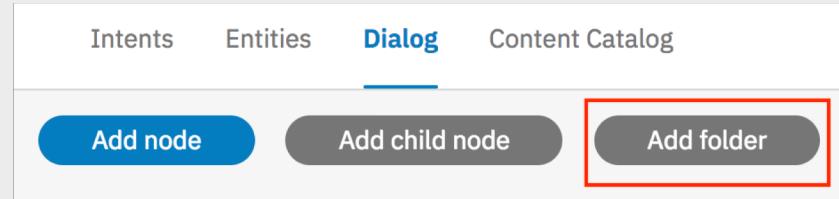
**Response 2:**

ok turning on the wipers for you.

# Folders

To help manage the Dialog flows, Watson Assistant allows you to create folders and store content within folders.

If you would like to hide or disable a set of logic, you could place it within a folder and set the condition on the folder to false.



The screenshot shows the Watson Assistant Dialog flow editor. On the left, there's a tree view of dialog nodes under a folder named 'Cognitive Car'. The folder 'Disabled logic' is highlighted with a blue box. It contains one node named 'schedule service' with the condition 'false' set. The main panel shows the details of this node, including its name, condition, and children (which are currently empty).

Dialog node name	If bot recognizes	Children
schedule service	#schedule_service	0

## Slots

In this lab, we'll use slots to collect multiple pieces of information as needed to complete a complex task for a user, such as scheduling service for your cognitive car.

### Condition #schedule\_service

Slot	Prompt
Service Type	What service do you want done?
Date	What day do you want to come in?
Time	What time on \$date do you want to come in?

# Schedule Service Node

1. Add a node under the welcome node.
2. Name the node schedule service
3. Click the Customize and enable Slots

The screenshot shows the 'Customize' interface for a node named 'schedule service'. At the top, there is a search bar containing 'schedule service' and a 'Customize' button. Below the search bar, the title 'Customize "schedule service"' is displayed, followed by two tabs: 'Customize node' (which is selected) and 'Digressions'. The main content area is titled 'Slots ⓘ'. It contains a descriptive text: 'Enable this to gather the information your virtual assistant needs to respond to a user within a single node.' To the right of this text is a blue toggle switch with the word 'on' next to it, which is highlighted with a dashed box. Below this section is another configuration option: a checkbox labeled 'Prompt for everything' with a descriptive text underneath: 'Enable this to ask for multiple pieces of information in a single prompt, so your user can provide them all at once and not be prompted for them one at a time.'

## Configure the Slots

1. Condition on #schedule\_service
2. Add the slots as shown in the table below

Slot	Check for	Save it as	If not present, ask	Required?
1	@servicetype	\$servicetype	What service do you want done?	Yes
2	@sys-date	\$date	What day do you want to come in?	Yes
3	@sys-time	\$time	What time on \$date do you want to come in?	Yes

4. Respond with: I'll schedule your \$servicetype for \$date at \$time.

# Results – Schedule Service slot

schedule service Customize X

If bot recognizes:

#schedule\_service - +

Then check for:

① Manage handlers

Check for	Save it as	If not present, ask	Type
1 @servicetype	\$servicetype	What service do you	Required <span>⚙️</span> <span>trash</span>
2 @sys-date	\$date	What day do you w	Required <span>⚙️</span> <span>trash</span>
3 @sys-time	\$time	What time on \$date	Required <span>⚙️</span> <span>trash</span>

➕ Add slot

Then respond with:

1. I'll schedule your \$servicetype for \$date at \$time.

Add a variation to this response

# Test it Out!

1. Open your Try it out panel and click clear for new conversation.
2. Try and schedule service.

Try it out

Clear Manage Context 13

Hi, I'm here to help. What can I do for you?

I'd like an oil change this friday

#schedule\_service

@servicetype:Oil change

@sys-date:2017-09-15

What time on 2017-09-15 do you want to come in?

8 am

#goodbyes

@sys-time:08:00:00

@sys-number:8

I'll schedule your Oil change for 2017-09-15 at 08:00:00.

Try it out

Clear Manage Context 13

Hi, I'm here to help. What can I do for you?

I'd like to schedule service

#schedule\_service

oil change

#schedule\_service

@servicetype:Oil chanqe

When do you want to come in?

friday

#goodbyes

@sys-date:2017-09-15

What time on 2017-09-15 do you want to come in?

8 am

#goodbyes

@sys-time:08:00:00

@sys-number:8

I'll schedule your Oil change for 2017-09-15 at 08:00:00.

# Configure Slot Handlers

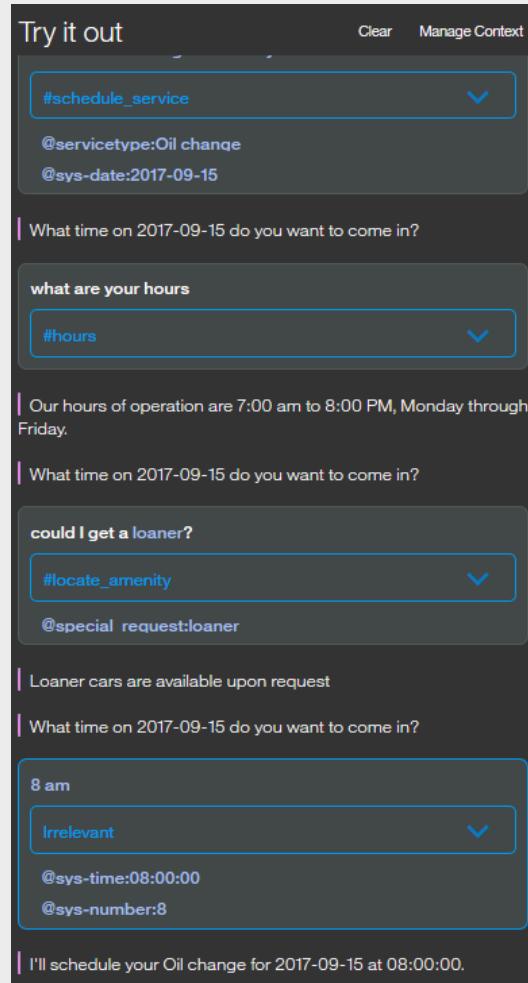
1. Edit the Schedule Service node
2. Click on Manage Handlers
3. Add handlers to respond to questions about loaner cars and hours

Handler	Condition on	Response
1	@special_request:loaner	Loaner cars are available upon request.
2	#hours	Our hours of operation are 7:00 AM to 8:00 PM, Monday through Friday.

# Test it Out!

1. Open your Try it out panel and click clear for new conversation.
2. Schedule service and verify your handlers are working.

**Note:** You have to be within the schedule service process to receive the answer provided by the Handler.

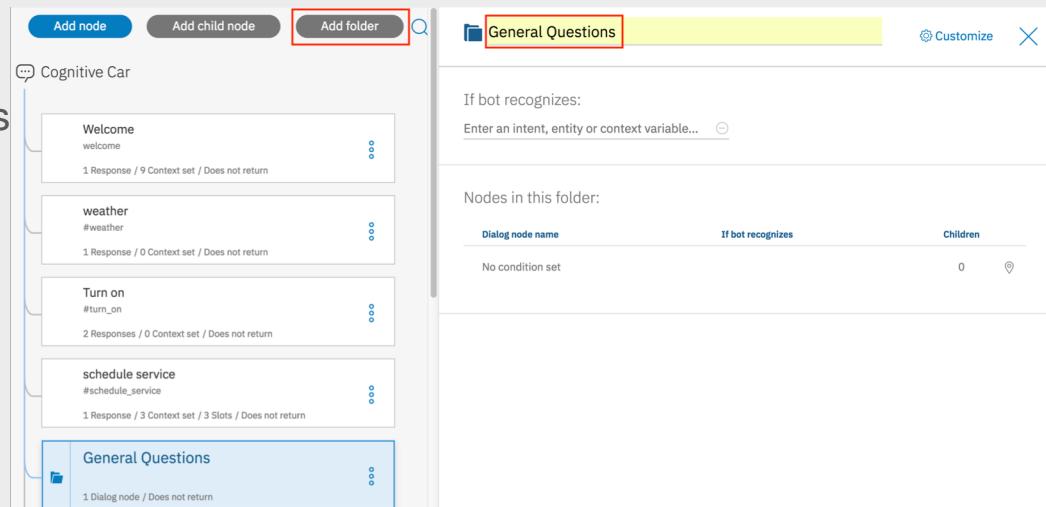


# Digression

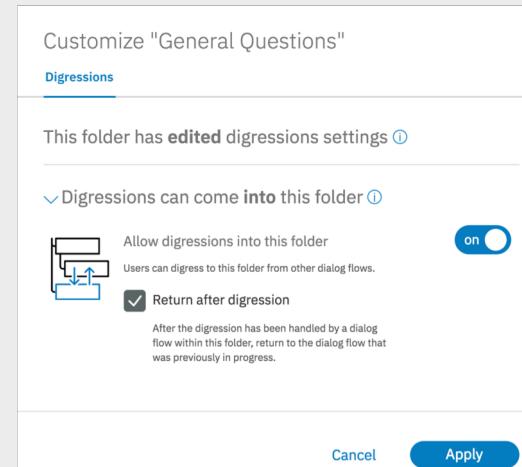
Digression is an alternative to Slot Handlers that can be reused in multiple places. Digressions allows to answer a common question outside the “schedule service” process and then return the user to where they left off in the process.

First we'll create a new folder to store common questions and enable digression.

1. Click the schedule service node and click Add Folder, enter “General Questions” as the folder name.
2. Select the Customize and Turn on “Digression” as well as check the “Return after digression” option.



The screenshot shows the dialog editor interface for a bot named "Cognitive Car". At the top, there are buttons for "Add node", "Add child node", and "Add folder". The "Add folder" button is highlighted with a red box. Below the buttons, the conversation tree is displayed. A new folder named "General Questions" has been added. The "Nodes in this folder" section shows that there are no nodes currently in the folder. The "If bot recognizes:" field is empty. The "Customize" button is visible at the top right.

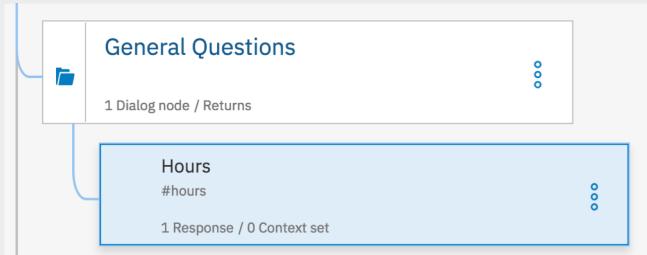


This screenshot shows the configuration dialog for the "General Questions" folder. The "Digressions" tab is selected. It displays a message stating that the folder has edited digressions settings. Under the "Digressions" section, there is a toggle switch labeled "on" which is turned on. Below the switch, there are two options: "Allow digressions into this folder" (with a description that users can digress to this folder from other dialog flows) and "Return after digression" (which is checked). A note below explains that after a digression is handled, the user will return to the previous dialog flow. At the bottom, there are "Cancel" and "Apply" buttons.

# Digression Continued

Next, we'll add a question within the General Questions folder.

1. Select the General Questions folder and click the Add Child Node button.
2. Enter “hours” as the condition and “Our hours of operation are 9:00 AM to 5:00 PM.” as the response.



This screenshot shows the configuration for the "Hours" dialog node. At the top, the node name "Hours" is displayed with a "Customize" link. Below it, under "If bot recognizes:", there is a text input field containing "#hours" with a minus sign and a plus sign button next to it. Under "Then respond with:", there is a list item: "1. Our hours of operation are 9:00 AM to 5:00 PM." with a minus sign button to its right. At the bottom, there is a placeholder text "Add a variation to this response". Ellipses are present on the right side of the configuration sections.

# Digression Continued

## Remove the Slot Handler for “#hours”

1. Select the Schedule Service node
2. Click on Manage Handlers
3. Remove handler for “#hours”

Manage handlers for "schedule service"

Handlers are how your virtual assistant will respond when the user's answer to a prompt is not found. These handlers will be checked before trying the "Not found" responses in a slot.

If answer to any prompt is not found and:

If bot recognizes	Respond with	More
1 @special_request:loaner	Loaner cars are available upon requ	
2 #hours	Our hours of operation are 7:00 AM	

[Cancel](#) [Save](#)

# Digression Continued

The final step is to turn on digression away for the “Schedule Service” process.

1. Select the Schedule Service node and click Customize.
2. Select the Digression tab and turn on “Digressions can go **away from** this node” as well as select the “Only digress from slots to nodes that allow returns” option.

schedule service

Customize

X

Customize "schedule service"

[Customize node](#) [Digressions](#)

This node has **edited** digressions settings ⓘ

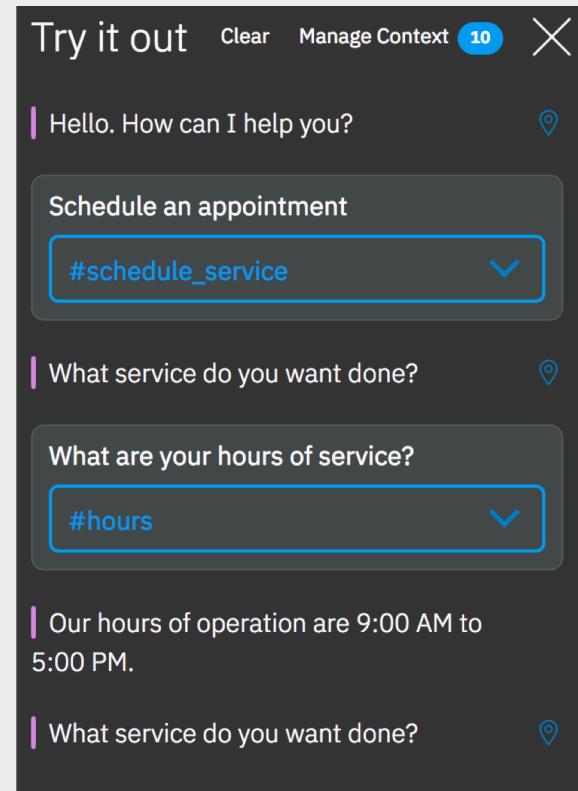
▼ **Digressions can go away from this node** ⓘ

 Allow digressions away while slot filling    
Users can divert the conversation away from this node in the middle of processing slots.

Only digress from slots to nodes that allow returns  
If a user goes off topic, only nodes with digressions that allow returns will be considered.

# Test it Out!

1. Open your Try it out panel and click clear for new conversation.
2. Enter the Schedule service process and ask a question that is part of the general questions folder.
3. The service will digress and answer the general question and then continue with the Schedule service process.



Now you're ready...

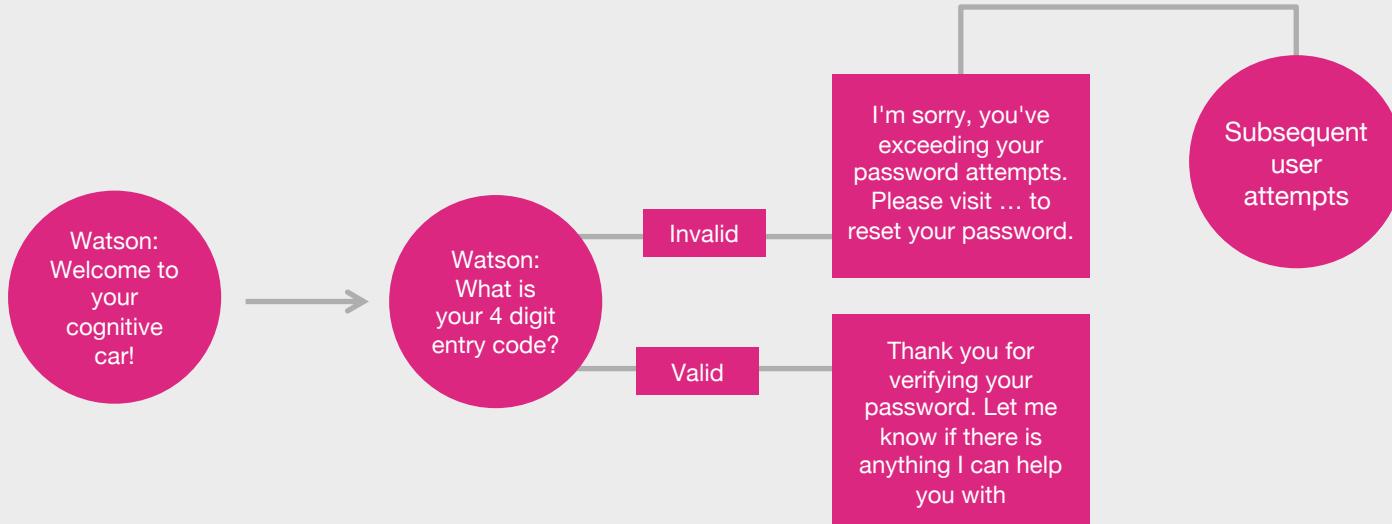
## Designing your first Multi-turn Interaction

Scenario: Let's imagine that your car has a 4 digit password that it needs from you in order to start the engine.

There are several user scenarios to account for:

- Rightful user knows password
- Rightful user attempts password
- Rightful user forgets password
- Child attempts password
- Carjacker attempts password

# Designing a Multi-turn Interaction



# Using Context

To simulate the external system passing in the context (such as passkey), we need to set context in our welcome node

```
1 {
2     "context": {
3         "AConoff": "off",
4         "passkey": 1234,
5         "lightonoff": "off",
6         "musiconoff": "off",
7         "appl_action": "",
8         "heateronoff": "off",
9         "volumeonoff": "off",
10        "wipersonoff": "off",
11        "default_counter": 0
12    },
13    "output": {
14        "text": {
15            "values": [
16                "Welcome to your cognitive car! anything I can assist you with?",
17                "Hello! How can I help you with your cognitive car?"
18            ],
19            "selection_policy": "random"
20        }
21 }
```

# Add Context to the Welcome Node (JSON)

1. Use your favorite text editor to copy the contents of context.txt from your lab files.
2. Select the welcome node.
3. Under Responses, select  to open the JSON editor
4. Under the first { line, paste the snippet at the right to add context.

```
1 {
2   "context": {
3     "AConoff": "off",
4     "passkey": 1234,
5     "lightonoff": "off",
6     "musiconoff": "off",
7     "appl_action": "",
8     "heateronoff": "off",
9     "volumeonoff": "off",
10    "wipersonoff": "off",
11    "default_counter": 0
12  },
13  "output": {
14    "text": {
15      "values": [
16        "Hello. How can I help you?",
17        "Hi, I'm here to help. What can I do for you?"
18      ]
19    }
20  }
}
```

# Add Context to the Welcome Node (Context Editor)

An alternative to adding context via JSON, the context information can be added via the Context editor.

1. Use your favorite text editor to open the context.txt file from your lab files.
2. Select the welcome node.
3. Under Responses, select  to open the Context editor.
4. Enter the information from the context.txt file.

Then set context:	
Variable	Value
\$ AConoff	"off"
\$ appl_action	Add value
\$ default_counter	0
\$ heateronoff	"off"
\$ lightonoff	"off"
\$ musiconoff	"off"
\$ passkey	1234
\$ volumeonoff	"off"
\$ wipersonoff	"off"

# Results – Adding Context

After context is set in a node, it can be referenced in conditions and responses with \$variable-name

Try it

1. Open your Try it out panel
2. Click clear for a new conversation
3. Click Manage Context.

You should see all of your context variables and their current value.

The screenshot shows a dark-themed interface for a 'Try it out' panel. At the top right, there are buttons for 'Clear' and 'Manage Context'. A red box highlights the 'Manage Context' button. Below these, a message from a bot says, 'Hi, I'm here to help. What can I do for you?'. The main area is titled 'Context variables' with a close button 'X' at the top right.

Context variables <span style="font-size: small;">ⓘ</span>	
\$Enter variable name	<input type="text"/>
\$timezone	"America/New_York"
\$AConoff	"off"
\$passkey	1234
\$lightonoff	"off"
\$musiconoff	"off"

# Password Request Node

1. Add a child node from welcome node for Password Request.
2. Name the node Password Request
3. Condition on “true” (create new condition).
4. Response = What is your four digit entry code?

The screenshot shows a conversational AI builder interface with two main panels. On the left, a tree view displays nodes: 'Cognitive Car' (parent), 'Welcome' (child of Cognitive Car), 'Password Request' (child of Welcome), and 'weather' (separate node). The 'Password Request' node is selected, highlighted with a blue border. On the right, the properties for the 'Password Request' node are shown. The title is 'Password Request'. Under 'If bot recognizes:', there is a condition 'true'. Under 'Then respond with:', there is a response template: '1. What is your four digit entry code?'. A placeholder 'Add a variation to this response' is also present. Navigation buttons at the top include 'Add node', 'Add child node', 'Add folder', and a search icon.

Add node   Add child node   Add folder  

Cognitive Car

Welcome  
welcome  
1 Response / 0 Context set / Does not return

>Password Request  
true  
1 Response / 0 Context set

weather

Customize X

Password Request

If bot recognizes:

true - +

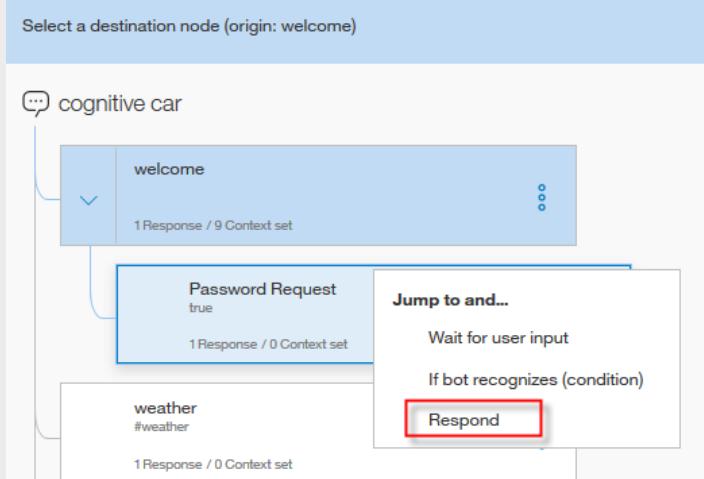
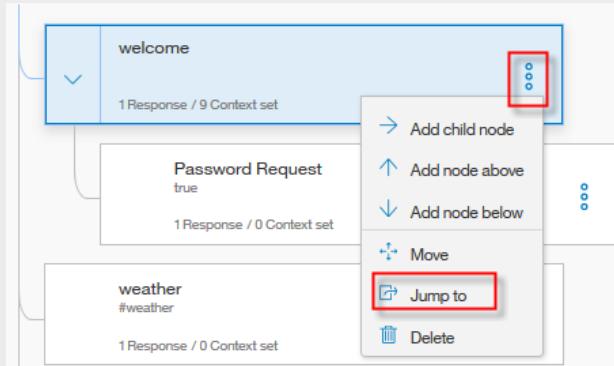
Then respond with:

1. What is your four digit entry code? -

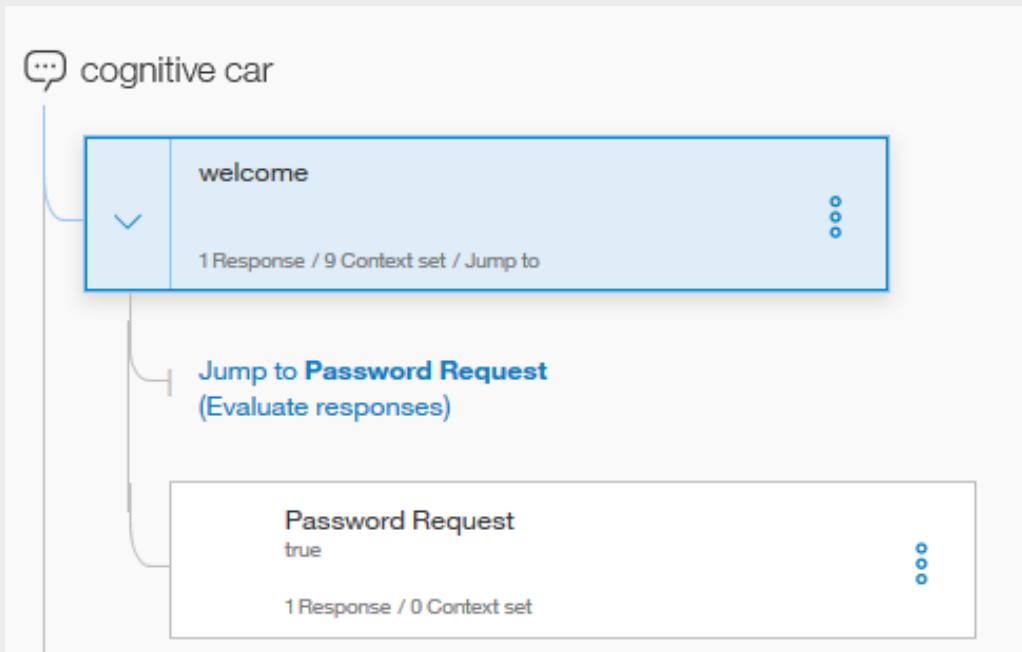
Add a variation to this response

# Create Jump to from Welcome to Password Request

1. Select welcome node
2. Click  and select Jump to
3. Select Password Request as the destination node.
4. Select Respond

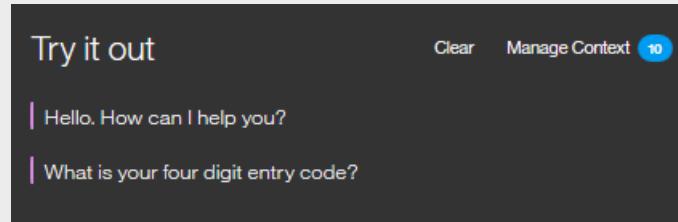


# Results - Password Request Node



# Test it Out!

1. Validate the Password Request node is being called in the Try it out panel.



# Dialog Condition – Checking for a Valid / Invalid Password

## Valid Password

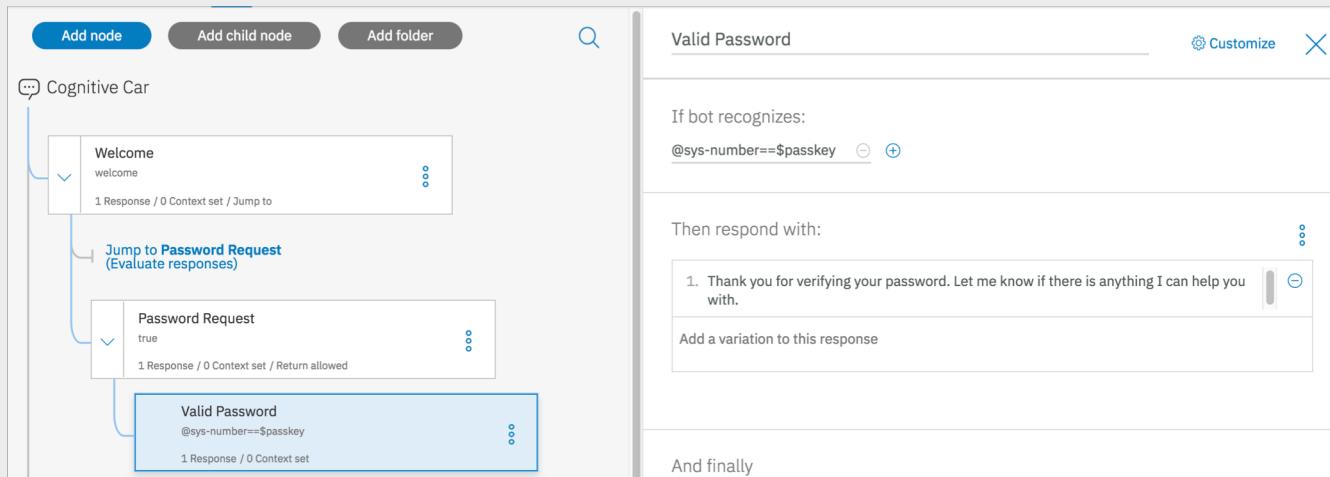
- Create a child of the password request node called Valid Password
- If no child nodes are evaluated as true, dialog jumps back to welcome
- Condition with @sys-number == \$passkey
- Evaluates if the entity number was present and matches to context passkey present\
- Create response for valid password

## Invalid Password

- Create another child node of the password request node, below your valid password node.
- Condition with anything\_else to capture anything that was not hit from prior sibling nodes
- In our scenario, this would happen if the incoming password did not match to the passkey.

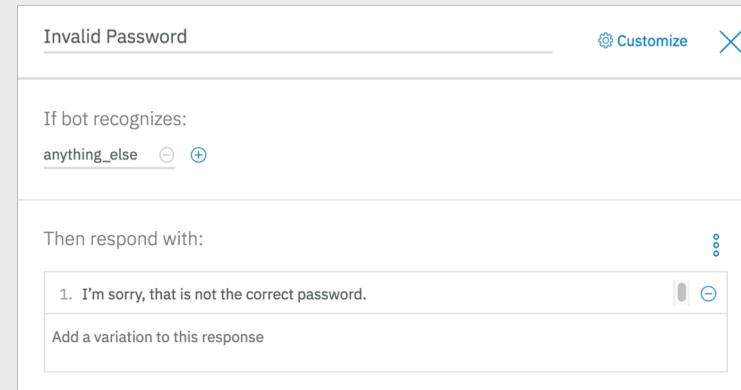
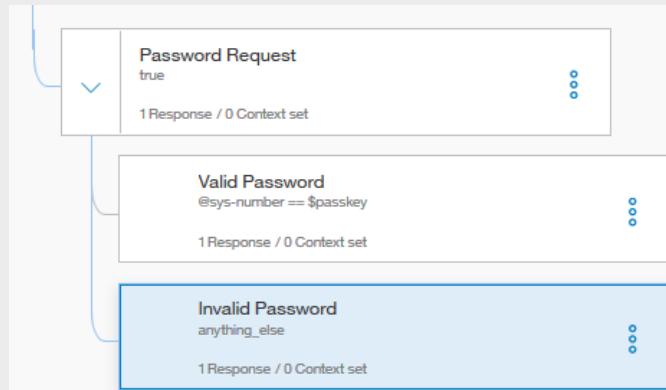
# Create Valid Password Node

1. Add a child node of the password request node
2. Name the node Valid Password
3. Condition with @sys-number == \$passkey
4. Response = Thank you for verifying your password. Let me know if there is anything I can help you with.



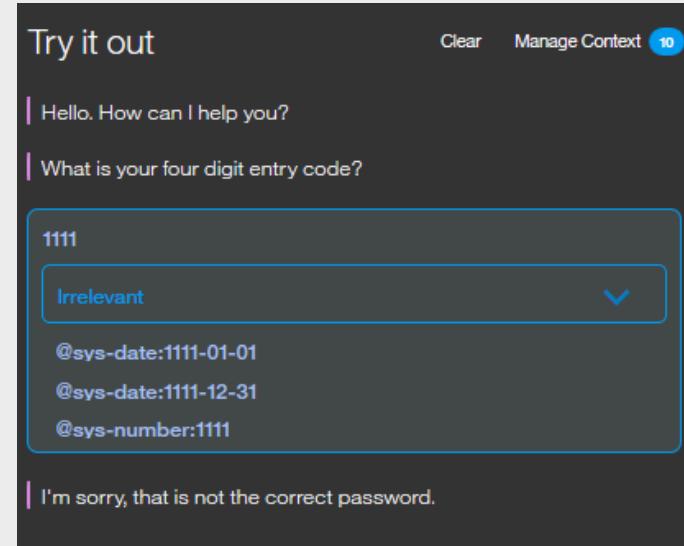
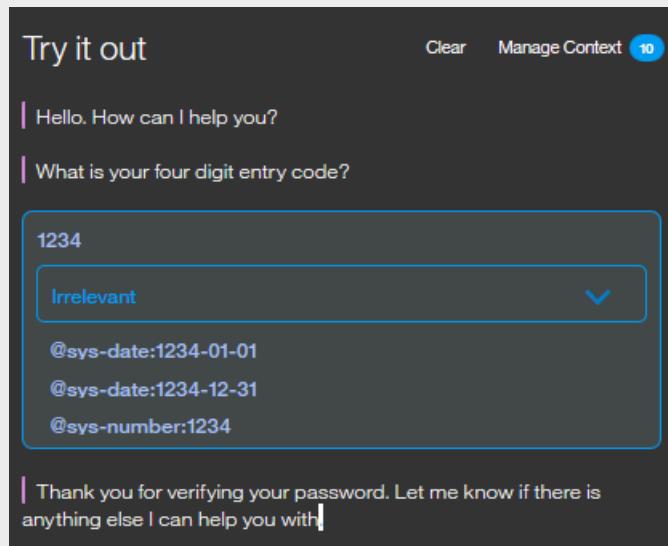
# Create Invalid Password Node

1. Add a second child node of the password request node
2. Name the node Invalid Password.
3. Condition with anything\_else.
4. Response = I'm sorry, that is not the correct password.



# Test Time!

1. Open your Try it out panel and click clear for a new conversation.
2. Test the two possible scenarios:
  - Correct Password (1234)
  - Incorrect Password



# Success!

Congratulations, you have officially asked the user a question, and evaluated their input!

Now one last final touch – when the user inputs an incorrect password, let's add a locking loop so that they cannot use the system

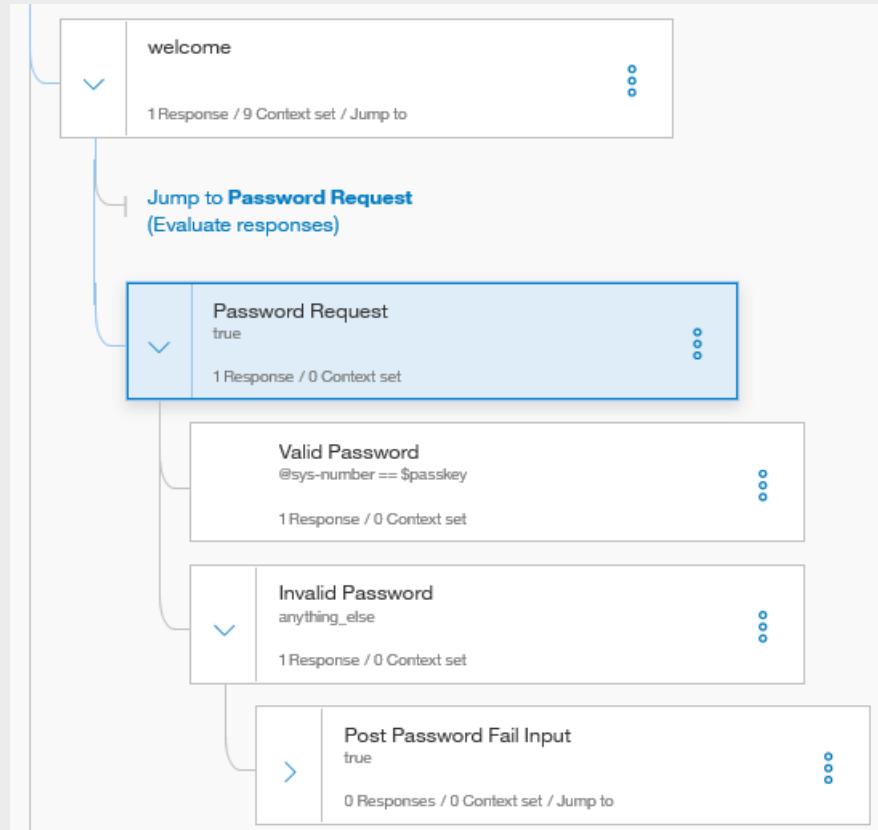
You can add more complex loops utilizing attempt counters, and multiple sequential follow-up questions based on the branching evaluations of inputs.

## Your Turn: Create a Simple Loop

After the system is Waiting for User Input after a password failure, we need to intercept the next incoming message and loop to the Invalid Password message.

1. Create a child of the Invalid Password node (anything\_else)
2. Name the node Post Password Fail Input
3. Condition as *true*; do not add a response
4. Add a JumpTo from this node to the Password Request

# Results – Simple loop



# Try it!

1. Open your Try it out panel and click clear for new conversation.
2. After entering an incorrect password, attempt to send a new message. The system should loop you back

The screenshot shows a "Try it out" panel with two messages from a bot:

Hello. How can I help you?

What is your four digit entry code?

**1111** Incorrect password

Irrelevant

@sys-date:1111-01-01  
@sys-date:1111-12-31  
@sys-number:1111

I'm sorry, that is not the correct password.

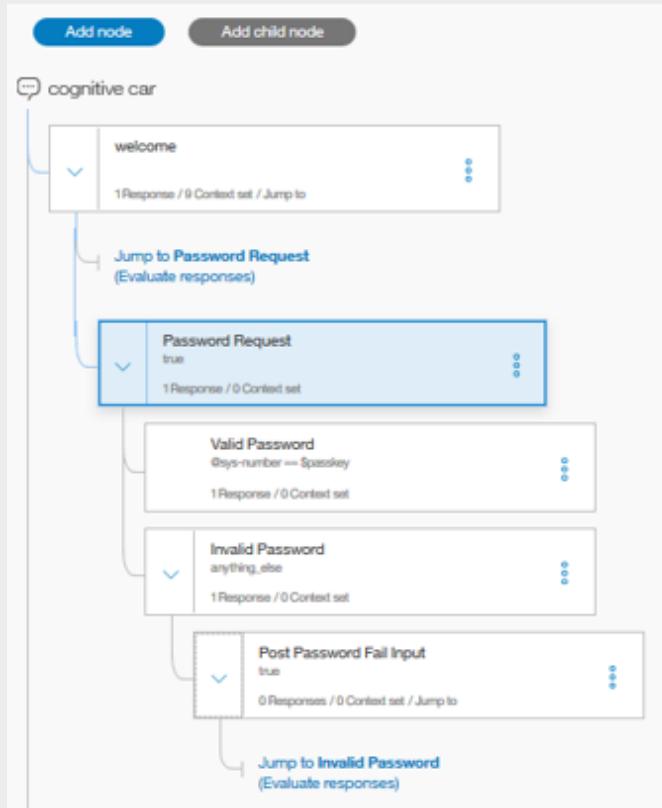
**1234** Correct password, but locked out

Irrelevant

@sys-date:1234-01-01  
@sys-date:1234-12-31  
@sys-number:1234

I'm sorry, that is not the correct password.

# Results – Multi-turn interaction



# So that was simple – What else can it do?

Here is a list of some common operations that are done via context on nodes. For example, if you wanted to extract an email, you could do so:

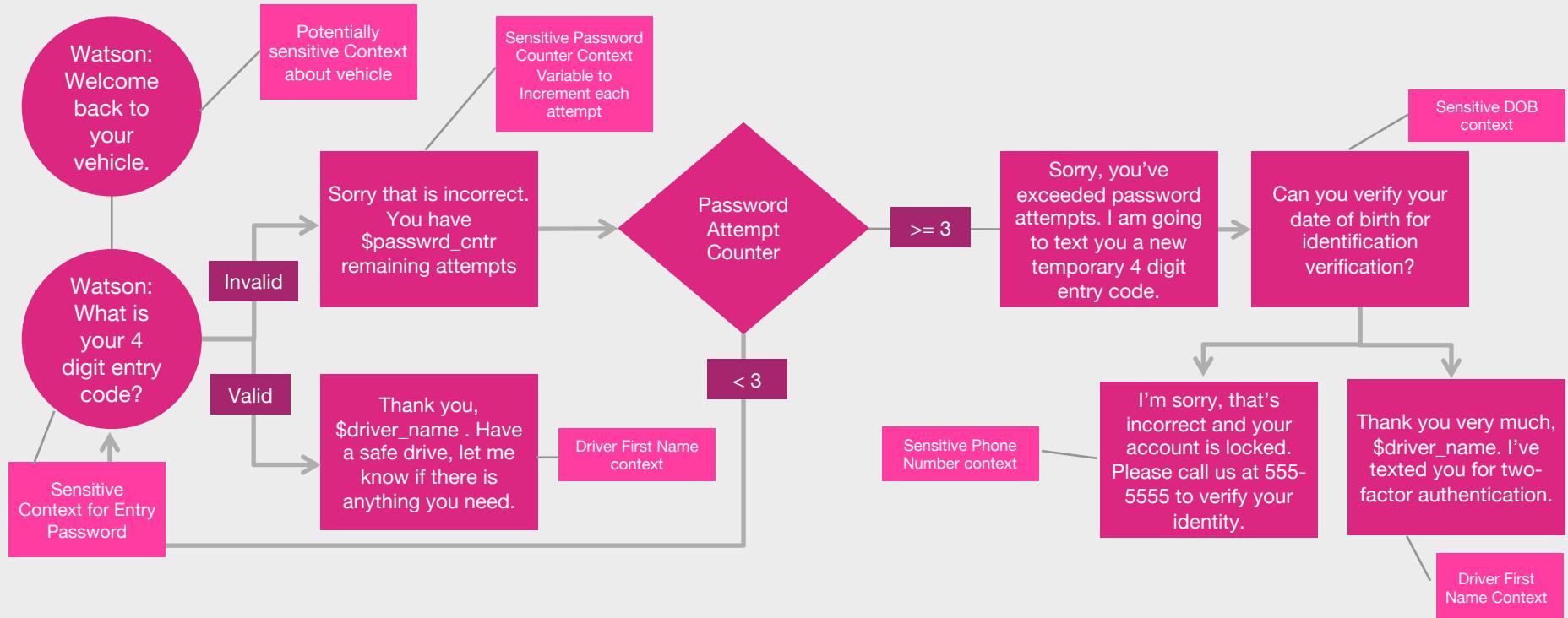
- `Input.text.matches('[0-9]+'), input.text.matches('^[0-9]{4}$')`
- `Input.text.extract(regexp, group_index)`
  - `Input.text.extract('/[d]+',0)`
  - Remember to escape the slashes
- `input.text.contains('yes')`
- `Input.text.equals(String)` and `input.text.equalsIgnoreCase(String)`
- `Input.text.length()`
- `Input.text.isEmpty()` -> check if a context variable or input is set

Evaluation syntax denoted by `<? Some expression ?>` - either via context variable or response\*

- `<? $age_number ?>` - output the variable data
- `<? $toppings.array.append('Cheese') ?>`
- `<? $toppings_array.removeValue('Onion') ?>`
- `<? $context.counter_example + 1`

The screenshot shows a portion of the Watson Assistant interface. At the top, there is a node configuration box with a grey header containing a right-pointing arrow and the text `intents[0].confidence>0.5`. Below this is a dark grey node body with a purple underlined expression `input.text.matches('^[0-9]{4}$')`, a red 'X' icon to its right, and a plus sign icon below it. To the right of the node body is a vertical ellipsis. At the bottom of the interface, there is a section labeled "Watson Response" with a green circular plus sign icon to its right. Below this label is a dark grey box containing the JSON-like text `{ "output": {`.

# Designing More Complex Multi-turn Interactions



# User Interface Lab

# Create the Assistant

1. Select the Deploy option from left navigation menu
2. Select “Add to an assistant”
3. Select “Create new”

Home Skills Assistants

## Assistants

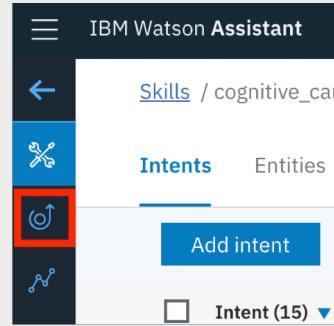
Build a virtual assistant that understands the unique business needs of your customers, and share it across multiple channels.

[Create new](#)

### Time to create an Assistant

An assistant is the user-facing component that manages the flow of information between your skills and your customers.

Click [Create new](#) to get started. [Learn more](#)



[Skills](#) / cognitive\_car-VictorP / Deploy

## cognitive\_car-VictorP

This skill is associated with assistants that you then deploy



Time to deploy your skill through an Assistant

To deploy a skill, first add it to an assistant. The assistant orchestrates the flow of information between your skill and your customers. Next, you can deploy the skilled assistant by using a built-in integration method.

[Add to an assistant](#)

# Add the Skill to the Assistant

1. Give the Assistant a name
2. Select “Create”
3. Select “Add Dialog Skill”

Time to give your Assistant a Skill [Learn more about skills](#)

New Dialog Skill  
Dialog uses Watson natural language processing and machine learning technologies to understand user questions and requests, and address them.

Add Dialog Skill



4. Choose “Add existing skill” from the top navigation menu
5. Select the displayed skill

Add Assistant

Create a new assistant

Name

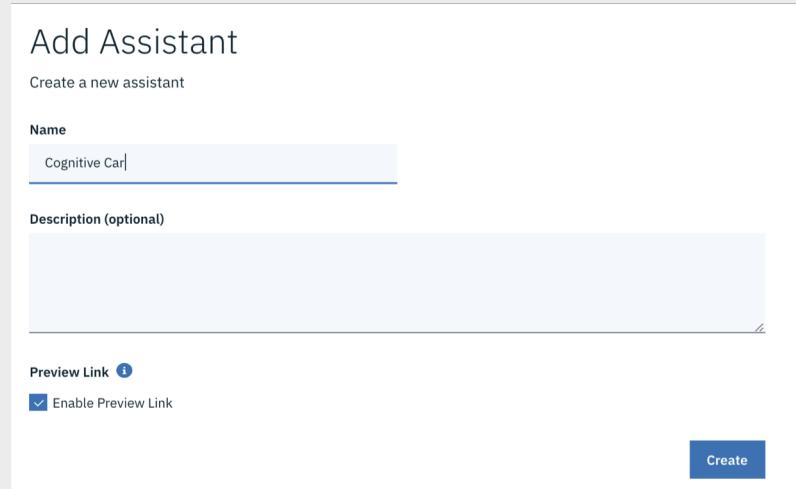
Cognitive Car

Description (optional)

Preview Link i

Enable Preview Link

Create

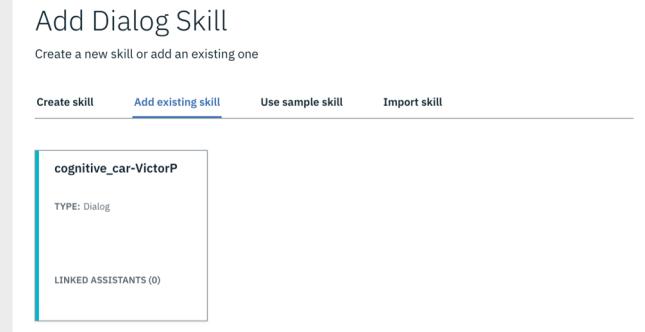


Add Dialog Skill

Create a new skill or add an existing one

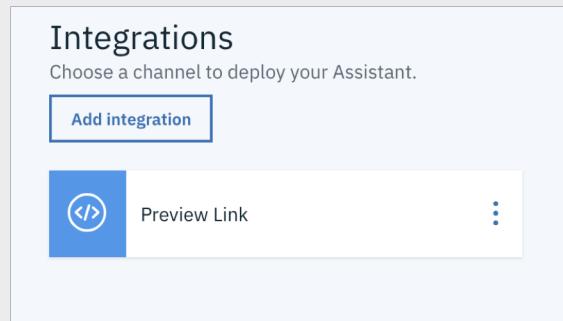
Create skill   Add existing skill   Use sample skill   Import skill

cognitive\_car-VictorP  
TYPE: Dialog  
LINKED ASSISTANTS (0)



# Create a Preview Link

1. Select “Preview Link” from the Integration pane
2. Provide a name for the Preview Link
3. Select the link in the “Try it out and share the link” section of the page
4. Select “Save Changes”



Preview Link Integration

Name

Cognitive Car Preview

Description

A public link you can share to test your assistant outside of the tooling.

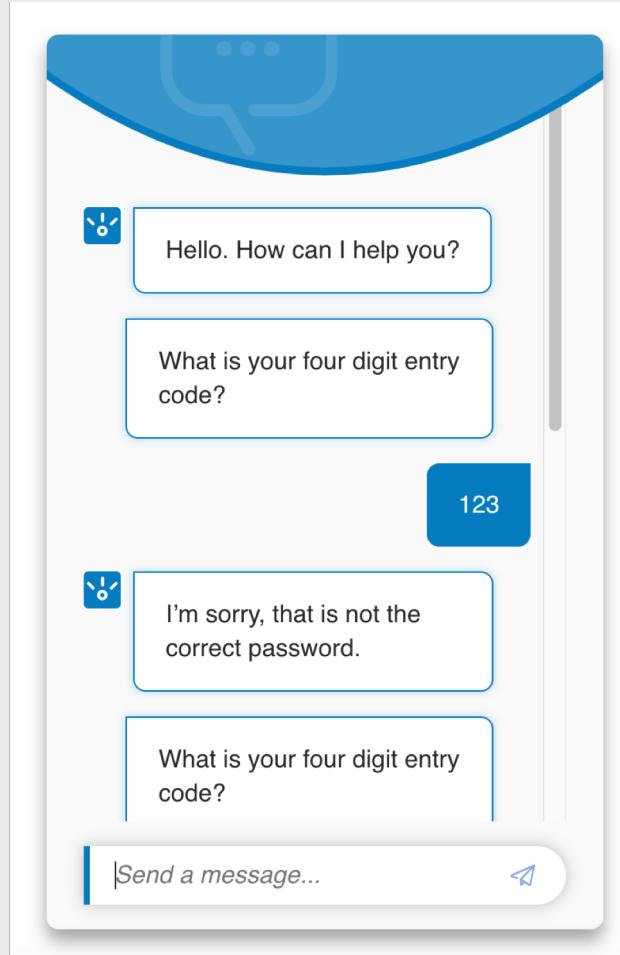
Try it out and share the link  
Use of the assistant embedded in this web page incurs billing charges.

Each message you send to the service through the chat window in the preview web page counts as an API call to the service, and is logged and billed as such.

<https://assistant-chat-us-south.watsonplatform.net/web/public/22cd2d65-686a-45da-ac1e-6d90954aa8b4>

Save Changes

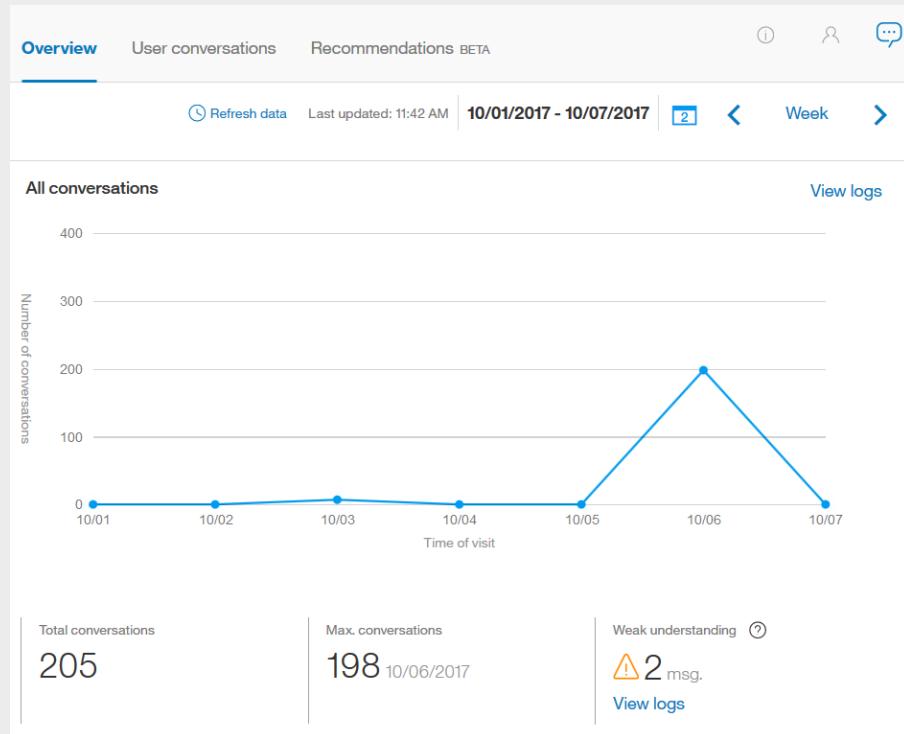
# Test the Assistant



# Improve Lab

# Improve Dashboard

- Tooling to understand your conversational activity at a glance
- Graph of live conversation volumes by day/week/month/year
- Dashboard data providing key metrics on the conversational system including top intents/entities
- Ability to drill into conversation logs directly from the dashboard
- Only shows actual conversations; not interactions made with Try It Out Panel

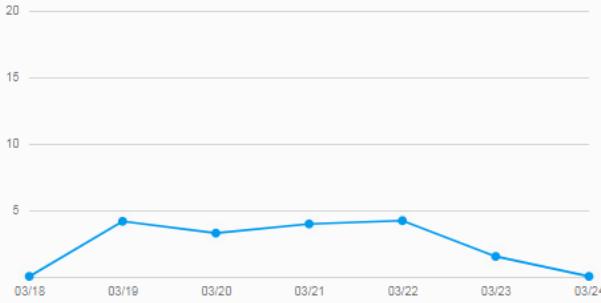


# Improve Dashboard - Continued

Avg. msg. per conversation ⓘ

Avg: 3.78

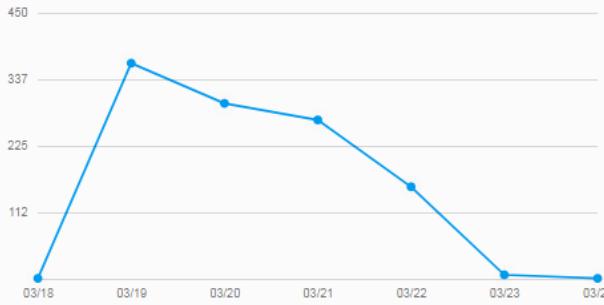
Avg. msg. per conversation



Total messages ⓘ

Total: 1,089

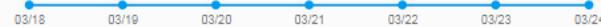
Total messages



Active users ⓘ

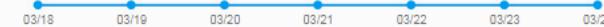
Total: No Data

Active users



Avg. conversations per user ⓘ

Avg. conversations per user



# Improve – User Conversations

Provides a history of conversations with end users. You can view, search, and filter this history, and use it to improve understanding through re-training.

The screenshot shows a user interface titled "User conversations". At the top, there is a search bar labeled "Search user statements..." and dropdown menus for "Intents" and "Entities", both set to "Last 90 days". Below this, a button says "Dates: Last 90 days" and a "Clear filters" link. The main area displays a list of conversations:

Action	User Statement	Response
	Original Understanding #shipping_request @yes:Yes	<b>Yes</b> 04/04/2017 @ 10:00 AM  Great! Would you like to run Detect My Serial Number Auto-Scan or would you like to enter the serial number and machine type manually? <ul><li><va:input>Run Auto-Scan</va:input></li><li><va:input>Enter Manually</va:input></li><li><va:input>I need help</va:input></li></ul>  
	Original Understanding #warranty_info	<b>warranty</b> 04/04/2017 @ 10:00 AM  In order to determine your best service options, I'll need to collect some information from you. Are you currently using your Lenovo machine? <ul><li><va:input>Yes</va:input></li><li><va:input>No</va:input></li></ul>  
	Original Understanding #greeting	<b>hey</b> 04/04/2017 @ 9:59 AM  Hello! I'm your virtual agent. What do you need help with today?  

# Improve – Watson Recommends

- Recommends utterances that will best impact the conversational model
- New recommendations provided on a daily basis
- Leverages unsupervised machine learning approach
- Available only to Premium users.

The screenshot shows a user interface for 'Recommendations BETA'. At the top, there are tabs for 'Overview', 'User conversations', and 'Recommendations BETA' (which is highlighted). On the right side of the header are icons for help, user profile, and more options. Below the header, a search bar contains the phrase 'where can a customer send a payment?'. To the right of the search bar are 'Save' and 'X' buttons. A purple horizontal bar follows. Underneath the search bar, the text 'Edit the intent understanding for this phrase' is displayed. Below this, the phrase is shown again with a link to 'Open conversation'. A section titled 'Most relevant intents' lists five options:

- #Information\_Payments Show Examples
- #Credit\_App Show Examples
- #Timeframe Show Examples
- Select another intent ▼
- Mark as irrelevant ?

A 'Skip to Next' button is located at the bottom right of the list.

# Advanced Topics

# Expression Language / Notation (Java)

Some Java constructs available in dialog

- Accessed by a constructor [new java.util.Date()] or by type locator [T(java.util.Date)] in SpEL
  - <? T(java.lang.Math).random() ?>
  - <? new java.util.Date() ?>
- Example:
  - context": {"max\_num": "<?  
T(java.lang.Math).max(\$var1, \$var2) ?>" }

Inline if/else statements work  
within expression evaluation.

- <?  
\$precanned\_resp.has(entities.small\_talk\_topic  
?.value) ?  
\$precanned\_resp.get(entities.small\_talk\_topic  
?.value) : "Sorry, I don't know"?>

JsonPrimitive	com.google.gson.JsonPrimitive
JsonObject	com.google.gson.JsonObject
JSONArray	com.google.gson.JsonArray
String	java.lang.String
Boolean	java.lang.Boolean
Integer	java.lang.Integer
Long	java.lang.Long
Double	java.lang.Double
Byte	java.lang.Byte
Short	java.lang.Short
Float	java.lang.Float
Character	java.lang.Character
Date	java.util.Date
Random	java.util.Random
Math	java.lang.Math

```
"context": {  
    "precanned_resp": {  
        "joke": "Knock Knock"  
        "weather": "I don't have the  
        weather.",  
        "feeling": "I feel great."  
    }  
}
```